Contract Signing and Employee Portfolio Verification Platform using Blockchain

<table>
<thead>
<tr>
<th>Group Member</th>
<th>UID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ling Lok Tin</td>
<td>3035569055</td>
</tr>
<tr>
<td><strong>Sze Ho Yin (Author)</strong></td>
<td><strong>3035688758</strong></td>
</tr>
</tbody>
</table>

*Supervisor: Dr. Wu Chen Shu*

Submitted on: 18th April, 2023
Abstract

Background check is an important step in the recruitment process. It verifies information on candidates’ resume and assists the company selecting capable candidates for the job. Currently, background checking procedure is done manually, e.g., calling referees, verifying contracts, and checking of documents. Current ways of background checking are labor-intensive and ineffective. The project, therefore, provides a digital solution based on blockchain to improve the background checking progress. Our solution aims to create a web platform that support employment contract signing on blockchain and resume verification. Employment history would be stored on blockchain, and recruiters can verify candidates’ job experiences by inputting some candidates’ basic information. Currently, the project is progressing as planned. Our team completed part of the research work regarding on blockchain, smart contract and method to integrate different components of the web platform. The next step is to continue the research work on smart contract development and implementation of the project.
Acknowledgements

We would like to show our gratitude for our team supervisor, Dr. Wu Chen Shu, who has given us helpful advice on technical ideas and implementation details of the project.

Also, we wish to acknowledge Miss Mable Choi for providing useful tips and skills to help construct the interim report.
# Contents

Abstract i

Acknowledgements ii

List of Figures iv

List of Tables iv

List of Abbreviations v

1. **Introduction** 7
   1.1. Background: Recruitment Process 7
   1.2. Current Employment Verification Process 7
   1.3. Objectives 7
   1.4. Project Contribution 7
   1.5. Outline 8

2. **Literature Review** 9
   2.1. Blockchain 9
   2.2. Non-Fungible Token (NFT) 9

3. **Methodology** 10
   3.1. Introduction 10
   3.2. Workflow 10
      3.2.1. Account Creation 11
      3.2.2. Contract Signing 11
      3.2.3. Tokenization 12
      3.2.4. Contract Termination 13
      3.2.5. Employee Portfolio Verification 13
   3.3. Technology Implementation 15
      3.3.1. Blockchain Development 15
      3.3.2. Smart Contract Development 17
      3.3.3. Database Development 18
      3.3.4. Backend Development 18
      3.3.5. Tokenization of Employment Contract 19
      3.3.6. Frontend Development 19
   3.4. Summary 20

4. **Project Development** 21
   4.1. Overview 21
   4.2. Platform Development 21
      4.2.1. Blockchain Development 22
      4.2.2. Backend Development 25
      4.2.3. Database Development 34
      4.2.4. Frontend Development 36
   4.3. Project Schedule 36
   4.4. Contribution 38
5. **Discussion** 39

5.1. Limitations 39
   5.1.1. Proof of Identity 39
   5.1.2. Legally Binding Power of NFT 40
   5.1.3. Solutions for Lost of Private Key 40

5.2. Future Plan 40
   5.2.1. Implementation of KYC and DAO 40
   5.2.2. Further Improvement of Current Web Interface 41
   5.2.3. Integration of System 41

6. **Conclusion** 42

References 43
List of Figures

3.1 High level overview of account creation process 10
3.2 High level overview of Process of Contract Signing 11
3.3 High level overview of deployment of smart contract 12
3.4 High level overview of tokenization 12
3.5 High level overview of Contract Termination 13
3.6 High level overview of Employee Portfolio Verification 14
4.1 File structure for our platform 21
4.2 File structure for backend and smart contract of our project 21
4.3 File structure for smart contract development in “fyp_server” 22
4.4 Code structure of “fyp_token.sol” file 23
4.5 Example of NFT meta data 24
4.6 File structure for backend development in “fyp_server” 25
4.7 File structure for “account”, “offer”, and “profile” 26
4.8 Code of fyp_server/sever/offer/path.js file 27
4.9 Code of fyp_server/server/offer/offer_view.js file 28
4.10 Code of fyp_server/server/offer/offer_bridge.js file 29
4.11 File Architecture for “service” folder in “fyp_server” 29
4.12 File structure for test in backend development in “fyp_server” 30
4.13 Example of unit test case retrieved from “test_account_bridge.js” file 30
4.14 File structure for database development 34
4.15 Schema of Account table 34
4.16 Schema of Agreement Table 35

List of Tables

4.1 API endpoint 31
4.2 Project Schedule 36
4.3 Work distribution 38
## List of Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>CID</td>
<td>Content Identifier</td>
</tr>
<tr>
<td>DOM</td>
<td>Document Object Model</td>
</tr>
<tr>
<td>ERC</td>
<td>Ethereum Request for Comment</td>
</tr>
<tr>
<td>EVM</td>
<td>Ethereum Virtual Machine</td>
</tr>
<tr>
<td>IPFS</td>
<td>Interplanetary File System</td>
</tr>
<tr>
<td>KYC</td>
<td>Known Your Customer</td>
</tr>
<tr>
<td>NFT</td>
<td>Non-Fungible Tokens</td>
</tr>
<tr>
<td>NPM</td>
<td>Node Package Manager</td>
</tr>
<tr>
<td>TPS</td>
<td>Transaction per second</td>
</tr>
</tbody>
</table>
1 Introduction

1.1 Background: Recruitment Process

Recruiting talented and competent personnel will undoubtedly benefit the company, and vice versa. [1] According to a survey, managers must spend 17% of their time managing underwhelming workers. [2] Not only does selecting incompatible candidates lead to poor performance but can also negatively affecting other employees and eventually harming the performance of the company. Recruitment, however, is a labor-intensive process that includes numerous processes, such as advertising, screening, interviewing, evaluating candidates, making decisions, and background checks. One of the most time and monetary consuming procedure is the employment verification process.

1.2 Current Employment Verification Process

Employment Verification Process is a pre-screening process to validate candidates’ working experience. There are multiple ways for Hiring Team to work on it, such as, calling reference parties, checking verification letters, and checking of contracts and agreements. [3] Yet, this process is labor-intensive, and it takes around 2-5 days for an employment verification process to be completed. [4] Not only does it increase the time frame for both employer and candidates to complete the employment process, but it also requires the company to hire talents for the work and eventually increases the operational cost. Some vendors can help with the employment verification process in lower cost and agile approach, i.e., checking verification letters and contracts. Yet, calling referees would always be the usual practice as during pre-screening stage employers will only obtain candidates’ resume. Besides, obtaining contracts or verification letters could raise potential privacy and credential issues. A digitalized, transparent, and immutable solution would solve the problem. Blockchain would be a good fit to it.

1.3 Objectives

This project intends to offer a web platform that enables companies to confirm the legitimacy of applicants’ resumes. The profile or résumé will be developed on blockchain, taking advantage of its immutability. No candidate could unfairly gain an advantage over the others with an honest and open résumé. The hiring firm could swiftly evaluate applications and exactly look for the candidates they require. It may guarantee the caliber of the new hires and, in turn, the performance of the business going forward. At the same time, this platform would hash the identity of candidates, therefore, only company that are permitted by candidates could view their profiles.

1.4 Project Contribution

It is hoped that this web platform offers a digital solution for hiring process and reduce the time and monetary cost of company in hiring procedures. Not only do the employers benefit from the project, but
job seekers can acquire the latest information on open positions and improve their job-hunting experience. This web platform allows employers and employees to sign their employment contracts on the blockchain and record them. Through retrieving contract signing record on the blockchain periodically, recruiting firm can simultaneously get a quick update of occupied jobs and provide accurate information for job seekers.

1.5 Outline

The report is structured into six chapters.

Chapter one offers an overview of current solutions and situation on hiring procedures and our proposed solutions. It also elaborates the significance of the project. Chapter two is a literature review which explain some important technology concept used in this project. Chapter three showcases the methodology used in the project. The high-level overview of workflow of the web platform and the technology stack use to construct the web platform. Chapter four explains the implementation details of the platform. Chapter five stated limitations of the project with its suggested solutions. It also follows with future plan for the project. Chapter six concludes the report and recall significance of each chapter.
2 Literature Review

This section will introduce some important technology concepts used in our project. They are blockchain (described in section 2.1) and non-fungible token (described in section 2.2).

2.1 Blockchain

Blockchain is a digitally, distributed database shared among the nodes of a computer network. [5] With its distributed features, it can run 24/7 and it can prevent from risk of point of single failure. This means building our platform based on blockchain allows user to use the service at any time. On the other hand, with its cryptographic principles and distributed features, data written on blockchain is immutable and data could be secured with zero-knowledge proof technique. With its immutable feature, it would make it impossible to make a false data written on blockchain. With zero-knowledge proofs, data can be validated by a verifier without leaking any information to the verifier. [6] It allows private documents, i.e., employment contracts, uploading on it as a proof of working experience without leaking any information to third party verifier during employment verification process.

2.2 Non-Fungible Token (NFT)

Cryptocurrency, like Bitcoin and Ethereum, is a fungible token which can be exchanged on a one-to-one basis. It means that one Bitcoin is always the same as another one Bitcoin. On the contrary, Non-Fungible Token (NFT) is a type of digital assets represent ownership of a unique item or a piece of content, in our project NFT is used to represent an employment record. It means that one NFT could be different from another NFT and each NFT is unique and cannot be replaced or exchange on a one-to-one-basis. [7] NFT is created and recorded on blockchain using smart contracts (piece of code that run on blockchain). It makes NFT veritable, secure and transparent as everyone can view the piece of code that guide the function and minting logic of the NFT. [8] Also, as NFT is recorded on blockchain, it inherits the immutable features of blockchain. It makes NFT a good alternative of certificate and employment record. Data representing the jobs and necessary for a job record would be stored as meta data in a NFT in this project. Yet, storing large amount of data to an NFT can greatly increase the gas fee, as a result, increasing our operation cost. Hence, a better approach is to first upload the data to a decentralized data storage and upload the link that directed to the location of the data in decentralized data storage to the meta data of NFT. This approach can reduce the size of data stored on NFT and reduce the gas fee.
3 Methodology

3.1 Introduction

This chapter presents the high-level overview implementation of the web platform. Section 3.2 describes the business and logic flow of the web platform. Section 3.3 showcases the technology stacks that used to develop the web platform.

3.2 Workflow

The web platform contains three major features, which are Account Creation, Contract Signing, and Employee Portfolio Verification. In this section a high-level view of workflow will be shown, details of technology implementation will be provided in later chapters.

3.2.1 Account Creation

Both employers and employees can create an account through the web interface. There are two types of accounts, namely, Individual and Business account. Business account is created for the convenience of employers to check the validity of candidates’ portfolio and initiate contract signing. Individual account allows clients to verify the public address of employer before they sign an employment contract and signing contract. Users can create an account with following steps (Fig. 3.1).

![Fig 3.1 High level overview of account creation process](image)

Firstly, both individual and business account applicants are required to provide their public address (address to receive and sent the NFT), email address and personal information, e.g., ID card number (for individual account) or company information (for business account). Secondly, upon KYC verification service to validate individual or company credential, accounts will be created. Lastly, public address and email address of account will be saved in our database to provide a smooth user experience for users.
For example, individual account users can verify companies’ public address through querying the database with companies’ name or email address. Meanwhile, business account users can retrieve candidates’ public address with candidates’ email address or name. Users no longer have to memorize the long public address of their counter party.

### 3.2.2 Contract Signing

Employers and employees can sign an employment contract and keep a record of it on the blockchain. The function of the employment contract would be the same as the employment contract sign on a paper. Yet, this process is now digitalized. The steps of contract signing are shown below (Fig. 3.2).

![Fig 3.2 High level overview of Process of Contract Signing](image)

Only company can upload the contract to initiate a job offer to candidate. To initiate the contract signing request, employers must fill in 1) target candidate’s email address or blockchain address, 2) employer’s private key (for contract signing), and 3) details of the job, i.e., the role of the job, the commencement date of the job, employment contract documents, etc. through web interface. Then, candidate will be notified with the event through email and can choose to accept, reject, or request for modification (contract signing request from employers) on our web platform. Once the candidate accepted the offer, the employment contract would upload to IPFS and notify both parties to sign the contract. Candidates and Company can sign on the employment contract with their private key. Once both parties have signed, it will trigger the execution of smart contract. Lastly, employment record will be tokenized into NFT and details of the job, hash of employee’s identity card number, information of the employer, i.e., employer’s public address, company name, etc. will be saved in metadata of the NFT. Lastly, NFT representing employment contract would be transferred to candidate’s blockchain address.
Note that document of employment contract would be encrypted to protect the privacy of both parties. Personal details, such as, HKID of candidates, will also being hashed before recorded to the NFT. Hashing will generate a cryptographic string which can be used to verify the integrity of the hashed messages or documents, while without leaking any information of the messages or documents. Besides, logic of minting and transferring of NFT will be written in smart contract, which will be deployed beforehand.

**3.2.3 Tokenization**

Every employment record will be tokenized as NFT and store to the blockchain. In this section, we will have a in-depth look to the procedure of the tokenization process for this project. The high-level overview of the procedures is shown in Fig 3.3 and 3.4.

![Fig 3.3 High level overview of deployment of smart contract](image)

Before the launch of the system, the smart contract will be deployed to the blockchain via alchemy service. Then, the alchemy service would return the NFT contract address to the system, which will be stored in our system. (Fig 3.3)

![Fig 3.4 High level overview of tokenization](image)
When the candidate accepts the offer, the NFT meta data (data include the encrypted employment contract, hash of candidate HKID, job title, job description, etc.) would be upload to the IPFS and the token_URI, a link locating the data on IPFS, would be returned to the system. When both parties signed the contract, our system would use the NFT contract address to locate the NFT smart contract on the blockchain and interacting with it via alchemy service. To mint a token, our system will call the mint function of the NFT contract to complete the tokenization steps. Eventually, NFT token would be minted and stored under candidate’s blockchain address. Token_id representing the token would be returned to both company and candidate for their reference.

3.2.4 Contract Termination

When either employer or employee decided to terminate the employment relationship, view the current token held and terminate the employment relationship on the web platform. Eventually, the NFT would be flagged to be terminated to provide information that the NFT is representing a terminated job record. The high-level overview of the procedures is shown in Fig 3.5.

![Fig 3.5 High level overview of Contract Termination](image)

Like tokenization, when our system receives terminate request from users for a specific token, it will interact with the NFT contract using the NFT contract address via alchemy service. Then, it will call the terminate function of the NFT contract and specify the NFT to be terminated, which will add a flag to the specific token and update the terminating time with current timestamp. (Fig 3.5)

3.2.5 Employee Portfolio Verification

The web platform also supports candidate’s employment portfolio verification service. The procedures are shown (Fig 3.6).
Fig 3.6 High level overview of Employee Portfolio Verification

Firstly, employer must enter the candidate’s blockchain address and candidate’s identity card number through the web interface. Secondly, using the blockchain address, backend server will call external API(s) for retrieving all NFTs data held by inputted (candidate’s) public address. Thirdly, backend will match 1) hash of identity card number with that on NFTs 2) using company address retrieved on NFTs to retrieve company name from database and the company name retrieved on NFTs. Lastly, if any pair get faults, it will be detected and both parties will be notified of the issue. Further proof should be provided from the candidates to the company. Moreover, details of job experience will be presented on the web interface to provide a user-friendly visualization.
3.3 Technology Implementation

3.3.1 Blockchain Development

Blockchain forms the backbone of the project. To consider which blockchain to be used, we will have following consideration, a) ease of development, b) transaction fees, c) transaction speed. Ethereum, Solana, and Polygon are the candidates for this project. Polygon is chosen based on above considerations; detail explanations described below.

**Ethereum**

Ethereum was launched in 2015 and it has more than 50.5 million smart contracts built on it. [9]

In terms of ease of development, large number of smart contracts deployed on Ethereum implies a relatively large number of libraries, terms to describe reusable code, supported on smart contract development in Ethereum. Smart contract is programs store and executed on blockchain. [10] In this project, we will write smart contract to execute the logic of minting and transferal of NFTs using Solidity. Solidity is the main programming language to develop smart contract on EVM. [11] It has a similar syntax with JavaScript, and hence, it will be convenience for our teams to develop with it as JavaScript is used on development of other components of the project. Also, there are varies of APIs supported for interacting with the blockchain. Hence, it is easy to develop on Ethereum with its great community of support.

In terms of transaction fee, minting and transferal of NFT on Ethereum cost at most $500 (USD) [12]. It is relatively expensive, comparing with Polygon less than $0.005 USD. In terms of transaction speed. Ethereum has a relatively have 20 TPS. Comparing with Ethereum is relatively slow in handling transaction, which can lower the efficient of our platform.

The advantage of Ethereum is the ease of development which Polygon also shares the same features. Yet, Ethereum has a higher transaction fee and transaction speed which is not capable for the web platforms when it goes scale. Hence, Ethereum is not being considered.

**Solana**

Solana is a web-scale blockchain that found in 2017. It uses Proof of History consensus mechanism to verify blocks, which is different from Proof of Work used in Bitcoin and Proof of Stake used in Ethereum 2.0 and Polygon. [13]
In terms of ease of development, as Solana is not EVM compatible, development of smart contract is different from Ethereum. At the same time, Solana has fewer decentralized applications running on it with about 350, while Ethereum has a much larger community with over 2900 decentralized application running on it. [14] There are also varies of APIs support on Solana for interacting with blockchain, such as Solana/web3.js for integration of web applications with blockchain and Anchor Framework for developing smart contracts on Solana. [15] With smaller community on Solana, there are less tools and forums helping when there are obstacles on the way of development. Besides, Rust, C, or C++ are used for smart contracts development on Solana, which syntax is much different from JavaScript. [15] It requires our team to take extra time to get familiar with its development environment, which is not favorable.

In terms of transaction fee, it is low. Each transaction and minting of NFT costs approximately $0.00025 USD, which is significantly lower than Polygon and Ethereum. [16] In terms of transaction speed, it is quick with 3155 TPS. [16]

The advantage of Solana is its high transaction speed and low transaction cost. Yet, Solana has a less friendly development environment and supports, which is not favorable for software development.

Polygon

Polygon was launched in 2017, supported with more than 7,000 decentralized applications. [17] It is a layer 2 protocol that run on the base of layer 1 blockchains, Ethereum. [17] Polygon provides a solution on scalability issue of Ethereum. [17] On the other hand, Polygon is EVM compatible implying that it has a similar development environment with Ethereum and applications on Ethereum can be run on Polygon. [18]

In terms of ease of development, as Polygon is EVM compatible tools and libraries developed in Ethereum can also be used on Polygon. Large number of smart contracts deployed on Ethereum implies a relatively large number of libraries supported on smart contract development in Polygon. EVM compatible features of Polygon also implies that Solidity will be used for the development of smart contract. Besides, there are varies of APIs supported for interacting with the Ethereum blockchain, such as, Web3.js for interacting with an Ethereum node, Ethere.js for interacting with Polygon wallet, and Alchemy NFT API for fetching NFT data. [19] Hence, it is easy to develop on Polygon with its great community of support.

In terms of transaction fee, minting and transferring NFT requires gas fee. [20] Gas fee is the transaction fee on blockchain. [20] Gas fee depends on the complexity of transaction. Therefore, transaction cost for NFT depends on the market price of gas and the complexity of the smart contract. Gas fee in Polygon is paid by MATIC, and a standard gas fee is around 0.005 MATIC, and 1 MATIC market price is $0.7819778 USD. [21] Transaction and minting fee in Polygon are less than $0.005 USD per transaction, which is low. In terms of transaction speed. Polygon has 65,000 TPS in comparison to Ethereum, which has 17 TPS. It
indicates that Polygon is relatively fast in handling transaction, which has highly efficient in handling our requested transaction and minting procedures. [22]

Polygon has a large community size and able to facilitate the development of the project. Besides, Polygon has a low transaction cost and high transaction speed. Hence, Polygon would be a good choice for the project.

### 3.3.2 Smart Contract Development

Smart contract is programs stored and executed on blockchain. [10] In this project, we will write smart contract to execute the logic of minting and transferal of NFTs. NFT is a unique digital identifier that cannot be copied and subdivided. [23] Also, NFT contains metadata that can store information. [23] In general terms, NFT can be treated as a tag to representing ownership. For example, if someone owns an NFT representing a particular dog, then they can proof that they own that dog. In this project, NFT will be minted for every employment record and the information of the employment details would be stored in the metadata of the minted NFT. In EVM compatible blockchain, there are three major smart contract token standards for NFT, namely, ERC-721, ERC-998, and ERC-1155. This project will be implemented with ERC-721.

**ERC-721**

ERC-721 is the simplest protocol for NFT. [24] Hence, it will be easier to implement. Ownership of NFT under this protocol is relatively easy to identify. [24] Yet, there are two key downsides for this protocol. First, it cannot be reclaimed by the issuer if the token is sent to a wrong address. [24] Second, it doesn’t support batch token transfers, which implies that multiple smart contracts are required to transfer multiple tokens. [24] It leads to a higher cost when transferring multiple NFT at a time.

**ERC-998**

ERC-998 is an extension of ERC-721. [25] It is a composable NFT and can contain multiple ERC-721 tokens. [25] This feature allows batch transaction of NFT. Instead of transferring ERC0721 NFT multiple times, users can make a single transaction of an ERC-998 token that contains multiple ERC-721 tokens. Yet, it is more complex and more difficult to implement comparing to ERC-721. [25]

**ERC-1155**

ERC-1155 is a protocol that support both fungible and non-fungible tokens minting. [24] Comparing with ERC-721, it is also a better option for batch token transfers as batch transfer of ERC-1155 tokens can be carried out with single smart contract. [24] It lowers the transaction cost of batch transfer of tokens. One significant downside of ERC-1155 is that it is harder to track an ERC-1155 token’s ownership. [24]
Rationale for using ERC-721

NFT in this project will serve as proof of working experience. Firstly, the NFT should be easy to check its ownership. For this reason, ERC-721 would be a better choice comparing with ERC-1155. Secondly, in this project, an NFT will be created only when two parties completed signing an employment contract and the NFT will be only transferred to the employee upon the completion of creation steps. Hence, batch transfer of NFT is not required in this project. It makes ERC-998 and ERC-1155 less significant to our project. With these two reasons, it makes ERC-721 a more compatible choice for the project.

3.3.3 Database Development

IPFS is used to save metadata of NFT, while PostgreSQL is used for storing account data.

IPFS

IPFS is a distributed, decentralized system for storing data. [26] With its distributed characteristics, the data will not be tampered or deleted. It can also easily be recovered even one of the data points crashed. [26] Data can be uploaded through IPFS API, like INFURA and Pinata. Once the data is uploaded to IPFS, a CID hash will be returned to act as the fingerprint of the content for searching and verifying integrity of the data being stored.

Storing data in blockchain could be expensive. [27] Hence, IPFS can play an important role in storing the metadata of NFTs. [27] Metadata can first upload to IPFS, API call to the IPFS content with the CID can be saved to the metadata of NFT. [27] As the API call link has a much smaller size comparing to the initial content saved, it would be much cheaper to mint the NFT. [27]

PostgreSQL

PostgreSQL is an object-relational database management system, which is supported by cross-platform. [28] Object oriented features of PostgreSQL support object features, including inheritance and user-defined data type. [29] It can help better organize our database and provide extensibility when we launch out new feature in the future. Besides, relational feature provides better organization of data for our project as data would be store with pre-designed schema.

3.3.4 Backend Development

Node.js and Express.js will be used.
Node.js

Node.js is an open source and cross-platform JavaScript runtime environment. It is developed in 2009. It is also used by Tech-Giant like Amazon, Netflix, eBay, Reddit, and with more than 30 million websites sit on it. It is supported with a large, many handy libraries can be found using NPM, the world’s largest software registry that support over 800,000 code packages. Libraries such as web3.js and Alchemy’s NFT API will be able to download in NPM.

Express.js

Express.js is a Node.js server-side framework. It provides built-in methods and middleware to ease the development of web and mobile application. Libraries and framework would ease our development as we do not have to start everything from scratch.

3.3.5 Tokenization of Employment Record

Hardhat and alchemy will be used to facilitate the minting process of NFT.

Hardhat

Hardhat is a Ethereum development environment for development, compilation, and deployment of smart contract. It helps manage and automate recurring task.

Alchemy

Alchemy is a web3 development platform which support developing tools to build dApps. It also serves as node provider which provide an ease way for developers to connect with the blockchain network using their node infrastructure through their API endpoint. In this project, we will use alchemy to mint, update and retrieve NFT.

3.3.6 Frontend Development

As our client would come from different background and may not have strong knowledge in programming. A user-friendly interactive web interface is needed to deliver our service. React.js will be used.
React.js

React.js JavaScript framework for building user interfaces. It would be easier for us to work on it given that our backend is written with JavaScript as well. Besides, React.js support building individual components for different part in a web page, which have high reusability and better organization. [36] On the other hand, React.js support virtual DOM, which allow keeping data state out of DOM and optimize efficiency of the app. [36] Lastly, it supports the integration with blockchain with library available on NPM as it is Node.js based as well. [36]

3.4 Summary

This chapter described the workflow of the web platform and introduced technology stacks that chosen to develop the web platform with justification of the choice. The next chapter will show the current progress of the project.
4 Project Development

4.1 Overview

This chapter presents the development of the project. Section 4.2 presented the detail implementation of the platform infrastructure. In section 4.2, part of our code base will be presented. Yet, limited to the length of the report, some details of our code implementation may not be shown in the report. The full code can be found in https://bitbucket.org/derek_szehoyin/fyp_server/src/main/ for the backend, while the full code for the frontend could be found in https://bitbucket.org/derek_szehoyin/reactapp/src/master/. Section 4.3 shows the project schedule. Section 4.4 shows the work distribution for this project.

4.2 Platform Development

This section will give a detail explanation on the platform infrastructure, which consisted of 4 major parts Blockchain (section 4.2.1), Backend (section 4.2.2), Database (section 4.2.3), and Fronted (section 4.2.4). Project structure of the platform is shown below (Fig 4.1).

Fig 4.1 File structure for frontend of our platform

Fig 4.2 File structure for backend and smart contract of our project
Our project consists of two parts, the frontend and backend. “Reactapp” is the front-code for this project (Fig 4.1) While “fyp_server” is the backend and smart contract code. (Fig 4.2) Both “reactapp” and “fyp_server” contains the “node_modules” folder which stores all the library and modules used in the project. (Fig 4.1, 4.2) The “package-lock.json” and “package.json” file stores the dependency of the project. (Fig 4.1, 4.2)

In “reactapp”, the folder “src” contains all the source code for different pages of the web user interface. (Fig 4.1)

In “fyp_server”, the folder “server” and “smart_contract” folder is self-explanatory. (Fig 4.2) The “contract_config.js” and “hardhat.config.js” file stores all the configuration data for blockchain development. (Fig 4.2) The “.sh” script file is used for running some configuration and setup command. (Fig 4.2)

**4.2.1 Blockchain Development**

This section explains the implementation detail of development and deployment of smart contract, which facilitate the function of tokenization of job record in this project.

**Project Structure**

The “smart_contract” project structure is shown (Fig 4.3). This part is developed under hardhat environment. It is located in “fyp_server” folder. (Fig 4.2)

![Fig 4.3 File structure for smart contract development in “fyp_server”](image-url)
In “contracts” folder, it contains ”fyp_token.sol” file which is a Solidity based smart contract to be deployed to blockchain network and guide the logic of minting and validating a NFT. In “test” folder, it contains the “test_fyp_token.js” file which contains the unit test cases to test the workability of the smart contract, “fyp_token.sol”.

**Smart Contract Development**

The skeleton and major function supported by the smart contract is shown. (Fig 4.4)

```solidity
pragma solidity ^0.8.17;
import "@openzeppelin/contracts/token/ERC721/extensions/ERC721Mintable.sol";
import "@openzeppelin/contracts/token/ERC721/extensions/ERC721Burnable.sol";
import "@openzeppelin/contracts/access/Ownable.sol";

contract FypToken is ERC721Mintable, ERC721Burnable, Ownable {
using Counters for Counters.Counter;
using Strings for uint256;

counter.Counter private _totalMinted;
mapping(uint256 => uint256) private _terminated;
mapping(uint256 => address) private _company_address;
mapping(uint256 => bool) private _terminated;

event minted(uint256 indexed value);

constructor() ERC721("yp_token", "yp_token") {
}

function isSigned(address, address, string) external view returns (bool) {
}

function mint(string memory message, ...) {
}

function terminate(uint256 token_id) public view returns (uint256) {
}

function company_address(uint256 token_id) public view returns (address) {
}

function terminate(uint256 token_id, uint v, uint8 r, uint8 s) public {
}

function beforeTokenTransfer(address, address, uint256 id, uint256) internal view override {
}
}
```

Fig 4.4 Code structure of “fyp_token.sol” file

The smart contract inherited the ERC721 token standard. There are several functions implemented to facilitate the NFT minting and terminate features. The isSigned() function takes in two address, two signature, and a string, which use for checking if two address signed on the same message with corresponding private key. It will return a Boolean to indicate it. The mint() function takes in a String, which is a link directing to the data store in IPFS, two address, and two signature. It is use for guiding the mint NFT logic, including verification of signature by calling isSigned() function, and will return the newly minted token id if successfully mint or return 0 if failed. The terminate() function will take in token_id as parameter, and return its terminated time if possible. The company_address() function will take in token_id as parameter, and return the blockchain address of the company that signed this contract. The terminate() function will take in token_id and a signature as parameter, and update the terminate time with
current timestamp if the company or employee make the signature and return the update timestamp, else do nothing and return 0. The _beforeTokenTransfer() function is to override the inherited function and to disable the transfer token function. (Fig 4.4)

**Smart Contract Deployment**

In this project, the smart contract is deployed on the polygon testnet (polygon-mumbai) network through the alchemy node services and Hardhat runtime environment using the “deploy.js” file as shown in (Fig 4.3). The address of the deployed is stored in “contract_config.js” file (shown in Fig 4.2), which will be used to locate the deployed contract on the polygon-mumbai network.

**NFT Minting**

To mint a NFT in this project, the meta data of NFT will first be uploaded to IPFS using our backend API endpoint (which will be discussed in section 3.2.2). Our backend API endpoint implemented the upload function using INFURA service. A sample of NFT meta data that uploaded to IPFS is shown below (Fig 4.5)

![Fig 4.5 Example of NFT meta data](https://fyp22041.infura-ipfs.io/ipfs/QmCC64na2J8BV4EWtT7RD44jEzJHMwfxYGhF9yK5qMTQgt)

The meta data is stored in JSON-format, with “details” specifying the detail information of the employment record, “employee” to store the information of the employee for the job record, “employer” to store the information of the employer for the job record, and “contract” stores the link that directed to the encrypted employment contract (which is in pdf) that uploaded to IPFS at the same time. (Fig 4.5) As we are using the INFURA service to complete the IPFS upload, `https://fyp22041.infura-ipfs.io/ipfs/QmCC64na2J8BV4EWtT7RD44jEzJHMwfxYGhF9yK5qMTQgt` specify the domain of the uploaded document, while “QmCC64na2J8BV4EWtT7RD44jEzJHMwfxYGhF9yK5qMTQgt” in this example is the CID (Content Identifier) to represent this piece of data that uploaded to IPFS. Users can locate the information on IPFS by combining two pieces of data.

Upon, uploaded the NFT meta data to the IFPS, the CID will be stored to the database. To mint the NFT, the link directed to the IPFS location, the blockchain address and signature of both parties will be retrieved from database. Then, the retrieved data will be pass to mint_nft() function in “NFT_service.js” file located under “service” folder (Fig 4.6). The mint_nft() function will interact with the smart contract via alchemy node service and mint the token. The token_id of the NFT will be returned when it is successfully minted, and token_id will be saved to the database.
NFT Termination

To update the NFT status to be terminated, the token_id of a job record will be retrieved from the database and pass to the terminate_contract() function in “NFT_service.js” file located under “service” folder (Fig 4.6) with the signature of it. The terminate_contract() function will interact with the smart contract via alchemy node service and call the terminate() function on the NFT smart contract to update the terminated time with current timestamp.

 Retrieving and Verification of NFT

To retrieve and verify the NFT hold by a specific blockchain address, it will be completed by alchemy API service which is implemented in get_nft() function located in “alchemy_service.js” file in the “service” folder. The get_nft() function will pass the blockchain address to the alchemy API endpoint to retrieve all NFT of our deployed smart contract held under the passed in blockchain address. Then, our backend function will loop through all the retrieved NFT to do the verification which will be discussed in section 4.2.2.

4.2.2 Back-end Development

This section explains the implementation detail of the file architecture, API endpoint developed, and the testing technique used in this project.

Project Structure

The backend uses a modularized structure for better organization purposes (Fig 4.6). The “server” folder is located in the “fyp_server” folder. (Fig 4.2)

Fig 4.6 File structure for backend development in “fyp_server”
In the “server” folder, it contains all the file related to the backend development. (Fig 4.6) The “server.js” file set up the server and connect to the database. (Fig 4.6) The “middleware.js” file contains all the self-implemented middleware. (Fig 4.6) The “configs.js” file contains all the configurations constant, such as the IPFS domain String. (Fig 4.6) The “util” folder contains all the self-implemented utility class used in the project. (Fig 4.6) The “test” folder contains all the unit test case for the project. (Fig 4.6) The “service” folder contains all the self-implemented service class. (Fig 4.6) The “model” folder contains all the data schema for the project. (Fig 4.6) The “offer” folder contains all the files required for the API route of /offer. (Fig 4.6) The “account” folder contains all the files required for the API route of /account. (Fig 4.6) The “profile” folder contains all the files required for the API route of /profile. (Fig 4.6)

Path, View, Bridge, Service Architecture

In the “account”, “offer”, and “profile” folder, it follows the path, view, bridge, service architecture. The path file contains all the API endpoint path and the middleware function. The view class will be responsible for unwrapping all the parameters of the request and delegating them to the bridge class, also it will handle error and wrap up the payload returned from bridge class with relevant response status code. The bridge class will be responsible for handle the business logic and calling different service from the “service” folder to complete the dedicated task. (Fig 4.7)
File for “offer” folder will be used as example to explain the implementation details.

Fig 4.8 Code of fyp_server/sever/offer/path.js file

In the “path.js” file, all the URI path for the RESTful API endpoint will write in here, with app.[method](\{path\}) format. The request will pass to middleware if needed. (Fig 4.8). Afterwards, the request will pass to the view class for further execution.
Fig 4.9 Code of fyp_server/server/offer/offer_view.js file

Each function in the Offer_View class in “offer_view.js” file will correspond to one RESTful API endpoint function. (Fig 4.9) Function in the View class will process the request and retrieve relevant parameters and dedicate them to the Bridge class function. Also, View class will wrap up the response from the bridge class as payload and add relevant status code for each API response before sending it back to the client. It will also catch error raised by the function in bridge class if possible.
Each function in the Bridge class will handle all the business logic required to complete the request, it will also dedicate some of the sub-tasks to the service class (which located in the “server” folder) if the task is too complicated. (Fig 4.10) The bridge class function would raise error message if errors happened or return the payload to the View class if task completed.

![Code of fyp_server/server/offer/offer_bridge.js file](image)

Fig 4.10 Code of fyp_server/server/offer/offer_bridge.js file

![File Architecture for “service” folder in “fyp_server”](image)

Fig 4.11 File Architecture for “service” folder in “fyp_server”
Different service is categorized into different class file which work as a helper function for executing sub-task from the bridge class function. (Fig 4.11)

Testing

In the “test” folder, it follows the structure of the “server” folder (Fig 3.11) For example, the “test_account_bridge.js” file is written with unit test cases that testing functions in the “account_bridge.js” file. (Fig 4.12)

![Fig 4.12 File structure for test in backend development in “fyp_server”](image)

In each automated test case, mocking technique is used. Mocking is the technique of pretending the existence of external functions, which ensure that only the target functions are being tested. (Fig 4.13)

![Fig 4.13 Example of unit test case retrieved from “test_account_bridge.js” file](image)
Sinon is a library for testing which support the mocking features. The `sinon.replace()` function is used to replace the service function that are not under testing. The last line of the unit test case `expect().to….` is an assertion statement that to compare if the actual result match the expected result. (Fig 4.12) The test case will only pass if they are matched. The statement following in `it(“….”)` is the statement to state the target function to be tested, the action, and the expected outcome of the test case. For example, “test create_account_bridge, delegate dict to mandatory field checking” shows that this test case is testing on the “create_account_bridge” function and it is expected that the dictionary object should be delegated to the “mandatory field checking” function. (Fig 4.13)

RESTful API

The web applications require API endpoint to provide functionality to all the users. A brief showcase of the API endpoint is shown below. (Table 4.1) RESTful API standard is used as it is self-explanatory, and it is a well-established standard commonly used in web development.

<table>
<thead>
<tr>
<th>Path and Method</th>
<th>arg/data (*mandatory field are highlighted with red)</th>
<th>usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>/account/{email} (GET)</td>
<td>raw-json:</td>
<td>get account chain address</td>
</tr>
<tr>
<td></td>
<td>{</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&quot;id&quot; : &quot;Y342343&quot; //hkid</td>
<td></td>
</tr>
<tr>
<td>/account/create (POST)</td>
<td>raw-json:</td>
<td>create new account</td>
</tr>
<tr>
<td></td>
<td>{</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&quot;email&quot;: &quot;abc @gmail.com&quot;,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&quot;password&quot;: &quot;1234&quot; ,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&quot;name&quot;: &quot;Chan Tai Man&quot;,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&quot;chain_address&quot;: &quot;234234&quot;,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&quot;type&quot;: &quot;BUSINESS&quot; // BUSINESS or INDIVIDUAL</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&quot;id&quot; : &quot;Y342343&quot; //needed only if type==INDIVIDUAL</td>
<td></td>
</tr>
<tr>
<td>/account/login (POST)</td>
<td>raw-json:</td>
<td>login to existing account</td>
</tr>
<tr>
<td></td>
<td>{</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&quot;email&quot;: &quot;abc @gmail.com&quot;,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&quot;password&quot;: &quot;1234&quot;,</td>
<td></td>
</tr>
<tr>
<td>/offer/ (POST) * only for logged in business account</td>
<td>form:{</td>
<td>post offer to candidate</td>
</tr>
<tr>
<td></td>
<td>'job_title': job_title,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>'candidate_email': candidate_email,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>'company_email': company_email,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>'contract': pdf file,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>'password': password,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>'job_description': job description</td>
<td></td>
</tr>
<tr>
<td>/offer (GET) * only for logged in account</td>
<td>query</td>
<td>get all received offer</td>
</tr>
<tr>
<td></td>
<td>?status=</td>
<td>//for individual account</td>
</tr>
<tr>
<td>Path</td>
<td>Description</td>
<td>Details</td>
</tr>
<tr>
<td>-------------------------------------------</td>
<td>-------------</td>
<td>---------</td>
</tr>
<tr>
<td>/offer/specific/:id (GET)</td>
<td>get specific offer</td>
<td>only for logged in account</td>
</tr>
<tr>
<td>example:</td>
<td></td>
<td>?id=396adae6-6b07-4c24-8fc9-2e1131eb63a2</td>
</tr>
<tr>
<td>/offer/accept (PUT)</td>
<td>update status of offer that are under REQ_REVIEW to REQ_SIGN and upload the offer to IPFS</td>
<td>only for logged in account</td>
</tr>
<tr>
<td>example:</td>
<td></td>
<td>?id=396adae6-6b07-4c24-8fc9-2e1131eb63a2</td>
</tr>
<tr>
<td>/offer/modify(PUT)</td>
<td>update status of offer that are under REQ_MODIFY to REQ_REVIEW update fields of the offer and notify the candidate that changes are made</td>
<td>only for logged account</td>
</tr>
<tr>
<td>/offer/sign (POST)</td>
<td>creating signature for the offer with specified offer id</td>
<td>only for logged account</td>
</tr>
<tr>
<td>/offer/sign (PUT)</td>
<td>sign a contract if both parties are signed, the offer status will be updated to</td>
<td>only for logged account</td>
</tr>
</tbody>
</table>

**Status Codes:**
- REQ_REVIEW
- REQ_SIGN
- COMPLETED
- REJECTED
- REMOVED
- REQ_MODIFY
- TERMINATED

**Example URLs:**
- ?status=REQ_REVIEW&status=COMPLETED
- ?id=396adae6-6b07-4c24-8fc9-2e1131eb63a2
- /offer/accept(PUT) id=5db68635-d022-4e8d-ba69-07e1755dcea3
- /offer/modify(PUT) id=5db68635-d022-4e8d-ba69-07e1755dcea3
- /offer/sign (POST) id=5db68635-d022-4e8d-ba69-07e1755dcea3
- /offer/sign (PUT) id=5db68635-d022-4e8d-ba69-07e1755dcea3
<table>
<thead>
<tr>
<th>Endpoint</th>
<th>Description</th>
<th>Example/Detail</th>
</tr>
</thead>
<tbody>
<tr>
<td>/offer/contract (GET)</td>
<td>* only for logged in account</td>
<td>query /id=</td>
</tr>
<tr>
<td></td>
<td></td>
<td>example: ?id=396adae6-6b07-4c24-8fc9-2e1131eb63a2</td>
</tr>
<tr>
<td>/offer/reject (PUT)</td>
<td>* only for logged in INDIVIDUAL account</td>
<td>raw-json:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>{ &quot;id&quot;: &quot;5db68635-d022-4e8d-ba69-07e1755dcea3&quot; //offer id }</td>
</tr>
<tr>
<td>/offer/remove (PUT)</td>
<td>* only for logged in BUSINESS account</td>
<td>raw-json:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>{ &quot;id&quot;: &quot;5db68635-d022-4e8d-ba69-07e1755dcea3&quot; //offer id }</td>
</tr>
<tr>
<td>/offer/terminate (PUT)</td>
<td>* only for logged in account</td>
<td>raw-json:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>{ &quot;id&quot;: &quot;5db68635-d022-4e8d-ba69-07e1755dcea3&quot; //offer id &quot;signature&quot;: &quot;0x5fbdb2315678afecb367f032d93f642f64180aa3 &quot; //signature for the offer on token_id }</td>
</tr>
<tr>
<td>/offer/terminate (POST)</td>
<td>* only for logged in account</td>
<td>raw-json:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>{ &quot;id&quot;: &quot;5db68635-d022-4e8d-ba69-07e1755dcea3&quot; //offer id &quot;secret&quot;: &quot;0x5fbdb2315678afecb367f032d93f642f64180aa3 &quot; //private key of the account }</td>
</tr>
<tr>
<td>/profile/record (GET)</td>
<td></td>
<td>query ?token_id=</td>
</tr>
<tr>
<td></td>
<td></td>
<td>example: ?token_id=1</td>
</tr>
<tr>
<td>/profile (POST)</td>
<td></td>
<td>raw-json:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>{ &quot;id&quot;: &quot;5db68635-d022-4e8d-ba69-07e1755dcea3&quot; //hkid &quot;chain_address:&quot;0xrewrjekh23r2342&quot; //blockchain address }</td>
</tr>
</tbody>
</table>

Due to limited spaces available for the report, the detail API documentation is not shown on the above table (Table 4.1). The detail version of API endpoint which provide the return value of each endpoint and...
error message response can be visited through following link: https://docs.google.com/spreadsheets/d/1Zt9E0wbXcZfjJ19BH2KuoYNMiH2tkxFsI0X3TvKvo/edit#gid=0.

## 4.2.3 Database Development

### Project Structure

Two data schemas are designed for the database, which is in “model” folder. (Fig 4.13) Schema of table Accounts is stored on file “account.js” (Fig 4.13) and schema of Agreements is stored on file “agreement.js” (Fig 4.14).

![Fig 4.14 File structure for database development](image)

### Database Schema

Account table stores the details of the user (Fig 4.15). Agreement table stores the details of the agreement (Fig 4.16).

![Fig 4.15 Schema of Account table](image)
In the Account table, ‘email’ is the primary key and used for login, ‘id’ is the hash of official id on the identity card issued by the government, ‘password’ is the hash of the password set by the account holder, ‘type’ is the type of the account which could be either BUSINESS or INDIVIDUAL, ‘name’ is the of the account holder, ‘chain address’ is the blockchain address, “createdAt” is the timestamp indicated the create time of the record, and ‘updatedAt’ is the timestamp when the record is updated. (Fig 4.15)

In Agreement table, ‘id’ is the primary key (the unique identifier of the record), ‘cid’ is the content id of the record of this agreement record on IPFS, ‘candidate_sig’ is the candidate signature on cid for validating credential on blockchain, ‘company_sig’ is the company signature on cid for validating credential on blockchain, ‘job_title’ is the title of the job, ‘job_description’ is the description for the job, ‘contract’ is the encrypted pdf employment contract in bytes, ‘candidate_email’ is the foreign key of Account table to identify the candidate of the agreement, ‘company_email’ is the foreign key of Account table to identify the company of the agreement, ‘status’ is the status of the record which could be [REQ_REVIEW, REQ_SIGN, COMPLETED, TERMINATED, REJECTED, REMOVED, REQ_MODIFY], “createdAt” is the timestamp indicated the create time of the record, ‘updatedAt’ is the timestamp when the record is updated, and ‘token_id’ is the NFT token id that the employment record associated with, if will be NULL originally and replaced once the NFT is minted. (Fig 4.16)
4.2.4 Frontend Development

A web-based user interface is developed for our platform. It is developed with React.js. This part is developed by my groupmate, Marcus Ling. Details of this part could refer to his report.

4.3 Project Schedule

Table 4.2 shows the project schedule. The project divided into six stages. Every stage shows the important task to be completed.

<table>
<thead>
<tr>
<th>Date</th>
<th>Task</th>
</tr>
</thead>
<tbody>
<tr>
<td>1st – 30th Sep</td>
<td><strong>Phase 1 Deliverable</strong></td>
</tr>
<tr>
<td></td>
<td>• Detailed project plan</td>
</tr>
<tr>
<td></td>
<td>• Project website</td>
</tr>
<tr>
<td>4th – 31st Oct</td>
<td>Research on Blockchain and NFT infrastructure</td>
</tr>
<tr>
<td></td>
<td>• Different blockchain project</td>
</tr>
<tr>
<td></td>
<td>• IPFS integration</td>
</tr>
<tr>
<td></td>
<td>• Tools to interact with Blockchain.</td>
</tr>
<tr>
<td></td>
<td>• Smart Contract Development</td>
</tr>
<tr>
<td>1st - 20th Nov</td>
<td>Smart contract development</td>
</tr>
<tr>
<td>21st Nov - 15th Feb</td>
<td><strong>Phase 2 Deliverable</strong></td>
</tr>
<tr>
<td></td>
<td>• First presentation</td>
</tr>
<tr>
<td></td>
<td>• Preliminary implementation</td>
</tr>
<tr>
<td></td>
<td>• Detailed interim report.</td>
</tr>
<tr>
<td></td>
<td>Core API endpoint development</td>
</tr>
<tr>
<td></td>
<td>• Internal call of API</td>
</tr>
<tr>
<td></td>
<td>• Integration of blockchain and platform backend</td>
</tr>
<tr>
<td></td>
<td>• With relevant testing</td>
</tr>
<tr>
<td>16th Feb – 15th March</td>
<td>Front end implementation</td>
</tr>
<tr>
<td></td>
<td>• Web interface for employers and employees</td>
</tr>
<tr>
<td></td>
<td>• Build relevant test case</td>
</tr>
</tbody>
</table>
16th March - 15th April

**Phase 3 Deliverable**
- Finalized tested implementation.
- Final Report
- Final presentation

Test and review the entire project

| May – Jun | Project Exhibition and Competition |
4.4 Contribution

This section evaluates the work distribution of our group. In this project, I took lead for the group, I also took lead on the research direction and schedule for the project. As shown in Table 4.3, both of us have equally share the work on researching for feasibility of the project. For the implementation of the project, I am responsible for coding the backend and smart contract for the project, which can be found in https://bitbucket.org/derek_szehoyin/fyp_server/src/main/. At the same time, my groupmate Marcus focus on the work of frontend development, which can be found in https://bitbucket.org/derek_szehoyin/reactapp/src/master/.

Table 4.3 Work distribution

<table>
<thead>
<tr>
<th></th>
<th>Sze Ho Yin</th>
<th>Ling Lok Tin (Marcus)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Research work</td>
<td>Done</td>
<td>Done</td>
</tr>
<tr>
<td>Frontend</td>
<td></td>
<td>Done</td>
</tr>
<tr>
<td>Backend</td>
<td>Done</td>
<td></td>
</tr>
</tbody>
</table>
5 Discussion

This section will discuss limitations of our solution (section 5.1), and our future plans for the project (section 5.2)

5.1 Limitations

There are some technical risks for the project. They are the proof of identity (section 5.1.1), legally binding power of NFT (section 5.1.2), and solutions for lost of private key (section 5.1.3).

5.1.1 Proof of Identity

In this project, we may not implement KYC process as most of them are costly and it is not the most viable product. Yet, KYC has served as a important component for the project as it serves as a gateway to ensure only valid data will be recorded to the blockchain. Hence, we would like to implement this function in future days. Besides, currently we didn’t implement mechanism to validate company’s blockchain address. It may raise to potential risk of making a employment record to a false blockchain address, which is not the company claimed on the NFT. It will reduce the credibility of the NFT as the public key of company is store in our platform database, which is centralized, there is a risk where our database is hacked or tampered.

There are several ways to solve the issue.

Firstly, it can be solved with law registration. A new law can be set to require company register a company license with their public key, which the identity of a company can be guaranteed by the government. Secondly, it can be solved with company claim. Company can broadcast the blockchain address owned by them to the public, which our system can validate the blockchain address claimed on the NFT with the company broadcasted blockchain address. Thirdly, a decentralized autonomous organization (DAO) can also solve the issue. A DAO is a decentralized autonomous organization, a type of bottom-up entity structure with no central authority. Members of a DAO own tokens of the DAO, and members can vote on initiatives for the entity. Smart contracts are implemented for the DAO, and the code governing the DAO’s operations is publicly disclosed [37]. We can build a DAO which serve as a decentralized database of storing the blockchain address of the company, and we corporate with several companies initially to set up the DAO. If new company are going to join our project, it can broadcast its blockchain address to members in DAO and members of the DAO can vote to decide if the newly blockchain address should be added to the DAO. As the data of valid company blockchain address has been moved to the blockchain, it can greatly reduce the risk of recording a false company blockchain address to the NFT.
5.1.2 Legally Binding Power of NFT

Although this project support signing employment contract through minting of NFT, it may not serve as the same legal binding power as signing employment contract on paper. Web3 and blockchain ecosystem is relatively new, hence, there are still room for further development on the rules and regulations of using these tools. Our platform could only run when the law grants the same legally binding power to the NFT as that of employment contract sign on paper. This is because users may not have the incentives to mint an NFT to represent an employment record if the NFT only serve as a proof of their job records. This issue can only be solved by government legislation.

5.1.3 Solutions for Lost of Private Key

Currently, we didn’t implement any mechanism to solve the problem of lost of private key. If candidate lost their private key, there have no mechanism to recover their NFT to their new blockchain address. Also, there have no mechanism to show a blockchain address is still inactive or lost. We believe there are solutions for the issues. Yet, there are solutions to recover private key using custodian service currently, for example, the private key recovery phrases. Hence, we believe the probability of happening a total loss of private key is insignificant and the NFT recovery features would have the least priority to be implemented in the future.

5.2 Future Plan

Currently, our platform is just a proof-of-concept (POC), which only the minimum viable product is implemented to show the feasibility of application of blockchain technology on employment contract signing and resume verification procedures. We have some plan to further improve the platform and make it more user friendly. The three major task to do in the futures are implementation of KYC and DAO (section 5.2.1), further improve of current web interface (section 5.2.2), and integration of system (section 5.2.3).

5.2.1 Implementation of KYC and DAO

Our top priority is to address the limitation of our current platform regarding the proof of identity issue (mentioned in section 5.1.1). As explained in the previous section, the main obstacle to implementing KYC service is the cost. However, we consider it a vital component for our platform, and we will implement it in the future if we obtain sufficient funds. Moreover, we believe that setting up a DAO would be the optimal solution to solve the company blockchain address issue (mentioned in section 5.1.1). This is because a DAO would be a fast and manageable solution, unlike relying on legislation or persuading companies to broadcast their blockchain address, which are unrealistic and infeasible options. Therefore, our immediate next steps are to build a protocol for the DAO and to seek cooperation with some companies to establish a DAO.
5.2.2 Further Improvement of Current Web Interface

Another area of improvement for our system is the web user interface, which we aim to enhance for a better user experience. Currently, our web interface does not integrate with any web3 custodian service. This means that users have to input the blockchain address and sign a contract using a third-party program or our API service. This could affect the user experience and increase the potential cybersecurity risk. Therefore, we intend to integrate our web interface with a third-party custodian service application, such as MetaMask, which would offer a more seamless user experience and reduce the potential cybersecurity risk.

5.2.3 Integration of System

Our system has the potential to deliver greater business value to the project by integrating with other functions. One of these functions is job posting and searching, which is the core functionality of platforms like LinkedIn and Jobs DB. This would enhance the user experience of our clients by offering them a comprehensive service. Another function is data analysis, which would allow us to use the user employment record data to generate business insights. Therefore, we plan to incorporate job posting and searching functionality in the future, as well as to apply machine learning or A.I. technology to analyze the data and provide meaningful insights for our platform users.
6 Conclusion

Due to inefficiency engendered by current labor-intensive background checking in hiring process, the contract signing, and employee portfolio verification platform serves as an effective digital solution to improve hiring process. This project targets to implement a web platform supporting employment contract signing on blockchain and resume verification. These projects do not only benefit companies by reducing hiring cost, but also smoothen the hiring process and provide better job-hunting experience for job seekers.

Our solution involves creating a web platform that supports employment contract signing and resume verification on blockchain. By updating latest employment record to blockchain and allowing recruiters to verify candidates’ job experiences by inputting their credential information. Our solution also provide termination and disabled transfer function of NFT, which comprehensively present details of job record with NFT. Our solution streamlines the background checking process and enhances its accuracy and efficiency.

Our solution is built on Polygon, a layer 2 blockchain built on top of Ethereum. The decision is made after careful consideration of various factors such as ease of development, transaction fees, and transaction speed. Instead of MERN stack, we used Postgres instead for our database. This is because a relational database could better organize the data and more fit to our project needs. We follow software engineer standards to build our web application which provide a user friendly user interface. It would be handier for recruiters and candidate without technical background to use our platform.

Currently, there are limitations for our solution. They are challenges related to proof of identity, legally binding power of NFTs, and solution for the loss of private keys. These limitations are important considerations for the future development of our project. Despite these limitations, we have outlined our future plans for the project, which include addressing the proof of identity issue through the implementation of KYC and DAO, further improving the current web interface, and integrating the system with web3 custodian service, MetaMask.

In summary, while our current platform is a proof-of-concept (POC) project, we have ambitious plans for the future development of the project. We are committed to addressing the limitations of our solution and continuously improving our platform to make it more user-friendly, secure and legally compliant. We are excited about the potential of blockchain technology in revolutionizing employment contract signing and resume verification procedures, and we look forward to further advancing our project in the future.
References


