Trusticate: Final Report
Blockchain-Verified Graduation Credentials

Pranay Periwal - 3035664697

FYP Supervisor: Dr. Liu, Qi
Group Member: Ye Ri Park - 3035712080
Date of Submission: Apr 17, 2023

Department of Computer Science
Abstract

Verifying educational credentials is a common practice during the hiring process, as candidates often misrepresent their educational background. However, those who obtained their degrees outside of their country may have trouble proving their qualifications. This verification process is usually done by third-party services or by contacting the educational institution directly, which can be costly and time-consuming. To simplify this process, blockchain technology can be used to store digital certificates issued by official educational institutions. Trusticate, a web application, has been developed to explore this approach using the Verifiable Credentials model, the Blockcerts library, and the Ethereum blockchain to issue and verify educational credentials. Trusticate provides a convenient way for academic organizations to register and utilize its technology. After a university creates its own Ethereum wallet, it can utilize the web application to generate graduation certificates for a group of students. Student information is imported via a CSV file, and the certificate is created in JSON format. Additionally, the certificate is made decentralized by being available on IPFS. By using Blockcerts-verifier, these credentials can be easily verified, simplifying the process of checking academic qualifications.
Acknowledgements

I would like to extend my appreciation to our supervisor, Dr. Qi Liu, for the opportunity to work on this final-year project. His invaluable advice and feedback help me move forward. Without his continued guidance,

I would like to also thank my team member, Ye Ri Park. Without her continuous effort, we could not have come to where we currently are.
# Table of Contents

List of Figures/Tables ........................................................................................................ iv  
List of Abbreviations ....................................................................................................... v  
CHAPTER ONE: INTRODUCTION .................................................................................. 1  
  1.1 Problem: Validating educational background ................................................................. 1  
  1.2 Current Solutions for validating educational background ................................................... 2  
  1.3 Blockchain ....................................................................................................................... 3  
    1.3.1 Blockchain and Its Benefits ......................................................................................... 3  
    1.3.2 Public-Key Cryptography (PKC) ................................................................................. 4  
    1.3.3 Blockchain and educational credentials ....................................................................... 4  
  1.4 Complexity of issuing blockchain-based credentials ....................................................... 5  
  1.5 Solution ......................................................................................................................... 5  
  1.6 Project Objectives ......................................................................................................... 6  
  1.7 Contribution ................................................................................................................... 7  
  1.8 Outline ........................................................................................................................... 8  
CHAPTER TWO: RELATED WORKS .............................................................................. 9  
  2.1 Approaches without using W3C Verifiable Credentials (VCs) ......................................... 9  
    2.1.1 OriginStamp .............................................................................................................. 9  
    2.1.2 Sony Global Education .............................................................................................. 10  
    2.1.3 Credly ....................................................................................................................... 10  
  2.2 Approaches using W3C Verifiable Credentials (VCs) ..................................................... 11  
CHAPTER THREE: METHODOLOGY ......................................................................... 12  
  3.1 W3C Verifiable Credential (VC) model ............................................................................ 12  
  3.2 Decentralized Identifiers (DIDs)...................................................................................... 13  
  3.3 Inter-Planetary File System (IPFS) .................................................................................. 14  
  3.4 Crypto Wallet ................................................................................................................. 15  
  3.5 Ethereum ....................................................................................................................... 15  
  3.6 Merkle Tree ................................................................................................................... 16  
  3.7 Blockcerts ...................................................................................................................... 17  
    3.7.1 Introduction to Blockcerts ......................................................................................... 17  
    3.7.2 Trusticate Ecosystem with Blockcerts ....................................................................... 18  
  3.8 Project flow diagram ..................................................................................................... 19  
  3.9 Step 1: Setting up issuing credentials .............................................................................. 20  
    3.9.1 Cert-issuer ............................................................................................................... 20
CHAPTER FOUR: PROJECT DEVELOPMENT ........................................................................ 24

4.1 Overview ................................................................................................................... 24

4.2 Project Schedule ....................................................................................................... 24

4.3 Blockchain-Related Development .......................................................................... 25
  4.3.1 Choosing Ethereum ......................................................................................... 25
  4.3.2 Development environment ............................................................................. 25
  4.3.3 Cert-tools development ................................................................................ 26
  4.3.4 Cert-issuer development ............................................................................... 33
  4.3.5 Blockcerts-verifier development .................................................................. 37

4.4 Frontend Development for the web application .................................................. 39
  4.4.1 Overview ......................................................................................................... 39
  4.4.2 Authentication flow ........................................................................................ 39
  4.4.3 Home page ..................................................................................................... 41
  4.4.4 Profile ............................................................................................................ 42
  4.4.5 Issue ............................................................................................................... 43
  4.4.6 Issuance History ............................................................................................. 46

4.5 Backend Development for the web application ................................................. 48
  4.5.1 Overview of the Backend Development ...................................................... 48
  4.5.2 REST API: Generate issuer ID ..................................................................... 48
  4.5.3 REST API: Issuing certificates ....................................................................... 50

4.6 Database Development for the web application ............................................... 52
  4.6.1 Firebase Firestore structure ......................................................................... 52
  4.6.2 Firebase Authentication ................................................................................. 54
  4.6.3 Firebase Storage Structure ............................................................................. 54

CHAPTER FIVE: FUTURE PLANS ................................................................................... 56

5.1 Overview ................................................................................................................... 56

5.2 Design individual certificate template ................................................................. 56

5.3 Improved verification of the university ................................................................. 56

5.4 Smart contract deployment to allow hosting of the web application ..................... 57

5.5 Using DID as a verification method ...................................................................... 58
List of Figures/Tables

FIGURE 1 DID ARCHITECTURE AND RELATIONSHIP OF COMPONENTS [7] ................................................................. 13
FIGURE 2 MERKLE TREE ........................................................................................................................................ 17
FIGURE 3 TRUSTICATE ECOSYSTEM IN VC MODEL .......................................................................................... 18
FIGURE 4 PROJECT FLOW DIAGRAM ................................................................................................................ 19
FIGURE 5 CONF.INI FILE CONTENT .......................................................................................................................... 27
FIGURE 6 SCREENSHOTS OF A VALID ‘ISSUER.JSON’ FILE .................................................................................. 28
FIGURE 7 REVOCATION LIST EXAMPLE WITHOUT A REVOKED CERTIFICATE .................................................... 29
FIGURE 8 REVOCATION LIST EXAMPLE WITH A REVOKED CERTIFICATE .......................................................... 29
FIGURE 9 SAMPLE CERTIFICATE TEMPLATE ........................................................................................................ 30
FIGURE 10 CERTIFICATE OF GRADUATION SAMPLE .......................................................................................... 31
FIGURE 11 SAMPLE CSV FOR STORING STUDENT DETAILS WITH THE SPECIFIC COLUMN NAMES .................. 32
FIGURE 12 PLACEHOLDER TEXTS ARE FILLED IN FOR UNSIGNED CERTIFICATE .............................................. 32
FIGURE 13 FILE STRUCTURE AFTER ISSUANCE ........................................................................................................ 33
FIGURE 14 SCREENSHOT OF CONF.INI FOR CERT-ISSUER ................................................................................... 34
FIGURE 15 SUCCESSFUL TRANSACTION DETAILS ON ETHERSCAN ................................................................. 35
FIGURE 16 SIGNED CERTIFICATE SHOWING ISSUER AND STUDENT DETAILS ..................................................... 36
FIGURE 17 SIGNED CERTIFICATE SHOWING MERKLE TREE PATH ......................................................................... 36
FIGURE 18 BLOCKCERTS-VERIFIER COMPONENT ON THE TRUSTICATE WEB APP ............................................. 37
FIGURE 19 VERIFICATION OF A CERTIFICATE USING BLOCKCERTS-VERIFIER ...................................................... 38
FIGURE 20 SCREENSHOT OF THE VERIFIER AFTER CLOSING THE MODAL ........................................................ 39
FIGURE 21 SIGN-UP MODAL ..................................................................................................................................... 40
FIGURE 22 LOGIN MODAL ......................................................................................................................................... 40
FIGURE 23 UNFILLED PROFILE PAGE AND ISSUER.JSON NOT UPLOADED ............................................................ 41
FIGURE 24 COMPLETELY FILLED IN THE PROFILE PAGE AND ISSUER.JSON UPLOADED CORRECTLY ............... 42
FIGURE 25 PROFILE PAGE SCREENSHOT ............................................................................................................... 43
FIGURE 26 ISSUE SCREEN IF THERE ARE NO ONGOING ISSUANCES ..................................................................... 44
FIGURE 27 PAGE TO ENTER BATCH DETAILS ........................................................................................................ 44
FIGURE 28 SUCCESSFUL VALIDATION OF CSV ....................................................................................................... 45
FIGURE 29 SCREEN AFTER SAVING THE DATA ........................................................................................................ 45
FIGURE 30 SUCCESSFUL BACKEND CALL TO ISSUE THE CERTIFICATES ............................................................ 46
FIGURE 31 HISTORY PAGE SCREENSHOT ............................................................................................................. 46
FIGURE 32 HISTORY DETAILS SCREENSHOT ....................................................................................................... 47
FIGURE 33 EXCEL FILE OF AN INDIVIDUAL ISSUANCE DETAILS ................................................................. 47
FIGURE 34 DEVELOPMENT ARCHITECTURE .......................................................................................................... 48
FIGURE 35 GET REQUEST FOR GENERATING ‘ISSUER.JSON’ ................................................................................ 49
FIGURE 36 GET REQUEST FOR GENERATING ‘REVOCATION-LIST.JSON’ ............................................................ 49
FIGURE 37 REST API ISSUING CERTIFICATES FLOW ............................................................................................. 51
FIGURE 38 POST REQUEST FOR ISSUING CERTIFICATES ..................................................................................... 52
FIGURE 39 SCHEMA FOR FIRESTORE ..................................................................................................................... 53
FIGURE 40 SCREENSHOT OF FIREBASE AUTHENTICATION ................................................................................ 54
FIGURE 41 FIREBASE STORAGE STRUCTURE ....................................................................................................... 55
FIGURE 42 ISSUER INFORMATION SHOWN USING BLOCKCERT’S VERIFIER BASED ON ISSUER.JSON ............. 57

TABLE 1 CONTRIBUTION OF EACH MEMBER ......................................................................................................... 7
TABLE 2 PROJECT SCHEDULE .................................................................................................................................... 25
# List of Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ECDSA</td>
<td>Elliptic Curve Digital Signature Algorithm</td>
</tr>
<tr>
<td>dApp</td>
<td>Decentralized application</td>
</tr>
<tr>
<td>DID</td>
<td>Decentralized identifiers</td>
</tr>
<tr>
<td>DLT</td>
<td>Distributed Ledger Technology</td>
</tr>
<tr>
<td>DNSSEC</td>
<td>Domain Name System Security Extensions</td>
</tr>
<tr>
<td>IPFS</td>
<td>InterPlanetary File System</td>
</tr>
<tr>
<td>P2P</td>
<td>Peer-to-peer</td>
</tr>
<tr>
<td>PoS</td>
<td>Proof of Stake</td>
</tr>
<tr>
<td>PoW</td>
<td>Proof of Work</td>
</tr>
<tr>
<td>SHA</td>
<td>Secure Hash Algorithm</td>
</tr>
<tr>
<td>VC</td>
<td>Verifiable Credential</td>
</tr>
<tr>
<td>W3C</td>
<td>World Wide Web Consortium</td>
</tr>
</tbody>
</table>
CHAPTER ONE: INTRODUCTION

This chapter introduces the project, Trusticate, for building a new platform to enable educational institutes to issue credentials such as graduation certificates on the blockchain. As faking educational credentials is a large-scale problem seen across industries, we wished to create a scalable product that was easy to adopt and integrate.

1.1 Problem: Validating educational background

The verification of an interviewee's educational background has become an essential and widespread practice among hiring companies. As the job market continues to become increasingly competitive, many job seekers are tempted to embellish their resumes to stand out from other candidates. As a result, statistics indicate that a staggering 44% of individuals have falsified their educational credentials in order to secure a job [1]. This has prompted many employers to take extra steps to verify a candidate's academic qualifications before extending an offer of employment.

It is important to note that the consequences of failing to verify a candidate's educational background can be dire. Research shows that nearly 70% of individuals caught lying about their educational qualifications have had their job offers revoked, been fired after starting the job, or received other forms of disciplinary action [1]. Such outcomes are not only a waste of valuable time and resources for hiring companies, but also pose a significant risk to the organization's reputation and financial well-being.

In addition to the risks posed by unverified credentials, other factors can also complicate the verification process. For instance, a candidate may misplace or lose their hard copy of a graduation certificate, thereby creating an obstacle in verifying their academic credentials. Obtaining a new copy of the certificate can be a lengthy process, taking up to ten working days if the document was not previously scanned and saved digitally.

Furthermore, verifying the educational background of candidates from unpopular or lesser-known institutes can pose significant challenges. In cases where a person's educational institution has shut down, the verification of academic credentials becomes particularly difficult, if not impossible.
In conclusion, the verification of a candidate's educational background is an essential practice that ensures that hiring companies are hiring the most qualified candidates for the job. The risks posed by unverified credentials, coupled with the challenges of verifying academic qualifications from unpopular institutes, makes this process even more crucial. Employers must invest in appropriate measures to ensure that they are not only hiring the most qualified candidates but also mitigating the risks associated with unverified credentials. This will go a long way in creating a more transparent and trustworthy job market.

1.2 Current Solutions for validating educational background

Currently, companies have two primary methods for verifying an applicant's educational background. The first involves outsourcing the process to a third-party service provider who specializes in the verification of academic credentials. This approach can be more efficient and convenient for companies, as it saves them from having to communicate directly with educational institutions to verify a candidate's degree. However, it can also be quite expensive.

The second method involves conducting the verification process in-house by directly contacting the institutions where the applicant studied. While this approach can be more cost-effective than outsourcing, it can also be quite cumbersome and time-consuming. It may take several days for the verification process to be completed, and there may be language barriers or other challenges associated with verifying international degrees and certifications. Moreover, the possibility of falsification of information may increase the risk of inaccurate results [2].

It is clear that the current processes for verifying educational credentials have limitations, and there is a need for improvement. Blockchain technology offers a promising solution that can radically transform the system. By leveraging the immutability and transparency of blockchain, educational institutions can create a decentralized, tamper-proof system for issuing and verifying academic credentials. This would provide companies with a reliable and secure method for verifying the educational background of candidates, while also reducing the risk of falsification and fraud.

Furthermore, blockchain technology can streamline the verification process, eliminating the need for time-consuming and tedious communication with educational institutions. With a blockchain-based system, companies can quickly and easily verify a candidate's educational credentials with just a few
clicks. This would significantly reduce the effort and resources required to verify academic qualifications and make the hiring process more efficient.

In conclusion, the current methods for verifying educational credentials have their limitations, and there is a clear need for improvement. Blockchain technology offers a promising solution that can transform the system by providing a secure and reliable method for verifying academic credentials. With the benefits of increased efficiency and reduced risk of falsification, blockchain-based systems could significantly improve the hiring process for companies and job seekers alike.

1.3 Blockchain
Section 1.3 details Blockchain technology and why it would be an important part of the solution to help combat falsified educational credentials.

1.3.1 Blockchain and Its Benefits
To elaborate, a blockchain is a secure and transparent digital ledger that records transactions and assets across a network of computers. This ledger is decentralized, meaning that it is not controlled by a single entity but is instead maintained by a network of users. Each block on the blockchain contains information about a specific transaction, and each block is connected to the previous block in the chain, forming a permanent and tamper-proof record of all transactions that have taken place.

The decentralized nature of blockchain technology provides several benefits, including enhanced security and privacy, increased transparency and accountability, and improved efficiency and speed of transactions. By eliminating the need for a central authority to manage and verify transactions, blockchains can reduce the risk of fraud, hacking, and other types of cybercrime. The use of cryptographic mechanisms ensures the authenticity and integrity of the data stored on the blockchain.

Additionally, blockchains are immutable, meaning that once a transaction has been recorded on the blockchain, it cannot be altered or deleted. This feature provides a complete and transparent record of all transactions, which can be used for auditing, compliance, and other purposes. Since every transaction is time and date stamped, the entire history of a transaction can be traced back to its origin, making it easier to track and verify information [3] [4].
In conclusion, blockchains provide a secure, transparent, and immutable way to record and track transactions and assets. The decentralized nature of blockchain technology allows for enhanced security, privacy, and efficiency, while immutability ensures the authenticity and integrity of the data stored on the blockchain.

1.3.2 Public-Key Cryptography (PKC)
Public-Key Cryptography (PKC), is a critical component of blockchain ecosystems. This feature enables users to send and receive digital documents securely, without the need for a third party to verify the transactions. In this system, a public key is used for receiving cryptocurrency transactions, while a private key is required to prove ownership or to spend the cryptocurrencies. The generation of public keys from private keys is possible, but the reverse is almost impossible due to the use of "trapdoor functions" or "one-way functions." These functions are algorithmic puzzles that are simple to solve in one direction but extremely difficult in the other direction [4].

To complete a transaction on the blockchain, it must be digitally signed using the appropriate private key. This digital signature serves as proof that the transaction has not been altered and can be verified as authentic using the corresponding public key. The use of PKC provides a high level of security and privacy in blockchain transactions, making it an essential feature of this technology.

1.3.3 Blockchain and educational credentials
Blockchain technology is suitable to stop the fraudulent generation of educational credentials because of its immutability. This means that the data about the credentials cannot be changed or removed once it has been entered into the blockchain.

To confirm the validity of an educational credential, it is necessary to compare the data of the presented credential to the data stored on the blockchain. This procedure can be used to verify that the presented credentials are legitimate and have not been tampered with. A match between the presented credential and the blockchain-stored data demonstrates that the presented data is authentic and has not been changed. However, if there is any discrepancy between the two sets of data, it is a sign that the presented credential cannot be trusted.
Therefore, blockchain technology provides an effective way to prevent the forgery of educational credentials, allowing for the creation of a secure and reliable system for verifying academic achievements.

### 1.4 Complexity of issuing blockchain-based credentials

Although blockchain-based educational credentials have the potential to benefit students, universities, and employers, their implementation has been challenging for universities. While some universities have begun to explore this approach, only a few of the top 50 universities have issued blockchain-based graduation certificates. MIT and HKUST are two examples of universities that have successfully implemented this technology.

The main obstacle to implementing this system is the technical expertise required to issue blockchain-based educational credentials. This process requires a high level of technical knowledge and expertise that may not be readily available in universities. There are currently only a few web applications or software-as-a-service (SaaS) products available that can issue blockchain-based credentials out-of-the-box, and they do not use the VC model. These have their own limitations, which are discussed in Chapter 2. As a result, universities must weigh the pros and cons of establishing an IT team to assist with the process.

However, implementing blockchain-based educational credentials is critical for the long-term benefit of students, universities, and employers. Therefore, a simple SaaS solution that bridges the technical knowledge gap and provides universities with the necessary tools to issue blockchain credentials would be highly beneficial. This would allow universities to take advantage of the benefits of blockchain technology without the need for extensive technical knowledge or resources.

### 1.5 Solution

Trusticate is a proposed platform that aims to simplify the process of verifying candidates' educational backgrounds by providing a centralized solution that brings together all the necessary tools. The platform is intended to assist with issuing and verifying students' educational credentials on the Ethereum blockchain, making the verification process secure, reliable, and efficient.
Trusticate provides a user-friendly interface that makes it simple for educational institutions to manage their students’ issued credentials and for recruiters to verify them. Trusticate offers a one-stop-shop solution for all aspects of credential verification. Recruiters can easily access the blockchain-stored data using Trusticate and verify the authenticity and accuracy of the credentials. This eliminates the need for manual verification processes and reduces the risk of fraudulent credentials being presented.

Overall, Trusticate provides a reliable, efficient, and user-friendly platform for issuing and verifying educational credentials on the blockchain. This innovative solution offers convenience and faster verification processes for recruiters, while providing educational institutions with a secure and trusted way to manage the credentials.

1.6 Project Objectives

The main objective of the project is to develop a blockchain-based web application that can be used to verify graduation certificates issued by educational institutions. The system will be built on Ethereum, a secure and decentralized layer 1 blockchain technology, which will be used to store educational credential information.

One of the key goals of the project is to make the technology easy to adopt, so that any educational institution can use Trusticate. The team will be developing a user-friendly web application that will be primarily used by issuers to create educational documents. The aim is to create a platform that is intuitive and easy to use, making it accessible to educational institutions of all sizes.

Trusticate's technology will provide convenience in the verification process of candidates' educational backgrounds. This will save time for both candidates and hiring companies, while also promoting fairness and honesty in the hiring process. With Trusticate, hiring companies can easily verify the authenticity of a candidate's educational credentials, providing greater confidence in the hiring process.

Overall, Trusticate aims to revolutionize the way that educational credentials are verified, providing a secure and transparent platform that can be easily accessed by educational institutions and employers.
alike. By leveraging the power of blockchain technology, Trusticate seeks to bring greater efficiency, fairness, and trust to the educational and employment sectors.

1.7 Contribution
For the completion of the project, the individual contribution in specific reference to coding are as mentioned in the table 1.7 below:

<table>
<thead>
<tr>
<th>Contribution</th>
<th>Pranay Periwal</th>
<th>Ye Ri Park</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Blockchain-related contribution</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Generating issuer.json for universities</td>
<td>✔️</td>
<td></td>
</tr>
<tr>
<td>Generating unsigned graduation certificates</td>
<td>✔️</td>
<td></td>
</tr>
<tr>
<td>Implementing revocation of certificates</td>
<td>✔️</td>
<td></td>
</tr>
<tr>
<td>Implementing issuing and signing graduation certificates</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>Generating IPFS links</td>
<td></td>
<td>✔️</td>
</tr>
<tr>
<td><strong>Web application contribution</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Signup and Login Page</td>
<td>✔️</td>
<td></td>
</tr>
<tr>
<td>Home page</td>
<td>✔️</td>
<td></td>
</tr>
<tr>
<td>Profile page</td>
<td></td>
<td>✔️</td>
</tr>
<tr>
<td>Issuing page</td>
<td></td>
<td>✔️</td>
</tr>
<tr>
<td>Issuance history page</td>
<td>✔️</td>
<td></td>
</tr>
<tr>
<td>Verification page</td>
<td></td>
<td>✔️</td>
</tr>
<tr>
<td><strong>Backend (Firebase/API) Contribution</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Setting up the firebase</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>API for getting issuer.json</td>
<td>✔️</td>
<td></td>
</tr>
<tr>
<td>API for issuing certificates</td>
<td>✔️</td>
<td></td>
</tr>
<tr>
<td>API for getting IPFS links</td>
<td></td>
<td>✔️</td>
</tr>
</tbody>
</table>

Table 1: Contribution of each member

Both members contributed in different ways to the project. Ongoing work is also being contributed to by both members. Overall, there was equal contribution.
1.8 Outline

This paper describes the software solution for allowing different educational institutes to be able to issue graduation certificates on the Blockchain (specifically Ethereum). Chapter 2 elaborates on the related works for blockchain-based educational credentials. Chapter 3 details the technologies and methodologies used to achieve the project objectives. The use of the Verifiable Credential model, Decentralized Identifiers and Inter-planetary File System are noted in this chapter. Also, a step-by-step introduction to the process of using the Blockcerts code library and building the web application is elaborated here. Chapter 4 describes the project's current status and the major future steps. Lastly, chapter 5 summarizes the project in a conclusion.
CHAPTER TWO: RELATED WORKS

This chapter explores the current progress and work being done in the domain of blockchain-based educational credentials. There are broadly two main approaches, and each has its own advantages and disadvantages.

2.1 Approaches without using W3C Verifiable Credentials (VCs)

To create a blockchain-based educational credential without using the W3C VC model standard means to create your own standard for your implementation. This can have some benefits but also some drawbacks. One of the main benefits to creating a custom standard specific to the ecosystem would be in terms of optimizing the solution for the specific use case. Inefficiencies found in the main standard can be improved upon. However, one potential downside of creating a custom standard is that it may not be interoperable with other systems, meaning that it would only work within a closed environment. This could lead to issues if a verifier outside of the ecosystem needs to verify the credential, as they would need to learn a new system.

2.1.1 OriginStamp

OriginStamp is a blockchain-based timestamping service that allows users to prove the existence and integrity of digital data at a certain point in time. It uses cryptographic hashing algorithms to create a unique digital fingerprint of a file, and then records that fingerprint on a blockchain network. OriginStamp uses a Merkle Tree structure to store multiple hashes together and ensure that the final hash is secure and immutable.

By timestamping data on the blockchain, OriginStamp creates a tamper-evident proof of existence that can be used to verify the authenticity and integrity of digital documents, images, videos, or other types of data. This can be useful in a wide range of industries, such as legal, financial, or scientific, where proof of data integrity is crucial. OriginStamp also provides an API that developers can use to integrate timestamping into their applications, as well as a web interface that allows users to easily upload and timestamp their files [5].

While this approach could potentially be beneficial for other forms of digital content, it may not be ideal for graduation certificates. There is a potential for the PDF file to be altered before it is uploaded to the blockchain, thus defeating its purpose. It would also mean that the graduation certificate is not truly
owned by the student. The public key of the student is not natively integrated into the certificate as the certificate would be the image/PDF file of the graduation certificate.

2.1.2 Sony Global Education

Sony Global Education is a subsidiary of Sony Corporation that provides educational services and solutions. In 2017, they announced the development of a blockchain-based system for managing and sharing educational records and achievements. The system uses Hyperledger Fabric, a blockchain framework developed by the Linux Foundation's Hyperledger project [6].

The system enables students to store their educational records on a secure, tamper-proof blockchain ledger, which can then be shared with educational institutions and employers as needed. However, most of the problems lie with the use of a privatized blockchain in Hyperledger Fabric for a public use case. By using Hyperledger Fabric, Sony Global Education would create a custom blockchain for issuing and verifying educational credentials. While Hyperledger blockchain is open source, they would need to build a large enough network to ensure security because it is a privatized blockchain. If there is limited adoption, not only is the security at risk, but it may also not be worthwhile for universities to invest in the solution. Assuming they can get widespread adoption, it is not certain that the resulting credentials would be interoperable outside of this Sony ecosystem. If students are forced into the Sony ecosystem, it is not truly decentralized [7]. It would also mean that the verifiers would have to adopt a new verification pattern for Sony Global Education issued credentials.

2.1.3 Credly

Credly is a digital credentialing platform that allows organizations to issue, manage, and verify digital badges and credentials. The platform enables individuals to showcase their skills and achievements by earning digital badges that can be shared on social media, online portfolios, and resumes.

Credly offers Open Badges. Open Badges is an open standard developed by the Mozilla Foundation that allows for the creation and sharing of digital badges. These badges contain metadata that describe the skill or achievement being recognized, as well as information about the issuer, recipient, and evidence of the achievement. Open Badges are interoperable, which means that they can be shared and verified across different platforms and systems [8].

In addition to using Open Badges, Credly also leverages blockchain technology to enhance the security and verifiability of its digital badges. Credly's blockchain technology is based on the Ethereum
blockchain. Although evidence from Github indicates they grabbed a portion of the Blockcerts issuing codebase, it is hard to determine exactly what they are doing as they have not made their technology publicly available or open-sourced. Furthermore, it is evident that they are not collecting receivers' public keys, therefore the badges that arise are not owned by the recipients [9].

2.2 Approaches using W3C Verifiable Credentials (VCs)
Several universities, including the Massachusetts Institute of Technology (MIT) and the Hong Kong University of Science and Technology (HKUST), have recently launched projects with their registrar offices to issue digital graduation certificates that are stored on the blockchain. These projects were developed using the open-source Blockcerts library, which provides a standard for creating, issuing, and verifying digital credentials using blockchain technology.

HKUST's project follows the Blockcerts library's open standard specifications, but they do not collect the public keys from the students. This means that the credentials are not owned by the students because their public key is not mentioned on the credential. However, the blockchain-based certificates issued by HKUST are still tamper-evident and can be used to prove the integrity and authenticity of the student's academic achievement [10].

On the other hand, MIT's project uses the Blockcerts app to collect the public keys from the students. Once the students are registered, the educational credential is sent to their wallet, and they have full control over whom to share the credentials with. This approach gives students full ownership and control over their academic credentials, allowing them to share them securely and conveniently with potential employers or other institutions [11].

One issue with MIT's approach is that there is no platform currently available for other universities to use the same technology. Each university would need to build and customize their own tools from scratch to issue blockchain-based academic certificates. Nonetheless, the development of such blockchain-based credential systems has the potential to revolutionize the way academic achievements are recorded and shared, providing more secure and transparent means of academic verification.
CHAPTER THREE: METHODOLOGY

Chapter 3 elaborates on the different technologies and tools used in the project. There are different technologies and concepts employed in building the solution. The W3C Verifiable Credential model, Decentralized Identifiers and IPFS are explored in this section. The main tool used is a code library called Blockcerts and it is used as the underlying technological layer for the product. There is also a discussion of the technologies used to building the platform such as React, Flask, Docker and Firebase in this chapter.

3.1 W3C Verifiable Credential (VC) model

Verifiable Credentials (VCs) are a significant development in the world of digital certificates since they provide a tamper-proof, decentralized, and secure way to manage and verify credentials. VCs can overcome the issues with traditional paper-based credentials since they are issued and verified electronically, providing an immutable and secure record of a person's educational achievements.

The W3C VC model defines a standardized way of issuing, storing, and presenting VCs. It allows issuers to create credentials using digital signatures and store them in a tamper-proof digital wallet on the holder's device. Verifiers can then request and verify the authenticity of these credentials without the need for intermediaries. The use of cryptography ensures that the credentials are secure and can only be accessed by the holder, ensuring privacy and security [12].

In our solution, educational institutions can issue VCs to their students, who can store them in a digital wallet on their device. These credentials can then be presented to hiring companies as part of the job application process. The hiring companies, acting as verifiers, can easily verify the authenticity of these credentials, saving time and resources, and ensuring that the candidate's educational qualifications are valid.

Overall, VCs offer a reliable and secure way to manage and verify credentials. They offer a promising solution to the challenges of the traditional paper-based system, providing a tamper-proof and decentralized system that can benefit educational institutions, students, and hiring companies alike.
3.2 Decentralized Identifiers (DIDs)

Decentralized Identifiers (DIDs) are unique identifiers that allow any party in a VC model to be identified without relying on a centralized system. DIDs are represented as a string of characters, and they are constructed using a method similar to a Uniform Resource Identifier (URI). For example, a DID could be represented as 'did:example:123456789abcdefghijk', where 'did' stands for "decentralized identifier", 'example' is the method used to generate the identifier, and the remaining string of characters is a unique identifier for the party.

![DID Architecture and Relationship of Components](image)

**Figure 1 DID Architecture and Relationship of Components [13]**

A DID is linked to a DID document that contains information about the owner of the DID, such as cryptographic public keys that can be used to verify the authenticity of the owner. A DID subject can be an individual or an organization. The DID and the DID document are stored on a Verifiable Data Registry, which can be a blockchain. A DID controller is the entity that can make changes to the DID document. Figure 1 shows an overview of some of the parts of the DID architecture [13].

One of the key features of DIDs is that they are designed to be decentralized, meaning that they are not linked to a specific database or centralized authority. This allows DIDs to be moved from one registry to another, making them ideal for use in VC models. Since VCs are decentralized, the owner of the VC can move it from one blockchain to another using their DID. This eliminates the need to reissue the VC, as the DID is linked to the owner and can be used to verify the authenticity of the VC on any registry. Overall, DIDs provide a secure and flexible solution for identifying parties in a VC model without relying on centralized systems.
3.3 Inter-Planetary File System (IPFS)

The Inter-Planetary File System (IPFS) is an open-source peer-to-peer networking protocol suite designed to organize and transfer data on the internet. It aims to change the current client-server model by storing files and data in multiple nodes, enabling clients to fetch data from different nodes, making it censorship-resistant.

IPFS uses content-addressing to identify and access files on the network instead of a URI. When a file is added to IPFS, a unique cryptographic hash of the file is created, which is then used to point to the data. This approach reduces data duplication as duplicate files will have the same hash, and therefore point to the same data.

The protocol seeks to address issues with data representation on the current World Wide Web with transfer protocols like HTTP. IPFS uses cryptographic hashes to verify the authenticity of files and data, preventing malicious actors from altering the data because any changes would alter the hash value.

Since IPFS is a distributed protocol, there is no centralization, meaning there is no single point of failure for hosted files. This makes the protocol more robust and resilient compared to traditional client-server models that rely on a central server. Additionally, IPFS tends to be faster because it fetches data from multiple nodes where data may be stored instead of just one server, improving performance and availability.

Trusticate uses IPFS for storing the signed educational credentials on it. This prevents any single point of failure for the educational credential. This is done to future-proof a student’s credential against any kind of downtime a university server may face. It also protects students from the event that their university must shut down. One drawback that needs to be worked on is that not all browsers allow for visiting an IPFS link directly. Once browsers natively support IPFS links, the workarounds would become redundant [14].
3.4 Crypto Wallet
A digital wallet is a software program that allows users to safely store and manage various types of digital assets, such as cryptocurrencies and digital documents. In the case of educational credentials, the digital wallet can store the verifiable credentials that students obtain from their educational institutions.

Similar to crypto wallets, when a user creates a digital wallet, they are given a set of private and public keys. The public key can be shared with educational institutions or potential employers to receive or share verifiable credentials. The private key, on the other hand, is used to confirm the holder's identity and authorize the sharing of credentials. Therefore, it is critical to keep the private key safe and not share it with anyone.

Digital wallets include a variety of security features, including encryption, multi-factor authentication, and biometric verification. These features guarantee that only the authorized user can access and manage the digital assets stored in the wallet. Additionally, digital wallets provide a user-friendly and convenient way for users to manage their digital assets and share them with others, making them an ideal solution for storing and sharing verifiable credentials [15].

3.5 Ethereum
Ethereum is a blockchain platform that started in 2015 and is the main blockchain that will be used in this project. It's run by the community and uses a peer-to-peer network. This means that each node functions as both a client and server. Nodes in Ethereum are responsible for maintaining the blockchain and processing transactions.

One of Ethereum's standout features is its support for smart contracts, which are contracts that are self-executing and written directly into code. Smart contracts remove the need for intermediaries like banks or legal systems and make transactions more automated, transparent, and secure. Ethereum, like other blockchain projects has its own native currency, that is called Ether. Ether is what is used to transact and interact with the Ethereum blockchain and its smart contracts. Every time a user needs to interact with the blockchain, they need to pay a small fee called Gas. This fee is what is distributed to the miners/validators as a reward for them keeping the blockchain safe.
In 2022, Ethereum transitioned from proof-of-work (PoW) to proof-of-stake (PoS) as its consensus mechanism. This change was made to address security, energy consumption, and scalability issues. PoS is a consensus algorithm that chooses validators to create and validate new blocks based on their ownership of the cryptocurrency. This incentivizes validators to behave honestly and not engage in malicious activities that could harm the network.

The switch to PoS has resulted in significant energy savings compared to PoW. It's also helped to improve scalability, allowing for faster and cheaper transactions. In summary, Ethereum is a community-run blockchain platform that uses a peer-to-peer network, supports smart contracts, and uses PoS as its consensus mechanism for security, energy efficiency, and scalability [16].

3.6 Merkle Tree

Merkle trees (also called Binary Hash Trees) are a binary tree data structure. These trees are used extensively in the field of blockchain. They help encode blockchain data in a more secure and efficient manner. Data of a transaction needs to be stored on the blockchain. In Merkle Trees, you take a transaction concatenating it with another transaction and run that data through a hash function. This hash value is then concatenated with a similarly generated hash value and then hashed again. This process continues until only one hash remains for the entire block of transactions.

The leaf nodes of the tree are hashes of individual transactions, the intermediate hashes are called ‘branches’ and the top hash is called ‘root’. The Merkle root is what is stored in the blockchain. In the general blockchain context, the Merkle Tree helps in the verification of a transaction. To verify the transaction exists, one would need the transaction ID and the block number containing the transaction. You would retrieve the Merkle root from the block along with the Merkle path (a set of hashes that connect to the transaction ID). Now the Merkle root can be recomputed using the transaction you are trying to verify and the Merkle path. If the root matches the root stored on-chain, the transaction is valid [17].

Figure 2 shows how a Merkle tree would look like. If we were to verify the membership of TX3 in the block, we would query the blockchain for TX3 and the network would return back Hash3 and Hash01.
With these 2 pieces of information, the Merkel root can be recomputed and compared with the Merkle root on the blockchain [18].

![Merkle Tree](Merkle_Tree.png)

**Figure 2 Merkle Tree [18]**

The benefit of using a Merkle Tree is that the entire content of the blockchain or the block does not need to be downloaded to verify a transaction exists in that block. This makes verification faster.

In Trusticate, Merkle trees are used to batch transactions. Instead of having every educational credential issuance be its own transaction, the transactions are batched together using a Merkle Tree. The root of the Merkle Tree is stored on the blockchain. The certificate given to students contains details such as the Merkle Root, blockchain transaction ID, the Merkle Path to the root, the hash of the transaction, etc. These details are used in the verification process. Using Merkle trees helps reduce gas fees and makes the issuance process much faster [19].

### 3.7 Blockcerts

#### 3.7.1 Introduction to Blockcerts

Trusticate uses Blockcerts, an open standard for building blockchain technology. Blockcerts allows creating, issuing, viewing, and verifying blockchain-based digital credentials. Blockcerts enables the creation of tamper-proof and independently verifiable digital certificates that are recorded on a blockchain, making them resistant to tampering and fraud. Blockcerts uses the Bitcoin and Ethereum blockchains, as well as other blockchains, to record and verify the credentials. Blockcerts was developed by Learning Machine, along with the MIT Media Lab, and it has been adopted by various organizations around the world for issuing digital certificates, including governments, and companies. This is project
contains a lot of low-level tools required to build out digital verifiable credentials. The latest version of Blockcerts uses the W3C VC standards for certificates. This allows us to keep Trusticate at the frontier of technology. Blockcerts provides us with some key tools to issue verifiable credentials such as ‘cert-tools’, ‘cert-issuer’, ‘cert-verifier-js’, and ‘blockcerts-verifier’ [20].

3.7.2 Trusticate Ecosystem with Blockcerts

The Blockcerts library will be used in the Trusticate technology stack. For the VC model mentioned in 3.1, the issuer is an educational institution, the holder is the student or a recipient, and the verifier is a hiring company. Figure 3 shows the ecosystem of our technology.

![Figure 3 Trusticate Ecosystem in VC Model](image)

The ecosystem works as follows:

1. The issuer sends recipients an invitation for a blockchain credential.
2. The holder accepts invitations and sends his/her blockchain public address to the issuer.
3. The issuer hashes credentials onto the blockchain.
4. The issuer sends blockchain credentials to the holder.
5. The recipient sends credentials to the verifier.
6. The verifier verifies the credentials stored on the blockchain.

This is the overview of the workflow of the blockchain technology of Trusticate. Following the VC model keeps us at par with the current state of technology. The Trusticate web app would incorporate
steps 3 to 6 on the web application. There will be an active effort to incorporate steps 1 and 2 in the future as well.

3.8 Project flow diagram

There are three major steps in our project, and Figure 4 shows the sequential order of the flow.

![Project flow diagram](image)

**Figure 4** Project flow diagram

The first step is to set up issuing credentials. The Ethereum wallet will be set up for issuing certificates because without the digital wallet, educational institutions are not able to add credentials to the blockchain. There would need to be Ether available in the digital wallet because that would be needed to pay gas fees. Blockcerts is an open-source library for issuing and verifying credentials, and it will be used for setting up cert-issuer and cert-tools. More details of the first step are discussed in Section 3.9.

The second step is to set up the verification procedure. Blockcerts-verifier will be used to validate the input credentials. It will also help point out certificates which have been revoked. More details of the second step are discussed in Section 3.10.

The third step is to build and deploy a web application for issuing certificates. One of the purposes of building blockchain-verified certificates is to reduce the processing time for issuing graduation certificates. The web application will be a user-friendly platform for educational institutions to easily issue for their students. More details of the third step are discussed in Section 3.11.
3.9 Step 1: Setting up issuing credentials

3.9.1 Cert-issuer
Cert-issuer is a tool that allows the institution wishing to issue certificates to create a transaction on the Bitcoin or Ethereum blockchain from the issuer, an institution, to the recipient, a student. This transaction would contain information about the certificate and the hash of the certificate. Version 3 of the cert-issuer code is in line with the W3C verifiable credential specifications. The documents issued by the tool are signed with MerkleProof2019 LD signature [21].

3.9.2 Cert-tools
Cert-tools is a command line tool built by Blockcerts to allow creating and instantiating a certificate batch. It also enables the import and export of credential information. A candidate’s degree details can be imported using a CSV file where every line of the CSV file includes details for one student. It creates a JSON file using an image of the certificate (base64 format) and based on the input data of students which can then be moved to cert-issuer for digitally signing and issuing on the blockchain [22].

3.9.3 Batch issuance and Ethereum wallet
The first step for an institute trying to issue certificates using blockchain (specifically the Ethereum blockchain) is that they would need to create an Ethereum wallet. This will give them a wallet address and their public and private keys which are used for digital signatures. The wallet address will be used as the issuer’s address for the certificates. The Ethereum wallet must also have some Ether which will be needed to pay for the gas fees related to adding transactions to the blockchain.

It is possible to issue one certificate in one transaction but that can become quite expensive as there are gas fees associated with each transaction. To make the issuing process more efficient, one would use batch issuance where one transaction includes multiple certificates [19]. A Merkle tree is built using different certificate hashes, and it is used in the Ethereum transaction by storing the root of the Merkle tree in the data field of the transaction [23]. This is as discussed is section 3.6.

A blockchain certificate given to a recipient contains the certificate itself as an image and a receipt. The receipt contains:

- Student details
  - Name
  - Email
  - Public key
- Credential awarded
- University (issuer) details
  - Issuer ID
  - Name
  - URL
  - Email
- Issuance date
- Certificate ID
- Nonce
- Proof type – MerkleProof2019
- The proof value derived from the Merkle Proof algorithm [19]

### 3.10 Step 2: Setting up the verification procedure

#### 3.10.1 Introduction to the verification procedure

After the educational credentials have been added to the block, the credentials can be verified. The key to using blockchain for verifying credentials is to simplify the verification procedure, and thus it is important to make sure that the verification works. Blockcerts-verifier, which is a tool from Blockcerts, is used to validate the credentials and is further discussed in the next section.

#### 3.10.2 Verification of certificate by the requester

Verification of a certificate involves a set of computations. To remove any manual or complex work in the verification process, a tool called ‘Blockcerts-verifier’ exists which takes a blockchain certificate as the input and outputs if it is valid or not. The internal computations involve the following:

- Calculating the hash of the certificate and checking if it is the same as the hash of the certificate on the receipt.
- Checking if the Merkle path on the receipt is valid.
- Checking if the Merkle root is the one stored on the Ethereum block at the mentioned transaction ID.

If all 3 checks are valid, this would confirm the authenticity of the certificate [19].
3.11 Step 3: Building and deploying a web application for issuing certificates

3.11.1 Introduction to Developing the web application
The process of issuing certificates must be simple for educational institutes to increase the usage of our technology. Trusticate’s web application will be built with React.js for the front-end. React.js is a front-end library built on JavaScript which can be used to build fast-scalable web apps. Flask, Docker and Firebase are used for the back-end. The details of the web platform are discussed in the next section.

3.11.2 Web platform for educational institutions
Our end objective would be to create a web app that can be used by universities to create certificates on the blockchain very easily. The current state of technology requires a lot of low-level technologically complex work in issuing certificates on blockchain. Our platform would allow universities to use their public-private keys associated with their Ethereum wallet to link to the issuance of certificates process very easily. A CSV file of the graduates with specific information such as name, degree, and the public address of graduates is required to create certificates on the fly. The issued certificates will be accessible via JSON files or generated links to verify.

3.11.3 Front-end
The frontend of Trusticate will be built using React. React is a free and open-source frontend JavaScript library built and maintained by Meta. It helps build interactive user interfaces and web applications quickly and efficiently with much lesser code than using vanilla JS. React follows a modular approach which prevents rewriting a lot of code. The code is broken down into smaller components which can be used in different places as needed. There are several tools and public components which React integrates very well and avoids having to redo a lot of code. This helps make the build process much faster [24].

3.11.4 Back-end
There are mainly three technologies used to setup the backend of Trusticate: Flask, Docker and Firebase. This discussion excludes the usage of cert-issuer and cert-tools which have been discussed earlier.
3.11.4.1 Flask
Flask is a micro web framework built on python. It does not require any heavy libraries and tools. It has no database abstraction. It is called a micro web framework because it is very lightweight and extensible. It is also very easy to get started with and would not hinder the development process. Since Trusticate needed a backend to serve specific simple queries, Flask would be ideal [25].

3.11.4.2 Docker
Docker is a way to containerize our backend blockchain infrastructure. Docker is a platform for developing, shipping, and running applications in a somewhat isolated environment. Trusticate uses Docker to containerize backend blockchain services such as cert-tools and cert-issuer. Flask is deployed in the Docker container as well. This enables the backend to call functions in cert-tools and cert-issuer to do issuance of educational credentials [26].

3.11.4.3 Firebase
Firebase is a set of backend cloud computing services and application development platforms provided by Google. It hosts databases, services, authentication, and integration for a variety of applications. It is quick to set up and use. Trusticate uses Firebase auth, Firebase Firestore, and Firebase Storage. Their usage is further elaborated in Section 4.6 [27].
CHAPTER FOUR: PROJECT DEVELOPMENT

4.1 Overview

This chapter goes over the project development process for Trusticate. Section 4.2 shows the project schedule. The project can be divided into three types of development: blockchain-related development, frontend development, backend API development, and lastly database development. Section 4.3 to Section 4.6 discuss each type of development in order.

4.2 Project Schedule

Table 2 is the overview of our project schedule including phase, time periods, milestones, and deliverables.

<table>
<thead>
<tr>
<th>Phase</th>
<th>Time Periods</th>
<th>Milestones</th>
<th>Deliverables</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>By 2 Oct. 2022</td>
<td>• Research blockchain technologies to be used for the project&lt;br&gt;• Explore Blockcerts library</td>
<td>• Detailed project plan&lt;br&gt;• Project webpage</td>
</tr>
<tr>
<td>2</td>
<td>Oct. 2022 - Nov. 2022&lt;br&gt;Dec. 2022 - 22 Jan 2023</td>
<td>• Set up cert-issuer and a private Ethereum wallet to issue dummy certificates from legitimate wallet address.&lt;br&gt;• Set up batch issuance.&lt;br&gt;• Able to create certificates for different students.</td>
<td>• Preliminary implementation&lt;br&gt;• Detailed interim report</td>
</tr>
<tr>
<td>3</td>
<td>Feb 2023&lt;br&gt;Mar. 2023 -</td>
<td>• Complete building the process from issuing to validation&lt;br&gt;• Make the issuing and verification process easy to scale&lt;br&gt;• Build the web interface for educational institutions to</td>
<td>• Finalized tested implementation&lt;br&gt;• Final report</td>
</tr>
</tbody>
</table>
The team has completed all 3 phases as planned. The web application is now ready to be used on localhost. The backend and frontend have been set up. Blockchain APIs are integrated into the backend.

4.3 Blockchain-Related Development
Section 4.3 explores the different aspects of the blockchain development undertaken for the completion of the project.

4.3.1 Choosing Ethereum
The decision to develop the Trusticate solution on the Ethereum blockchain was based on careful consideration of several factors, including the network's expansive usage and network effect. Alternatively, Bitcoin was also considered but ultimately ruled out due to its limitations in terms of functionality.

One of the key reasons Ethereum was chosen was because of its ability to support smart contract functionality. This was deemed to be an essential component of the Trusticate solution, particularly if it was to be hosted on the internet (as opposed to being run solely on localhost). Smart contracts would provide a secure and efficient means for transactions to be signed via the web, without the need to expose the private key of the wallet.

In contrast, if Bitcoin had been selected as the blockchain of choice, it would have been challenging to develop a hosted web application for Trusticate that could securely sign transactions using the university's private key. Bitcoin does not support smart contracts, and therefore, it would be difficult to implement the required functionality without compromising on security.

Overall, the decision to develop Trusticate on the Ethereum blockchain was made after careful consideration of its strengths and weaknesses, as well as the specific needs of the project. Ethereum's
support for smart contracts and its widespread adoption made it the ideal choice for building a secure and scalable solution that could meet the needs of the university and its stakeholders.

4.3.2 Development environment
Blockchain development is done in Python using Docker containers. Docker is used because the environment setup for Blockcerts source codes is difficult to maintain. To test the issuance of certificates, a legitimate crypto wallet and Ether are required. A personal Metamask wallet was used to manage the Ether. The blockchains that were issued to are Ethereum Goerli (testnet) and Ethereum mainnet.

4.3.3 Cert-tools development
As introduced in section 3.9.2, Cert-tools is a command line tool built by Blockcerts to allow creating and instantiating of a batch of unsigned certificates. There were various steps taken to complete the setup of cert-tools and they are further elaborated on below.

4.3.3.1 Creating a conf.ini file
Cert-tools requires setting up a ‘conf.ini’ file which contains the different configurations needed for issuing unsigned certificates. The different configurations are used further down the pipeline by other Python files of the cert-tools library.

The conf.ini file stores details about the issuing university. The university’s official website, university name, administrative email ID, links to where ‘issuer.json’ and ‘revocation-list.json’ will be hosted in the future (as explained in section 4.3.3.2), and the university’s public key. All of this can be seen in Figure 5.
The conf.ini file also stores other information about the path of local files and folders and where they are stored on the system. This is used by other Python programs for storing the created unsigned certificates.

If there are additional fields that need to be added to each certificate, they are also stored in the conf.ini file. For instance, Trusticate uses additional fields to store information about the degree obtained (such as Bachelor of Engineering) and the credential awarded (such as First class honours). The template structure for these fields is stored in the conf.ini file and is used by the programs later.

### 4.3.3.2 Generating issuer.json

Cert-tools provides a file called ‘create_issuer.py’. This was fine-tuned by the team to meet the objectives of the project. The code takes as input the conf.ini file and outputs an issuer.json file for the university. This issuer.json file serves as the id of the university in the Blockchain domain. The issuer.json contains properties as shown in Figure 6. These field values are extracted from the conf.ini file. There is one additional field called ‘image’ which stores the logo of the university as a base64 string.
This file needs to be hosted on the internet at the endpoint that was mentioned in the conf.ini file. The endpoint will be stored as the value for a field called ‘issuer_id’ when issuing certificates. This is necessary as during the verification process of a certificate, the verifier program will check the ‘issuer_id’ on the certificate, go to the link mentioned, and check the fields of the issuer.json file such as the public key stored on the issuer.json and compare that to what is stored on the certificate. Also other checks are in place such as the issuer.json publicKey[0].createdAt date needs to be before the creation date of the certificate.

### 4.3.3.3 Generating revocation-list.json

A program called ‘create_revcation_list.py’ was created to issue ‘revocation-list.json’ for universities. The purpose of this JSON file is to allow a university to revoke different credentials that were previously awarded to students for various reasons. This revocation list would need to be hosted at the endpoint mentioned in the conf.ini file as it is also shown on the issuer.json file (as seen in Figure 6).

Figure 7 shows a sample revocation-list.json hosted on the internet. In this figure, there is no revoked certificate. Figure 8 shows a certificate with a revoked certificate. To revoke a certificate, the ‘id’ of the certificate and the reason for revocation must be added to the array “revokedAssertions”. This will mark a credential revoked when the verifier program tries to verify a certificate.
Figure 7 Revocation list example without a revoked certificate

```
// https://trusticate.xyz/v1/hku/revocation-list.json
{}
```

Figure 8 Revocation list example with a revoked certificate

```
// https://trusticate.xyz/v1/hku/revocation-list.json
{}
```

4.3.3.4 Creating a certificate template

The certificate template refers to a JSON file that acts as a template for the graduation certificates that are yet to be issued and contains all the fields for the graduation certificates’ JSON file. This template will have the values filled in for all the common fields, such as issuer details. The rest of the fields that will be personalized are filled in with a placeholder. Creating this certificate template improves efficiency when issuing certificates for 100s of students.
Figure 9 shows a screenshot of a sample template made for The University of Hong Kong. The issuer details are again extracted from the conf.ini file. The display.content field will contain the HTML code for displaying the graduation certificate image for each student. This image is personalized and discussed in section 4.3.3.5.

4.3.3.5 Generating personalized image templates

To display an image for the graduation certificate while using the Blockcerts-verifier tool, HTML code needs to be added to each certificate in the display.content field of the JSON file of the certificate. This HTML code is hardcoded into the certificate JSON. However, the HTML code needs to be personalized for every student during generation. In order to achieve this, a separate python program was created and invoked during the generation of individual certificates. This program does the following steps:
1. Creates a string of HTML code that contains gaps for student details such as name, degree obtained, honours classification, date, etc.

2. Using the `.format()` function, the string of HTML code is formatted with individual student details.

3. This HTML code now contains a personalized image. This code is converted to an image using the ‘imgkit’ library. The image is then lastly converted to a base64 string. A sample graduation certificate image can be seen in Figure 10.

4. Finally, to ensure the image displays correctly on the verifier tool, the image is displayed in a `div` with appropriate width, height, and position properties.

![Certificate of graduation sample](image)

*Figure 10 Certificate of graduation sample*

### 4.3.3.6 Issuing unsigned certificates

The last step while using cert-tools is to issue unsigned certificates. Here unsigned refers to the fact that these certificates have not been signed by the private key of the university yet. All details on the certificate are filled in and correct and would await being signed by the cert-issuer library (as discussed in section 4.3.4). Another Python program ‘instantiate_v3_certificate_batch.py’ uses the conf.ini file, the previously generated certificate templated, and a CSV file containing the roster of the students’ details generates the unsigned certificates for all students in the roster.
It is important to note that the CSV file must adhere to a specific format, with columns in a specific order and with specific names. Figure 11 shows a sample CSV file and its column names. The public key of each student is needed as it will be stored on the individual student graduation certificate. Storing this public key on the student’s certificate acts as proof of ownership of the certificate.

<table>
<thead>
<tr>
<th>name</th>
<th>publicKey</th>
<th>identity</th>
<th>hasCredential</th>
<th>educationalCredentialAwarded</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pranay Periw</td>
<td>0x9584731ee69B</td>
<td>pranayp@con</td>
<td>Bachelor of Engineering</td>
<td>Second Class Honours</td>
</tr>
<tr>
<td>Julie Park</td>
<td>0x4471Bdcd177d</td>
<td>u3571208@con</td>
<td>Bachelor of Engineering</td>
<td>First Class Honours</td>
</tr>
</tbody>
</table>

Figure 11 Sample CSV for storing student details with the specific column names

All the placeholder texts on the template are replaced with actual student data extracted from the CSV. Figure 12 can be seen and compared with the template file in Figure 9. A random ID is generated for every certificate and stored on the certificate. The ID is relevant when trying to revoke a certificate as shown in section 4.3.3.3.

Figure 12 Placeholder texts are filled in for unsigned certificate
After the issuance is done, the unique ID is used as the file name for the certificate and stored under unsigned_certificate as seen in Figure 13.

![Figure 13 File structure after issuance](image)

### 4.3.4 Cert-issuer development

#### 4.3.4.1 Introduction to cert-issuer development

The primary objective of utilizing the cert-issuer Blockcerts library is to generate graduation certificates via transactions on the Ethereum blockchain. The process was initially tested on the Ethereum Goerli testnet to ensure successful issuance before being tested on the Ethereum mainnet. Sufficient Ether in the issuing address is a crucial requirement for the issuance process to be successful. For testing purposes, a team member used a personal digital wallet. The issuance process entails multiple steps, which can be found in the subsequent subsections for further information.

#### 4.3.4.2 Creating a Config File

The config file is for configuring cert-issuer. Every university has different information for the conf file. There are several fields required to be specified in the config file.

- **issuing_address = <issuing-address>**
  
  This field indicates the public key of the issuing university must be specified.

- **usb_name = <path_to_usb>, key_file = pk_issuer.txt**
  
  When issuing certificates to the Ethereum blockchain, the private key that corresponds to the issuing address is required to digitally sign the transactions. Since private keys must be carefully dealt, private keys are stored in a USB. These two fields are to indicate the path to USB and the file name pk_issuer.txt.

- **chain = <ethereum_goerli | ethereum_mainnet>**
  
  Since cert-issuer offers issuing to both the Bitcoin and the Ethereum blockchain, the blockchain must be specified. Ethereum goerli and mainnet have been used, so it was specified as either ethereum_goerli or ethereum_mainnet.
• **verification_method = URL**

Merkle trees must be verified, and this field indicates the verification method of Merkle Proof ID. Since the JSON of the issuer identity is stored at the specified endpoint, the verification method is set to URL.

• **unsigned_certificates_dir = <path-to-unsigned-certificates>**

Unsigned certificates must be generated before issuing via cert-tools. This field specifies the path to unsigned certificates.

• **blockchain_certificates = <path-to-blockchain-certificates>**

Issuing certificates generates signed certificates that can be used to verify the authenticity of the certificates. This field specifies the path to where these signed certificates are to be stored.

Figure 14 shows a sample conf.ini file for cert-issuer. The chain and verification method may be the same for all issuance, but other fields may need to be modified for each issuance.

```ini
[IssuerInformation]
issuing_address = 0x1d415BEff60fe93405a31a301871af0aA3d3F678
usb_name = ./
key_file = pkIssuer.txt

[EthereumConfig]
chain = ethereum_mainnet
verification_method = URL

[DataDir]
unsigned_certificates_dir = data/unsigned_certificates
blockchain_certificates_dir = data/blockchain_certificates/47afPn6o6vMmhL4UF6tN
```

**Figure 14** Screenshot of conf.ini for cert-issuer

The conf.ini file is updated for each university based on data retrieved from the database. Once the file is updated correctly, the updated fields are used for issuance in the next steps.
4.3.4.3 Handling certificates in a batch

The issuance of certificates in a batch is enabled by the Merkle tree. The CertificateBatchHandler, which is present in cert-issuer, systematically iterates through the certificates. The certificates are initiated as an ordered dictionary with iterations performed in the order of insertion. The CertificateBatchHandler returns a batch of certificates organized in a Merkle tree structure.

4.3.4.4 Connecting to the Ethereum network

Following the preparation of a batch of certificates and configurations, the subsequent step is to establish a connection with the Ethereum network. The EthereumServiceProviderConnector in cert-issuer is utilized to configure the connectors to the Ethereum blockchain. This connector sends broadcasts to all available Ethereum APIs and known nodes on the Ethereum network to optimize the chances of a successful connection. By transmitting to multiple nodes, it also enhances the probability of effective transaction processing and minimizes the risk of network failure.

4.3.4.5 Issuing transactions to the Ethereum network

The EthereumTransactionHandler is the main controller of the issuance process. It verifies the balance in the issuing address and if there is an adequate amount of Ether available, it initiates the transaction. The handler performs tasks such as creating, signing, verifying, and broadcasting the transaction in a sequential manner. Upon calling the issue_transaction() function, the transaction ID is returned. This transaction ID can be verified through Etherscan. An illustration of successful transaction details visible on Etherscan can be observed in Figure 15.

![Etherscan Transaction Details](image)

**Figure 15** Successful transaction details on Etherscan
For instance, Figure 15 displays the particulars such as transaction hash, status, block number, the number of block confirmations, and timestamp. Transaction fees and gas prices are involved in this process. These fees cannot be avoided, but issuing certificates in a batch can minimize the overall costs.

### 4.3.4.6 Generation of signed certificates

Once the certificates have been successfully issued on the Ethereum blockchain, the cert-issuer library saves the signed certificates in the path specified in the configuration file. These signed certificates are in JSON format and contain information about the issuer and the students. Figure 16 displays an example of a successfully signed certificate that includes this information.

#### Figure 16 Signed certificate showing issuer and student details

Additionally, Figure 17 shows another part of the signed certificate that includes the Merkle tree path and the proof value for the certificate. This proves that the Merkle tree has been generated accurately for the batch issuance process.

#### Figure 17 Signed certificate showing Merkle tree path
The signed certificates also contain other details, such as the certificate image in base64 and the certificate type. These signed certificates can be used to verify the authenticity of the certificates through the verifier.

### 4.3.5 Blockcerts-verifier development

Blockcerts provides a ‘blockcerts-verifier-js’ library that allows integration of their verifier into web applications. However, this proved to be complicated because the component was not easy to integrate with a React project. The component was built using polymer 3 and there were some configuration changes that needed to be made in order to use it in the Trusticate web app. The component can be seen in Figure 18.

![Figure 18 blockcerts-verifier component on the Trusticate web app](image)

Now the signed graduation certificate JSON file can either be dropped into the component or the (IPFS) link to the certificate can be typed in the input box. This will lead to the verification of the certificate as seen in Figure 19.
Verification of a certificate using Blockcerts-verifier

The Blockcerts verifier will do the Proof Verification for the certificate and then a Status Check to ensure the certificate is valid or not. Proof verification involves the following 8 steps:

1. Getting transaction ID
2. Computing local hash
3. Fetching remote hash
4. Comparing hashes
5. Checking Merkle Root
6. Checking Receipt
7. Parsing issuer keys
8. Checking Authenticity

The status check involves checking the revocation status of the certificate. Since the certificate passes all checks, it is shown as ‘Verified’.
Figure 20 Screenshot of the verifier after closing the modal.

After closing the initial verification modal, the personalized certificate image along with other metadata is shown in Figure 20.

4.4 Frontend Development for the web application

Section 4.4 describes the different pages and components built on the web application of Trusticate.

4.4.1 Overview

The aim of the web application is to provide a user-friendly platform for educational institutions to issue certificates to the Ethereum blockchain. To achieve this goal, the web application is built using React for the frontend and Flask in a Docker container for the backend. The current version of the web application offers three main services, namely generating an 'issuer.json' file to identify the university, enabling batch issuance of certificates, and providing a verification feature for the issued certificates.

4.4.2 Authentication flow

There is a complete authentication flow created for the Trusticate web application. This involves registration, login, and sign-out functionalities.
For registration, 4 inputs are taken from the user as shown in Figure 21:

1. University Name
2. University website
3. University admin email ID
4. Password

The Firebase function `createUserWithEmailAndPassword()` is used to pass the email ID and the password to Firebase Authentication. The University name, website, and email id are also saved to the Firebase Firestore. Upon signup, the user is directly logged in.

For the normal login option, the email ID and password are taken as input as shown in Figure 22.
For login, the Firebase function ‘signInWithEmailAndPassword()’ is called with the email ID and password. Firebase compares the values stored in Firebase Authentication and returns the User data if found. At this point, sessionStorage is used to save an auth token for the user. When accessing a private link in the dashboard view, the auth token is checked if it is stored or not before displaying any results.

For signout, the ‘signOut()’ function from Firebase Auth is called. When the signout button is clicked, the stored auth token in sessionStorage is removed i.e. it is unset. This ensures that the user cannot access the private links without being logged in.

### 4.4.3 Home page

The home page of the dashboard is where the user is first directed to on login and signup. A personalized greeting message is displayed to the user. If the user has not set up the profile correctly, they will see 2 red boxes on the home page as seen in Figure 23. The first box is checking if all details on the profile page are filled in or not. If they are not filled in, it will be red in color. The 2\textsuperscript{nd} red box is checking if the ‘issuer.json’ file i.e. the JSON file that serves as the ID for the university has not been uploaded to the endpoint the university said they would upload it to.

![Welcome Admin @ University of Hong Kong](image)

**Figure 23** Unfilled Profile page and issuer.json not uploaded
Once the profile details are filled in, and the issuer.json file has been correctly uploaded to the endpoint it is supposed to be hosted at, the boxes will turn green as seen in Figure 24. At this point, the university can go ahead to the issue tab to issue graduation certificates.

![Welcome Admin @ University of Hong Kong](Welcome_Admin.png)

**Figure 24** Completely filled in the Profile page and issuer.json uploaded correctly

---

### 4.4.4 Profile

The Profile Page serves an important role in creating issuer.json and revocationList.json files. Universities can update their information, such as their name, admin email, website URL, public key, and the URLs where issuer.json and revocationList.json files will be stored. A screenshot of the Profile Page is shown in Figure 25, where universities can also update their logos.
The "Edit" toggle enables users to edit their profile details, and the "Save" button saves the changes to the database. Once the profile details are finalized, users can click on the "Generate issuer.json" button, which downloads two files: issuer.json and revocation-list.json. Universities need to upload these JSON files to the specified endpoints.

4.4.5 Issue

The issue page allows the university to instantiate a new batch of blockchain graduation certificates as seen in Figure 26.
Figure 26 Issue screen if there are no ongoing issuances.

Upon clicking on the ‘Create new graduation batch’ button, the user is redirected to the page where they can add details about the issuance as seen in Figure 27.

Figure 27 Page to enter batch details

On this screen, the user can add the Graduation month and year of the batch along with their faculty. This page accepts a CSV file containing the data of the students. The CSV file must be of the specific format with the exact 5 columns with the exact names as seen in the Figure. After uploading the file, the user must first validate it by clicking the validate button. If the CSV is in the correct format, the
user will see a green success message and can then save the information. After saving, the issuance data will be stored on Firestore.

The data is stored so as it is possible that the university may not immediately wish to issue the certificates on Ethereum. Hence, they can come back later and will not have to reupload the information. Figure 28 shows the successful validation of a CSV file and Figure 29 shows the data displayed on the previous page of issuance after saving.

![Figure 28 Successful validation of CSV](image)

![Figure 29 Screen after saving the data](image)

After saving the data, the issue button can be clicked. This will call an API which will start the issuance process on Ethereum on the backend. If successful, there will be a new message as shown in Figure 30. At this point, the issuance is complete from the user’s perspective. They can continue to check the status of the issuance on the History page of the website.
4.4.6 Issuance History

When the History tab is clicked, a table of issuances is shown. Figure 31 is a screenshot of the History page.

![Figure 31 History Page Screenshot](image)

The table displayed on the web application shows information about the batches of certificates issued by the user. The table contains columns with the batch name chosen by the user, the date of issuance, the number of students included in the batch, and the current status. The status can either be "success" or "issuing". If the status is "issuing", it means that the certificates have not been issued to the Ethereum blockchain yet, as there are multiple steps involved in the issuance process. Once the transaction has been successfully committed to the blockchain, the status changes to "success".

When a batch has been successfully issued, users can view the details of the certificates by clicking on the corresponding row in the table. This opens a screen as seen in Figure 32.
The table displays a list of students with their signed certificates that can be viewed and downloaded. Universities may wish to download all certificates to distribute them to students, and they can do so by clicking the "Download all certificates" button, which generates a zipped file of all the certificates with the filenames as students' names. The "Download an Excel file" button automatically downloads an Excel file that looks like Figure 33.

The Excel file is similar to the roster that universities uploaded at the "Issue" tab, with an additional column for the IPFS link. Since certificates in JSON format may be inconvenient, students can verify their graduation using the IPFS links, which are decentralized and permanent. This allows students to verify their graduation at any time.
4.4.8 Summary
To conclude the description of the frontend of Trusticate, the web application is a very simple, user-friendly alternative to manually issuing blockchain credentials using the Blockcerts library oneself. It provides the features to create the issuer.json, instantiate and issue a batch of certificates, and verify them, all in one place.

4.5 Backend Development for the web application

4.5.1 Overview of the Backend Development
The web application's backend is built using a Flask application that runs on Docker and utilizes REST APIs. Firestore is used for the database and is discussed in section 4.6. To test the REST APIs, port 5000 of Docker was mapped to local port 8082, and Postman was used for testing. Figure 34 illustrates the web application's architecture, including how REST APIs interact with the frontend and database. The subsequent subsections will provide more detailed explanations of each REST API.

![Figure 34 Development Architecture](image)

4.5.2 REST API: Generate issuer ID
On the Profile Page, users can download two JSON files: ‘issuer.json’ and ‘revocation-list.json’. ‘issuer.json’ serves as an identifier for the university, while ‘revocation-list.json’ is utilized to revoke certificates. When users click the button to generate JSON files, the React application sends a POST request to the ‘/trusticate/certtools/generate_issuer_json’ endpoint, including the school ID in the message body. Next, the configuration file in Firestore is updated with the user’s profile details. Then, using cert-tools, issuer.json is generated. The Flask library’s send_file() method is utilized to return the file as a response from the REST API. Figure 35 presents a screenshot of the Postman POST request to the endpoint for generating issuer.json.
The React application sends a POST request to the `trusticate/certtools/generate_revocation_json` endpoint when the user clicks the button to generate the `revocation-list.json` file. The message body of the request contains the school ID. The configuration file is updated with the user's profile details stored in Firestore. Using cert-tools, the `revocation-list.json` file is generated. The Flask library's `send_file()` function returns the generated file as the response to the request. Figure 36 displays a screenshot of a successful request to the `trusticate/certtools/generate_revocation_json` endpoint for generating the `revocation-list.json` file, with a status of 200 OK.

**Figure 35** POST request for generating ‘issuer.json’

**Figure 36** POST request for generating ‘revocation-list.json’
4.5.3 REST API: Issuing certificates

In the Issue Page of the web application, users can initiate the issuance process by clicking on the 'Issue' button. This action triggers a POST request, which includes the school ID and batch ID in the message body, to the endpoint '/trusticate/issue_batch'. The REST API for this endpoint can be divided into nine steps, and the flow is depicted in Figure 37.

When the API is called, it first updates the configuration files for cert-tools and cert-issuer by sending the university ID (UID) to Firebase and retrieving the profile details. Next, it fetches the student details for the specified batch issuance from Firebase using the UID and batch ID. Once the issuance process is completed, the signed certificates are stored in Firebase Storage (step 6). The API then updates the issuance status to True so that the universities can view the details of the issuance on the frontend.
Furthermore, Web3Storage is used to generate IPFS links, which are stored in Firebase. Finally, when all the steps are completed successfully, the API returns a status code of 200 to the frontend. The screenshot in Figure 38 shows that the API returns a 200 status code, indicating that the issuance process has been completed successfully.
Figure 38 POST request for issuing certificates

4.6 Database Development for the web application

The choice of database, as mentioned earlier, for the project was Firebase. Firebase was used as it was quick to setup and initialize. Three different tools of Firebase are employed in the project: Firebase Firestore, Firebase Authentication and Firebase Storage.

4.6.1 Firebase Firestore structure

Firestore is a NoSQL database. NoSQL databases are ways of storing data different from tabular databases. They are not relational databases. Firestore is a type of document NoSQL database. The comparison would be that every table and row in a relational database is a container and document respectively in the Firestore database. Each document contains data as a set of key-value pairs. In Firestore, each document can have further subcollections [28].

The project uses one main collection to store the data called ‘schools’. Each school would have its own document under this collection. There would be several fields storing the data for the school. Once the school creates an issuance for graduation certificates, a new collection called ‘issuance’ is created under the ‘schools’ collection. Each ‘issuance’ collection would have new documents for every student. The schema for this can be visualized in Figure 39.
The expand on the diagram, each document under the ‘school’ container would contain the fields as shown. The name, website, and email are received during the signup process. Now, every school document would have a subcollection called ‘issuance’. The issuance documents would have details specific for that issuance including the status which is ‘true’ or ‘false’ depending on whether the certificates in that issuance batch were successfully issued on Ethereum or not. Lastly, every issuance would have another subcollection called ‘students’ which would contain documents storing the details of each student. These details are extracted from the CSV file uploaded by the university.
The IDs for each document are randomized alphanumeric strings given by default by Firebase. Each document data is pulled as needed using Firebase JavaScript functions on the front end.

4.6.2 Firebase Authentication

Firebase authentication is an abstraction that Firebase provides to the authentication flow for any application. The abstraction is in terms of not having to waste time thinking about the storage and handling of passwords, email addresses, etc. The data is stored and managed by Firebase. Firebase uses a modified version of the scrypt hashing algorithm for hashing and storing passwords. The method used by Trusticate for authentication is using email. This was chosen as Trusticate expects the administrator to sign up with the university email ID. Figure 40 shows how the user looks once they have been added to the Firebase Authentication [29].

![Figure 40 Screenshot of Firebase Authentication](image)

4.6.3 Firebase Storage Structure

The REST API discussed in Subsection 4.5.3 generates signed certificates that are saved in Firebase Storage to enable universities to easily access them in JSON format at any time. Figure 41 illustrates the Firebase Storage structure, which has a top-level folder named "Trusticate" that contains folders for each university. Within each university folder, there are subfolders for each batch ID that represents a specific issuance. In each batch folder, there is a folder named "signed certificates" that stores JSON certificates for all students included in the batch. The separate batch folders are created in anticipation of future developments that may require storing other document types, such as unsigned certificates.
Figure 41 Firebase Storage structure
CHAPTER FIVE: FUTURE PLANS

5.1 Overview
This chapter presents the future plans of the project. Section 5.2 evaluates the future steps needed to improve the project such as smart contract development for hosting the web application on the internet, revocation using the web application, allowing universities to design their own certificate, and improving the verification of the university during signup.

5.2 Design individual certificate template.
Trusticate presently uses a standard certificate template that is fitting for graduation certificates; however, certain universities may opt to use their own unique template. In order to enhance the Trusticate project, the web application ought to furnish a service that enables universities to upload their own certificate templates. The current certificate is created using HTML, and allowing the university to modify the HTML code could aid in facilitating this process. Nevertheless, trying to do it using no-code may prove to be difficult since determining the location of each field may require more intricate development. This may entail the use of artificial intelligence to locate blank fields. The team will investigate how to make the certificate templates more dynamic.

5.3 Improved verification of the university
The existing registration process for Trusticate is uncomplicated, requiring only the university name, website, and admin email for a user to register. However, this simplicity presents a risk of fake graduation certificates being created by unverified users. To prevent this, additional measures of verification need to be introduced to ensure that the universities are legitimate. While issuer details on certificates can identify fake universities, it is still crucial to minimize the potential for fake certificates to be generated. The Trusticate team is confident that the registration system can effectively address this issue.

An automatic verification of a falsely created university that has issued certificates happens during the verification step. When using the Blockcert’s verifier tool, it displays the domain of the website where the ‘issuer.json’ file is stored (as seen in Figure 42). Hence, if the ‘issuer.json’ file is not stored on the official university servers, it will be visible during verification. However, we would like to prevent such malpractices in the first place by ensuring that the registration of users is from official university administrators.
Multiple solutions have been proposed for enhancing the registration system of Trusticate. One possible solution is to authenticate email addresses by verifying their domain name. As each university has its own email domain, like "connect.hku.hk" for HKU, the registration process can examine if the domain is valid. Not only the domain but also the sender’s email ID must be verified. For instance, if the email ID is ‘admin@hku.hk’, verifying the email address by sending a verification email would help prove ownership of the email account. Another solution is to authenticate websites by scrutinizing the SSL certificate, verifying the domain, assessing the privacy policy, and evaluating the domain's age. Amalgamating these solutions would solidify the legitimacy of new university sign-ups on Trusticate.

5.4 Smart contract deployment to allow hosting of the web application

Trusticate presently runs on localhost, both the frontend and the backend. The backend, as discussed in section 4.5, is running through a docker container because the cert-issuer and cert-verifier are python code bases with extensive files. Even though the backend is running on localhost, the certificates are being legitimately published to Ethereum. However, the end objective is to make the project available via the internet as hosted on trusticate.xyz. This brings a new problem to light. The private key of the university, which was okay to input locally as the code is all running on localhost, is no longer okay to submit on the internet. The way around this is to use smart contracts and signing of transactions through a wallet like Metamask.

The challenge for the future is to convert the code bases of cert-tools and cert-issuer to smart contracts. This is an extensive task as the code is quite lengthy and tricky. There is a lot of usage of different python libraries which would need to be transpiled to either Vyper or Solidity. It may be possible to use
Chainlink External Adapters to do some of the computation off-chain to limit gas fees. This process is much more complicated and would take longer [30].

5.5 Using DID as a verification method

The current method for identifying the issuer is via a JSON file that is hosted on the endpoint given by the user. Although the current method is one of the common and convenient ways to verify the issuer, the identities are not decentralized. The authentication power is given to Trusticate, making educational institutions rely on Trusticate for identifying themselves. To maintain a more decentralized registry, the team plans to incorporate using DIDs as the verification method (Refer to section 3.2). With DIDs, more details can be added to the DID document, and they provide credentials that enforce the identities. The team will focus on generating valid DIDs and issuing and verifying credentials that are bound to DIDs.
CHAPTER SEVEN: CONCLUSION

The verification of educational credentials is important in the hiring process, but it is a time-consuming task for employers. Trusticate aims to save time and resources for employers by verifying the educational credentials of candidates in an extremely quick and easy way using the blockchain system. The project targets to allow legitimate educational institutions to use Trusticate to issue students’ graduation certificates and to get rid of the tedious and time-consuming work for employers.

The Trusticate project has been completed and works on localhost. All 3 main parts of the development process were completed including Blockchain development, frontend development, and backend development. The project allows for a user friendly UX to enable universities to sign up and deploy graduation certificates on the blockchain efficiently and easily. The platform allows for management of the certificates, including easy verification. The certificates are also deployed on IPFS for decentralization. Trusticate has made graduation credentials easy to access and easy to verify.
References


