PRIVATE PERMISSION BLOCKCHAIN FOR OPTIMIZED INVOICE MANAGEMENT SYSTEM

Thansinee Pichaibunditkun

A Thesis in Partial Fulfillment of the Requirements
for the Degree of Master of Science Program in Information Technology
Graduate School
Thai-Nichi Institute of Technology
Academic Year 2022

Thesis Topic	Private Permission Blockchain for Optimized Invoice Management System
By	Thansinee Pichaibunditkun
Field of Study	Information Technology
Thesis Advisor	Asst.Prof.Dr.Ferdin Joe John Joseph

The Graduate School of Thai-Nichi Institute of Technology has been approved and accepted as partial fulfillment of the requirement for the Master's Degree

…….. Dean of the Graduate School
(Assoc.Prof.Dr.Warakorn Srichavengsup)
Month…… Date…… Year……..……

Thesis Committees

…….. Chairman
(Assoc.Prof.Dr.Kuntpong Woraratpanya)

…….. Committee
(Dr.Sarayut Nonsiri)

…….. Committee
(Dr.Pramuk Boonsieng)

…….. Advisor
(Asst.Prof.Dr.Ferdin Joe John Joseph)

THANSINEE PICHAIBUNDITKUN : PRIVATE PERMISSION BLOCKCHAIN FOR OPTIMIZED INVOICE MANAGEMENT SYSTEM. ADVISOR : ASST.PROF.DR.FERDIN JOE JOHN JOSEPH, 67 PP.

Blockchain technology has become popular in the recent years due to Bitcoin and Ethereum which are potential in terms of security, transparency and sharing in the same network by Peer-To-Peer network. Though this technology has high costing, it is worth to use in the businesses to reduce fraudulences from inside and outside of organization. Invoice system is a technology that all companies invest to protect the data of invoice management because it is important for analysis by the trends of business each year. In this paper the use of blockchain technology to protect the data of invoice by developing the invoice management system with smart contract for invoice rules to approve, reject, and to protect the data is proposed to find out changes of invoice in blockchain network system by a web application in intranet to manage the limitation of small business. The conceptual framework and the output obtained by the developed system shows the feasibility to develop a private blockchain system for invoice management.

Graduate School Student's Signature……

Field of Study Information Technology Advisor's Signature……

Academic Year 2022

Acknowledgement

First of the foremost, I would like to express my deep appreciation to Asst. Prof. Dr. Ferdin Joe John Joseph accepted for my ideas and supported my Master's Degree. Without him, the dissertation would not be possible and completed. Furthermore, I would like to express my gratitude to my research committee, Assoc. Prof. Dr. Kuntpong Woraratpanya, Dr. Sarayut Nonsiri, and Dr. Pramuk Boonsieng for their encouragement and valuable suggestions on my research.

Moreover, I would like to express my sincerest thank you Ms. Arunotai Uoonjai who always supported and recommended the ideas of front end and coordinate for completely the research in a technicians part. My life would be harder without her assistant.

Especially, a special thank you to Mrs. Benjamas Pichaibunditkul for her positive attitude and motivated me to do this paper.

Finally, I would like to thank you all support from my friends and Thai-Nichi Institute of Technology that is a part of my thesis achievement. I am grateful for the opportunity from the Thai-Nichi Institute of Technology that provided me the different knowledge from my background.

Thansinee Pichaibunditkun

Table of Contents

Table of Contents (Continued)

List of Figures

List of Figures (Continued)

Chapter 1
Introduction

1.1 Statement of the Problem

A business can be described as an organization or enterprising entity that engages in professional, commercial, or industrial activities [1]. The business can be for profit entities or a non-profile organization which depends on the purpose of the business owner. The range of businesses is also different with sole proprietorships, partnerships to large, international corporations. Furthermore, business comprises a variety of fields; for example, real estate, banking, and so on. However, all businesses still have the same objective to survive, grow, and have stability.

In addition, each organization is composed of various departments, such as Human Resources, Sales, Marketing, and Finance, which work together to achieve the organization's objective. Consequently, many organizations share information in each department so to work together to find ways to initiate direct and indirect profit. Therefore, information security is a major concern in every organization, especially in the Finance Department.

The Finance Department is responsible for obtaining and handling money and other assets on behalf of the organization. It also has the responsibility to provide an economic analysis to improve the organization's business strategies. As such, the Finance Department has to provide many documents about the organization's situation; for example, a permanent file (ex. company certificate and financial statement), purchasing document (ex. purchase request and purchase order), sales tax and input tax documents, paying off debt (ex. receipt voucher and invoice), and other documents. From the example, it can be seen that there are numerous important documents, which should not be disclosed outside the organization because this would have an impact on the stability and strategies.

An invoice is not the most important financial document, but it shows a customer's requirement and explains about the business's cash flow and finances. Mostly, small businesses also do not give precedence to document management, and document security is essential not only for data protection compliance, but also for maintaining trust

with the customer. However, trust is not the only reason to provide document security, but there are also many other reasons, such as minimizing the risk of a data breach, the financial impact of a data breach, customer confidence in the organization's products and services, reputational damage, and the rise of cybercrime. [2] Findex Group Limited, a financial consultant company in Australia, has provided an article on their website about the invoice hacker or Invoice Redirection, as cybercriminals are impersonating businesses and suppliers, accessing emails, and intercepting invoices. The hacker sends the email with an invoice, including changes to their bank account details and asks the person to pay to the hacker's account. These business email scams cause businesses significant financial damage, thus accounting for 63 percent of all business losses reported by Scamwatch. The average loss is nearly US $30,000/year [3].

Blockchain has been a buzzword for the past few years, which people know from Bitcoin, Ethereum, and other digital currencies. Blockchain is a shared, immutable ledger that facilitates the process of recording transactions and tracking assets in a business network. An asset can be tangible (a house, car, cash, and/or land) or intangible (intellectual property, patents, copyrights, and/or branding). Virtually anything of value can be tracked and traded on a blockchain network, consequently reducing risk and saving costs for all involved [4]. In Thailand, many industries would like to apply blockchain technology to their working operations because people trust the system that has enhanced security, greater transparency, instant traceability, increased efficiency and speed, and automation.

Hence, blockchain in the invoice management system would be helpful for a user who works in the financial field because he/she could track any transaction from the past until the present by using a hashing algorithm in the blockchain. As a consequence, the document would be saved in a private blockchain only by approval of the validators (a proof-of-work process) to build the new block and if consensus succeeds, add the block to the chain. Moreover, the block or document could not be changed that is the reason why blockchain would be appropriate for the financial field.

Encryption is at the heart of what makes a block special. Blocks are assigned a cryptographic hash, which are generated exclusively for its data and with the hash for the previous block. Hash work is like a fingerprint for accessing the data if the hash becomes completely different. Thus, that block's adjusted hash would also be passed to, and

recorded with, the subsequent block. In case a hacker changes one bit, the recorded hash for that block would not match the block's new hash, and the blockchain would alert the administrators [5]. Hence, the security of blockchain has received the trust by many people.

In this study, blockchain technology was used for an invoicing system by utilizing a private blockchain to offer small businesses with the confidence, security, and efficiency that they would need at a reduced cost of investment.

1.2 Objectives of the Study

1.2.1 To offer a blockchain-based invoicing management solution for usage in a private network and restrict authorization to the relevant parties.

1.2.2 To determine if a system would be suitable for usage by small business firms.

1.2.3 To deliver the system’s counterfeit invoice result.

1.3 Scope and Limitations

1.3.1 The prototype would only be used for small businesses.

1.3.2 The Hyperledger Besu would be used, which is an open-source Ethereum client for only a private network.

1.4 Structure of the Thesis

1.4.1 Chapter 1 Introduction

1.4.2 Chapter 2 Literature Review

1.4.3 Chapter 3 Research Methodology

1.4.4 Chapter 4 Production and Results

1.4.5 Chapter 5 Conclusion and Future Research

Chapter 2
Literature Reviews

This research provided the design and developed a prototype of an invoice system by using blockchain technology to prove that this form of technology would be suitable for an invoice system in a small business. The literature review provided the related technology, tools, and research.

2.1 Related Technologies

2.1.1 Blockchain technology

2.1.2 Consensus type

2.1.3 Smart contract

2.1.4 Hashing algorithm

2.1.5 Web3

2.2 Related Tools

2.2.1 Hyperledger Besu

2.3 Related Research

2.3.1 One method for implementing privacy protection of electronic invoices based on blockchain

2.3.2 Performance analysis of Hyperledger Besu in a private blockchain

2.1 Related Technologies

2.1.1 Blockchain technology

Blockchain technology refers to a chain of time stamped and immutable blocks, which are linked to each other using cryptography [6]. Peer-to-peer network has been applied for blockchain technology that each node is interconnected to another node. However, each transaction must validate the transactions and achieve consensus by a "miner" to create the proof of work. The blockchain stores the data by “Distributed Ledger Technology”. When a transaction occurs, every node would receive an announcement (Figure 2.1).

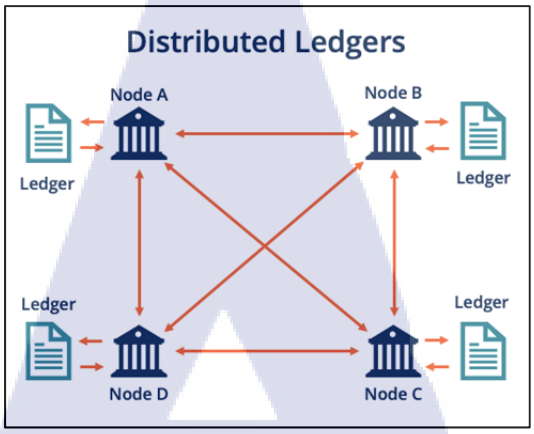

Figure 2.1 Distributed ledgers

In 2008, Nakamoto introduced Bitcoin that was the first proposed cryptocurrency, which introduced the blockchain as a distributed infrastructural technology [7]. Bitcoin allows the user to securely transfer cryptocurrencies named “Bitcoins” without a centralized regulator. This is also the concept for Ethereum, NXT, and Hyperledger Fabric.

2.1.2 Consensus type

Blockchain technology has a key aspect of determining which the user publishes to the next block [8]. There are many possible consensus models used for permission as the blockchain does not need to have the trust of a third party to provide the validation. Hence, the consensus type is more important for the approval node to the block. This is as follows:

- Proof of Work Consensus Model (POW)

There is a consensus process by using mathematics to proof the transaction that uses more time, and it would also use a miner to validate the transaction. The miner would receive the compensation after proofing the transaction to the blockchain. This consensus model is used for Bitcoin in a public blockchain.

- Proof of Stake Consensus Model (POS)

The validator must have the asset for completing the transaction. If the validator has a lot of assets, he/she would have a high opportunity to receive the authority

to complete the transaction and get the compensation. This consensus model is used for Ethereum in a public blockchain.

- Proof of Authority Consensus Model (POA)

The validator is clearly determined by everyone or a group of authority. The consensus would only be approved by the validator to verify the transactions and build new blocks. It is mostly used for a private blockchain, such as Hyperledger Besu.

2.1.3 Smart contract

In the 1990s, a cryptographer named Szabo coined the term “smart contract” and defined it as “a set of promises, specified in digital form, including protocols within which the parties perform on the other promises.” [9] Since the concept was accepted, smart contracts have evolved, especially after the introduction of Bitcoin in 2009.

Smart contracts are simply programs stored on a blockchain that run when predetermined conditions are met [10]. They are normally used to automate the execution of an agreement, and all participants would be immediately certain of the outcome without any intermediary’s involvement or time loss. A smart contract is mostly used when releasing funds to the appropriate parties, registering a vehicle, sending notifications, or issuing a ticket. When the transaction is completed, the blockchain would be updated; therefore, the transaction could not be changed. The transaction could only be accessed by a person who has been given permission.

2.1.4 Hashing algorithm

Hash functions are useful for a security application by a mathematical function, which converts the numerical input value into another compressed numerical value. Cryptographic hash functions use a combination of mathematical functions that convert the input of a variable size to an output of a fixed size, which is referred to as a hash. They are designed in such a way that the results are irrevocable, i.e., the hash function should only go one way.

SHAs are one of the most widely used hashing algorithms because of their effectiveness. SHAs are used to maintain the veracity of the files and serve the

purpose of ordinary hash functions, which is to check for duplicate data or file corruption during a transfer with the help of checksums. [11]

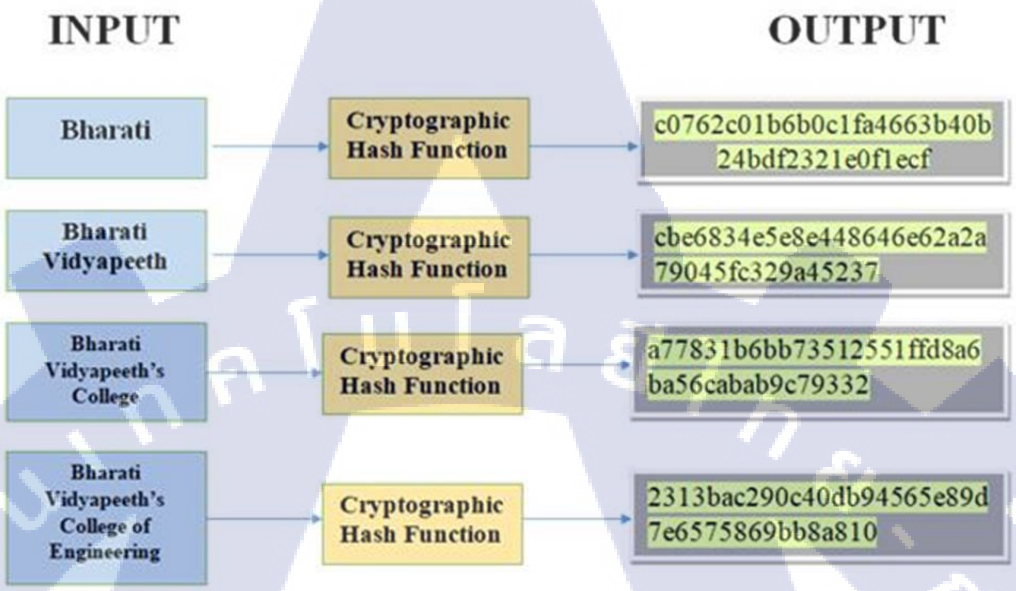

Figure 2.2 Text outputs after hashing

2.1.5 Web3

Web3 is a concept for a decentralized web to be built, operated, and owned by its users. Web3 puts power in the hands of individuals rather than corporations.

When Web 1.0 was established in 1989 at CERN, Geneva by Berners-Lee, the first inception was mainly static websites owned by companies with no interaction with a user - individual. Web 1.0 was known as “the read-only web” then the world wide web evolved to Web 2.0 in 2004. Web 2.0 began with the social media platforms to be “Read-Write” and has continued to do until the present. The companies providing content to the users also began to provide platforms to share user-generated content and engage in user-to-user interactions. Mostly, companies get the profit from the increasing number of users to do advertising. [12]

Figure 2.3 The evolution of the World Wide Web (Web 1.0 - 2.0)

The promise of 'Web3.0' was coined by Ethereum co-founder Gavin Wood for solving the problem that crypto adopters experienced because the website required too much trust. As a consequence, the concept of Web 3.0 was created to use blockchain, cryptocurrencies, and non-fungible tokens (NFTs) to give power back from many companies to the users.

2.2 Related Tools

2.2.1 Hyperledger Besu

Hyperledger Besu is an open source Ethereum client developed under the Apache 2.0 license and written in Java. It runs on public and private networks. Hyperledger Besu includes a command line interface and JSON-RPC API for running, maintaining, debugging, and monitoring the node in an Ethereum network [13]. Moreover, Hyperledger Besu has had smart contract development both in public and private networks.

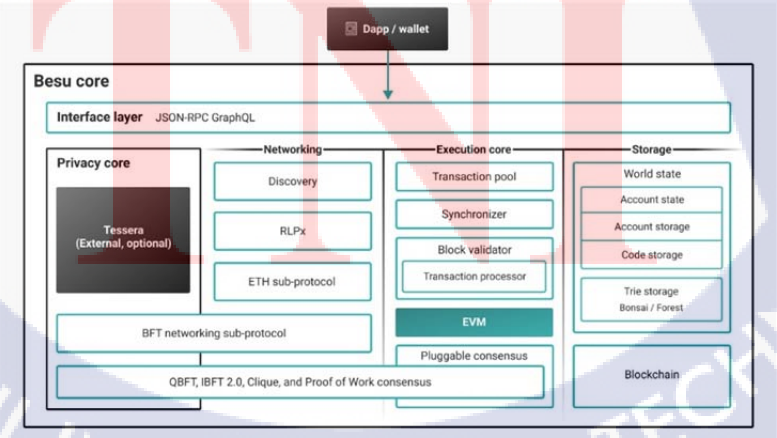

Figure 2.4 Architecture of Hyperledger Besu

The private network of Hyperledger Besu is not connected to Ethereum Mainnet or an Ethereum testnet. The private network has to use a different Chain ID and proof of authority consensus (IBFT2.0 or Clique). Figure 5 compares the difference between IBFT2.0 and Clique.

Property	Clique	IBFT 2.0
Finality	Forks	No Fork
Validators Required	1	4
Liveness	½ validators live	>= ⅔ validators live
Speed	Faster than IBFT 2.0	Slower than Clique

Figure 2.5 The difference between IBFT2.0 and Clique

2.3 Related Research

2.3.1 One method for implementing privacy protection of electronic invoices based on blockchain

The Chinese government had a concern about electronic security in 2019. The government had promoted blockchain technology for researchers by providing scholars and focusing on cryptography. As such, an invoice system is one research project that has worked in China.

For the electronic invoice in China, organizations need to apply for a legal certificate from the CA Center then they can provide the information of the electronic invoices, which is encrypted and stored in a block. However, if the invoice is completely stored in the block and the user needs to check the invoice, he/she would need to enter the index information to search for the existence of the related invoices. If the invoices exist, the system would compare the hash value of the invoices element information with the hash value provided by the user (Figure 2.6).

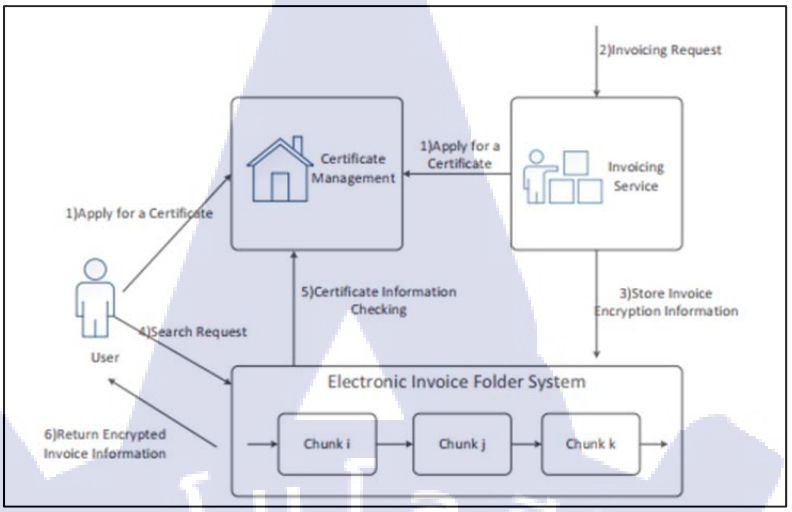

Figure 2.6 Model of an electronic invoice folder system [14]

The system is divided into the API layer, capability layer, foundation layer, and storage layer with a different focus for providing a stable and reliable service for the system (Figure 2.7).

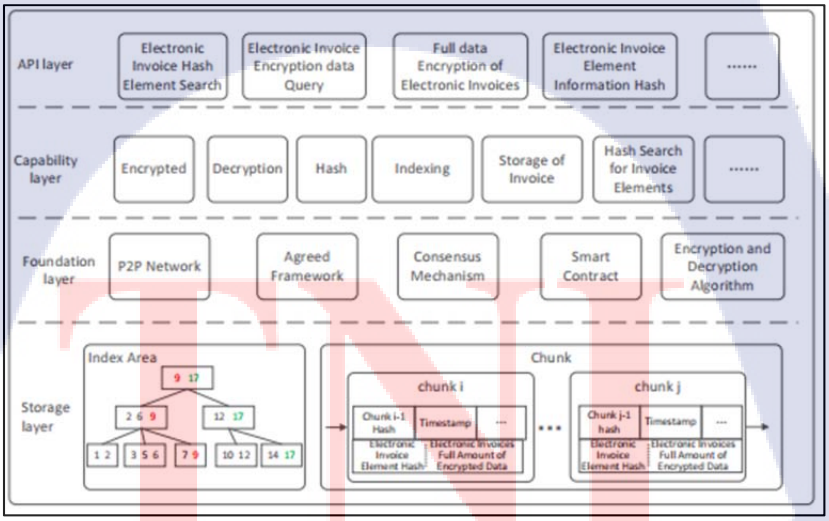

Figure 2.7 The architecture for an electronic invoice folder system [14]

API layer is concerned with the entry point through the system which wraps the internal function and assembles the services needed to expose an external base.

Capability layer divides the capabilities of the basic services of the system, but it is finer and smaller than the various services provided by the API layer. The main purpose of dividing the capability layer is to improve the reusability of the system's functions and to build highly cohesive and low-coupling system modules. [14]

Foundation layer is for supplying some of the basic support in the system. It is mainly proposed for providing a reliable and stable basic service for the system; for example, peer-to-peer network, consensus mechanisms, and smart contract.

Storage layer comprises two parts, which are the index area that stores information in the way of a B+ tree, and the full amount of encrypted data for the electronic invoice is stored in the system area.

In summary, three types of storage have been compared, which are C1 that is an IRS public repository of electronic invoices and is a public repository, C2 is third party platforms that provides relevant services and capabilities, and C3 is a blockchain system which is mentioned in this paper. The compared results are shown in Figure 2.8.

Scenario	*transmission security*	*query security*	*storage security*	*privacy protection*	*prevention of information abuse*
C1	N	N	Y	N	Y
C2	N	N	N	N	N
C3	Y	Y	Y	Y	Y

Figure 2.8 Compared results for the paper

Blockchain technology can provide transmission security, query security, storage security, privacy protection, and prevention of information abuse by the ability of the blockchain. Thus, the author recommends the use of a blockchain to provide an invoice system instead of using other forms of storage to store the invoice's information.

2.3.2 Performance analysis of Hyperledger Besu in a private blockchain [15]

Recently, Ethereum has become one of the mainstream blockchain platforms for developing enterprise applications. There are many Ethereum clients using various programming languages like Go, JavaScript, Python, and Java. However, their

blockchains do not meet the specific need of the enterprises. In particular, the capability to enforce membership (permission), provide high performance, and perform private transactions, which only allows participants of those transactions to access the metadata or payload. Hyperledger Besu has recently gained much attention in constructing decentralized enterprise applications. This research studied the performance and scalability of Hyperledger Besu, the impact of system configurations, and chain parameters on the performance metrics of the throughput and latency, which had not been thoroughly studied. This paper would propose a load balance-based performance evaluation architecture (Figure 2.9).

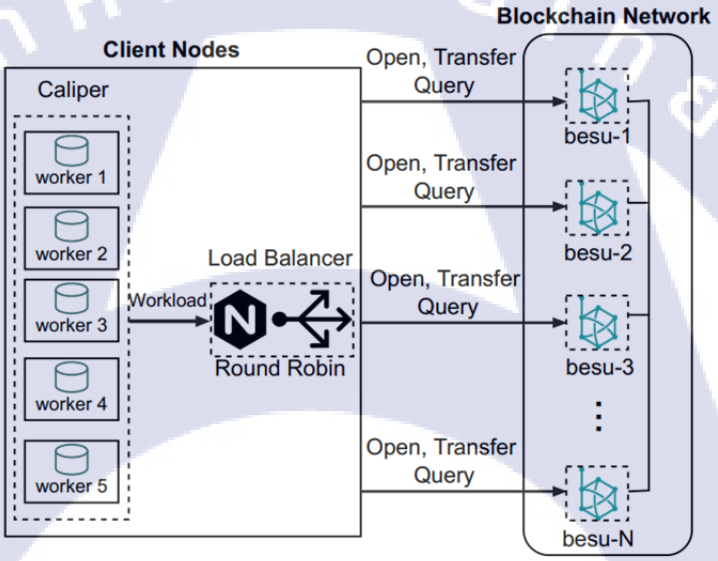

Figure 2.9 Hyperledger Besu performance evaluation architecture

POA is more popular than POW in Hyperledger Besu. As such, this paper presented an in-depth experimental evaluation of Hyperledger Besu for its POA consensus algorithms (the POAs of Hyperledger Besu are Clique, IBFT 2.0, and QBFT), which showed its performance characteristics under various configurations. Through extensive benchmarks, critical parameters were identified that would have an impact on the performance and scalability of Hyperledger Besu, and the extent of the influence and the root causes of the identified bottlenecks were analyzed through log analysis.

Property	Clique	IBFT2	QBFT	Ethash	PoS
Type	PoA	PoA	PoA	PoW	PoS
Finality	No	Yes	Yes	No	No
Liveness	1/2	1/3	1/3	1/2	1/2
Throughput	High	High	High	Low	Medium
Quorum	1/2	2/3	2/3	1/2	1/2
BFT	No	Yes	Yes	Yes	Yes
Usecases	Private, Rinkeby, Goerli	Private	Private	Public	Public

Figure 2.10 A comparison of Hyperledger Besu's consensus protocols

Clique is a POA consensus protocol presented in the Ethereum Improvement Proposal (EIP) 225. In this protocol, this was fixed to set the "signers" as the authorization to collect and execute the transection to create or mine a block. The signer would be allowed to seal a block every ⌊N/2⌋ + 1 blocks.

IBFT 2.0 is Istanbul Byzantine Fault Tolerance (IBFT), which is a variant of the Practical Byzantine Fault Tolerance (PBFT). IBFT is suitable for the blockchain network, which was first proposed informally in EIP650. IBFT uses a leader-based (or voting-based) consensus for approving the block through three phases of pre-prepare, prepare, and commit. Before each round, the selected proposer would propose the new block proposal and broadcast it with a pre-prepared message. Then, other validators would enter the state of pre-prepared and broadcast to prepare the message. When receiving the 2f+1 prepared messages, the validators would enter the prepared state and then broadcast the committed message. Finally, the validators would wait for the 2f+1 committed message to enter the committed stage so to attach the proposed block to the chain.

QBFT was created to resolve the safety and liveness issues from IBFT from the whole system being caught by two honest nodes locking onto a different block. Quarum blockchain was proposed and developed a variant of IBFT to "QBFT". However, the phases of work were the same as IBFT in order to pre-prepare, prepare, and commit. Nevertheless, finally, a new block was inserted into the chain by the proposer after the 2N/3 committed messages. If a consensus was not achieved among all the

validators before the predefined time, a round change would happen with all the clock times being reset.

In this paper, experiments were set up to conduct testing, which included the network size, node flavor, load balancing, consensus algorithm, and block period seconds. The control variate method was also set up to be explored (Figure 2.11).

Test Parameters	Varying	Fixed
Send rate	100 → 10,000 with step=100	8-LB-2C7.5G-QBFT-1S
Network size(N)	4 → 36 with step=2	⋆-LB-2C7.5G-QBFT-1S
Load balance(LB)	LB, NLB	8-⋆-2C7.5G-QBFT-1S
Node flavor(F)	1C7.5G, 2C7.5G, 2C15G, 4C15G, 4C30G, 8C30G, 16C60G	8-LB-⋆-QBFT-1S
Consensus(C)	Clique, IBFT2, QBFT	8-LB-2C7.5G-⋆-1S
Block time(BT)	1S, 2S, 3S, 4S, 5S	8-LB-2C7.5G-QBFT-⋆

Figure 2.11 Tested parameters in the experiments

In this paper, the experiments were set up based on the cloud and each node was run by a docker. They used the benchmark tool Hyperledger Caliper v0.4.2 with Ethereum SDK v1.4 to connect and test the deployed Hyperledger Besu blockchain networks. Over 22,500,000 transactions were generated and sent to be deployed in the system.

For the result, Hyperledger Besu compared the result of the paper of Hyperledger Fabric (HLF) and private Ethereum, which were experiments with Hyperledger Caliper to benchmark and evaluate the performance. It was found that Hyperledger Besu had limited scalability, block time, and block size. Nonetheless, the consensus and transaction types had the same performance or at a better level from other blockchains.

Chapter 3
Methodology

This research designed and developed the prototype of an invoice system based on blockchain technology. The system provided a private blockchain, which was appropriate for a small business as it had low security for the data system. There were seven steps used to conduct the work (Figure 3.1).

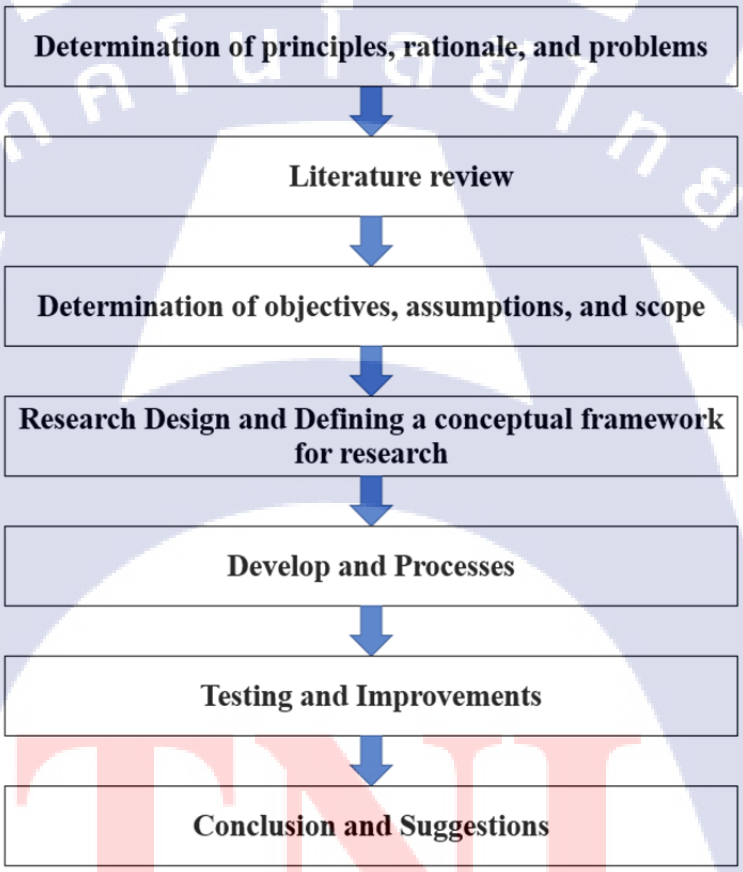

Figure 3.1 Work process for the research

3.1 Determination of the Principles, Rationale, and Problems

Blockchain technology has been gaining much interest for using in an invoice system because an invoice is a very important document for the company, as it tells a lot of undisclosed information, such as the customer's information, rate of the product, and

so on. For protection from hacking invoice information or making a fake invoice, the author selected blockchain technology for the invoice system by using six criteria for consideration of using blockchain technology (Figure 3.2).

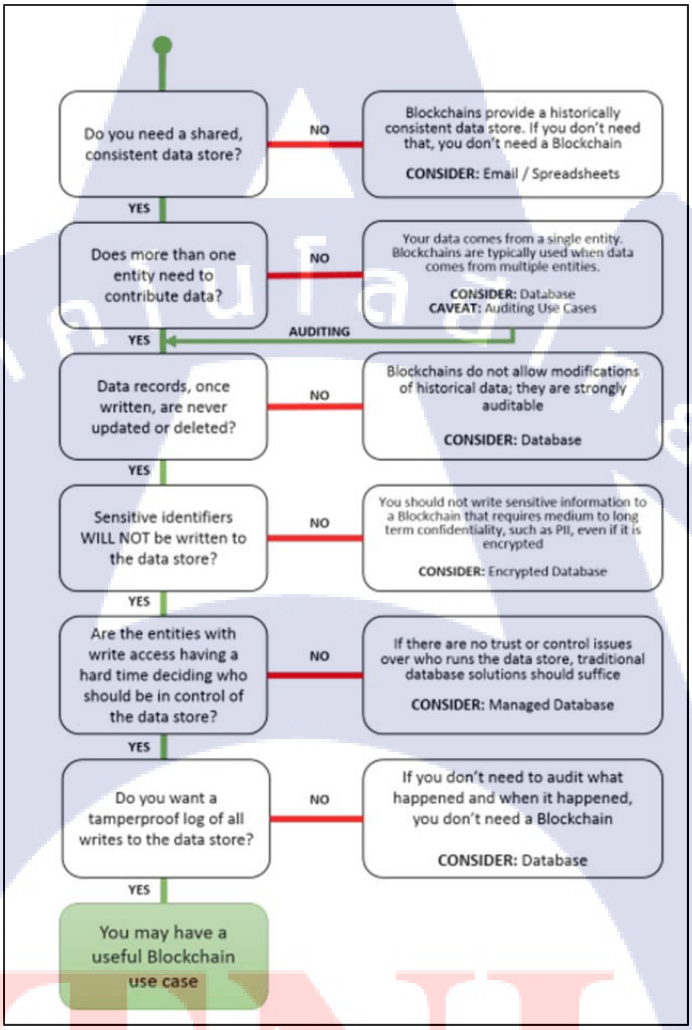

Figure 3.2 Criteria for consideration for using blockchain technology

In considering the criteria, an invoice management system should use blockchain technology because an invoice is one of the spreadsheets for storing and needing to be shared between the company and clients. Therefore, an invoice would be contributed by more than one person, which they would require information from the seller, and the accountant would need to provide the data for approval by an authorized person.

However, an invoice cannot be duplicated by the invoice ID or data. This is the reason why blockchain technology could be applied for this system because the data would not change and update the information, but the data would be recorded in the system as always. Hence, this data should be controlled by an authorized person, such as an accountant or the owner of the company because it is a payment documentation, which is concerned with the income of the company and all the customer's details.

Moreover, every six months to one year, auditing should be processed in each company. That is the main reason for providing blockchain technology for rechecking the data in the invoice, tracking the real income-outcome of the company, and reconfirming that the data were not changed from the system.

3.2 Literature Review

The author started reviewing the document, which was related to the invoice system by using the blockchain technology field and studying the problem in each document. The document scope was in the years 2018 - 2022, and the related tools technology, such as Solidity Language, Hyperledger Besu, and Smart Contract was used. The stages of the literature review has been provided in Chapter 2.

3.3 Determination of the Objectives, Assumptions, and Scope

This process was mentioned in Chapter 1 of this research. The author designed and developed a prototype of an invoice system, which would be appropriate for a small business by using private blockchain technology.

3.4 Research Design and Defining a Conceptual Framework for the Research

For the conceptual framework, this research used Hyperledger Besu for the private blockchain then created the wallet by Metamark to do the smart contract. However, the framework needed to work in the web browser then the author provided the Front-End process for creating the webpage to input the invoice to request for the approval (POA) to the current block. The Back-End process was created for the API to be connected to the blockchain then integrated for testing whether the web page was working or not.

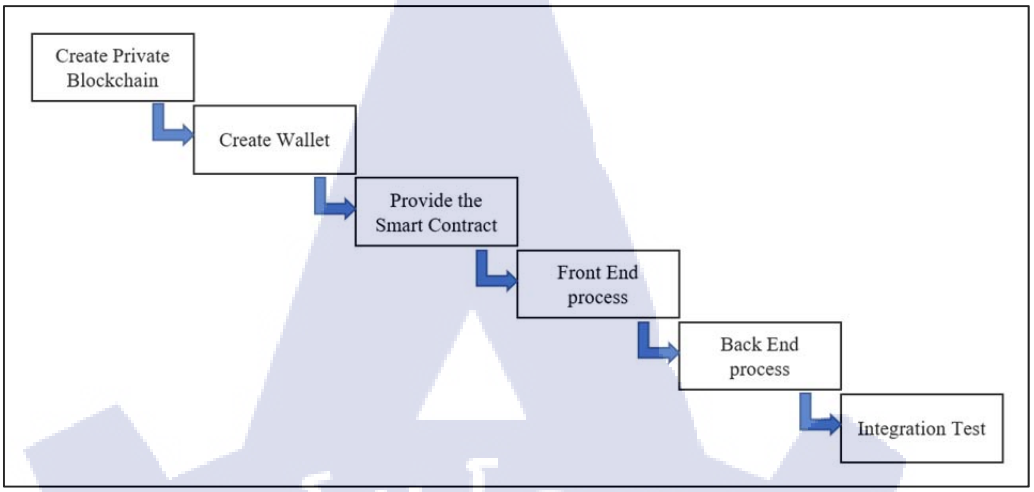

Figure 3.3 Conceptual framework for research

3.5 Development and Processes

Through the development of the system, the smart contract provided the condition of the system first for finding the ways of programming, such as the data which the system would need to get an Invoice ID, the total value of the invoice for finding the way for checking the information then the contract was run by a remixing system for implementing, complying, and deploying the smart contract to run the transaction. For the contract, the author used the modified coding of the smart contract from ERC-20 and ERC-777 to be applied in a small business and selected only the important function of the standard. Nevertheless, the contract was implemented in a secure way to protect the important data in the blockchain and fix the condition of the contract in each piece of information.

Because the smart contract was used for providing the condition of the system, a connection with the wallet was needed for the process of approval by the Metamask wallet, which is a global wallet for Ethereum and easy to use. Moreover, the webpage provided interface services for the user's use on the local host, which was connected with the blockchain and wallet by API. All the development programs are shown in Figure 3.4.

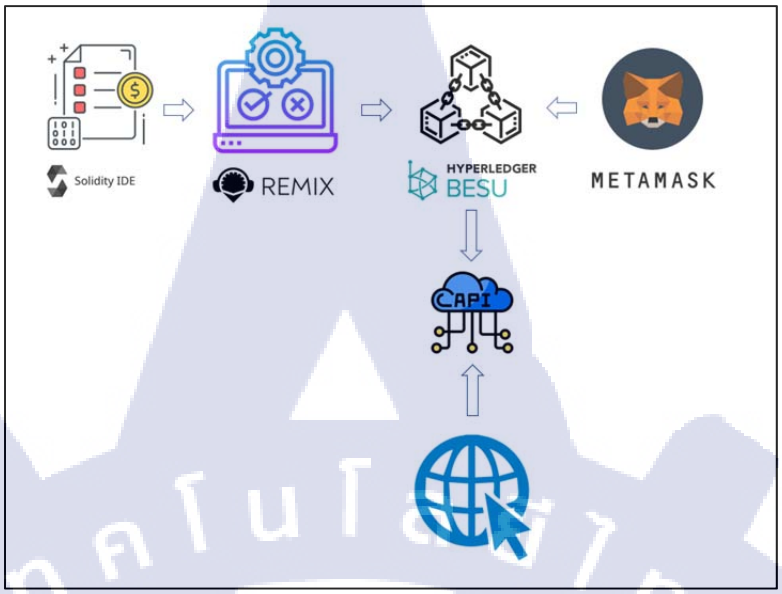

Figure 3.4 Developing the programming of the invoice system

3.5.1 Smart contract

The smart contract of the program was modified from the standard. As such, mostly contracts would need to proceed by the need of the user by getting information from the small company that their need in the invoice system was “To check the invoice number and value was not changed and can recheck all the time”. Then, the contract would need to get two items, which would be the Invoice ID and Value (invoice file) from the user (Figure 3.5).

```
function add(string memory _id, string memory _value) public returns (bool){
    // _id = id invoice ,  _value = data hash
    if ((bytes(_id).length) <= 0 || (bytes(_value).length) <= 0){
        return false;
    }
    else {
        ID.push(_id);
        valueInvoice[_id] = _value;
        hashData[_value] = _id;
        return true;
    }
}
```

Figure 3.5 A smart contract to get the result from the user

After receiving the data needed from the user, the condition of checking the process in the contract was provided for fixing the data type and returning the information for processing the use of the interface in the frontend and the backend work in the next step. This contract used the data type of Boolean to call data from true and false. Moreover, the website provided a simple template and was matched with the user. As a consequence, the contract hashed the data by using “keccak256”, which was one of the SHA-3 algorithms for hashing the generator for collecting and matching the data when the user needed to recheck the information from the blockchain. The coding of checking the information is shown in Figure 3.6 and checking the data is shown in Figure 3.7.

```
}

function checkmatch(string memory _id, string memory _value) external view  returns (bool){
    bool isAns = _checkID(_id);
    // require(bytes(_id).length > 0, "InVoice not null");
    if (isAns == false){
        return false;
    }
    else {
        if (bytes(_value).length != bytes(valueInvoice[_id]).length) {
        return false;
        }

        return keccak256(abi.encodePacked(_value)) == keccak256(abi.encodePacked(valueInvoice[_id]));
    }
}
```

Figure 3.6 Smart contract coding to check the data from the user

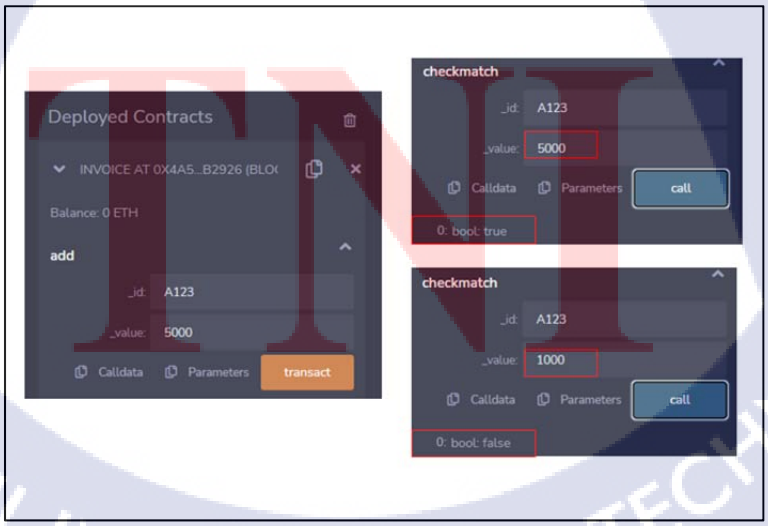

Figure 3.7 Deployed contracts for checking the data

3.5.2 Hyperledger Besu

Hyperledger Besu had to set up at least four nodes to provide the system from the consensus (IBFT2.0 requirement) and for approving the block; thus, the liveness should stay >= ⅔ validators live. Moreover, the requirement of Hyperledger Besu was Install Java JDK 11 – 16 and Hyperledger Besu packaged binaries for running the system. After installing all the requirements, the file of the configuration provided the .json language for configuring the blockchain, chain ID, and other conditions of the blockchain. For the coding of the configuration file, this included [16]:

• chainId is the transection signature as the network ID, which provided data of the chain information and type of blockchain.

• Muirglacierblock is a milestone block.

• Ibft2 provided a specific consensus of the protocol in the blockchain that would use the IBFT2 as POA.

• Blockperiodseconds is the minimum block time that could be set; for example, if we set two for “blockperiodseconds”: 2, the new block would be provided every two seconds.

• Epochlength is the number of blocks, which would reset all the votes.

• Requesttimeoutseconds is the timeout of consensus, which would change the round. Normally, a double number of the blockperiodseconds would be used.

• Nonce: This would be set at 0x0 that refers to containing information about the genesis block (first block).

• Timestamp is creating the date and time of the block.

• extraData is a recursive length prefix (RLP) encoded string (which is space efficient) containing the validator’s addresses of the IBFT 2.0 private network.

• gasLimit is the total gas limit for all transactions. (This blockchain did not use the gas limit for consensus. Thus, free gas used "0x1fffffffffffff".)

• Difficulty is the difficulty to create a new block.

• mixHash is the unique identifier of the block.

• Coinbase is a network coin base account, which is where all block rewards for this network would be paid.

For additional information after getting the condition of the configuration details, the information of the private keys of each node with a balance was set up from this file to verify the validator's information of each node. The coding of the configuration and node structure is shown in Figures 3.8 and 3.9.

Figure 3.8 Coding of the configuration of the blockchain

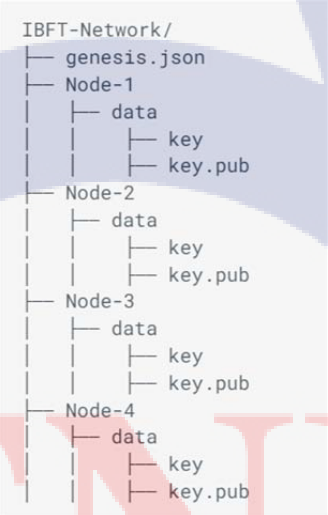

Figure 3.9 Structure of each node in setting up the system

To create the genesis block (start Node1), the following command should be run:

```
besu --data-path=data --genesis-file=..\genesis.json --rpc-http-enabled --rpc-http-api=ETH,NET,IBFT --host-allowlist="*" --rpc-http-cors-origins="all" --rpc-http-port=8545
```

The command line would allow the user to enable: [17]

• Nodes and accounts permission using --permissions-nodes-config-file-enabled and --permissions-accounts-config-file-enabled.

• The JSON-RPC API using --rpc-http-enabled.

• The ADMIN, ETH, NET, PERM, and IBFT APIs using --rpc-http-api.

• All-host access to the HTTP JSON-RPC API using --host-allowlist.

• All-domain access to the node through the HTTP JSON-RPC API using --rpc-http-cors-origins.

Then, find the enode URL display from Node1 (Figure 3.10) as a peer and update the permission to each node. Furthermore, put the enode URL on each node and run the node to peer all the nodes together. If the nodes are peered together and completely, the system would start as shown in Figure 3.10.

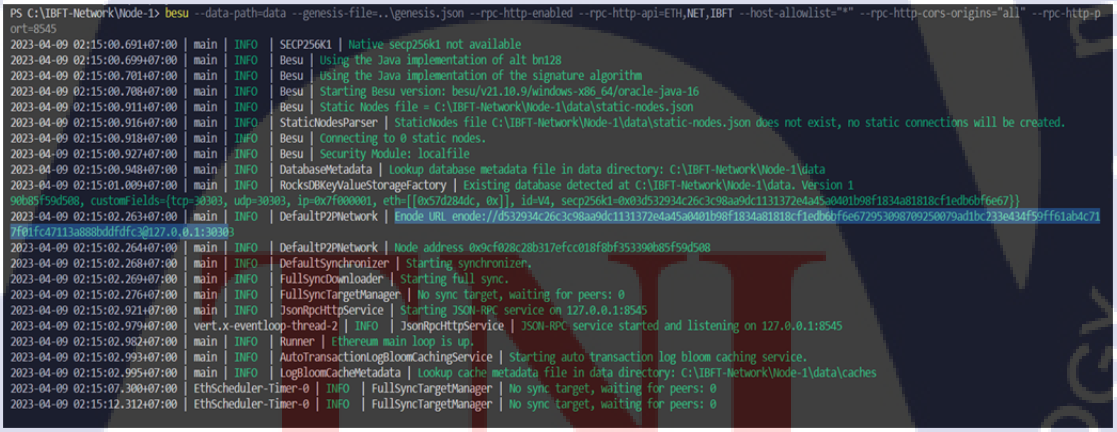

Figure 3.10 Structure of each node in setting up the system

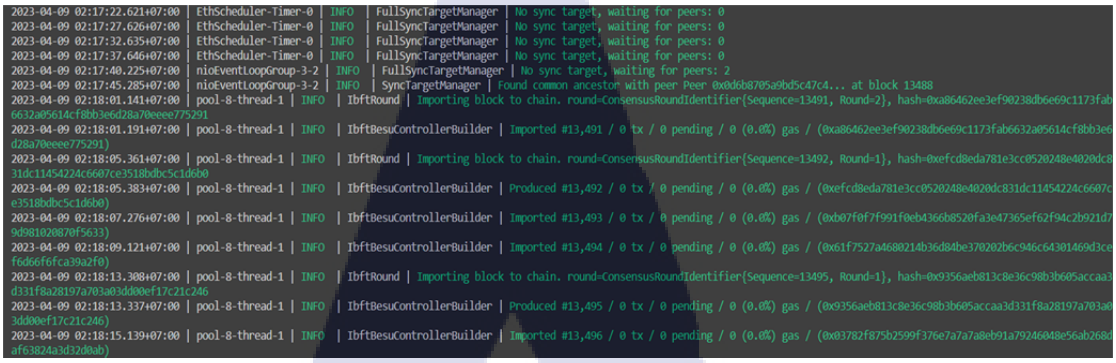

Figure 3.11 All nodes would be linked together and start the blockchain

For nodes 2 - 4, a different command was used for connecting to the same enode but had a different port (Figure 3.12).

```
Node 2
besu --data-path=data --genesis-file=..\genesis.json --bootnodes=enode://
d532934c26c3c98aa9dc1131372e4a45a0401b98f1834a81818cf1edb6bf6e67295309870925007 9ad1bc233e434f59ff61ab4c717f01fc47113a888bddfdfc3@127.0.
0.1:30303 --p2p-port=30304 --rpc-http-enabled --rpc-http-api=ETH,NET,IBFT --host-allowlist="*" --rpc-http-cors-origins="all"
--rpc-http-port=8546

Node 3
besu --data-path=data --genesis-file=..\genesis.json --bootnodes=enode://
d532934c26c3c98aa9dc1131372e4a45a0401b98f1834a81818cf1edb6bf6e67295309870925007 9ad1bc233e434f59ff61ab4c717f01fc47113a888bddfdfc3@127.0.
0.1:30303 --p2p-port=30305 --rpc-http-enabled --rpc-http-api=ETH,NET,IBFT --host-allowlist="*" --rpc-http-cors-origins="all"
--rpc-http-port=8547

Node 4
besu --data-path=data --genesis-file=..\genesis.json --bootnodes=enode://
d532934c26c3c98aa9dc1131372e4a45a0401b98f1834a81818cf1edb6bf6e67295309870925007 9ad1bc233e434f59ff61ab4c717f01fc47113a888bddfdfc3@127.0.
0.1:30303 --p2p-port=30306 --rpc-http-enabled --rpc-http-api=ETH,NET,IBFT --host-allowlist="*" --rpc-http-cors-origins="all"
--rpc-http-port=8548
```

Figure 3.12 Command for nodes 2 - 4

3.5.3 Metamask

To connect the wallet through the blockchain for connecting to the smart contract with the blockchain, the Metamask wallet was used as the gateway to the blockchain apps between the smart contract (on the remix webpage) to the Hyperledger Besu (blockchain).

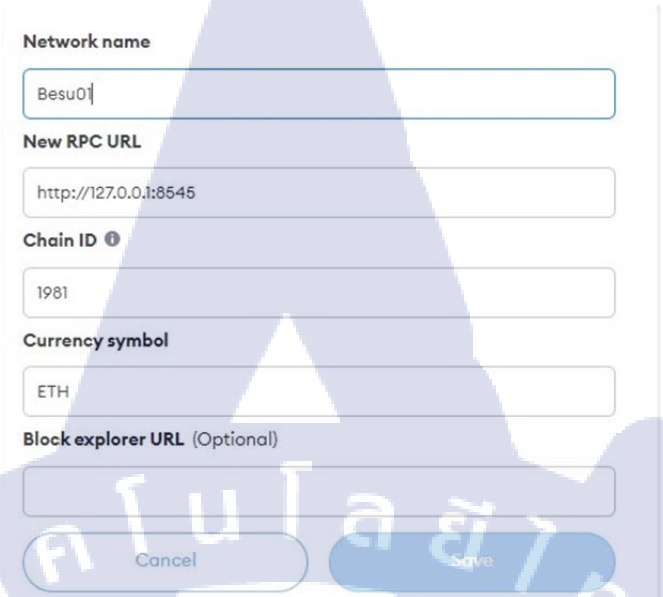

Figure 3.13 Setting up the network of Hyperledger Besu at Metamask

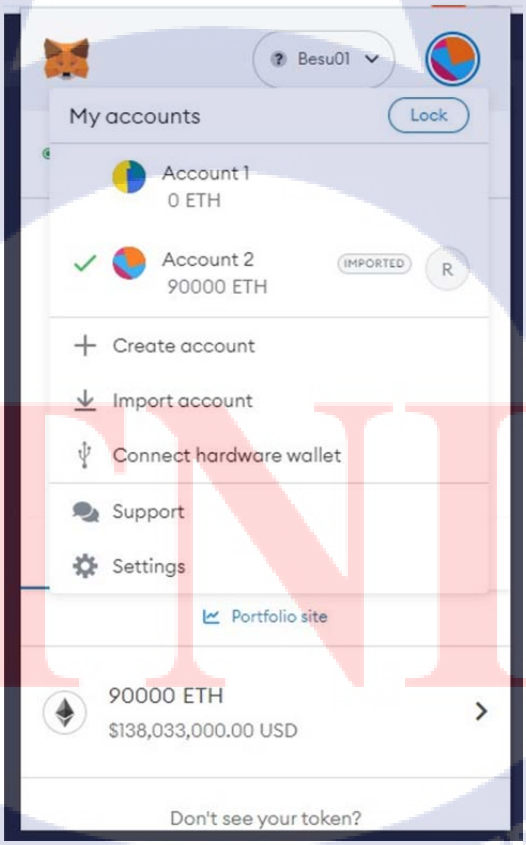

Figure 3.14 Configuration Metamask was done for transection

After setting up the network with Hyperledger Besu in Metamask, the remixed contract was complied and ready to deploy for running the smart contract. Then, the Metamask was connected with the environment on the deploy page (Figure 3.15).

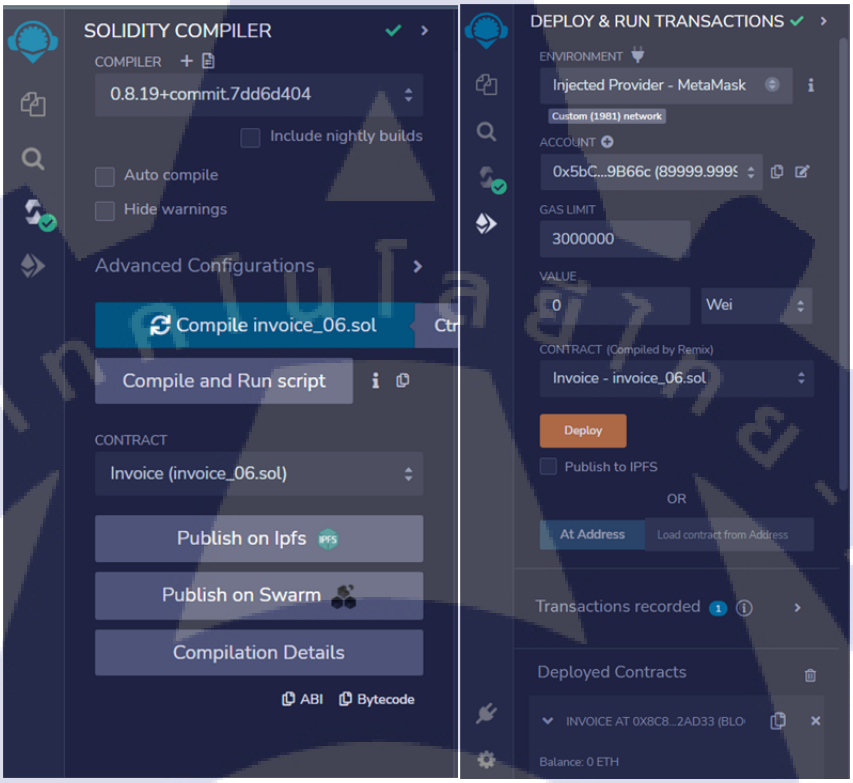

Figure 3.15 Compiling and deploying the smart contract

If the smart contract was deployed, the file would show on the deployed contract to display the transection of the smart contract and could provide the transection for checking the process of the contract or condition. However, all the transections which happened in the contract were sent to the Metamask for approval from the wallet, and all the activities were saved on the wallet.

3.5.4 API

This system was concerned with the user's interface and needed to provide the web page. API was used for connecting between the frontend and backend for communication and providing the service to the user. However, before running the API, the Hyperledger program was recommended to be run with Postman. [18] All the coding from the contract and preparation for the frontend page had to pass and get a result (Figure 3.16).

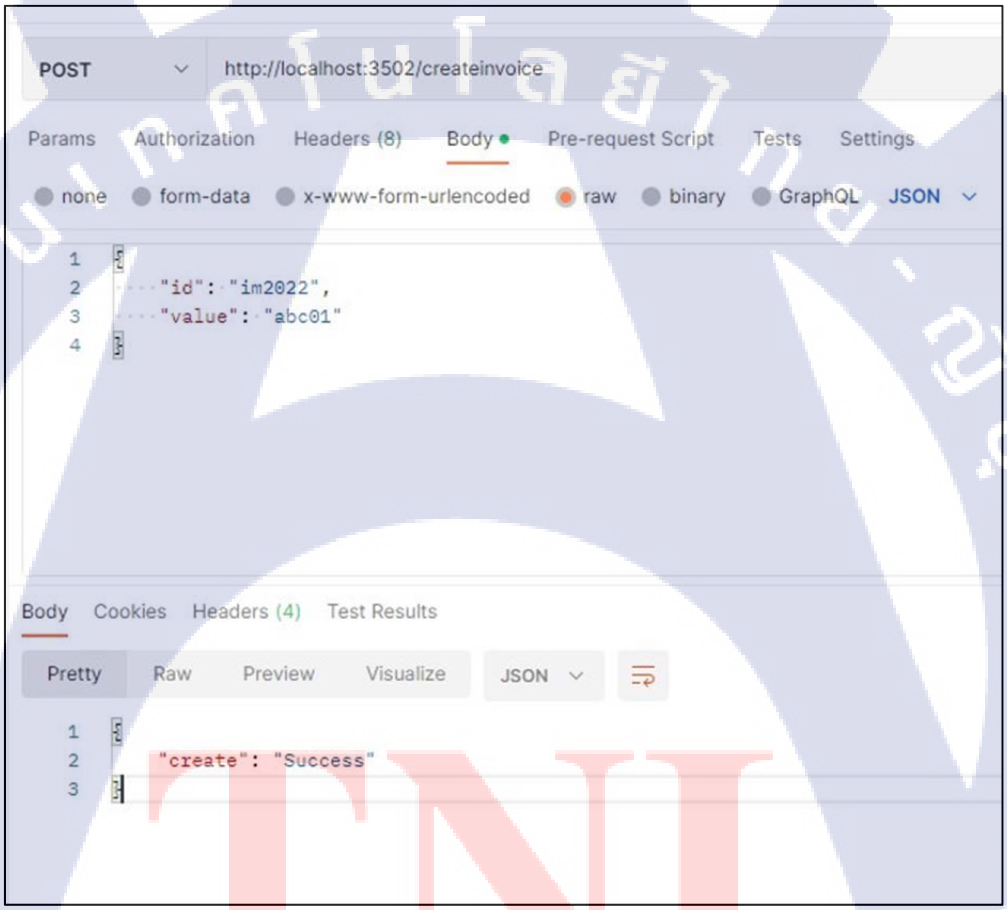

Figure 3.16 Postman for creating an invoice request

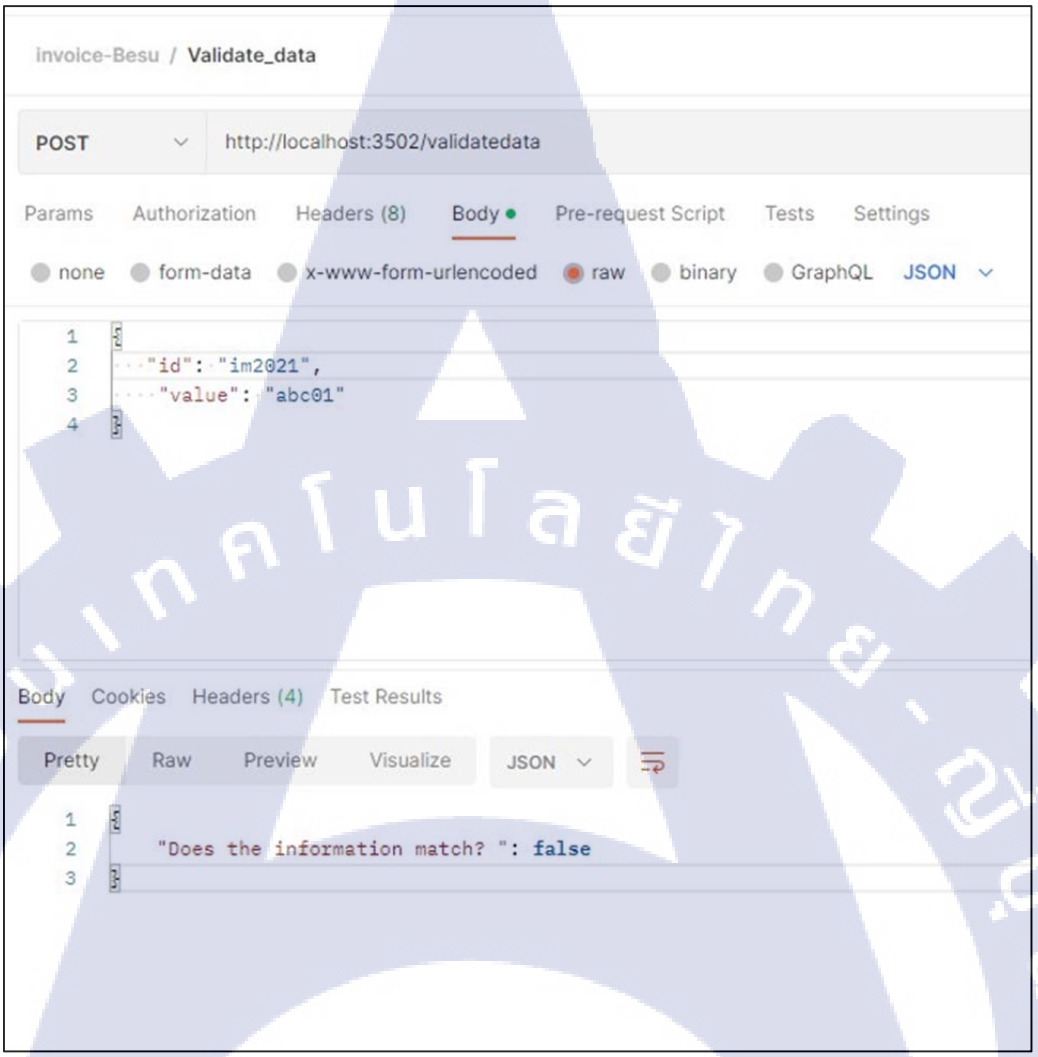

Figure 3.17 Postman for validating a data request

3.6 Production and Results

This stage would be provided in Chapter 4 for the frontend and backend processes and the flowchart of all the pages. This was provided for the frontend to understand more about the function and workflow of this paper done for the invoice management system for a small business.

3.7 Conclusion and Recommendations

This stage would be provided in Chapter 5 for a summary of the total paper and suggest future research to be a benefit for people who would be working on this project or who would apply this research to their work.

Chapter 4
Production and Results

This chapter describes the testing whether the system could be used with a smart contract or not. Additionally, the flow of the program was provided for easy understanding between the frontend and backend. The various webpages comprising the Dashboard page, Create page, Check page, Search page, and Validate page are also included.

Python language was used for the API file, which imported the function as a “hashlip” for hashing the file from a .pdf file or .jpeg file to hashing the data (sha256) then returning the hash password to the user. However, JSON was imported for the call function from the smart contract to the website.

The problem of this program was the core error when the blockchain was provided on the path of the frontend on http://127.0.0.1:5500, but backend used the path http://127.0.0.1:3502. As a consequence,, the program had to use the core middleware for adding the origins of the path to the call function (Figure 4.1).

Figure 4.1 Origin of the coding

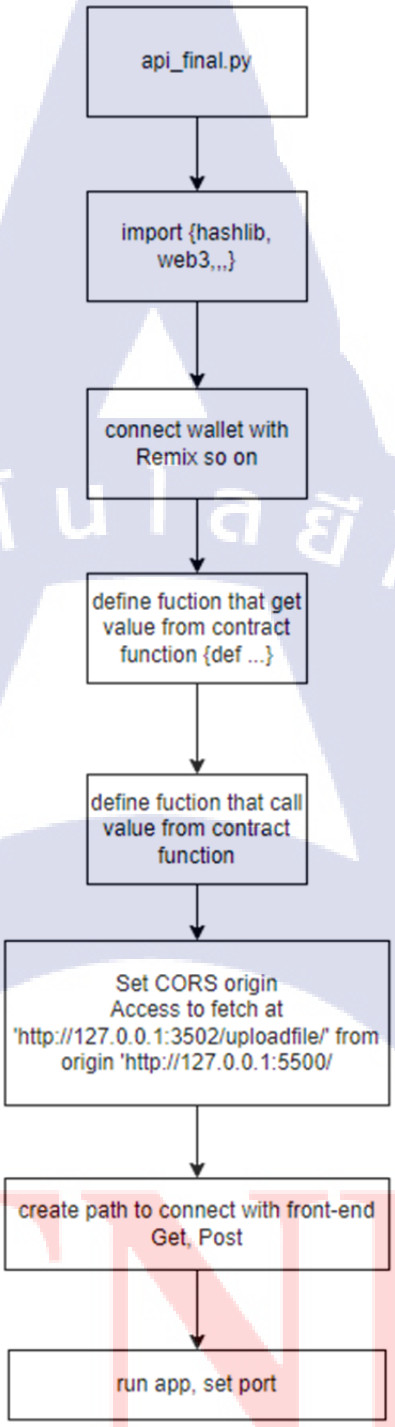

Figure 4.2 Flowchart of the API file

4.1 Dashboard Page

This page showed the total number of invoices in the system by the index page. This was the only page that could use the method to show the total number from the variable “getSizeID” to alert the number of invoices in the blockchain system.

```
def getSizeID():
    _num = contract.functions.getSizeID().call()
    return(_num)
```

```
@app.get("/totalinvoice")
def totalinvoice():
    return {'Total':getSizeID()}
```

Figure 4.3 Coding of the dashboard page

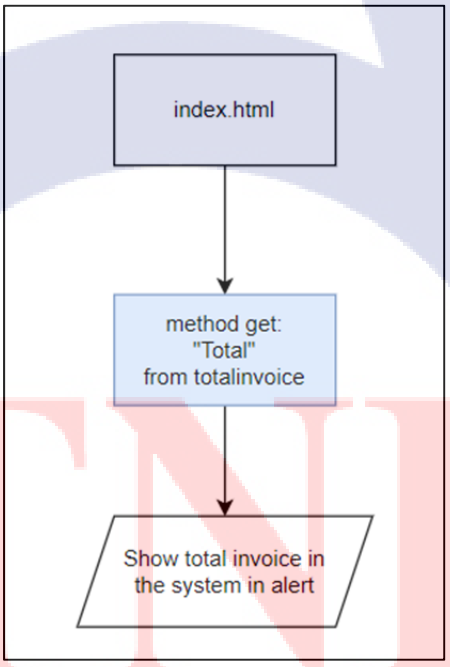

Figure 4.4 Flow of the dashboard page

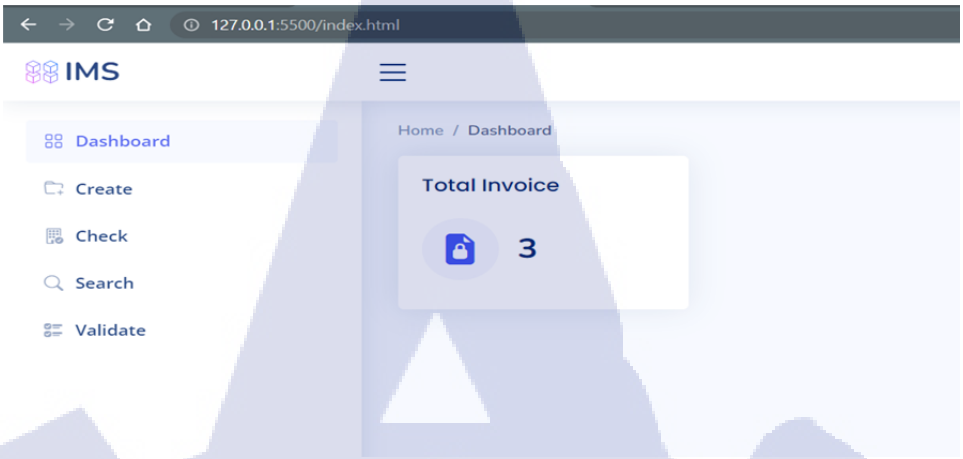

Figure 4.5 Dashboard page from the website

4.2 Creating an Invoice Page

The invoice system created the invoice for the blockchain. The created invoice page was the page for uploading the information of the invoice. This included the Invoice ID and Invoice value, which could not be duplicated or repeated in the system and uploaded the file for hashing in the system to show that the invoice had been verified. Moreover, Web3 was applied on the Api.py file for protecting the data in the blockchain.

```
def add(_id, value):
    store_contact = contract.functions.add(_id, _value).buildTransaction({"chainId":
1981, "from": '0xDb27962ef68be525Dbc3f0983d2Aa00332dCd926', "gasPrice":
web3.eth.gas_price, "nonce":
web3.eth.get_transaction_count('0xDb27962ef68be525Dbc3f0983d2Aa00332dCd926') })
    # Sign the transaction
    sign_store_contact = web3.eth.account.sign_transaction(store_contact,
private_key=pk)
    # Send the transaction
    send_store_contact =
web3.eth.send_raw_transaction(sign_store_contact.rawTransaction)
    transaction_receipt = web3.eth.wait_for_transaction_receipt(send_store_contact)
    return True
```

Figure 4.6 Coding of creating an invoice page 1

Moreover, this program did not use the database for storing the data but used the hash function for hashing the data to SHA256 to be provided as the key of the file to access and for checking if the invoice had been created in the system yet. However, this page would return the value as “Success” in the case the file update was a “Success” or “Fail” when the file had been duplicated in the system (Figures 4.7 – 4.10).

```python
@app.post("/uploadfile/")
async def create_upload_file(ID: str = Form(...),file: UploadFile = File(...)):
    data = file.file.read()
    hash = hashlib.sha256(data)
    _id = ID
    _value = hash.hexdigest()
    canCreate = isCreateInvoice(_id,_value)
    if (canCreate):
        add(_id,_value)
        return {'create':'Success'}
    else:
        return {'create':'Fail'}
```

Figure 4.7 Coding of creating an invoice page 2

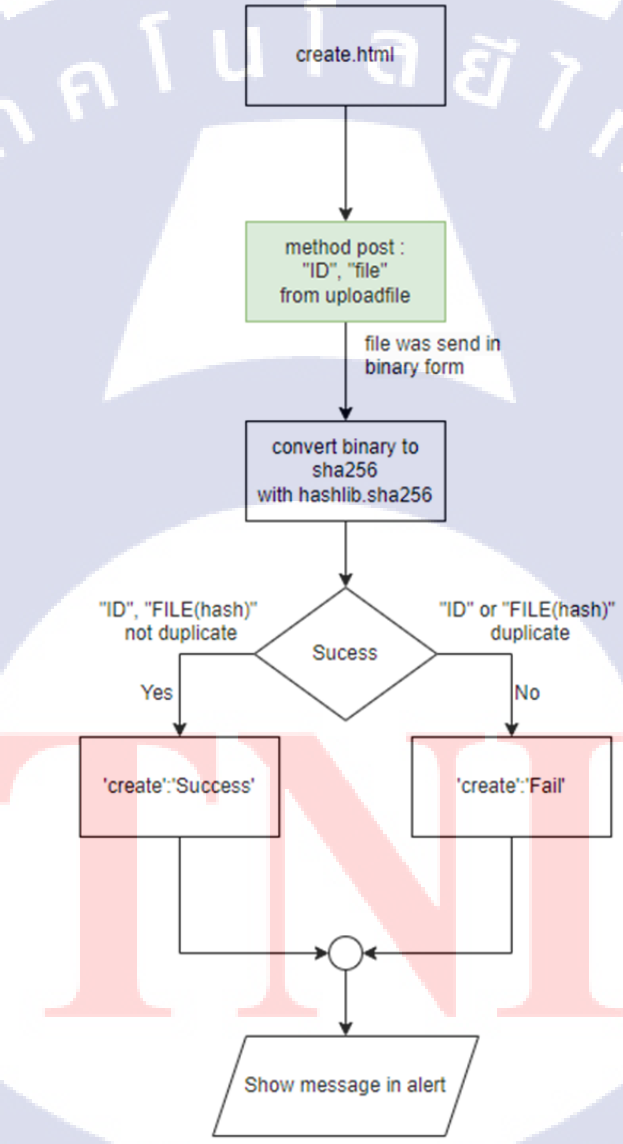

Figure 4.8 Flow of creating an invoice page

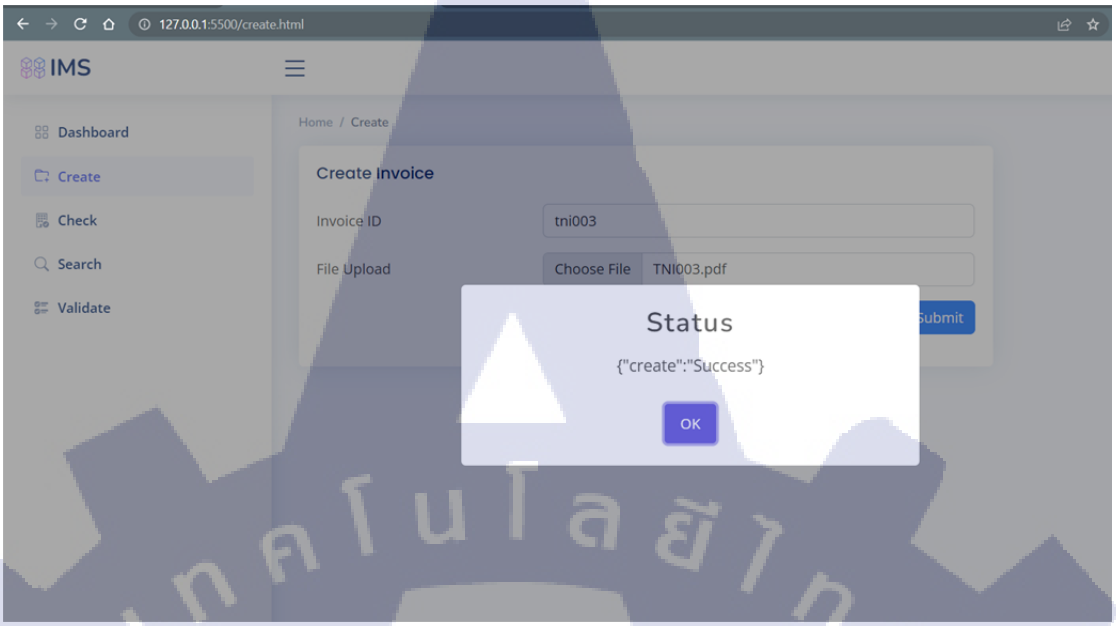

Figure 4.9 Return status: “Success”

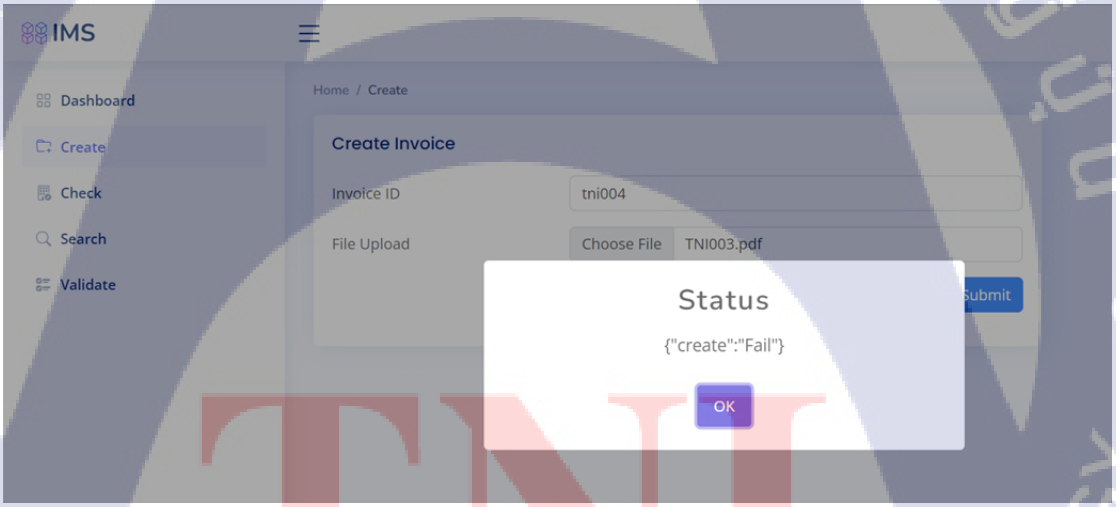

Figure 4.10 Return status: “Fail”

4.3 Checking the Data Page

After creating an invoice page to the system, the data would be rechecked from the user to prevent any duplicated work from the user’s part and check if the invoice had been updated in the system yet. However, the invoice ID would have to provide the full date for finding the information in the system that would match with the invoice. If the

system returned the answer as “true”, the data were in the system, but if not, then the answer would be “false” (Figures 4.11 – 4.12).

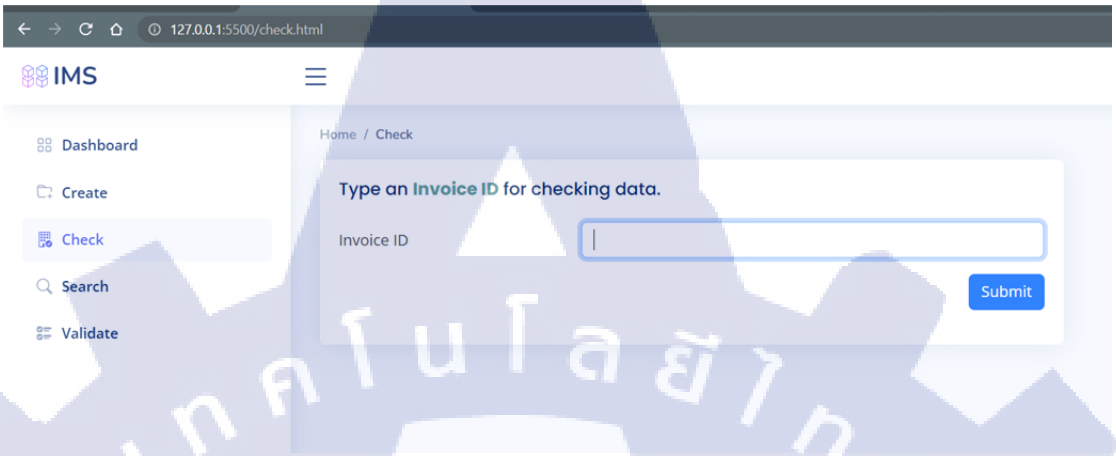

Figure 4.11 Checking the data page

```
def checkIDInvoice(_id):
    isID = contract.functions._checkID(_id).call()
    return(isID)
```

```
@app.post("/checkid")
async def checkID(ID: str = Form(...)):
    _id = ID
    return checkIDInvoice(_id)
```

Figure 4.12 Coding checking the data page

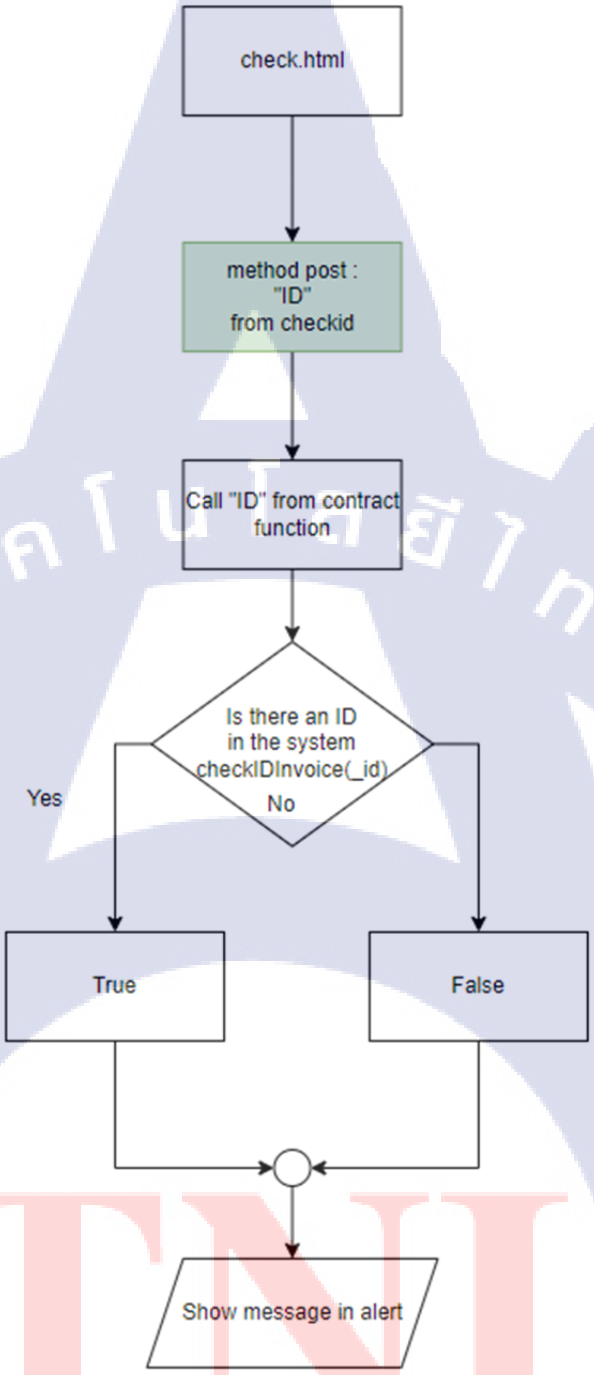

Figure 4.13 Flow of checking the date page

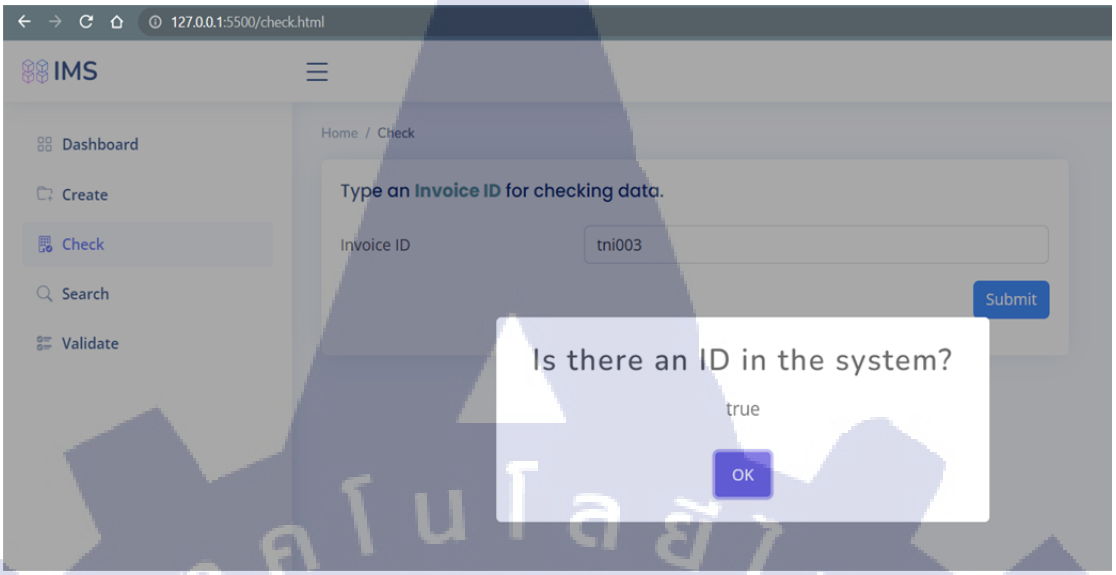

Figure 4.14 Return status: “true”

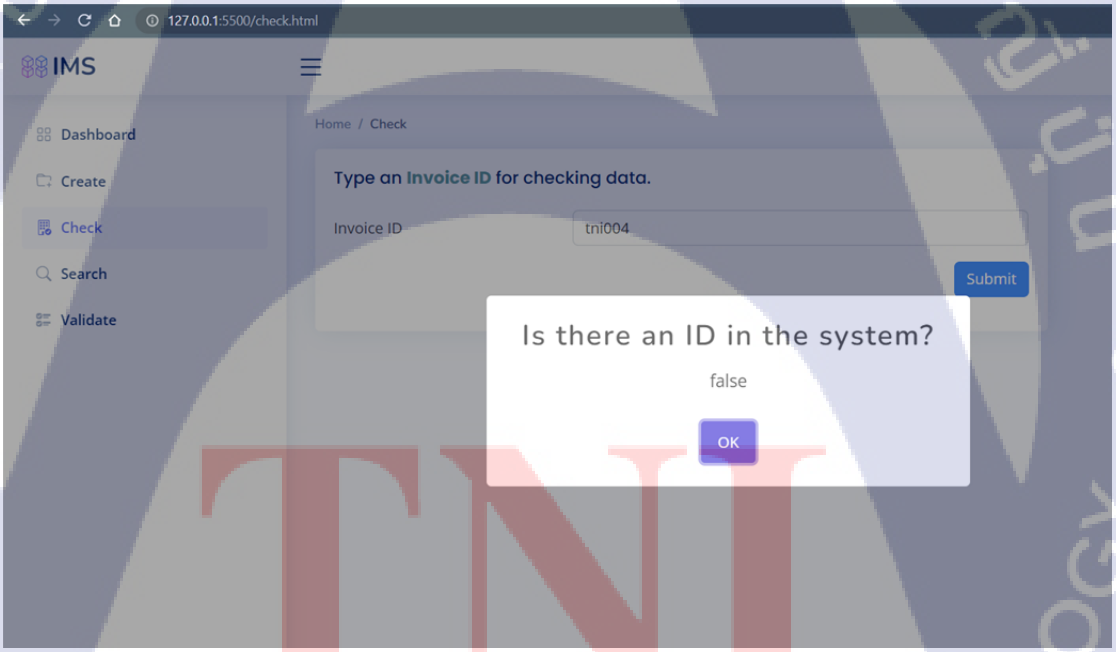

Figure 4.15 Return status: “false”

4.4 Searching the Data Page

From creating the page, this program hashed the file to the hashing key value by the SHA256 function. Therefore, the data was checked from the hashing key because it could be trusted and would not change because it had already received the transaction to the blockchain. This page was provided for getting the hashing key to the user for using in the validate page.

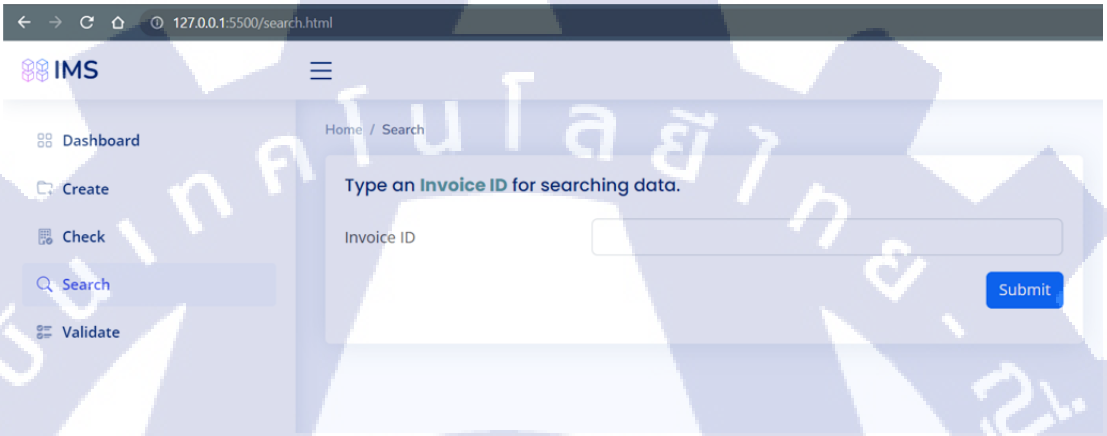

Figure 4.16 Searching the data page

```
def getValue(_id):
    value = contract.functions.getValue(_id).call()
    return(value)
```

```
@app.post("/searchwithid")
async def searchWithID(ID: str = Form(...)):
    _id = ID
    isData = checkIDInvoice(_id)
    if (isData):
        return getValue(_id)
    else :
        return 'INVOICE ID NOT MATCH'
```

Figure 4.17 Coding of searching the data page

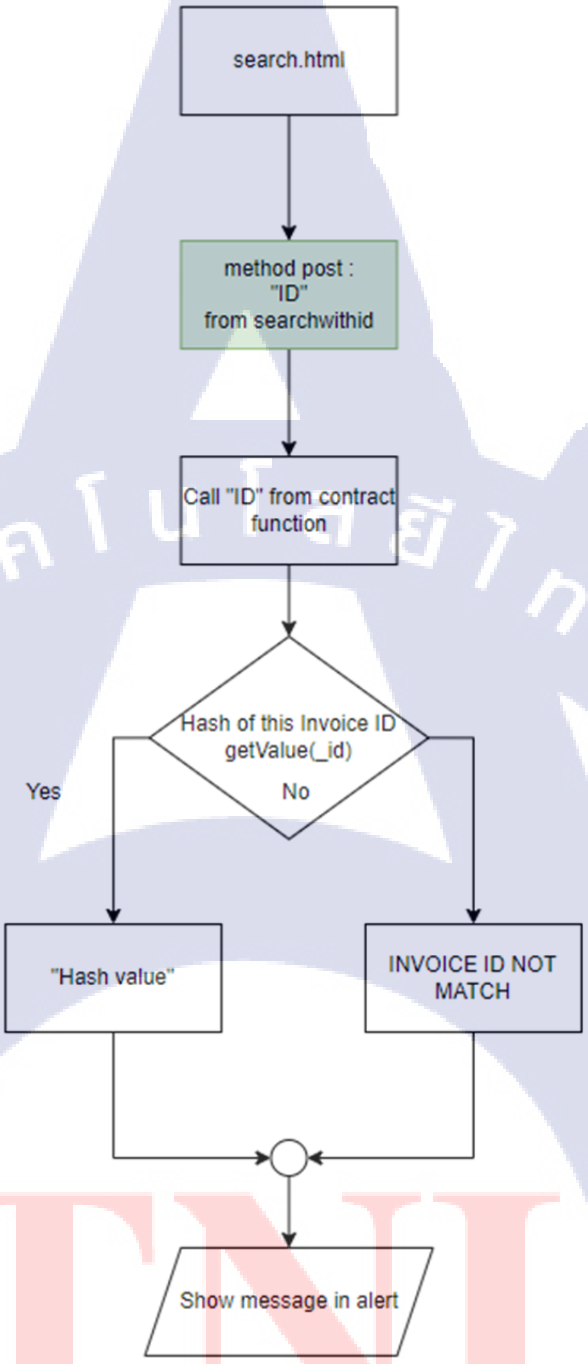

Figure 4.18 Flow of searching the data page

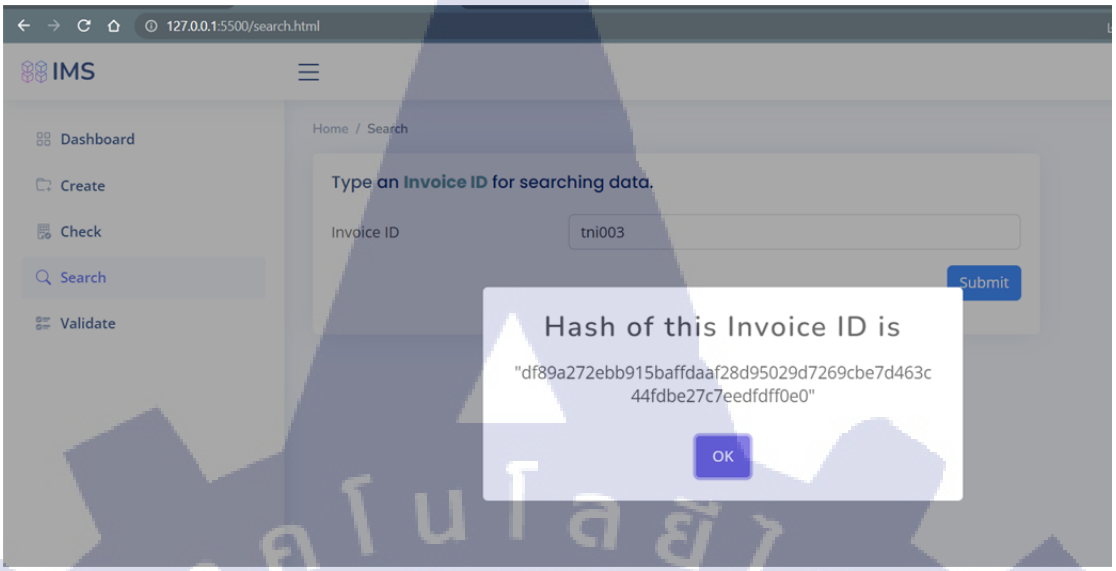

Figure 4.19 Return searching the data page if the invoice was created in the system, then get the hash key

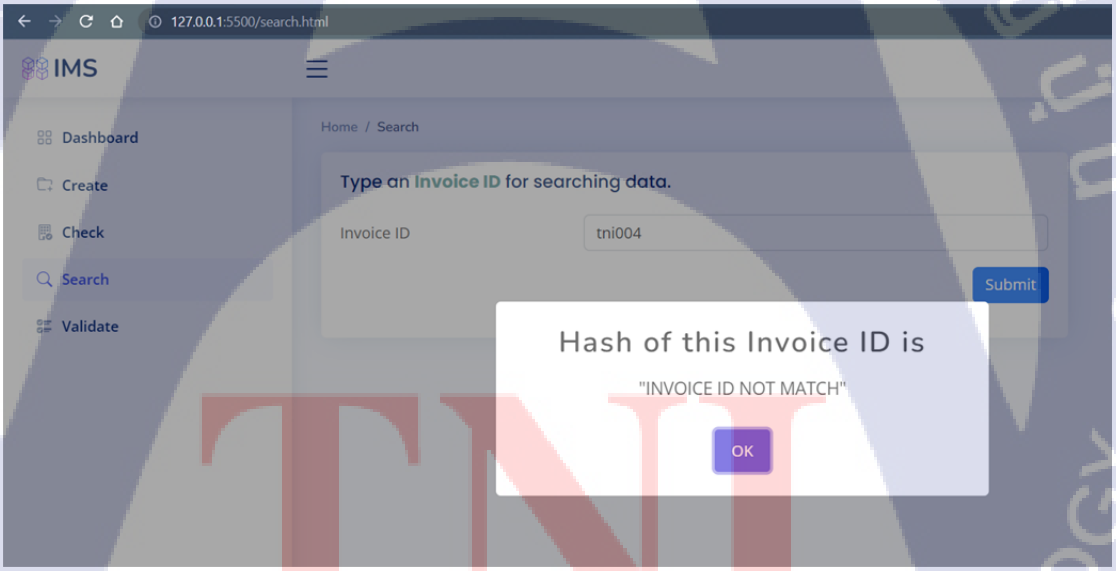

Figure 4.20 Return searching the data page if the invoice was not created in the system, then get “Invoice ID not Match”

4.5 Validating Page

Validating the data was the key of this program. After receiving the hashing key from the search page, the invoice ID had to reconfirm the match with the hashing key. Consequently, this page would return “Match” for the matching ID and Value or “Not Match” in the case the ID and value did not match in the system.

Figure 4.21 Validating page

```
def ValidateData(_id,_value):
    validate = contract.functions.checkmatch(_id,_value).call()
    return(validate)
```

```
@app.post("/validatedata")
async def ValidateData_api(ID: str = Form(...),VALUE: str = Form(...)):
    _id = ID
    _value = VALUE
    ValidateData(_id,_value)
    if (ValidateData(_id,_value)):
        return 'MATCH'
    else:
        return 'NOT MATCH'
```

Figure 4.22 Coding of the validating page

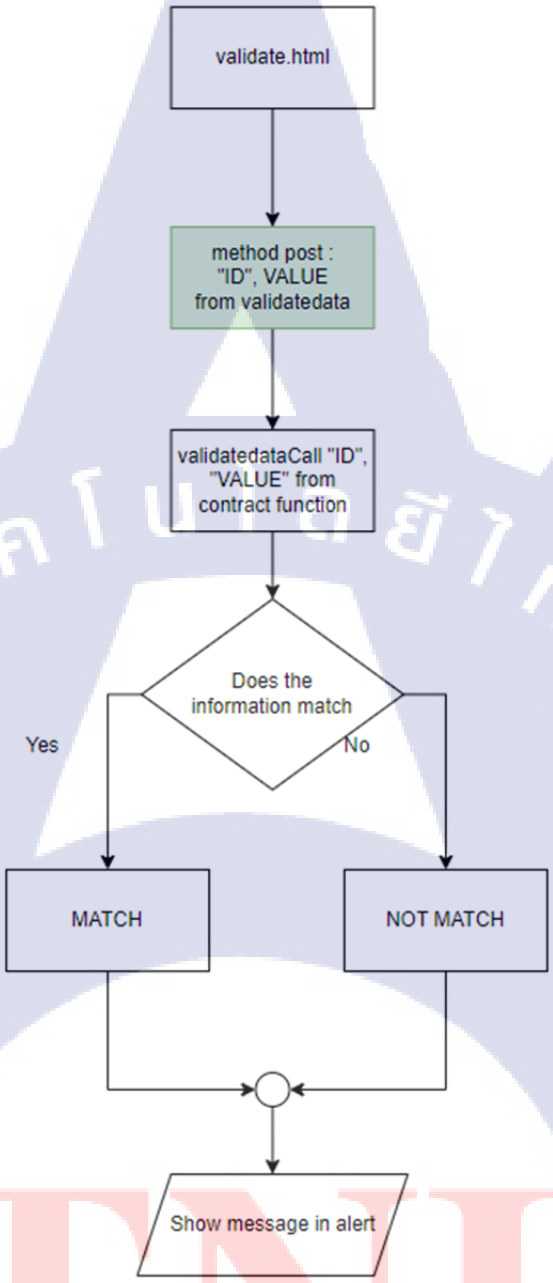

Figure 4.23 Flow of the validating page

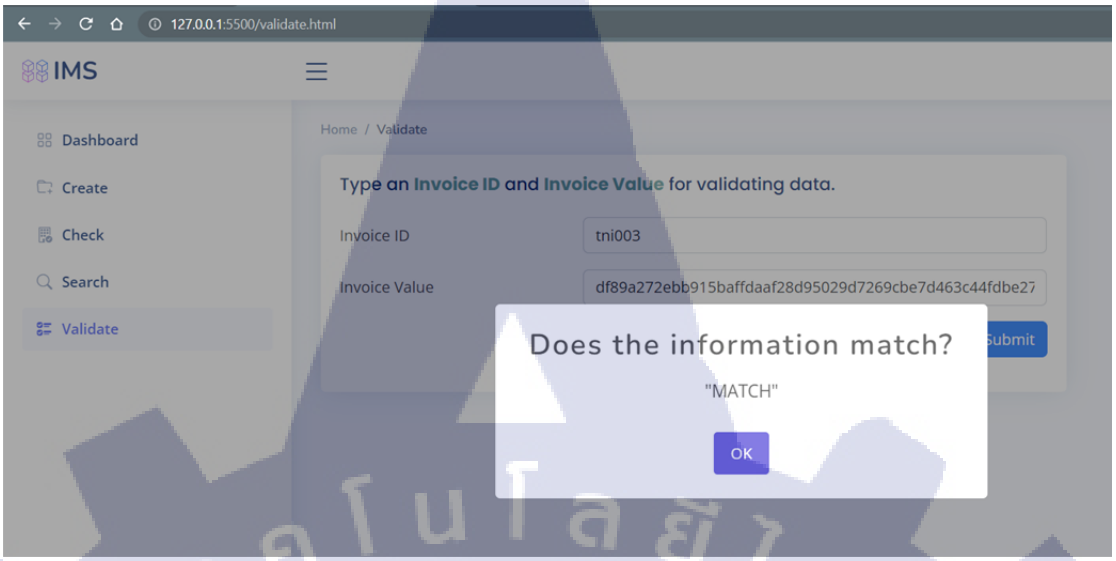

Figure 4.24 Return the validating page if the ID and Value did not match

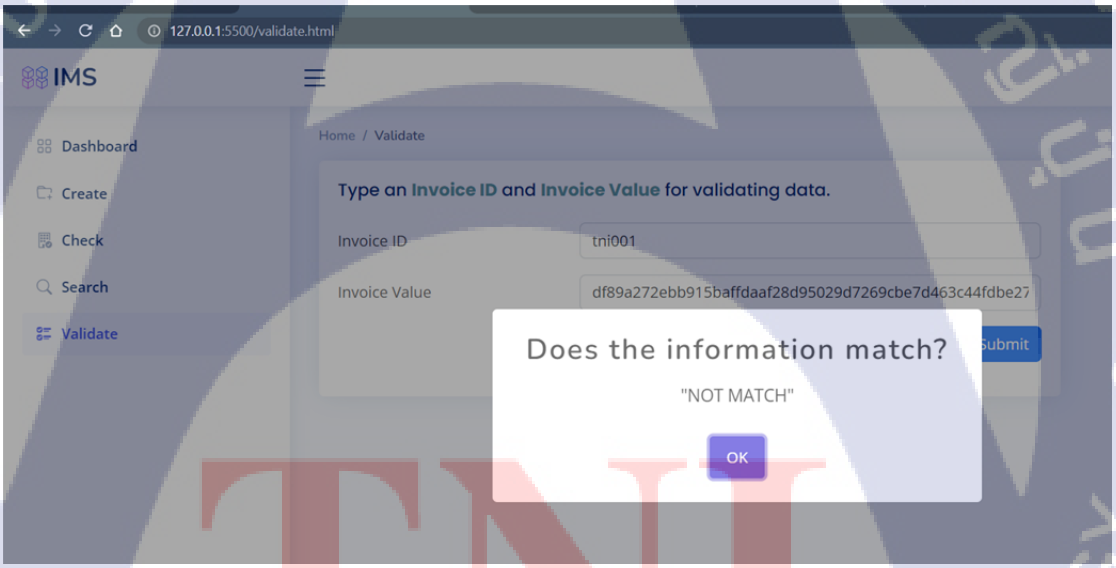

Figure 4.25 Return the validating page if the ID and Value did not match

Chapter 5
Conclusion and Future Research

This chapter summarizes all the results of this research of an invoice management system for a small business which would be provided by blockchain technology. This would be one option for a small business to find an opportunity to use the high security of data. A plan for future research is also provided.

To offer a blockchain-based invoicing management solution for usage in a private network and restrict authorization to the relevant parties.

Referring to the user, the system would require four nodes to build up the chain, which would require two-thirds of the validators to approve the blockchain. As a result, the user could offer authorization for validating the data before approving the block. The required system could then achieve the purpose of the desired investigation.

To determine if a system is suitable for usage by small business firms.

By using Hyperledger Besu's technology, basic users may have SSD storage, which is common in small businesses. The frontend application, on the other hand, would use the local web browser to upload the data. As a result, the software and the standard requirements were produced for usage in small businesses.

To deliver the system’s counterfeit invoice result.

By comparing the hash number and system, the system was able to supply the hash function for counterfeiting the transformed data.

In conclusion, an invoice management system would be a popular application for usage in businesses since an invoice is a financial statement document that is used for financial auditing. As a result, the system should be transparent and simple to use. The author recommends the use of high security for small businesses in order to have a high degree of protection that they could operate on the company’s computer to save

resources. Furthermore, Hyperledger Besu is a blockchain technology that is enterprise-friendly and makes use of the power of a little computer.

In the future, the study should be extended to medium to large businesses, and the concept could be employed to provide an invoicing management system. This should be shared from a peer-to-peer network to a different chain ID or another computer in another branch of the organization to determine the performance of blockchain technology in a public network. Web3 technology is still in development, and there have been several updates for security and user convenience. However, future research might have the use of the full Web3 developed invoice management application with other financial functions, or provide AI technology to identify the words to summarize the information and go through the system.

References

References

[1] The Economic Times, *"What is 'Business,"* [Online]. Available: https://economictimes.indiatimes.com/definition/business. [Accessed: September 4, 2022].

[2] Findex Group Limited, *"Cybercrime on The Rise with Invoice Hacking,"* [Online]. Available: https://www.findex.com.au/insights/article/cybercrime-on-the-rise-with-invoice-hacking. [Accessed: September 4, 2022].

[3] Australian Cyber Security Centre, *"Rajiv's Story,"* [Online]. Available: https://www.cyber.gov.au/acsc/view-all-content/guidance/tax-scam-stories/rajivs-story. [Accessed: September 4, 2022].

[4] IBM, *"What is Blockchain Technology?,"* [Online]. Available: https://www. ibm.com/topics/what-is-blockchain. [Accessed: September 4, 2022].

[5] S. J. Bigelow, *"Blockchain: An Immutable Ledger to Replace the Database,"* [Online]. Available: https://www.techtarget.com/searchitoperations/tip/ Blockchain-An- immutable- ledger-to- replace-the- database. [Accessed: September 5, 2022].

[6] M. Ahmed et al., "Securing medical forensic system using hyperledger based private blockchain," *Proceedings of the 23rd International Conference on Computer and Information Technology, ICCIT,* Dhaka, Bangladesh, December 19-21, 2020, no page.

[7] S. Nakamoto, *"Bitcoin: A Peer-to-Peer Electronic Cash System,"* [Online]. Available: https://bitcoin.org/bitcoin.pdf. [Accessed: August 20, 2022].

[8] D Yaga et al., "Blockchain Technology Overview," *Proceedings by the National Institute of Standards and Technology Internal Report 8202, NISTIR 8202,* New Jersey, USA, October 6-9, 2018, pp. 2-4.

[9] A. M. Antonopoulos and G. Wood, *Mastering Ethereum,* USA : O'Reilly Media, Inc., 2019.

[10] IBM, *"Smart Contracts Defined,"* [Online]. Available: https://www.ibm.com/th-en/topics/smart-contracts. [Accessed: October 1, 2022].

[11] S. Kam et al., "Secure hashing algorithms and their comparison," *Proceedings of the 6th International Conference on Computing for Sustainable Global Development, INDIACom,* New Delhi, India, March 13-15, 2019, pp. 788-792.

[12] Ethereum, *"Introduction to Web3,"* [Online]. Available: https://ethereum.org/en/web3/. [Accessed: April 6, 2023].

[13] Hyperledger Besu, *"Hyperledger Besu for Private Networks,"* [Online]. Available: https://besu.hyperledger.org/en/stable/private-networks/. [Accessed: September 30, 2022].

[14] J Yang et al., "One method for implementing privacy protection of electronic invoices based on blockchain," *Proceedings of the IEEE International Conference on Power Electronics Computer Applications, ICPECA*, Shenyang, China, January 22-24, 2021, pp. 99-104.

[15] C. Fan et al., "Performance analysis of hyperledger besu in private blockchain," *Proceedings of the 4th IEEE International Conference on Decentralized Applications and Infrastructures, DAPPS*, New Jersey, USA, August 15-18, 2022, pp. 64-73.

[16] T. Heg, *"Hyperledger Besu: How to Create an Ethereum Genesis File,"* [Online]. Available: https://consensys.net/blog/quorum/hyperledger-besu-how-to-create-an-ethereum-genesis-file/. [Accessed: February 15, 2023].

[17] Hyperledger Besu, *"Create a Permissioned Network,"* [Online]. Available: https://besu.hyperledger.org/en/stable/private-networks/tutorials/permissioning/. [Accessed: March 24, 2023].

[18] Hyperledger Besu, *"Run with Postman,"* [Online]. Available: https://besu.hyperledger.org/en/stable/private-networks/tutorials/quickstart/?h=postman#run-with-postman. [Accessed: March 24, 2023].

Appendices

Appendix A.

Command code for start node (.text)

Node 1:

besu --data-path=data --genesis-file=..\genesis.json --rpc-http-enabled --rpc-http-api=ETH,NET,IBFT --host-allowlist="*" --rpc-http-cors-origins="all" --rpc-http-port=8545

Node 2:

besu --data-path=data --genesis-file=..\genesis.json --bootnodes=enode://d532934c26c3c98aa9dc1131372e4a45a0401b98f1834a81818cf1edb6bf6e672953098709250079ad1bc233e434f59ff61ab4c717f01fc47113a888bddfdfc3@127.0.0.1:30303 --p2p-port=30304 --rpc-http-enabled --rpc-http-api=ETH,NET,IBFT --host-allowlist="*" --rpc-http-cors-origins="all" --rpc-http-port=8546

Node 3:

besu --data-path=data --genesis-file=..\genesis.json --bootnodes=enode://d532934c26c3c98aa9dc1131372e4a45a0401b98f1834a81818cf1edb6bf6e672953098709250079ad1bc233e434f59ff61ab4c717f01fc47113a888bddfdfc3@127.0.0.1:30303 --p2p-port=30305 --rpc-http-enabled --rpc-http-api=ETH,NET,IBFT --host-allowlist="*" --rpc-http-cors-origins="all" --rpc-http-port=8547

Node 4:

besu --data-path=data --genesis-file=..\genesis.json --bootnodes=enode://d532934c26c3c98aa9dc1131372e4a45a0401b98f1834a81818cf1edb6bf6e672953098709250079ad1bc233e434f59ff61ab4c717f01fc47113a888bddfdfc3@127.0.0.1:30303 --p2p-port=30306 --rpc-http-enabled --rpc-http-api=ETH,NET,IBFT --host-allowlist="*" --rpc-http-cors-origins="all" --rpc-http-port=8548

Appendix B.

Source Code for Create the Genesis Block (.json)

MILP Code

```
{
   "genesis": {
    "config": {
      "chainId": 1981,
      "muirglacierblock": 0,
      "ibft2": {
        "blockperiodseconds": 2,
        "epochlength": 30000,
        "requesttimeoutseconds": 4
      }
     },
     "nonce": "0x0",
     "timestamp": "0x58ee40ba",
     "extraData":
"0xf83ea00000000000000000000000000000000000000000000000000000000000000000d594c2ab482b506de561668e07f04547232a72897daf808400000000c0",
     "gasLimit": "0x1fffffffffffff",
     "difficulty": "0x1",
     "mixHash":
"0x63746963616c2062797a616e74696e65206661756c7420746f6c6572616e6365",
     "coinbase": "0x0000000000000000000000000000000000000000",
     "alloc": {
       "627306090abaB3A6e1400e9345bC60c78a8BEf57": {
        "privateKey":
"c87509a1c067bbde78beb793e6fa76530b6382a4c0241e5e4a9ec0a0f44dc0d3",
        "comment": "private key and this comment are ignored.  In a real chain, the private key should NOT be stored",
        "balance": "900000000000000000000000"
       },
```

```
        "f17f52151EbEF6C7334FAD080c5704D77216b732": {
          "privateKey":
"ae6ae8e5ccbfb04590405997ee2d52d2b330726137b875053c36d94e974d162f",
          "comment": "private key and this comment are ignored.  In a real chain, the
private key should NOT be stored",
          "balance": "90000000000000000000000"
        },
        "5bCCe4e7206d85C6B8727562b58BaBE298e9B66c": {
          "privateKey":
"0334910fc3f9450d5506a6572415d902e8872b93b7aa620892aab698e62b9289",
          "comment": "private key and this comment are ignored.  In a real chain, the
private key should NOT be stored",
          "balance": "90000000000000000000000"
        },
        "Db27962ef68be525Dbc3f0983d2Aa00332dCd926": {
          "privateKey":
"052392f7ff63575c99b2e17a547fc6e35974b5d19dc71b17b260d860bbb120b5",
          "comment": "private key and this comment are ignored.  In a real chain, the
private key should NOT be stored",
          "balance": "90000000000000000000000"
        }
       }
    },
    "blockchain": {
      "nodes": {
        "generate": true,
        "count": 4
      }
    }
  }
```

Appendix C.

Source Code of Smart Contract (.sol)

```
// SPDX-License-Identifier: MIT
pragma solidity  ^0.8.0;

contract Invoice {
    string [] private  ID; // id invoice
    mapping (string=> string) private valueInvoice; // id invoice = data hash
    mapping (string=> string) private hashData; // data hash = id invoice

    function add(string memory _id, string memory _value) public returns (bool){
        // _id = id invoice ,  _value = data hash
        if ((bytes(_id).length) <= 0 || (bytes(_value).length) <= 0){
            return false;
        }
        else {
            ID.push(_id);
            valueInvoice[_id] = _value;
            hashData[_value] = _id;
            return true;
        }
    }

    function getSizeID() external view returns (uint){
        return ID.length;
    }

    function getValue(string memory _id) public view returns (string memory){
        return valueInvoice[_id];
    }

    function _checkID(string memory _id) public   view   returns (bool){
        if(bytes(valueInvoice[_id]).length <= 0){
```

```
            return false;
        }
        else{
            return true;
        }
    }

    function _checkValue(string memory _value) public    view   returns (bool){
        if(bytes(hashData[_value]).length <= 0){
            return false;
        }
        else{
            return true;
        }
    }

    function isCreateInvoice(string memory _id,string memory _value) external
view  returns (bool){
        bool isID = _checkID(_id);
        bool isValue = _checkValue(_value);

        if (isID || isValue){
            return false;
        }
        else
            return true;
    }

    function checkmatch(string memory _id, string memory _value) external view
returns (bool){
        bool isAns = _checkID(_id);
```

```
        // require(bytes(_id).length > 0, "InVoice not null");
        if (isAns == false){
            return false;
        }
        else {
            if (bytes(_value).length != bytes(valueInvoice[_id]).length) {
            return false;
            }

            return keccak256(abi.encodePacked(_value)) ==
keccak256(abi.encodePacked(valueInvoice[_id]));
        }
    }

    function getID(string memory _value) external view returns(string memory){
        bool isAns = _checkValue(_value);
        if(isAns == false){
            return 'null';
        }
        else{
            return hashData[_value];
        }
    }
}
```

Appendix D.

Source Code for API (.py)

```
# pip install python-multipart
import hashlib
from ctypes import addressof
import json
from traceback import print_tb
from web3 import Web3
from web3.middleware import geth_poa_middleware

from fastapi import FastAPI, File, UploadFile, Form
import uvicorn
from fastapi.params import Body
from fastapi.middleware.cors import CORSMiddleware

pk=
'0x052392f7ff63575c99b2e17a547fc6e35974b5d19dc71b17b260d860bbb120b5'
ganache_url = "http://127.0.0.1:8545"
web3 = Web3(Web3.HTTPProvider(ganache_url))
web3.middleware_onion.inject(geth_poa_middleware, layer=0)
abiContract =
json.loads('[{"inputs":[{"internalType":"string","name":"_id","type":"string"},{"inte
rnalType":"string","name":"_value","type":"string"}],"name":"add","outputs":[{"inte
rnalType":"bool","name":"","type":"bool"}],"stateMutability":"nonpayable","type":"
function"},{"inputs":[{"internalType":"string","name":"_id","type":"string"}],"name
":"_checkID","outputs":[{"internalType":"bool","name":"","type":"bool"}],"stateMu
tability":"view","type":"function"},{"inputs":[{"internalType":"string","name":"_val
ue","type":"string"}],"name":"_checkValue","outputs":[{"internalType":"bool","nam
e":"","type":"bool"}],"stateMutability":"view","type":"function"},{"inputs":[{"intern
alType":"string","name":"_id","type":"string"},{"internalType":"string","name":"_v
alue","type":"string"}],"name":"checkmatch","outputs":[{"internalType":"bool","na
me":"","type":"bool"}],"stateMutability":"view","type":"function"},{"inputs":[{"inte
rnalType":"string","name":"_value","type":"string"}],"name":"getID","outputs":[{"i
```

```
nternalType":"string","name":"","type":"string"}],"stateMutability":"view","type":"f
unction"},{"inputs":[],"name":"getSizeID","outputs":[{"internalType":"uint256","na
me":"","type":"uint256"}],"stateMutability":"view","type":"function"},{"inputs":[{"i
nternalType":"string","name":"_id","type":"string"}],"name":"getValue","outputs":[
{"internalType":"string","name":"","type":"string"}],"stateMutability":"view","type"
:"function"},{"inputs":[{"internalType":"string","name":"_id","type":"string"},{"int
ernalType":"string","name":"_value","type":"string"}],"name":"isCreateInvoice","ou
tputs":[{"internalType":"bool","name":"","type":"bool"}],"stateMutability":"view","t
ype":"function"}]')
addressContract =
web3.toChecksumAddress('0x8C8582e2331B731289ff06BefA2382b836b2AD33') #
FILL ME IN
contract = web3.eth.contract(address=addressContract, abi=abiContract)

def getSizeID():
    _num = contract.functions.getSizeID().call()
    return(_num)

def add(_id,_value):
    store_contact = contract.functions.add(_id, _value).buildTransaction({"chainId":
1981, "from": '0xDb27962ef68be525Dbc3f0983d2Aa00332dCd926', "gasPrice":
web3.eth.gas_price, "nonce":
web3.eth.get_transaction_count('0xDb27962ef68be525Dbc3f0983d2Aa00332dCd92
6') })
    # Sign the transaction
    sign_store_contact = web3.eth.account.sign_transaction(store_contact,
private_key=pk)
    # Send the transaction
    send_store_contact =
web3.eth.send_raw_transaction(sign_store_contact.rawTransaction)
    transaction_receipt = web3.eth.wait_for_transaction_receipt(send_store_contact)
```

```
    return True

def checkIDInvoice(_id):
    isID = contract.functions._checkID(_id).call()
    return(isID)

def checkValueInvoicce(_value):
    isValue = contract.functions._checkValue(_value).call()
    return(isValue)

def ValidateData(_id,_value):
    validate = contract.functions.checkmatch(_id,_value).call()
    return(validate)

def getID(_value):
    id = contract.functions.getID(_value).call()
    return(id)

def getValue(_id):
    value = contract.functions.getValue(_id).call()
    return(value)

def isCreateInvoice(_id,_value):
    isCreate = contract.functions.isCreateInvoice(_id,_value).call()
    return(isCreate)

app = FastAPI()

origins = [
    "http://localhost:5500",
    "http://localhost:3502",
```

```
    "http://127.0.0.1:5500"
]

app.add_middleware(
    CORSMiddleware,
    allow_origins=origins,
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)

@app.get("/")
def read_root():
    return {"Hello": "World"}

@app.get("/totalinvoice")
def totalinvoice():
    return {'Total':getSizeID()}

@app.post("/createinvoice")
async def createInvoice(payload: dict = Body(...)):
    _id = payload['id']
    _value = payload['value']
    canCreate = isCreateInvoice(_id,_value)
    if (canCreate):
        add(_id,_value)
        return {'create':'Success'}
    else:
        return {'create':'Fail'}

@app.post("/uploadfile/")
```

```
async def create_upload_file(ID: str = Form(...),file: UploadFile = File(...)):
  data = file.file.read()
  hash = hashlib.sha256(data)
  _id = ID
  _value = hash.hexdigest()
  canCreate = isCreateInvoice(_id,_value)
  if (canCreate):
      add(_id,_value)
      return {'create':'Success'}
  else:
      return {'create':'Fail'}

@app.post("/searchwithid")
async def searchWithID(ID: str = Form(...)):
   _id = ID
   isData = checkIDInvoice(_id)
   if (isData):
      return getValue(_id)
   else :
      return 'INVOICE ID NOT MATCH'


@app.post("/checkid")
async def checkID(ID: str = Form(...)):
   _id = ID
   return checkIDInvoice(_id)

@app.post("/validatedata")
async def ValidateData_api(ID: str = Form(...),VALUE: str = Form(...)):
   _id = ID
   _value = VALUE
```

```
    ValidateData(_id,_value)
    if (ValidateData(_id,_value)):
        return 'MATCH'
    else:
        return 'NOT MATCH'


# @app.post("/getdatawithid")
# async def getDataWithID(payload: dict = Body(...)):
#     _id = payload['id']
#     return { 'Data Invoice is ':getValue(_id)}

@app.post("/getdidwithdata")
async def getIDWithData(VALUE: str = Form(...)):
    _value = VALUE
    isID = getID(_value)
    if (isID == 'null'):
        return False
    return { 'ID Invoice is ':isID}

if __name__ == "__main__":
#   # uvicorn.run(app, host="0.0.0.0", port=3502)
    uvicorn.run(app, port=3502)
```