



Anh Nguyen

Build a blockchain application using React and Solidity

Metropolia University of Applied Sciences

Bachelor of Engineering

Information Technology

Bachelor's Thesis

6 May 2022

Abstract

Author: Anh Nguyen
Title: Build a blockchain application using React and Solidity
Number of Pages: 54 pages
Date: 6 May 2022

Degree: Bachelor of Engineering
Degree Programme: Information Technology
Professional Major: IoT and Cloud Computing
Supervisors: Kimmo Sauren, Senior Lecturer

The purpose of the thesis was to build a blockchain cryptocurrency application. The system was used to connect and send Ethereum through the blockchain network. Another goal of this project was to study the blockchain technology and other related concepts such as Ethereum, smart contract, and Solidity.

The project was built using React with fundamental aspects such as State and React Hooks. Solidity was used to write the smart contract. All the implementation was done on Ropsten test network which is a blockchain testing environment for developers.

The result was a React cryptocurrency application that allows the user to connect to Ethereum wallet Metamask, send transaction that attached with a Gif to be stored permanently on the internet. Moreover, the user can view the details as well as list of all transactions. The project implementation can be used as a base for testing a project before deploying it on Mainnet the main Ethereum network with extended features in further development.

Keywords: Blockchain, React, Solidity, Smart Contract, Ethereum

Contents

List of Abbreviations

1	Introduction	1
2	Blockchain Technology	2
2.1	History	2
2.2	How it works	3
2.3	Blockchain glossary	4
2.4	Benefits and drawbacks	5
2.4.1	Benefits	5
2.4.2	Drawbacks	6
2.5	Use cases and applications	6
2.6	Types of blockchain	7
3	Ethereum	9
3.1	Terminology	9
3.2	Transactions	10
3.3	Nodes	11
3.4	Decentralized application and concept of Web3	12
3.5	Networks	13
3.5.1	Public networks	13
3.5.2	Private networks	13
3.6	Ethereum 2.0	13
4	Smart contract	15
4.1	How it works	15
4.2	Ethereum smart contract	16
4.3	Smart contract languages	16
4.4	Benefit	17
4.5	Limitation	17
4.6	Solidity	18
5	JavaScript	21
5.1	A brief history	21
5.2	JavaScript frameworks and libraries	22

5.2.1	React	22
5.2.2	Vue.js	23
5.2.3	Node.js	23
5.3	The advantages and disadvantages	24
5.3.1	The advantages of JavaScript	24
5.3.2	The disadvantages of JavaScript	25
6	React	26
6.1	Brief history	26
6.2	Fundamental features of React	27
6.2.1	Virtual DOM	27
6.2.2	JSX	28
6.2.3	Component	28
6.2.4	Props and state	29
6.2.5	React Hooks	30
7	Project Implementation	31
7.1	Initializing and setting up React application	33
7.2	Writing Solidity smart contract	34
7.3	Deploying smart contract	36
7.4	Connecting smart contract to React	37
7.5	Connecting to wallet and sending transactions	41
7.6	Retrieving the list of transactions	43
8	Results	45
9	Conclusion	54
	References	55

List of Abbreviations

DLT:	Distributed ledger technology
PoW:	Proof-of-Work
PoS:	Proof-of-Stake
ETH:	Ether
NFT:	Non-fungible Token
EVM:	Ethereum Virtual Machine
EOA:	Externally owned account
DAPP:	Decentralized application
UI:	User Interface
ABI:	Application Binary Interface
JSON:	JavaScript Object Notation
HTTP:	HyperText Transfer Protocol
SPDX:	Software Package Data Exchange
OOP:	Object oriented programming
HTML:	HyperText Markup Language
CSS:	Cascading Style Sheets
ECMA:	European Computer Manufacturers Association

ES:	ECMAScript
iOS:	iPhone Operating System
NPM:	Node Package Manager
CLI:	Command-line Interface
NPX:	Node Package Execute
JIT:	Just-In-Time
SPA:	Single-page Application
DOM:	Document Object Model
JSX:	JavaScript Syntax Extension
XML:	Extensible Markup Language
URL:	Uniform Resource Locator
API:	Application Programming Interface

1 Introduction

Blockchain has gained significant concern as the technology behind cryptocurrency Bitcoin since its creation in 2008. Blockchain is a decentralized ledger controlled by peer-to-peer network which is resistant to modification and data-tampering. Thus, blockchain has changed the way of processing transaction, handling data and providing other services. Money transfer is one of the most common use cases of blockchain. Using blockchain transaction can be processed automatically without the verification of intermediaries using smart contract; therefore, the speed is improved. Furthermore, transaction stored on blockchain is immutable and shared across all members in the network meaning that data security and transparency are guaranteed. As a result, blockchain is considered as an effective platform for financial services.

The purpose of the project was to build a full stack blockchain cryptocurrency application using React as front-end and Solidity as smart contract programming language. Due to time constraints and new knowledge, the project focused on basic functions of payment application which were transferring and logging transactions. A basic smart contract was also written for keeping track of all transactions; therefore, the data history is searchable as required. The goal of this final project is to acquire the understanding of blockchain and smart contract as blockchain's key element.

The thesis contains nine main parts. Following the Introduction, Chapter 2 introduces and explains how blockchain technology works as the foundation of the project. Secondly, chapter 3 discusses Ethereum concepts in expanding the application of blockchain since it was released. Smart contract with its programming language Solidity are the main contents of chapter 4. Next, chapter 5 covers JavaScript's theories to get acquainted with React, a JavaScript library in chapter 6. Chapter 7 focuses on described how the project was achieved as well as its structure. Finally, the last chapters highlight the main results of the project and a suggestion for further studies.

2 Blockchain Technology

Blockchain is an immutable digital ledger [1] which is updated and shared among several computers also known as “nodes” in the network [2]. Blockchain is a type of distributed ledger technology (DLT) which has no central administrative functions and the data is shared and synchronized across multiple network participants.

“Block” refers to where the data and the state are stored consecutively and “chain” refers to the reality that each block references the block before it and they form a chain of blocks [3]. As a result, blockchains are resistant to modification because changing data in a block requires the change in all subsequent blocks, which represents the consensus of the network [2].

2.1 History

Cryptographer David Chaum, the pioneer in cryptography, introduced a blockchain-like protocol in the dissertation in 1982 [3]. Later in 1991, the first blockchain prototype was reported by Stuart Haber and W. Scott Stornetta [3] in their research project about time-stamping digital documents [4]. In the research they discussed the use of chaining digital documents together to validate the integrity of a document [4] as well as to secure it from data tampering [5]. The project was the inspiration for many cryptographers and computer scientists, which led to the formation of Bitcoin, the first cryptocurrency based on blockchain [5].

The first concept of blockchain was introduced in 2008 when Satoshi Nakamoto released Bitcoin whitepaper. Nakamoto used a Hashcash-like method as a principal component in Bitcoin to timestamp blocks without third-party validation and to control the speed of adding a new block to the chain. In 2009, the first transaction in the world was 10 Bitcoin from Nakamoto to Hal Finney as a reward.

The creation of blockchain-based platform Ethereum in 2013 was considered as second revolution after Bitcoin. Ethereum enhances the potential of blockchain and its use cases by encouraging developer to build and develop in new decentralized platform with new technologies such as smart contract and decentralized application.

2.2 How it works

Record of data in blockchain is resistant to tamper and verified using consensus mechanism to reach the agreement between nodes. Proof-of-work (PoW) and proof-of-stake (PoS) are two types of common consensus mechanisms currently being used.

- PoW: is known as “mining” [3] and done by miners, who solve the puzzle using computer resources to add new block to the network and earn the reward in ETH [2]. However, PoW is consuming a lot of electricity and can only perform limited transaction per seconds which can cause network congestion if exceed [3].
- PoS: is done by validators who have the same responsibility with miners in PoW. Users need to stake ETH to become a validator. In PoS, validator does not use computational resources because they are not mining as the block selection methods are randomized. The chosen validator earns rewards for creating new blocks and verifying other blocks they do not create when they are not chosen. The verification is also called “attesting”. Stake is used for quality assessment since the validator will lose their stake when defrauding the chain and malicious behaviour. [2.] However, PoS is still under development now and Ethereum is planning to upgrade to PoS in 2022 with Ethereum 2.0 project.

Figure 1 explains the process of adding a transaction into the blockchain using PoW.

How does a transaction get into the blockchain?

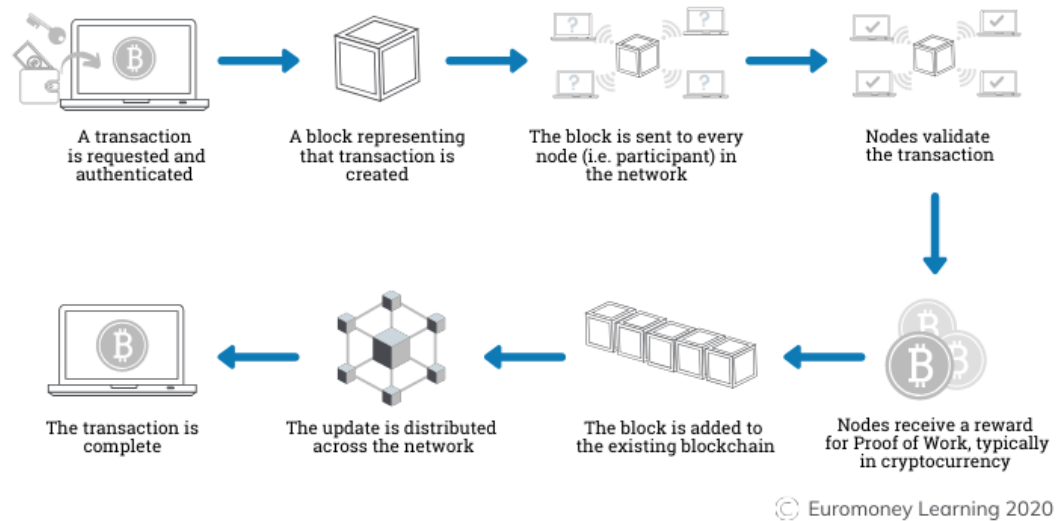


Figure 1. How blockchain works. Copied from Euromoney [6].

When a transaction is requested, it is recorded as a block that representing data. The block is broadcast to the network with other blocks waiting for processing. The nodes take and validate the transactions by solving a difficult puzzle as the result is 64-digit hexadecimal number called hash [3]. The fee for validating is sent the miner as the rewards. Next, the block is added to the blockchain and connected to the previous as well as the next blocks [1]. A chain of blocks is formed to increase the data security and prevent data-tampering.

2.3 Blockchain glossary

This section covers a few basic glossaries that mentioned in each transaction.

- **Address:** is a string of alphanumeric character used to receive and send transaction. Address starts with 0x in Ethereum.

- Hash: is the output of mathematical function that confirms the transaction. Each block includes the hash of previous block and its own value.
- Public and private keys: are created together in each wallet. Public key allows user to receive the funds in the account. Private key acts as a password to prove the ownership of an account. Private key is a 256-bit number and difficult to remember. Therefore, seed phrase is a 12-word phrase which derived from private key that also gives the access to the wallet and be the backup for private key.
- Wallet: a digital wallet that keeps, manages and transfers the cryptocurrencies. The wallet has an address to perform transactions. A wallet can be a cold wallet or a hot wallet. A hot wallet is connected to the internet while the cold wallet is not. [7.]

2.4 Benefits and drawbacks

Like other technology, blockchain has its advantages and disadvantages.

2.4.1 Benefits

Compared to centralized technology, blockchain presents efficiencies due to its security, transparency, automation and trust.

- Security: Data is shared and synchronized among the nodes based on the consensus mechanism. All validated records are immutable and cannot be altered.
- Transparency and trust: All participants can access the same recorded data at the same time regardless of locations. As transactions are time-stamped, the data is trusted and transparent.
- Speed and automation: Transactions can be executed using smart contract when the pre-determined terms are met without the

attendance of intermediary like bank or lawyer to confirm anything. It helps to increase efficiency and speed as well as money-saving. [1.]

2.4.2 Drawbacks

This section studies the top weaknesses of blockchain.

- Limited throughput: there is a restriction on how many transactions per second that blockchain technology can process. Bitcoin's throughput is about seven transactions per second and Ethereum's is 30 transactions per second which is a small number comparing to Visa [3]. Scalability is challenging as it can cause network issues [8].
- Environmental impact: computational resources are used to verify the transactions that consume a lot of electricity which leads to negative environmental sustainability [3].
- Asset loss: cryptographic key is an approach to protect access to a blockchain wallet. If the key and seed phrase are lost, it is impossible to recover and regain access because of decentralized system; therefore, the asset in wallet is lost permanently [8].

2.5 Use cases and applications

Blockchain technology is used in various industries for different purposes, such as supply chain, finance, healthcare, game, government [1], fuel, travel and transportation [3]. Furthermore, blockchain can also be implemented in centralized system for data security and cost reduction [5]. The following are noticeable examples of blockchain's use cases nowadays:

- Cryptocurrency: the most important use of blockchain is digital currency such as Bitcoin and Ethereum. Users can transfer and exchange cryptocurrency as a payment for products and services and the transactions are stored on blockchain [2].

- Smart contract: a program that automatically executed when given conditions are met.
- Banking: with blockchain, sending money and other financial situation can benefit from automatic and quick transaction verification also during non-business hours [8].
- NFTs: NFT shorts for non-fungible tokens, digital token for ownership of digital art works and real assets [3].
- Supply chain: in logistics and supply chain monitoring, blockchain is used to trace the good's origin from harvesting to consumption [8].

2.6 Types of blockchain

There are four different types of blockchain network according to IBM, American technology corporation [1].

- Public blockchain networks: In public blockchain, anyone which internet connection can join to view, send and verify the transactions without providing personal information. However, the drawbacks are computational resources requirement, lack of privacy and security for enterprise [1]. Proof-of-work (PoS) and proof-of-stake (PoW) are two mainly used consensus mechanisms. In addition, the well-known public blockchain networks are Bitcoin and Ethereum. [3.]
- Private blockchain networks: Private blockchain network is controlled and governed by an organization or enterprise for working with private data. In the network, the organization allows only validated participants to verify the transactions and maintain the ledger. [7.]
- Permissioned blockchain networks: Organizations generally set up permissioned blockchain networks after private blockchain for additional security and management. It is used to determine the authorized participants in certain transactions in the network. Like private blockchain, the participant needs to gain the permission or

access to participate in; however, they can only perform permitted tasks. Furthermore, permissioned blockchain network also known as hybrid blockchain has features of both public and private. [1.]

- Consortium blockchains: Consortium blockchains has similar feature with permissioned blockchain; however, the difference is that the responsibilities of managing the blockchain are shared between multiple organizations. This type of blockchain is optimal for collaboration and security. [1.]

3 Ethereum

Ethereum is a blockchain-based platform which is the technology behind Ether (ETH) the second largest cryptocurrency in the market and various decentralized applications [9]. It was first proposed by Vitalik Buterin in 2013 and developed by Ethereum Foundation in 2014.

In Ethereum, Ethereum Virtual Machine (EVM) is a single, standard computer that all participants on the network agree on and have a copy of its state. The agreement is reached using consensus mechanisms when obtaining the consent of at least 51% of the nodes. Currently, Ethereum uses proof-of-work (PoW) mechanism.

In addition, the participant can request EVM to perform arbitrary computation. When the request also called transaction is transmitted, the computation is verified and executed by other participants on Ethereum network. As a result, it causes the change of EVM's state. Both transaction's record and current state are stored on the blockchain. [2.] Transactions are transferred within Ethereum accounts and the sender needs to small fee by Ether as the cost for executing [10].

3.1 Terminology

This section introduces common terminologies in Ethereum that used in the project.

- Smart contract: computer program that executes on Ethereum blockchain when conditions are met.
- Ether (ETH): cryptocurrency of Ethereum ecosystem that supporting a pricing mechanism. Gas is small fee paid in ETH for executing smart contract since broadcasting each transaction requires the use of computational resources. The gas is the reward to the miner who validates the transaction, execute, commit it to the blockchain and

transmit it to the network. The fee prevents malicious from submitting infinite transactions as it runs out of ETH.

- Wei and gwei are main denominations since transaction is small Ethereum. Wei has value of 10^{-18} ETH and is the smallest amount of ether. Gwei or giga-wei, has its value in ETH is 10^{-9} , which is often used to present gas fee.
- Minting is the procedure through which new ether is created by Ethereum protocol when a miner creates new block. The reward gives to miners as the encouragement.
- Account: an object is used to store ether. An account can be externally owned account (EOA) or smart contract. EOA is created and controlled by user with private key. Both types of account can interact with smart contract as well as receive and transfer ether. [2.]
- Wallet: the interface that keeps private keys for Ethereum account interaction.

3.2 Transactions

An Ethereum transaction is initiated by an EOA to change the state of the network. Transferring ether between two accounts is the simplest example about transaction.

Transaction lifecycle is described in the following processes:

- Transaction hash is generated after submitting transaction.
- Transaction is broadcast to the network with other transactions.
- A gas fee is required, a miner takes the transaction and start to verify it. Processing time depends on efficiency of the network and the miner at the time.

- The transaction is added to the block when the number of confirmations is adequate for the network to process and recognize the transaction.

There are three types of transactions:

- Regular: a transaction between two wallets.
- Contract deployment: a transaction without the recipient address, while data field is filled with the execution code during deployment.
- Execution of a contract: a transaction that the beneficiary is smart contract address. [2.]

3.3 Nodes

An application connects to the Ethereum node first in order to interact with the blockchain. The connection allows application to view data from blockchain, send transaction through the network.

Nodes are computers which are running software and performing functions in Ethereum client. A client is the implementation that dictates the way network and blockchain functions by validating transactions in each block. Nodes basically communicate with other nodes to store all the states of the blockchain and update it when the consensus is achieved. The collections of nodes and their broadcasting construct the Ethereum network.

Access to a node is the requirement when deploying smart contract. It can be obtained by running own node, using node services like Alchemy or connecting to a public node like Infura. [2.]

There are three different types of nodes that consume data in different ways on the network.

- Full node: verifies all the blocks and stores entire blockchain information as well as maintains all the states. It can also provide requested data. [2.]
- Light node: verifies only essential small record of blockchain and requests other data from full node [9]. Light node can be used for low-capacity and lighter devices, such as mobile phones [2] which have limited data storage capacity [9].
- Archive node: stores everything in full node and creates an archive of previous states. Archive node is designed for chain analytics, block explorers, wallet vendors and more. [2.]

3.4 Decentralized application and concept of Web3

Decentralized application or DApp, is Ethereum-based application [9] using a smart contract and a front-end user interface (UI) [2]. Unlike traditional application whose back-end code is running on centralized database, back-end code in a dApp running on a decentralized network and according to Ethereum, smart contract can be considered as dApp's back-end.

Based on the context of Ethereum, web3 refers to dApps running on the blockchain network. Web3 applications are private because they do not require an exchange of personal data for providing services. Moreover, they also commit data integrity, zero downtime and resistance to restriction.

However, dApp has a few limitations due to its principles. It is difficult to maintain a dApp because data and code are immutable after deploying on the network. Another problem occurs as transaction throughput now is about 20 transactions per second, it will cause network congestion if more transactions need to be processed. [2.]

3.5 Networks

Ethereum offers different environments called networks, with the purpose of development, testing and production. There are two types of networks in Ethereum: public and private.

3.5.1 Public networks

In public network, everyone with internet connection can view, create and verify transactions on public blockchain.

- Mainnet: is the only public Ethereum production blockchain and used for actual transactions with value in real world on distributed ledger.
- Testnets: are instance of a blockchain to be used for testing before deploying to Mainnet. Testnet ETH is separated from official ETH and can be acquired by faucets webapps. Arbitrum Rinkeby, Görli, Kintsugi, Kovan, Optimisic Kovan, Rinkeby and Ropsten are examples about common test networks today. [2.]

3.5.2 Private networks

Nodes in private networks are isolated from the public network. Private blockchain networks are built to test Ethereum application before deploying it, which allows more time saving than public testnet. Furthermore, organizations build consortium network acting as private intranet to keep data confidential. [2.]

3.6 Ethereum 2.0

Ethereum 2.0, also known as Serenity or Eth2, which under development now, is expected to provide a scalable and sustainable network to increase transaction throughput from about 20 transactions per second now to 100.000 transactions per second [10].

Eth2 is designed to be upgraded in following phases:

- Phase 0 (The Beacon Chain): launched in December 2020, it first introduced separated Beacon Chain, the chain that uses proof-of-stake (PoS) as a new type of consensus mechanism.
- The Merger: expected to be shipped in third or fourth quarter of 2022, it will merge current Mainnet with The Beacon Chain as well as end the use of proof-of-work and move to proof-of-stake.
- Shard chains: expected to be released in 2023, includes multi-phase upgrade. Its purpose is to improve Ethereum as a more scalable and capable network. [9.]

4 Smart contract

Smart contract is computer program stored on the blockchain that run when predefined conditions are satisfied. All parties participating in smart contract can exchange money, property and other valuables without third-party verification. The outcome of the agreement's execution is guaranteed as it is enforced by the blockchain. In addition, smart contract can automate the workflow, when the conditions are met, the next step is triggered automatically. [11.]

Smart contract now is implemented in various industries such as banking and finance, supply chain management, insurance and other use cases.

4.1 How it works

According to Nick Szabo, an engineer who first introduced the term "smart contract" in 1994, a vending machine is the metaphor for a smart contract. It is also known as a digital vending machine. To receive desired outputs, it is required to put right inputs. [2.]

Smart contracts are following "if/then" logical statement written on the blockchain. When the predefined conditions are met and verified, a computer network executes the functions. [11.] The process is also described user account's interaction with a smart contract in Ethereum [2]. When the transaction is finished, the blockchain will be updated. Therefore, the transaction is immutable and worked entirely as programmed [2] and only authorised participants can view the results [11].

Inside the smart contract, involved parties need to agree on the conditions that control the transactions, on which approaches transactions and data are represented on the blockchain, investigate all exceptions and design a solution for resolving disputes.

Nowadays, there are several templates, tools, web interfaces for easily building smart contract provided by companies that apply blockchain to business, alongside with programming by developer. [11.]

4.2 Ethereum smart contract

Smart contract is also a type of Ethereum account. It has balance and can send transactions through blockchain network when deploying the smart contract. Gas fee is requirement for the sender when deploying contract to keep the network secure from spamming.

Additionally, compilation is important before deploying contracts because it needs to be in bytecode that web application and EVM can understand it. When working with web application, Application Binary Interface (ABI) is a JSON file that explains the contract and contract's functions for the application. In Ethereum, ABI is used to interact with contracts from outside of the blockchain as well as between contract-to-contract interaction.

The following things are needed when deploying the contracts: the bytecode from compilation, gas fee, deployment script and an approach to Ethereum node. [2.]

4.3 Smart contract languages

In Ethereum, smart contract can be written using several programming languages with syntax familiar to JavaScript or Python.

- Solidity: a widely used and beginner-friendly language for writing smart contract inspired by C++, Python and JavaScript.
- Vyper: a Pythonic programming language running on EVM. Vyper follows simple and secure principles; therefore, it has limited features compared to Solidity.

- Yul: an intermediate programming language suitable used by experienced developers for high-level optimizations. Yul+ is Yul's extension with new added features. [2.]

4.4 Benefit

Smart contracts have various benefits comparing to traditional contracts.

- Automaticity, speed and accuracy: The smart contract is executed immediately when the conditions are met. In addition, it can avoid errors when preparing documents manually.
- Security: The blockchain technology is secure because of distributed and decentralised system, transaction information is protected and difficult to hack or exploit vulnerability. Within blockchain, each record is linked to the preceding and subsequent records, hackers need to impact the entire chain to change the single record.
- Transparency: Because there is no human intervention and because of the security, the information is safe and remain consistent.
- Savings: Without the presence of third-party to verify transactions, the related fee and time delays are eliminated. [11.]

4.5 Limitation

By default, smart contract cannot send HTTP request; therefore, it is unable to access data from outside blockchain network. Moreover, using external material could put the agreement in jeopardy. However, Oracles was developed to solve the issue, it acts a bridge between blockchain and outside. It takes external data and transmit it into blockchain for smart contract to use. [2.]

4.6 Solidity

Solidity is an object-oriented programming language for writing smart contract. It is supported by various blockchain platforms, especially Ethereum Virtual Machine [2].

Ethereum co-founder Gavin Wood came up with the idea about Solidity in 2014. Later in 2014, the concept was further developed by Christian Reitwiessner, Ales Beregszaszi and many Ethereum distributors. The first release was version 0.1.2 in August 2015. Until now Solidity is still under development with the Ethereum Foundation's sponsorship and the latest version is 0.8.13. [6.] Moreover, there is a community of collaborators involved in adding new features, building system, improving the documentation, reporting and fixing issues on GitHub [12].

Solidity was influenced by current programming languages such as C++, Python, Java. It is statically typed curly-braces, contract-oriented, high-level programming language. Because the similarities with other languages, it also has common features as inheritance, libraries that reusable code can be called from different contracts and complex custom types. [2.]

Figure 2 shows the Solidity code that store and update a variable inside the smart contract.


```

1  // SPDX-License-Identifier: GPL-3.0
2
3  pragma solidity >=0.7.0 <0.9.0;
4
5  contract Owner {
6
7      address private owner;
8
9      event OwnerSet(address indexed oldOwner, address indexed newOwner);
10
11     modifier isOwner() {
12         require(msg.sender == owner, "Caller is not owner");
13         _;
14     }
15
16     constructor() {
17         owner = msg.sender;
18         emit OwnerSet(address(0), owner);
19     }
20
21     function changeOwner(address newOwner) public isOwner {
22         emit OwnerSet(owner, newOwner);
23         owner = newOwner;
24     }
25
26     function getOwner() external view returns (address) {
27         return owner;
28     }
29 }

```

Owner.sol hosted with ❤ by GitHub [view raw](#)

Figure 2. Example of Solidity code. Adapted from inext.io [13]

In every file, SPDX-License-Identifier and “pragma” keyword are added at the top to enable compiler features and to declare the version of Solidity. In the next line, “contract” acts as a purpose of a class in normal OOP that contain attributes and methods. Inside the “contract”, the “event” is member contract that can be inherited and served as function that called later. The “event” stores the transaction log arguments each time it is emitted. The logs are kept on the blockchain and can be accessed with the address of the contract [13.] In Solidity, there is special data type called “address” that represent 20-bytes ETH address. Next, “function” keyword used to define executable block of code to modify the state of the contract. Additionally, because the smart contract is immutable, “constructor” can be called once when the contract is created and when the “constructor” is finished executing, the code will be deployed on the blockchain.

As Solidity is a new language and primarily designed for blockchain, it remains limitations for example security issues, short of documentation, user support,

ability to test and verify the program. In standard programming languages, code can be debugged; however, it is required different approach when working with Solidity because the code is immutable. When the smart contract is deployed, it is impossible to modify the code, data or logic inside. Following the best practices, for every possible case, the developers need to perform testing the code in testnet network before deploying to the Mainnet. [14.]

5 JavaScript

JavaScript is an interpreted, lightweight, object-oriented programming language. It is a scripting language that known as a core component of the Web, alongside HTML and CSS. While HTML and CSS have responsibilities for the structure and the style of the website, JavaScript allows to add functionality and interactive behaviours, it can improve the user interface and experience of a website.

JavaScript is dynamic-typed, multi-paradigm language with first-class function. It supports prototype-based object construction and functional programming. Its syntax is based on C and Java with purpose of decreasing the new concepts when studying new language. [15.]

In addition, JSON shorts for JavaScript Object Notation, was defined by Douglas Crockford, is a concept that derived from JavaScript. JSON is a standard text format for storing and transferring data over the network on the basic of JavaScript object syntax. JSON can be read and generated by other programming languages as it is text-based format. The data is a collection of name/value pairs separated by a comma and the pairs are required double quotes around it. [16.] JSON has become a useful data format because it is independent, human-readable and easy for data-interchanging.

Initially, JavaScript is designed to run in a host environment which provides the specific communication mechanisms with the outside world. And the browser is the common host environment for JavaScript. [15.] Furthermore, with the advent of Node.js and React Native, JavaScript now can be used to build server-side and mobile applications.

5.1 A brief history

Brendan Eich created JavaScript in 1995 when he was working at Netscape Corporation. Eich designed JavaScript to run as a scripting language for the competition of Netspace's web browser, Netscape Navigator and Microsoft

Internet Explorer. He built and completed the first version in only ten days. First the language was called Mocha, later it was changed to LiveScript in May and finally was JavaScript when it was first introduced in December 1995. [17.]

Although there is a similar in the name and the syntax between Java and JavaScript, there is a different in the core mechanism of the languages.

In 1997, JavaScript was adopted by ECMA International for standardization [17]. It is crucial to ensure the web's interoperability through different web browsers when interpreting JavaScript. Since the introduction of ECMAScript 2015 or ES6, JavaScript has been updated in June every year.

In 2008, the creation of Node.js JavaScript runtime environment run on Chrome V8 created in the same year, provided an important development for JavaScript later. With Node.js, JavaScript code can be executed outside a web browser; therefore, it allows the code to be run on server-side. Nowadays, JavaScript is well-suited to numerous types of application.

Today, JavaScript is one of the most popular programming languages in the world, especially it is core technology of the Web [17].

5.2 JavaScript frameworks and libraries

JavaScript programming language is widely used to build frameworks and libraries for simplifying the web and application development.

5.2.1 React

React is a front-end JavaScript library used to build user interfaces released in 2013 by Meta, formerly known as Facebook.

In addition, React Native is used to build mobile application for Android, iOS based on React and native components.

5.2.2 Vue.js

Vue.js is also a front-end JavaScript framework for building UIs and single-page application followed Model-View-ViewModel design pattern [17]. It allows developers to extend HTML with built-in and user-defined attributes called directives. Moreover, Vue.js has refined reactivity system that automatically keep track and update based on the state change. Vue.js was created by Evan You in 2014 and now it is maintained by open-source community with members from all around the world.

5.2.3 Node.js

Unlike React and Vue.js, Node.js is a back-end JavaScript runtime environment run on Chrome's V8, a JavaScript engine. It is a tool that allow JavaScript code run outside the web browser. [17.] Node.js was written in 2009 by Ryan Dahl. In many years, JavaScript developers use Node.js to share the software, manage and publish the code dependencies.

Npm stands for Node Package Manager is the package manager for JavaScript platform developed by Isaac Z. Schlueter in 2010. As npm is default package manager for Node.js, to use npm, Node.js needs to be installed first in the computer.

Command Line Interface (CLI) is one important component of npm. With CLI developers can download and install packages from a terminal. The installed packages are defined in package.json file that written in JSON. In the file, name and version using semantic versioning scheme of the dependency are specified. [18.] Moreover, coming with npm, Node Package Execute (npx) is a tool to run Node packages without installing. npx is useful to use single-time package and to avoid versioning problems, unnecessary packages installation.

5.3 The advantages and disadvantages

Like all other programming languages, JavaScript has its strengths and weaknesses. The reasons are related to the way JavaScript executed in client-side and the use of it in server-side [19].

5.3.1 The advantages of JavaScript

This section discusses the benefits of using JavaScript.

- **Popularity:** According to Stack Overflow's survey, JavaScript is one of the most popular and commonly used programming languages in the world as it is the core language to build websites, expand to server-side code and mobile applications. There are huge amounts of JavaScript tutorials, libraries, projects, resources that improve the productivity for the developer and learner. The frameworks and libraries written in JavaScript by technology companies also have large active communities supported by all developers and collaborators from over the world.
- **Speed:** JavaScript is executed on client environment; therefore, the execution is fast as it does not totally depend on web server's support and simple tasks can be completed instantaneously. In addition, XMLHttpRequest Object is a special object that allows to transfer the data between web browser and web server without reloading the page. Moreover, now many browsers support JIT stands for Just In Time compilation to enhance the performance of JavaScript by compiling the code at run time.
- **Simplicity:** The JavaScript's syntax was based on Java and C. It is considered as a simple language to learn and implement than other popular languages such as C++.
- **Versatility:** JavaScript is primarily for front-end development but with the advent of Node.js, it is now can be used in various aspects of

application. It is possible to create an entire JavaScript application by using Node.js, a back-end application framework Express.js and JSON-like document database MongoDB [19]. Furthermore, JavaScript interacts well with other programming languages.

- Update: Since the release of ECMAScript 2015 (also called ES6) added the significant changes for JavaScript, ECMA International has updated JavaScript each year. [19.] New edition is published in every June and ES12 edition or ECMAScript 2021 is the latest version.

5.3.2 The disadvantages of JavaScript

Besides the benefits, JavaScript also has a few limitations.

- Client-side security: Since the JavaScript code is executed directly in client's browser and it is visible to the user, bug and oversights can be exploited for malicious purposes. For the vulnerability, JavaScript is disable in the browser but certain features on the website required JavaScript are unfunctional. [19.]
- Browser support: Different browser interprets JavaScript code differently because it depends on browser vendor to decide how to implement the code inside it and the old browser stops supporting new functions. Therefore, the good practice is to check and run the test to ensure the best performance.

6 React

React is an open-source front-end JavaScript library for building user interfaces by UI components [20]. It is declarative, component-based, efficient, flexible tools commonly used for creating single page application (SPA) [21].

React introduces various fundamental features that make it significant such as Virtual DOM, components, JSX, props, state and Hooks. It is simple to build reusable interactive UIs, update the corresponding components based on the change of data which make it easier to read and reuse the code as well as solve the bug [20]. Using React can increase the speed of web development process, save time and money for individual and the business.

Nowadays, React is still maintained by active communities working in the project with the goal of evolving React core, making it easy and fast. It is also one of the reasons makes React popular since its release. There are enormous available React libraries that generally meet and support the requirement of developers, such as Gatsby for static website, Next.js for server-side rendering, React-spring for animations, Formik for forms, React Native for mobile application and state management with Redux, MobX, Flux, XState, Recoil. Having knowledge in UI libraries is one of the most vital skills for front-end developer today.

Currently the well-known companies include Netflix, Airbnb, Facebook, Instagram, Dropbox use React as their technologies. It indicates the proof of the value and efficiency of React as the most influential front-end library. React is expected to maintain its position in near future with rapid and sustainable development. [21.]

6.1 Brief history

React was developed by Jordan Walke, a software engineer at Meta, formerly known as Facebook. Back in 2011, when Facebook application grew stronger, it was crucial to update the code base and add new features to the system. The

issues were the difficulty in cascading update as well as in code maintenance and management. Therefore, Jordan Walke created an original prototype of React called FaxJS to make the operation more effective. [21.] The first use of React was in Facebook's newfeed in 2011. Later in 2012, it was launched on Instagram after the acquisition from Facebook. [22.]

At JSConf US in May 2013, Jordan Walke made FaxJS an open-source and introduced it as React. In 2014, new feature React Developer Tools was announced to become an extension of Chrome Developer Tools and in the same year, React introduced React Hot Loader plugin. [21.]

In March 2015, React v0.13 was released which added new features for supporting ES6. Later in March, a framework for creating mobile application using React called React Native was introduced. And in 2015, React was implemented to enhance UI development by well-known companies such as Netflix and Airbnb. From there, there are a huge number of React libraries to handle various tasks that released by the ecosystem around it, such as React Router, Redux, MobX in 2016, React Fiber in 2017 and recently Hooks in 2019. [22.]

6.2 Fundamental features of React

This section introduces an overview about main React concepts to understand how React works.

6.2.1 Virtual DOM

DOM or Document Object Model is an application programming interface for HTML and XML documents. It represents the web documents as objects and nodes, for that reason, programming languages can interact with the web page and can modify the structure, content and style of the document. [23.]

In React, every DOM object has its corresponding virtual DOM object. The virtual DOM is a programming concept that acts as a representation of a DOM object.

When changes are made, React will update every virtual DOM objects. The “diff” process compares the differences between the new virtual DOM and the previous version. Next, React updates only the required DOM based on the changes that was made. [24.] The entire process is called reconciliation.

6.2.2 JSX

JSX stands for JavaScript XML and it is a JavaScript syntax extension. XML is Extensible Markup Language used to store and transmit data like HTML. It is recommended to use JSX with React the describe the UI components [20].

Figure 3 shows the variable declaration written in JSX.

```
function getGreeting(user) {  
  if (user) {  
    return <h1>Hello, {formatName(user)}!</h1>;  
  }  
  return <h1>Hello, Stranger.</h1>;  
}
```

Figure 3. JSX syntax.

Like HTML, JSX tags also includes attributes, tag names and children. In JSX, JavaScript expression is the attribute inside curly braces.

To convert JSX into JavaScript, React uses Babel, a JavaScript compiler for converting ES6 code into previous JavaScript versions that the browser supports. [25.]

6.2.3 Component

React is a component-based library for building components that represent logical reusable part of UI. Theoretically, component is just JavaScript function. The return value from this function is HTML which written in JSX, a syntax for

combining JavaScript and HTML easily. There are two types of components: functional component and class component. [20.]

6.2.4 Props and state

Props and state are basically plain JavaScript objects. [26]

- **Props:** stand for properties, are passed to React component like the parameter in a function. Props is immutable when declaring a component. [20.]
- **State:** is built-in React object that stores information about the value. It is considered “private” as state is created and controlled inside the component. The component triggers the render update when the state changes.

As mentioned below, there are functional component and class component. They are distinguished by props and state.

- **Stateless component:** also known as functional component, that accepts props and return JSX. It is mainly used for displaying data and rendering UI.
- **Stateful component:** also known as class component, has both props and state. Class component always needs to include “extends React.Component” that give the permission to access functions of React.Component and render() method that returns JSX code. It is responsibilities for communication between client-side and server-side, processing data and state management.

It is recommended that since version 16.8 use Functional component with Hooks to make it stateful because it is easier and predictable than Class component. [26.]

6.2.5 React Hooks

React Hooks are features added to React in version 16.8. Hooks allow developer to use state and other features without writing a class. With Hooks, developers can reuse stateful logic without updating component system. Moreover, Hooks are useful to improving rendering performance and it provides a new way to combine current React concepts: props, state, ...

Below are basic built-in Hooks that widely used in React:

- **useState:** accepts initial value of state and it will return a current stateful value and a function used to update the state.
- **useEffect:** accepts a function that performs side effects which are not allowed in the main body of the function component. `useEffect` accepts function passed to it as first argument and second argument is optional. The function run after every time the render is completed; however, to optimize the process, there are a few ways to set the conditions for running the side effects.
- **useContext:** accepts a context object returned from `React.createContext` and returns value “prop” of `<.Provider>` component as current context value of the context. `useContext` allows to work with Context API and share data without passing any props.

In addition, there are a few additional built-in Hooks and Library Hooks as well. [20.] Building custom Hooks is a useful option to handle reusable stateful logic in different components. The name of custom Hooks starts with prefix “use”.

An important rule is that Hooks can only be called inside at the top level of React functional components as well as from custom Hooks [20].

7 Project Implementation

This section explains how the application was created with three main features: connecting to the wallet, sending and listing transactions.

In the project, Node.js was installed in advance because it used npm commands to install dependencies and packages. Figure 4 illustrates the folder structure of the project in Visual Studio Code.

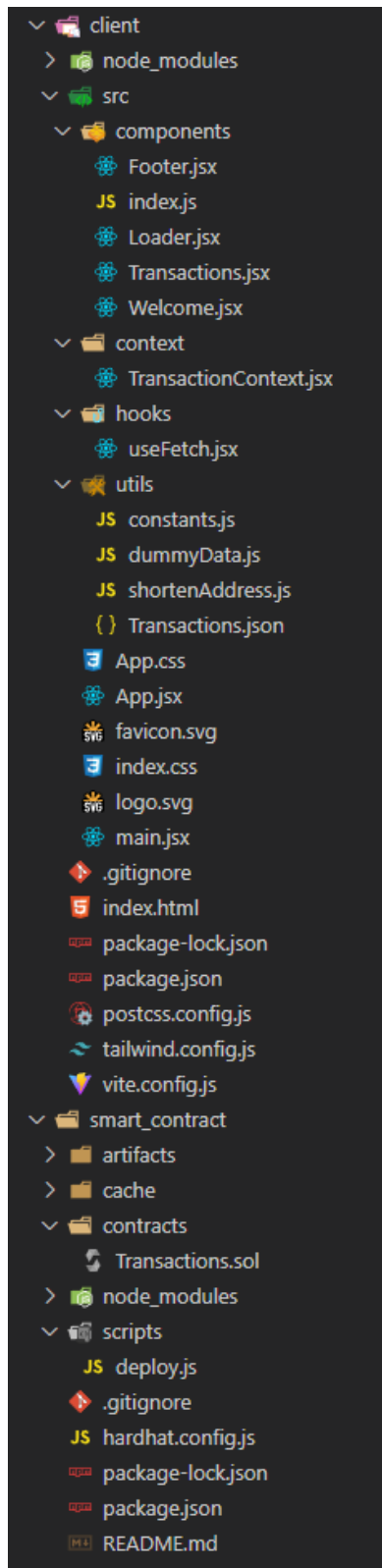


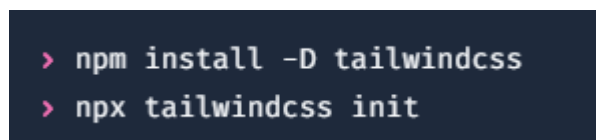
Figure 4. Project structure.

The project includes two main folders: client and smart_contract. Client is where React application was created, while smart_contract folder is the place that stored the smart contract code file and the script to deploy it.

7.1 Initializing and setting up React application

In client folder, Vite was used to initialize React. It is a front-end build tool that installs, builds and starts React application faster. After setting up, use npm run dev command to start the project and the application was running on localhost:3000.

Tailwind CSS is the framework for user interface and styling. It allows to style the design and build modern website with simple utility classes instead of writing CSS code. Figure 5 shows the commands for installing Tailwind via npm.

A terminal window with a dark background showing two commands: > npm install -D tailwindcss and > npx tailwindcss init.

```
> npm install -D tailwindcss
> npx tailwindcss init
```

Figure 5. Tailwind installation commands.

The commands created tailwind.config.js file which is an optional file for customizations. Tailwind directives also added to index.css file after importing a few of customized configurations.

React-icon and ethers were two additional packages to be installed. Ethers is the library that allows React to easily interact with blockchain and smart contract.

In src, new folder called components was created to store all components: Loader.jsx for loading spinner animation, Welcome.jsx for connecting wallet and sending transactions, Transaction.jsx for listing transactions. To import all components in one line or export all components from single file, index.js file was created in components folder. Finally, components were imported in App.jsx file to form the structure of the application.

7.2 Writing Solidity smart contract

In smart_contract folder, command `npm init -y` was used to initialize the project, it acted as a starting point of smart contract. The next step was downloading Hardhat, Ethers.js and other plugins using command in listing 1.

```
npm install --save-dev hardhat @nomiclabs/hardhat-waffle ethereum-waffle chai
@nomiclabs/hardhat-ethers ethers
```

Listing 1. Installing Hardhat and other necessary dependencies.

Hardhat is the Ethereum development environment, it allows to run Solidity locally and test smart contract before deploying. Moreover, Hardhat also supports additional plugins for extended functionality.

To create basic Hard project structure, run `npx hardhat` and select create a basic sample project. In basic sample project, two folders for keeping smart contract code file and script to deploy smart contract were created. Solidity smart contract that keeps track all transactions through blockchain was written in new file called `transactions.sol` inside `contracts` folder. Listing 2 shows the first half part of smart contract codes used in the project.

```
pragma solidity ^0.8.4;
contract Transactions {
    uint256 transactionCount;
    event Transfer ( address from, address receiver, uint256 amount,
string message, uint256 timestamp, string keyword );
    struct TransferStruct { address sender; address receiver; uint256 amount;
        string message; uint256 timestamp; string keyword; }
    TransferStruct[] transactions;
    ...
}
```

Listing 2. Solidity smart contract.

As shown in listing 2, the process started with declaring the version of Solidity in form of semantic versioning. Next, a contract which is similar to class in OOP languages, called `Transactions` was defined. Contract `Transactions` contains declarations of state variable, event, struct and functions. Firstly, variable `transactionCount` with type `uint256` holds the number of transactions. Secondly, the event was declared to be emitted later. After emitting, the event allows to

query the action that happened on the blockchain in transaction logs. Next, struct or structure, is similar to an object that specifies what properties the transaction needed to have. Finally, an array was defined to store all transactions. Therefore, transactions variable is an array of TransferStruct.

In construct Transactions, there are three main functions to decide what happens within a contract.

- **addToBlockchain:** This is the main function of smart contract as shown in Listing 3.

```
function addToBlockchain(
address payable receiver, uint256 amount,
string memory message, string memory keyword
) public {
    transactionCount += 1;
    transactions.push(
        TransferStruct(
            msg.sender, receiver, amount, message,
            block.timestamp, keyword
        )
    );
    emit Transfer(
        msg.sender, receiver, amount, message,
        block.timestamp, keyword
    );
}
```

Listing 3. addToBlockchain function.

The function is public function and do not return anything. When addToBlockchain function is called from React, parameters are passed into it with receiver, amount, message and keyword. The function tasks are increasing transactionCount by 1 and push a transaction into transactions array as well as passing necessary parameters. Finally, to make the transfer, event Transfer was emitted and passed the same parameters as above. Overall, addToBlockchain is a function to transfer and store all transactions on blockchain.

- **getAllTransactions:** Listing 4 shows the function for getting all transactions.

```
function getAllTransactions() public view returns
(TransferStruct[] memory) {
return transactions; }
```

Listing 4. getAllTransaction function.

It is a public, view-only function that returns transactions as an array and get it from the memory.

- **getTransactionCount:** The function for getting the count of transaction is shown in Listing 5.

```
function getTransactionCount() public view returns (uint256) {
return transactionCount; }
```

Listing 5. getTransactionCount function.

It is a public, view-only function that returns transactionCount as a number.

7.3 Deploying smart contract

In this section, smart contract was deployed using Hardhat, MetaMask and Alchemy on Ropsten test network. Deployment script was written in deploy.js file and listing 6 illustrates the main function of it.

```
const main = async () => {
  const transactionsFactory = await
hre.ethers.getContractFactory("Transactions");
  const transactionsContract = await transactionsFactory.deploy();
  await transactionsContract.deployed();
  console.log("Transactions address: ", transactionsContract.address);
};
```

Listing 6. Main function of deploy.js

In the function, transactionsFactory generated the instances of the contract. Calling deploy() on transactionFactory started the deployment. After running the script, the transaction deployed and the address of deployed smart contract on the blockchain was logged in the terminal.

To deploy the smart contract, a wallet with ETH is required to pay the gas fee. First was creating Ethereum account to send and receive transaction in Ropsten test network using MetaMask, a virtual wallet. As in test network, there are several faucets offering test ETH to fund the deployment by entering MetaMask account address. Another configuration to connect to Ethereum network was Alchemy. It is a platform that providing nodes to communicate with Ethereum chain. HTTP key was obtained after creating new application with development environment, Ethereum chain and Ropsten test network.

Next, `hardhat.config.js` file needed to be updated with added plugins and dependencies as shown in listing 7.

```
require('@nomiclabs/hardhat-waffle');

module.exports = {
  solidity: '0.8.4',
  networks: {
    ropsten: {
      url: '',
      accounts: ['']
    },
  },
};
```

Listing 7. `hardhat.config.js` file

In listing 7, `hardhat-waffle` is the plugin to build smart contract testing. Solidity version and name of the test network were specified in order to export. URL was the HTTP key from Alchemy and account was the private key from MetaMask.

To deploy the contract, run `npx hardhat run scripts/deploy.js --network ropsten`. The result printed in the console was the address of deployed contract on Ethereum testnet.

7.4 Connecting smart contract to React

In `src` folder, new folder `utils` was created and also `constants.js` file inside it. The file contains the address of deployed contract above and ABI from `transaction.json` which also automatically generated from the deployment. This

file keeps all information in React app to interact with smart contract using ethers.js.

All the interaction stored in TransactionContext.js file. React context API was used to connect to the blockchain and it allows not to write the logic across all components but in one centralized place.

To interact with smart contract, it was needed to fetch an instance of Ethereum contract using function shown in Listing 8.

```
const getEthereumContract = () => {
  const provider = new ethers.providers.Web3Provider(ethereum);
  const signer = provider.getSigner();
  const transactionContract = new ethers.Contract(contractAddress,
contractABI, signer);

  return transactionContract;
}
```

Listing 8. Function to fetch the contract.

Provider, signer and contract are three concepts to create the instance. Provider is the node provider that gives the application the access to the blockchain. Signer represents Ethereum account that can sign the transactions. And the contract is an Ethers.js object that refers to a deployed contract. Moreover, when connecting to the wallet, the application had the access to “ethereum” object which allows it to send the request the user’s account.

The following functions were declared to connect to the MetaMask wallet and to send the transaction using smart contract.

- **checkIfWalletIsConnected:** the async function is called at the start of application to check if there is any connected account for rendering the UI. The code of function is displayed in listing 9.

```

const [currentAccount, setCurrentAccount] = useState("");
...
const checkIfWalletIsConnected = async () => {
  try {
    if(!ethereum) return alert("Please install Metamask");
    const accounts = await ethereum.request({ method:
'eth_accounts' });
    if (accounts.length) {
      setCurrentAccount(accounts[0]);
      getAllTransactions();
    } else {
      console.log('No account found')
    }
  } catch( error) {
    console.log(error);
    throw new Error("No ethereum object");
  }
};

```

Listing 9. checkIfWalletIsConnected function

In the function, method `eth_accounts` returns an array representing the current connected address. If there is connected account, the first item is set as current account and the function also returns all record of transactions in `getAllTransaction()` function that defined in next section.

- **connectWallet:** The function used to connect with MetaMask wallet as shown in listing 10.

```

const [currentAccount, setCurrentAccount] = useState("");
...
const connectWallet = async () => {
  try {
    if (!ethereum) return alert("Please install MetaMask.");
    const accounts = await ethereum.request({ method:
'eth_requestAccounts' });
    setCurrentAccount(accounts[0]);
  } catch (error) {
    console.log(error);
    throw new Error("No ethereum object");
  }
};

```

Listing 10. connectWallet function

Calling this function with method `eth_requestAccounts` requests MetaMask to be opened in the browser. It returns all the addresses that connected to the application as an array and function sets the first address as current address.

- **sendTransaction:** This function contains all the logic for sending and storing transactions. Firstly, it was needed to call all contracts related using variable `transactionContract` and get data from input of the form in application.

Listing 11 shows the sending action started with the call to method `eth_sendTransaction` and accepted a few parameters such as `from` address, `to` address, gas fee and value written in hexadecimal.

```

    await ethereum.request({
      method: 'eth_sendTransaction',
      params: [{
        from: currentAccount,
        to: addressTo,
        gas: '0x5208', // 21000 GWEI = 0.000021 Ether
        value: parsedAmount._hex, // 0.00001
      }]
    });
...
const transactionHash = await transactionContract.addToBlockchain(
  addressTo, parsedAmount, message, keyword);
  setIsLoading(true);
  console.log(`Loading - ${transactionHash.hash}`);
  await transactionHash.wait();
  setIsLoading(false);
  console.log(`Success - ${transactionHash.hash}`);

```

Listing 11. Sending Transaction action.

The `addToBlockchain` function was called to get the transaction hash or id. The loading state was set to true to wait for the transaction to be finished. When it finished, the loading state was false and the successful console was logged.

In `TransactionContext.jsx`, `TransactionProvider` the parent function of three above functions, needs to get `{children}` parameter from the props and returns `TransactionContext.Provider` as shown in Listing 12.

```

return (
  <TransactionContext.Provider value={{ connectWallet, currentAccount,
  formData, setFormData, handleChange, sendTransaction, transactions, isLoading
  }}>
    {children}
  </TransactionContext.Provider>
);

```

Listing 12. `TransactionContext.Provider` component.

In main.jsx file, wrap the application with TransactionContext.Provider to pass all data from TransactionContext.jsx to any children components. As the result, Welcome and Transaction components have the access the data inside the value above.

7.5 Connecting to wallet and sending transactions

As Welcome component is the child of TransactionContext.Provider, it has access to data and functions of TransactionContext. In Welcome.jsx component, the main concepts were Connect Wallet button, form and Submit button.

- **Connect Wallet button:** Clicking button triggered MetaMask connection with the properties shown in Listing 13.

```

    {!currentAccount && (
      <button
        type="button"
        onClick={connectWallet}>
        <p>Connect Wallet</p>
      </button>
    )}

```

Listing 13. Connect Wallet button.

currentAccount state and connectWallet function were derived from TransactionContext. The button is rendered if there is no current account connected.

- **Form and Submit button:**

To handle with input from the form, Input component which is basic arrow function were created to reuse it as shown in Listing 14.

```

const Input = ({ placeholder, name, type, value, handleChange })
=> (
  <input
    placeholder={placeholder}
    type={type}
    step="0.0001"
    value={value}
    onChange={ (e) => handleChange(e, name) }
  />
);

```

Listing 14. Input component.

The component created basic HTML input tag and returned basic version of Input that contains placeholder, type, step for value as the ETH value is large and onChange. onChange is call-back function that get the event and call handleChange function which interacts with the input that accepts keyboard event as pressing key and the name of the event. Listing 15 shows the handleChange function.

```
const handleChange = (e, name) => {
  setFormData((prevState) => ({ ...prevState, [name]:
    e.target.value }));
};
```

Listing 15. handleChange function.

handleChange function returns an object having previous state value and new value from e.target property. Therefore, it dynamically updates input depending on the name of the input.

Listing 16 describes handleSubmit function attached with Submit button.

```
const handleSubmit = (e) => {
  const { addressTo, amount, keyword, message } = formData;
  e.preventDefault();
  if(!addressTo || !amount || !keyword || !message) return;
  sendTransaction();
};
...
{isLoading
  ? <Loader />
  : (
    <button
      type="button"
      onClick={handleSubmit} >
      Send now
    </button>
  )}
```

Listing 16. handSubmit function and Submit button.

handleSubmit function destructures all properties from form data and prevents reload when submitting data. Next, it checks the filled data and execute sendTransaction function. When the transaction is processing, the application renders Loader component which is

spinner animation. And when it is finished, the Submit button is rendered.

7.6 Retrieving the list of transactions

Two functions were created in TransactionContext.jsx serving the purpose of listing the transactions.

- `checkIfTransactionsExist`: sets up the current number of transactions which stored in local storage.
- `getAllTransactions`: returns all transactions that created using function in Solidity smart contract.

`useEffect` Hooks is used with `checkIfWalletIsConnected` and `checkIfTransactionsExist` as parameters because they need to be called at the start of application to fetch data and render UI.

Transaction details such as address from, address to, amount, message, gif and timestamp are stored in a card component as in listing 17.

```
{ transactions.reverse().map((transaction, i) => (
<TransactionsCard key={i} {...transaction} />
))}
```

Listing 17. How transactions are rendered in the card.

All transactions from current account which derived from TransactionContext. In the loop, each transaction is displayed in transaction card with its properties.

Additionally, a custom hook was defined for fetching the Gif from Giphy platform based on keyword inside the card in Listing 18.

```

const useFetch = ({ keyword }) => {
  const [gifUrl, setGifUrl] = useState("");
  const fetchGifs = async () => {
    try {
      const response = await
fetch(`https://api.giphy.com/v1/gifs/search?api_key=${APIKEY}&q=${keyword.split(" ").join(" ")}&limit=1`);
      const { data } = await response.json();
      setGifUrl(data[0]?.images?.downsized_medium.url);
    } catch (error) {
      setGifUrl("");
    }
  };
  useEffect(() => {
    if (keyword) fetchGifs();
  }, [keyword]);
  return gifUrl;
};

```

Listing 18. Custom hook for Gif.

In `fetchGifs` function, requested the response from Giphy API with following parameters: `api_key` provided by Giphy, `q` as keyword query and limit to only one result. After that, destructured the data from response and set the state of `GifUrl`. Finally, called `fetchGifs` function inside `useEffect` to fetching Gif when there is a keyword.

8 Results

This section describes the outcome of the project. The following steps are carried out to run and use the application.

Step 1: Run `npm run dev` in the client directory to start the project. The application is running on `localhost:3000` as shown in figure 6.

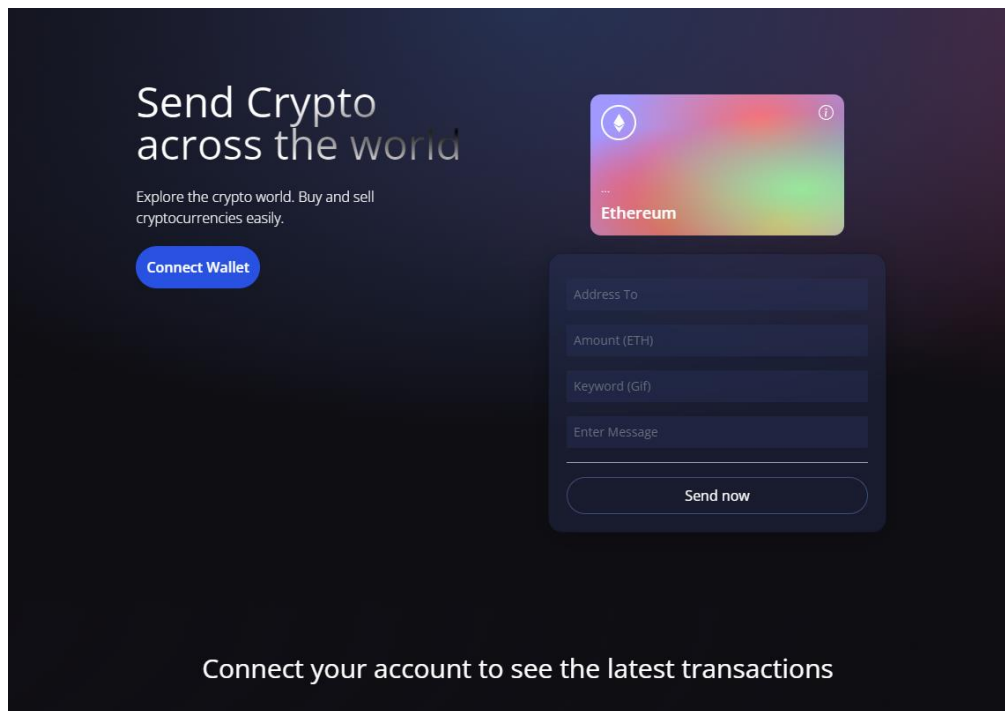


Figure 6. The application's view.

Step 2: Click Connect Wallet button to trigger MetaMask connection and connect the wallet to application. Figure 7 illustrates the MetaMask interface after hitting Connect Wallet.

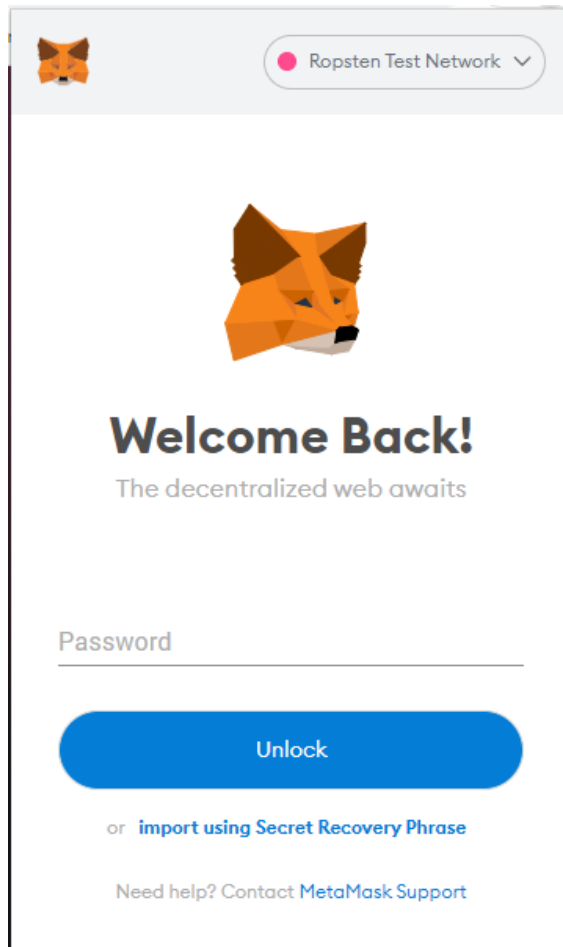


Figure 7. MetaMask interface as an extension in Chrome.

Step 3: Enter MetaMask password and choose the account to connect on Ropsten Test Network. When the connection is successful, Connect Wallet button disappears and the Ethereum address starting is shown on the card as shown in Figure 8.

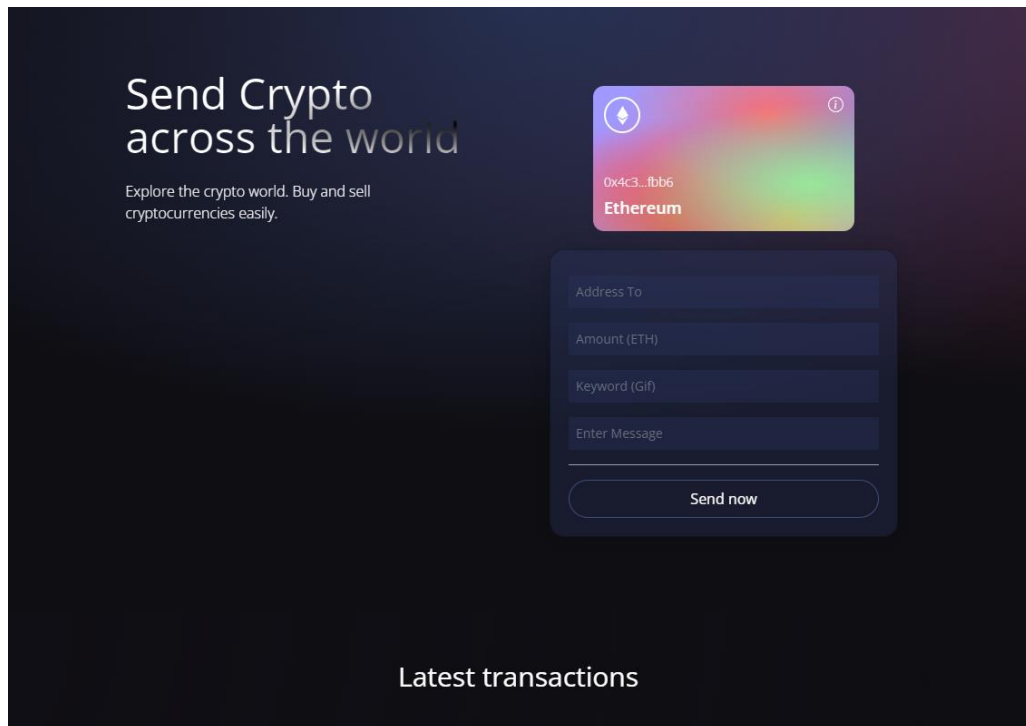


Figure 8. Application after connecting to wallet.

Figure 9 shows that the account 1 is now connected with the application.

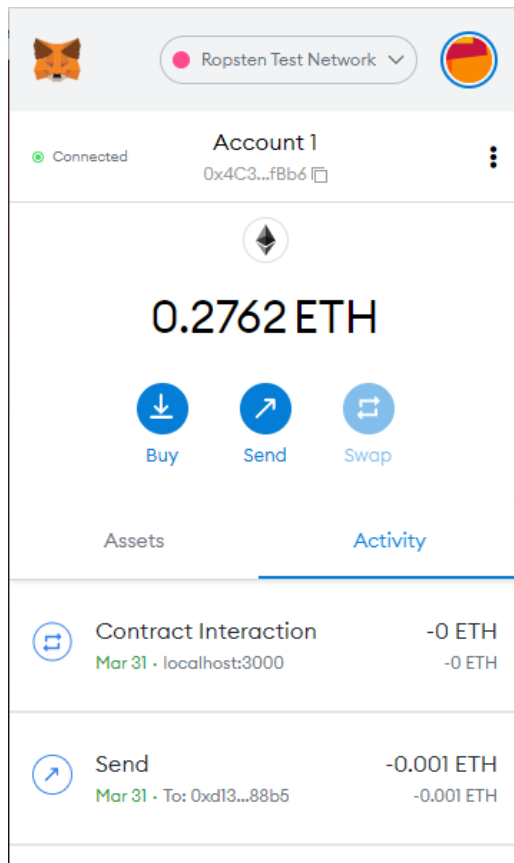


Figure 9. MetaMask wallet when login in.

Step 4: Fill the card with the address, amount, keyword and message like the example in figure 10 to send the transaction to another account.

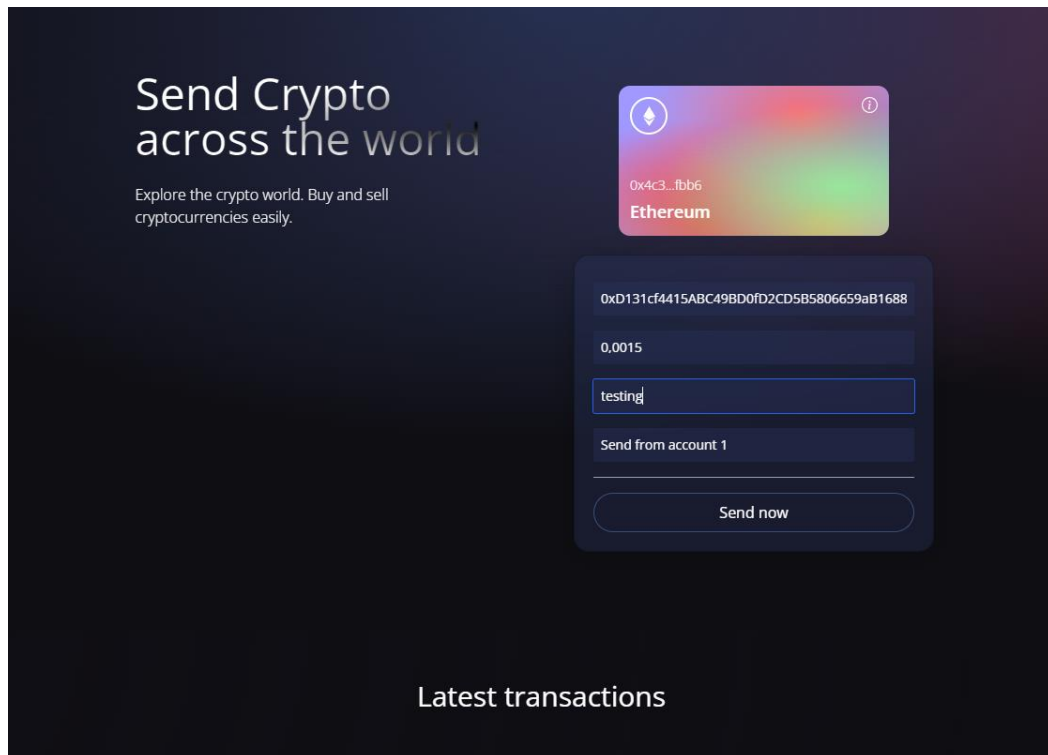


Figure 10. Example of sending transaction.

Step 5: Click Confirm to process the transaction in Metamask notification as illustrated in figure 11.

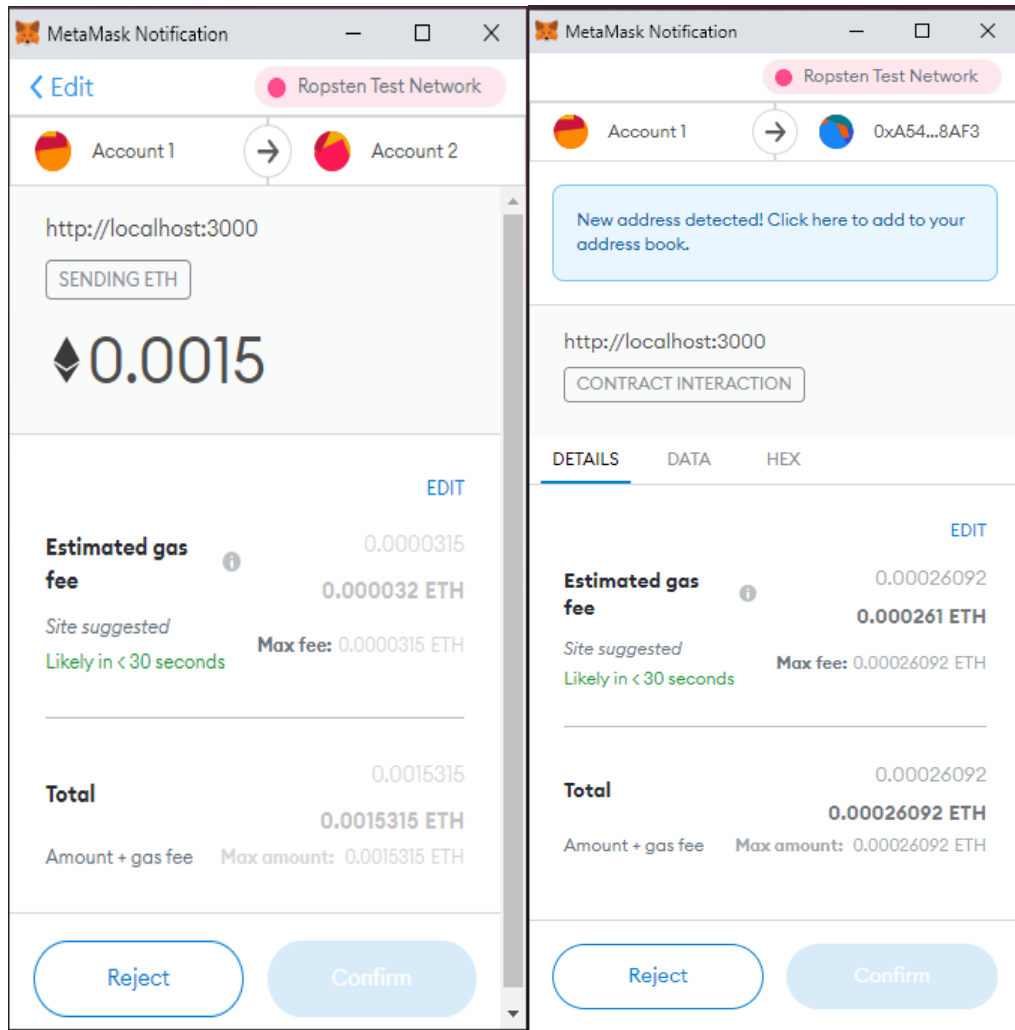


Figure 11. First and second notification to confirm the transfer.

In first notification, MetaMask asks to confirm the sending of 0.0015 ETH to another account. Second notification is the contract interaction and in here the user is interacting with Solidity smart contract.

Step 6: Reload the page when the loading is complete. The transaction below is displayed in the Latest transactions part. Figure 12 illustrates the transaction details with the information about addresses, amount in ETH, message and a gif based on the given keyword as well as transaction time.

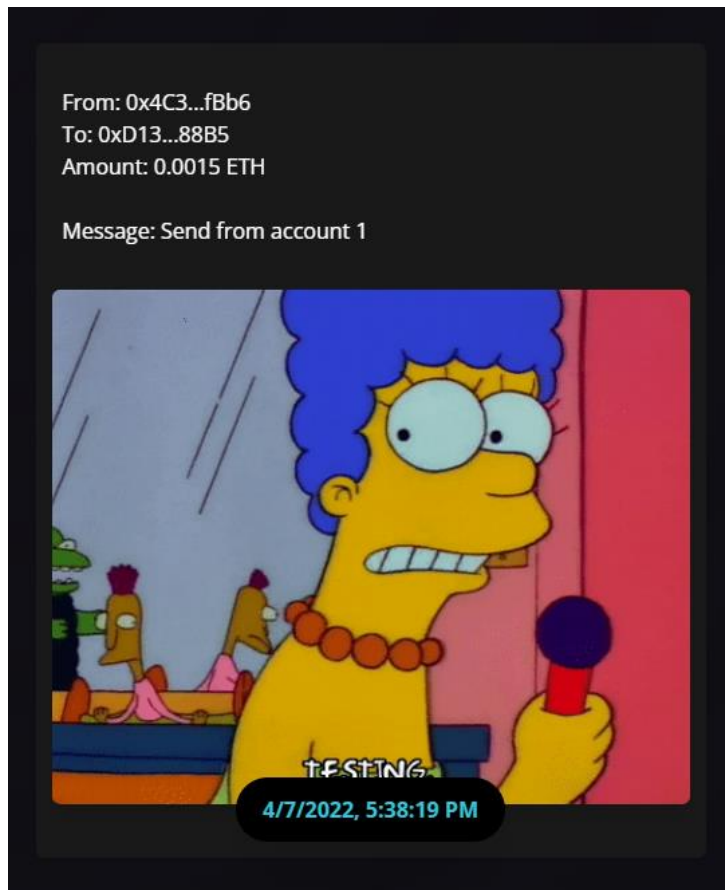


Figure 12. Details of the transaction.

Step 7: Open MetaMask wallet to check the balance and the transaction history. Compared to figure 9, the balance in figure 13 is reduced because the amount transferring to another account and the gas fee for transaction execution.

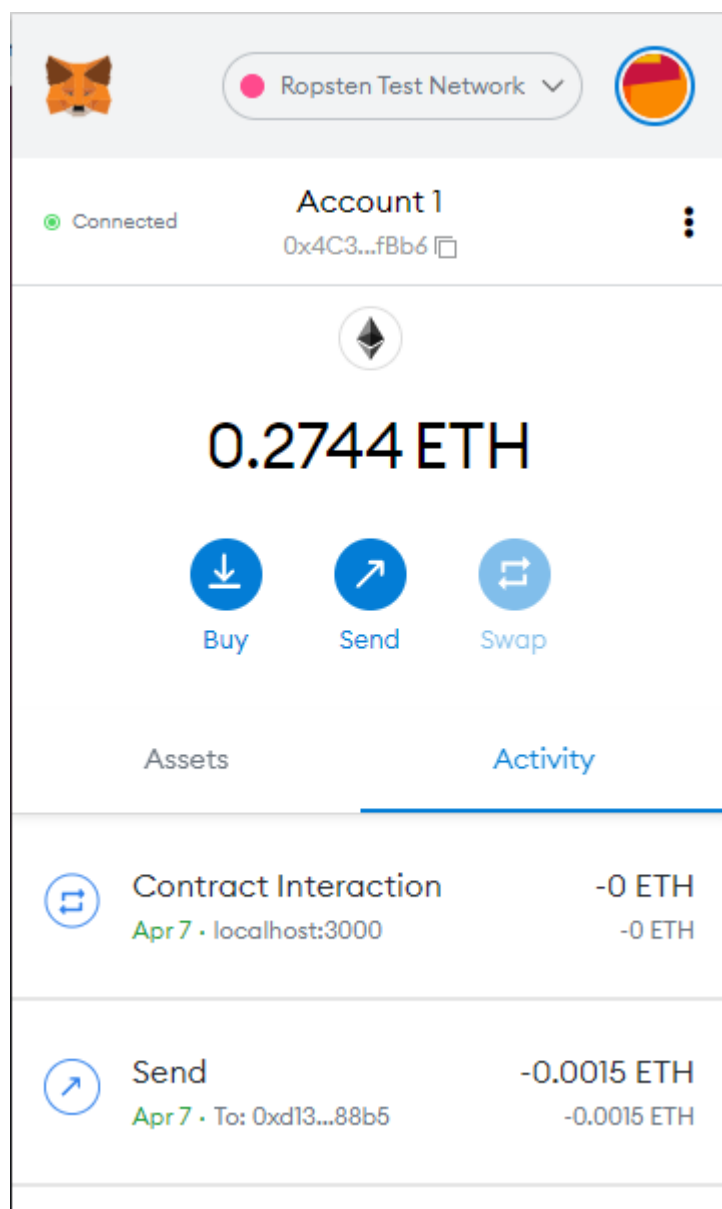


Figure 13. MetaMask wallet of account 1.

Step 8: Click the link in first or second line in each transaction to open transaction details on ropsten.etherscan.io as shown in figure 14.

The screenshot shows the Etherscan interface for a transaction on the Ropsten Testnet. The transaction is successful and has 4 block confirmations. It was sent from an address starting with 0x4c3fc358 to an address starting with 0xd131cf44. The value is 0.0015 Ether, and the transaction fee is 0.000031500140049 Ether. The gas price is 0.00000001500006669 Ether, and the gas limit is 21,000. The transaction is of type 2 (EIP-1559) with a nonce of 9 and position 24. The input data is 0x.

Etherscan
Ropsten Testnet Network

All Filters Search by Address / Txn Hash / Block / Token / Ens

Home Blockchain Tokens Misc Ropsten

Transaction Details

Overview State

[This is a Ropsten **Testnet** transaction only]

Transaction Hash: 0x4d7d1c126cd987927b2d91791696a1b10f5b6d330262139465076d205f9990c6

Status: **Success**

Block: 12171556 4 Block Confirmations

Timestamp: 3 mins ago (Apr-07-2022 02:37:31 PM +UTC)

From: 0x4c3fc358be6f5d59a6181523f3ca7c1f97f3fbb6

To: 0xd131cf4415abc49bd0fd2cd5b5806659ab1688b5

Value: 0.0015 Ether (\$0.00)

Transaction Fee: 0.000031500140049 Ether (\$0.00)

Gas Price: 0.00000001500006669 Ether (1.500006669 Gwei)

Gas Limit & Usage by Txn: 21,000 | 21,000 (100%)

Gas Fees: Base: 0.000006669 Gwei | Max: 1.500007898 Gwei | Max Priority: 1.5 Gwei

Burnt & Txn Savings Fees: **Burnt:** 0.00000000140049 Ether (\$0.00) **Txn Savings:** 0.00000000025809 Ether (\$0.00)

Others: Txn Type: 2 (EIP-1559) Nonce: 9 Position: 24

Input Data: 0x

[Click to see Less](#)

ⓘ A transaction is a cryptographically signed instruction from an account that changes the state of the blockchain. Block explorers track the details of all transactions in the network. Learn more about transactions in our [Knowledge Base](#).

Figure 14. Transaction Details on Etherscan within Ropsten Testnet.

This proves that there is an actual transaction sent by Ethereum address through the blockchain and can be accessed via any Block Explorer platform.

9 Conclusion

The goal of the project was to study decentralized technology blockchain and its key elements which are smart contract as well as creating a cryptocurrency transfer application where the user can send a transaction with a rapid and simple process.

The result was a React application that connects to the blockchain with Solidity was used to interact with smart contract. Each transaction sent by the user was attached with a Gif and stored permanently on Ethereum network. The application worked as a basic example to demonstrate the benefit of applying blockchain in providing transparency and trust as well as optimising the time and cost in the transaction.

Overall, the project was successful as the author achieved the goals that were stated at the beginning of this thesis. Based on this study, applying the concept of blockchain into a React application assisted effectively in consolidating the theories. For further developments, the application can be deployed to the domain as it is hosted on localhost now. Using this application the user can use their own Metamask account and send Ethereum through the blockchain. Because cryptocurrency commonly has a high risk related to scam or deceit, it is recommended to enhance the security and reliable of the application.

References

- 1 What is blockchain technology? [online]. IBM Corporation.
URL: <https://www.ibm.com/topics/what-is-blockchain>. Accessed 22 April 2022.
- 2 Pace Katerina, Smith Corwin, Pal Anurag, Wackerow Paul, Joshua, il3ven, Stephenson Tap, Wilden Chiara, Richards Sam, Cordell Ryan. Ethereum Development Documentation [online]. Ethereum; 2022.
URL: <https://ethereum.org/en/developers/docs>. Accessed 22 April 2022.
- 3 Darlington Nick. What is Blockchain Technology? [online]. Blockgeeks; 25 November 2021.
URL: <https://blockgeeks.com/guides/what-is-blockchain-technology>. Accessed: 22 April 2022
- 4 Making sense of bitcoin, cryptocurrency and blockchain [online]. PWC.
URL: <https://www.pwc.com/us/en/industries/financial-services/fintech/bitcoin-blockchain-cryptocurrency.html>. Accessed: 22 April 2022.
- 5 Blockchain [online]. Binance Academy.
URL: <https://academy.binance.com/en/glossary/blockchain>. Accessed: 22 April 2022.
- 6 How does a transaction get into the blockchain? [online]. Euromoney Learning.
URL: <https://www.euromoney.com/learning/blockchain-explained/how-transactions-get-into-the-blockchain>. Accessed 22 April 2022.
- 7 A Blockchain Glossary for Beginners [online]. Consensys.
URL: <https://consensys.net/knowledge-base/a-blockchain-glossary-for-beginners>. Accessed: 22 April 2022.
- 8 Rodeck David, Curry Benjamin. What is Blockchain? [online]. Forbes Advisor; April 2022.
URL: <https://www.forbes.com/advisor/investing/cryptocurrency/what-is-blockchain>. Accessed: 22 April 2022.
- 9 Etherem [online]. CoinDesk; April 2022.
URL: <https://www.coindesk.com/price/ethereum>. Accessed 19 April 2022.

- 10 Kelly Liam J, Millman Rene, Graves Stephen. What is Ethereum 2.0? [online]. Decrypt; 24 March 2022.
URL: <https://decrypt.co/resources/what-is-ethereum-2-0>. Accessed 19 April 2022.
- 11 What are smart contracts on blockchain? [online]. IBM Corporation; 2022.
URL: <https://www.ibm.com/topics/smart-contracts>. Accessed 16 April 2022.
- 12 Solidity Explained - What is Solidity? [online]. Moralis Blog; 22 June 2021.
URL: <https://moralis.io/solidity-explained-what-is-solidity>. Accessed 2 April 2022.
- 13 Carlomagno Marcos. Understanding Solidity by example [online]. ITNext; 15 October 2021.
URL: <https://itnext.io/understanding-solidity-by-example-b2394f7cea8f>. Accessed 2 April 2022.
- 14 Potter John M. The problem with Solidity [online]. Crypto Circus Top; 13 May 2018.
URL: <https://medium.com/crypto-security-top/the-problem-with-solidity-be7e6c277a58>. Accessed 2 April 2022.
- 15 About JavaScript [online]. MDN web docs; April 2022.
URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript/About_JavaScript. Accessed 10 April 2022.
- 16 What is JSON? [online]. W3schools.
URL: https://www.w3schools.com/whatis/whatis_json.asp. Accessed 10 April 2022.
- 17 Sheromova Vasilisa. Everything you need to know about JavaScript [online]. Exyte; 26 November 2020.
URL: <https://exyte.com/blog/everything-you-need-to-know-about-javascript>. Accessed 10 April 2022.
- 18 What is npm? [online]. W3schools.
URL: https://www.w3schools.com/whatis/whatis_npm.asp. Accessed 10 April 2022.

- 19 The advantages and disadvantages of JavaScript [online].
FreeCodeCamp; 5 December 2019.
URL: <https://www.freecodecamp.org/news/the-advantages-and-disadvantages-of-javascript/>. Accessed 10 April 2022.
- 20 Getting Started [online]. React; 2022.
URL: <https://reactjs.org/docs/getting-started.html>. Accessed 13 April 2022.
- 21 Arancio Stephen. ReactJS: A brief history [online]. Medium; 5 August 2021.
URL: <https://medium.com/@sjarancio/reactjs-a-brief-history-3c1e969a477f>. Accessed 13 April 2022.
- 22 Banks Alex, Porcello Eve. Learning React [e-book]. Sebastopol, USA: O'Reilly Media; 2020.
URL: <https://learning.oreilly.com/library/view/learning-react-2nd/9781492051718/ch01.html#reacts-past-and-future>. Accessed 13 April 2022.
- 23 Introduction to the DOM [online]. MDN web docs; 18 February 2022.
URL: https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Introduction. Accessed 13 April 2022.
- 24 Framework main features [online]. MDN web docs; April 2022.
URL: https://developer.mozilla.org/en-US/docs/Learn/Tools_and_testing/Client-side_JavaScript_frameworks/Main_features. Accessed 13 April 2022.
- 25 What is React? [online]. W3schools.
URL: https://www.w3schools.com/whatis/whatis_react.asp. Accessed 13 April 2022.
- 26 Manning Jammes, Schiefer Roman, Farwell Corey, Chereches Ovidiu. Props vs state [online]. Github; 6 July 2017.
URL: <https://github.com/uberVU/react-guide/blob/master/props-vs-state.md>. Accessed 13 April 2022.