Tampere University

Luong Dang Hai

THE ETHEREUM BLOCKCHAIN: USE CASES FOR SOCIAL FINANCE APPLICATIONS

Faculty of Information Technology and Communication Sciences
Bachelor of Science Thesis
April 2019

ABSTRACT

Luong Dang Hai: The Ethereum blockchain: Use cases for social finance applications
Bachelor of Science Thesis
Tampere University
International Degree of Science and Engineering (B.Sc)
April 2019

Centralized network solution have been around for a long time, despite having a considerable issue of trust, in which users need to rely on the implementation of the system. During unfortunate incidents such as centralized server hacking attacks, users' data can be stolen and distorted, as well as not available while requested. Blockchain is discovered and believed to be a distributed network solution which can mitigate the above issue.

This bachelor's thesis studies how blockchain network can be integrated into a social financial mobile application. The research is completed by developing a smart contract and connect it with the mobile application. The smart contract is written in the Solidity programming language and run on the Ethereum network.

Keywords: blockchain, smart contract, solidity, ethereum, finance

PREFACE

I would like to thank my employer Bankify Oy for suggesting this bachelor thesis and making this thesis possible. I want to also thank to my supervisor, Dr. Marko Helenius and my English teacher, Mrs Sarina Lewis for the aid I was given to improve this thesis and making it possible.

In Tampere, Finland, 1st April 2019

Luong Dang Hai

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF PROGRAMS AND ALGORITHMS

LIST OF SYMBOLS AND ABBREVIATIONS

Θ	Asymtotic Notation Theta
API	Application Programming Interface
Blinky	A finance mobile application owned by the Bankify company which helps users to split shared costs and develop saving target.
Cost-Split	The term used in the Blinky application to define the shared cost between a user group (buying food, items together)
ETH	Ether, the currency of the Ethereum blockchain
JSON	Javascript Object Notation
JWT	JSON Web Token
Pony-fill	A term used in the Javascript language which means providing the missing functionality in the standard library of the host run-time environment.
Programmable Blockchain	Blockchains which can run smart contracts, developed by one or more programming languages.
REST	Representational State Transfer
TX	Transaction
URL	Uniform Resource Locator
World computer	An alternative name for the Ethereum network, denoting its feature to run computer programs in the global scale.

1 INTRODUCTION

Developing Internet-based applications, which send and receive data over the Internet, can be developed by making a client which users will interact, and a server to manage the application's data (for example, facts, scientific knowledge, personal information and even legal proof). Any organizations which need to serve data online are required to set up this client and server system. Hence, there is a huge trust being placed onto the systems which will be responsible for making sure the data is available and reliable. Most of the time, those systems do not fail everyone's trust, until there are unexpected, either objective or subjective incidents, such as unauthorized access by a hacking group, or by malicious people trying to gain their advantages from the data. Such examples can be found even on big corporations: Facebook and Google failed to protect their customer's data [15], [39].

Finance has played an important role to the human society since the early ages. After the Internet was invented, finance, among other industries, was given a huge enhancement on how the services can be delivered to customers. There are many companies which are offering such financial services such as PayPal [18], Visa [19], etc. Those services also rely heavily on the setup stated above, which generates a risk when the trusted system fail to protect customer's data. Those who can bypass the system's security can compromise a large amount of money from the customer[31]. Therefore, there is a need for a better system that will protect data from the mentioned issue. Luckily, one such solution was discovered which is called "blockchain". Blockchain leverages the power of the majority where everyone can have a chance of governing the data over the Internet, which makes it extremely hard to execute a malicious attack. Blockchain also embraces the good sides of backing up data since everyone can, and is responsible for keeping one copy of the data.

By learning about the concept of blockchain, how it can achieve to solve the above mentioned problems, the research problem of this thesis is to develop a solution based on blockchain for a social finance mobile application, featuring a new way for users to split the cost between friends and handle the payment without directly interact with the payment methods. The objective is to show what is the solution, how the solution can be developed and then integrated to the system of the mentioned mobile application. After that, the solution will be evaluated objectively with key requirements of a financial application.

Chapter 3 presents the brief theoretical definition of blockchain, the advantages of using

blockchain, a special feature of it called smart contracts and, then introduces a blockchain network that will be used to develop and deploy the smart contract to integrate to the mobile application system mentioned in later chapters. Chapter 4 will describe the necessary concepts while developing a smart contract, which involves explaining the special features of smart contract's development programming language called Solidity. Chapter 5 describes the current Blinky application system and the requirement needed for the new solution with blockchain. After that, the description about implementing the solution is explained in Chapter 6, and then being evaluated in the Chapter 7. Chapter 8 concludes the results presented in Chapter 6 and 7, and then propose the usefulness and possibility to bring the solution to mass-usage.

2 RELATED WORK

Even though blockchain is a new technology, its benefits have been discovered and proved in many industries, such as health care [11], supply chains management [35], insurance [42], the Internet of Things [33] and especially finance industry with banking [14]. Thus, blockchain has become a compelling topic for new theses. For example, Jutila *et al.* suggested blockchain's advantage for financial services and provided applications in the conceptual level [20]. However, there was not a specific application implementation for an example financial service. Therefore, this thesis is intended to provide such information: implementing a smart contract for a social financial application, which records the event happened in a social cost split event.

Blockchain's main structure is a permanent database and collects new records continuously. Victoria *et al.* also based on this idea and explored using this new technology as a way to reliably manage important records [22]. The author presented an example of maintaining a land registry system on a blockchain network and evaluated using various standards on preserving digital records. Even though there are some limitations on using blockchain as a long-term data store, this new technology is suitable for a short-term need provided that the system is properly secured and managed. However, the research only stops at a conceptual level, and the need for reliably keeping digital data can also benefit other use cases, for example, between friends and families as a small history log. This thesis got the motivation from the mentioned paper to execute a practical solution with a financial mobile application.

The support for developing smart contracts on blockchain is not fully accessible at the time of writing. The blockchain systems do provide their frameworks, programming languages, semantics, etc. However, there are not many comprehensive and beginner-friendly tutorials to get started with developing applications on the blockchain. One of the accessible and beginner-friendly guides provides a tutorial where the reader can develop a smart contract as a back-end service for a web application [9]. The book chose Ethereum blockchain as the platform to make the tutorial, as the platform is the second-largest blockchain network while supports a Turing-complete programming language to produce full-featured computer programs. The book took a noticeable effort in explaining the concepts and required steps to maintain a smart contract service without relying on the skill level of the reader in terms of front-end development. This thesis aims to supplement a practical example of applying the guide on the book and make a small smart contract acting as a database server.

3 BACKGROUND

This chapter explains the key concepts of a blockchain, a new blockchain feature called smart contract and introduces a platform that provides the capability to develop one which is the Ethereum blockchain. Understanding the key ideas is the prerequisite to understand: i) the revolutional idea of blockchain; ii) the solutions proposed in later chapters.

3.1 Blockchain Definitions

A blockchain is a structure to store data in a continuously growing list of records, which are also referred to as "blocks" [32]. These "blocks" are linked and secured by cryptography. The whole chain containing the "blocks" is protected and maintained by a network of computers, which is also called a `node network`, through a consensus mechanism.

A blockchain can be compared to a self-organizing book which can add pages to itself infinitely. Each block is similar to a page, which contains a page number to identify itself. All pages are linked to each other by the book spine, and each page number implies that the next page is certainly the current page number plus one. With this setup, the reader can easily jump to the page as needed, and navigate between pages. A blockchain is constructed in a similar way as that book while the page numbers are replaced by unique identifiers such as a hash value, the book spine is removed and instead each page contains the identifier for the previous page [12].

3.1.1 Content creation on the blockchain

Content addition to the blockchain is similar to new content addition to a book. For example, anyone can get a digital version of a thesis, write edit and annotate. There has to be a solution to validate which printed out versions of that thesis is the original one. One such solution is comparing all copies of the same book. If the same page is showing different content between versions, the majority which hold the same content will be correct. To the blockchain context, everyone will have the same copy of data and have the right to add new content to it. Each time content is added there will be a broadcast event to everyone, each of whom can verify the data and after the successful verification, the next set of data will be added to a new block [24].

Blockchains improve the content creation by allowing more complex content which can be added into itself. These contents can be transactions data, messages, texts, conditions and even computer programs, or scripts which run when the conditions are met.

3.1.2 Blockchain's capabilities

From the definition and the mechanism of content creation mentioned above, it is derived that a blockchain is able to:

- Transfer values, such as electronic cash [24].
- Exchange message to each others as a form of signed message (to which the receiver is able to verify the sender) [5]
- Store records of data such as company shares, bonds, ... [34]
- Run optional scripts each time a condition is met [3, s. 20]

For the purpose of this thesis, finding use cases for a social finance application, only the data storing and script executing will be discussed. This is due to the nature of a computer application: i) Running business logic, and ii) Storing users' data.

3.1.3 Transparency and democracy

These capabilities mentioned in Sub-Chapter 3.1.2 can be found in current blockchains. For example, the records of company shares are saved in that company, legal authority. Those middlemans make sure that all shareholders's stakes are recorded correctly. Each time one needs to transfer the shares, all those middlemans need to be notified and then they will verify and execute the transfer. Those outlined steps might be broken if a chain is corrupted or a human error occured.

A blockchain prevents these issues with the reliability of data existence, automatic agreement bindings and no authority in control. All participants in a blockchain will keep a record of that company's shares and share transfer will be handled by a smart contract automatically [4]. Anyone with access to the Internet can safely make an agreement to each other without actually meet or know beforehand, since the rules and execution of a smart contract are automated. Each time new data is requested to record, the whole network will verify it with cryptography. Later, the data will be immutable, extremely hard to temper with [24][5]. Furthermore, no one owns the system. Instead, it is maintained, verified and controlled by everyone who joins the network.

3.2 Smart contracts

Running smart contracts is a new feature of a blockchain [41], is a "computerized transaction protocol that executes the terms of a contract" [37]. As mentioned in Sub-Chapter 3.1.1, any request to run the smart contract codes will be sent with a set of conditions. When the predefined conditions are met, the codes from the smart contract will be executed [3].

3.2.1 Comparison between smart and traditional contracts

In a traditional contract, each party will hold a copy. The judges and the legal system will enforce and make sure that the contract terms will be respected and followed. Manual work in traditional legal systems are not required in smart contracts. Since they are computer programs, all the execution rules are automated and enforced by computer logic.

A whole new possibility has been able to be realized since more complex rules and logic can be applied thanks to smart contracts. Running smart contracts on blockchain can also be understood as running computer programs in a world computer, since the contract will be executed anywhere in the world, which ensures the availability of the contract (zero downtime). Furthermore, since the contract execution will be made by every node joining the network, "an extreme level of fault tolerant" can be achieved [7].

With a smart contract, two people on the blockchain, for example Anna and Bob can agree to certain terms and conditions without actually meet and know each other before hand. They do not need to spend time and money from different third party entities to formalize and legalize the contract. Instead, what is required from them is only running the smart contract, and all the rules will be automated according to the rules defined in the contract itself.

3.2.2 Supported platforms for smart contracts

Since running smart contracts is a new feature in blockchain, not all blockchain networks have the full support for this feature, including the *bitcoin* network [23]. However, there are 3 most developed blockchain networks that support this new feature: NEO [30], EOS [25] and Ethereum [26].

Table 3.1 showed 3 most popular smart-contract-supported blockchains with some past metrics [10][8][36]. EOS blockchain has the largest recent developer commits, however, its maturity is merely 1.5 years comparing to Ethereum's 5 years. That leads to lots of Projects have been built on Ethereum, which confirms its reliability to work with as a smart contract platform. With the mentioned analysis, it is clear that Ethereum will be chosen

as the platform for developing a smart contract in this thesis.

Criteria	NEO	EOS	Ethereum
Maturity	3.5 years	1.5 years	5 years
Developer Activity past 12 months	155 commits	4795 commits	973 commits
Number of Projects	<50	93	2133
Supported Programming Languages	C#, VB.Net F#Java, Kotlin, Python	C++	Solidity

Table 3.1. *Popular Smart contract-enabled platforms and comparisons.*

3.3 The Ethereum Blockchain

After the successful release of the first blockchain platform called *bitcoin*, the financial industry has shown gradual increase in the adoption of blockchain technology [16]. However, the problem that *bitcoin* solved was only sending electronic money between peers, and the community demands a next generation of blockchain that can achieve more, for example, running computer programs with consensus-backed security. That was the reason for the born of Ethereum blockchain to become a programmable blockchain in the year 2014 [6]. Ethereum took a different approach to blockchain by allowing everyone to add their own operations and complexity rather than just a money transaction [3]. Therefore, Ethereum is suitable to realizing complex business logic [38].

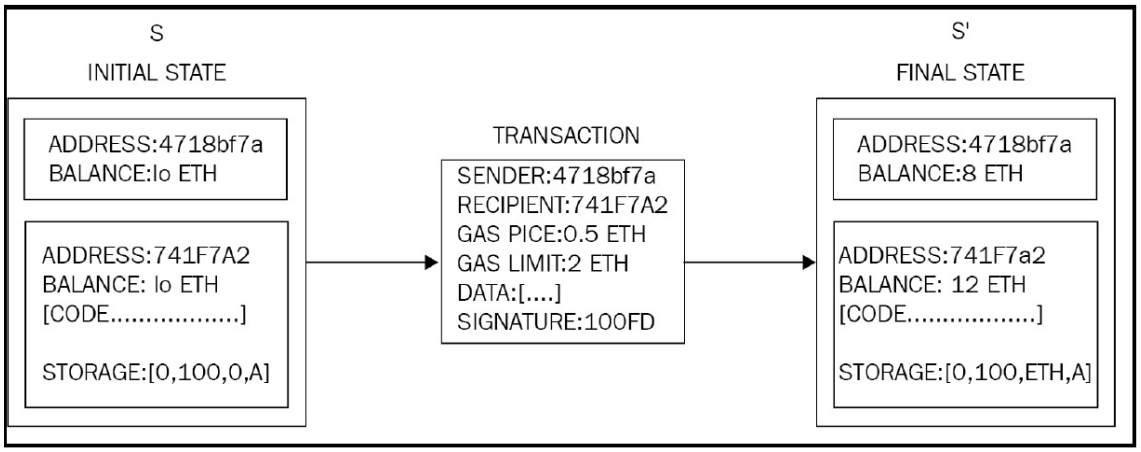

Figure 3.1. *Ethereum Transaction [3]*

The design of Ethereum is similar to a state machine [38][3][40]. Every time one wants to change the data on Ethereum, a transaction is created. The transactions will be collected and processed incrementally, each of which will describe how the data will transform from the current state to the next one, as in Figure 3.1

There are currently two Ethereum networks running: Ethereum Classic (ETC) [29] and Ethereum (ETH) [26]. In this thesis, the Ethereum ETH will be used to discuss and

implement the smart contract due to the speculation that the later Ethereum is more popular, advanced and actively maintained. The currency from Ethereum will be refered as "Ether" after this.

3.3.1 Smart contract execution on Ethereum

In Ethereum, each smart contract has its own address after being released to the network. To run one, an user with another Ethereum address will need to initiate a transaction to the contract, which optionally contains the amount of Ether to transfer, the message to invoke some function on the smart contract, and amount of fee (GAS) to be paid for the miners. The transaction will be verified by the network which consumes GAS gradually and if the transaction is invalid, the remaining fee will be refunded to the sender. Otherwise, the amount of Ether specified will be transfer and the code invocation will be activated on the transaction, which will run a specified function on the smart contract.

4 SMART CONTRACT DEVELOPMENT

Smart contracts are simply programs which run on top of a blockchain network. Hence, they can be developed similarly as normal computer programs, by using programming languages to compile the logic of the software, which then after that being compiled to machine instruction in order to execute. In the Ethereum blockchain, Solidity is introduced to be one of the programming languages which have the support to compile smart contract code [28].

Solidity is a modern language inspired by many predecessors such as C++, Python and JavaScript [28]. It is built with the support for Ethereum Virtual Machine (EVM), which is the platform to execute smart contracts, hence becomes a contract-oriented programming language [28]. Together with any other modern counterparts, Solidity features inheritance, statically typed and libraries to ease the development process. To understand later code examples shown in later chapters, the language's features which are contracts, function modifiers, addresses will be described in details

4.1 Contracts

Contracts can be known as the soul of Solidity, and a contract's design reassembles to that of classes in other object-oriented programming languages, with the appearance of state variables equivalent to class properties, functions to class methods. For example, Program 4.1 shows a basic contract.

Program 4.1 shows a contract `IntegerStorage` which contains a state variable of type `uint`. This contract also provides two functions, first of which is `getInt()` which returns the value of the state variable `storedInt`, and the second is `setInt(uint newInt)` which receives a parameter of type `uint` and will set that parameter's value into the state variable.

```
1  pragma solidity ^0.4.0;
2
3  contract IntegerStorage {
4      uint storedInt; // This is the state variable
5
6      function getInt() public returns uint {
7          return storedInt;
8      }
9
10     function setInt(uint newInt) public {
11         storedInt = newInt
12     }
13 }
```

Listing 4.1. *A simple contract in Solidity.*

4.2 Addresses

Address is a value type in the Solidity language to denote the Ethereum address, each of which contains a 20-byte value equalling to the size of an Ethereum address [28]. The address can be from a real person holding an Ethereum wallet, or another contract, since contracts also own an Ethereum address. This type `address` includes some members, which is identical to methods and property of an object in other object-oriented programming languages, such as `balance` and `transfer`. By having this special value type, it is possible to programmatically send and receive money inside an Ethereum smart contract.

4.3 Function modifiers

Functions in Solidity contracts can have modifier to change the behavior of its own [28]. With function modifiers, methods can be protected by checking a precondition before executing. Program 4.2 shows a contract which includes a simple function modifier.

In Program 4.2, the contract `owned` is assigned an Ethereum address coming from the caller of the constructor function, which is included in `msg` variable, as the owner when being created. During the lifetime of the contract, the function `close()` can only be effectively executed if the caller's address is equal to the owner's address.

4.4 Events

In order to get the best out of smart contracts, there has to be a method to communicate with them from the applications, website, etc. Thus, events are designed as a contract feature in Solidity. Each event contains the data denoted from the time which its contract is created. When being called, the transaction's logs will store the event. Program 4.3 and 4.4 demonstrates a simple interaction from a web application with a smart contract.

```
pragma solidity ^0.4.0;

contract OwnedWithModifier {
    function OwnedWithModifier() public { owner = msg.sender; }
    address owner;

    // This contract defines a modifier and it will
    // use it in function close()
    // The function body is inserted where the special symbol
    // '_' in the definition of a modifier appears.
    // This function modifier ensures that only the
    // owner of this smart contract can execute the close()
    // function and in other cases, an exception will
    // be thrown
    modifier onlyOwner {
        require(
            msg.sender == owner,
            "You need to be the owner to call this function"
        );
        _;
    }

    // This function can only execute if the owner
    // is calling it
    function close() public onlyOwner {
        // Do something inside this function
    }
}
```

Listing 4.2. *Simple function modifier in a contract [28].*

```
pragma solidity ^0.4.0;

contract DepositOrderReceiver {
    event DepositOrderReceiver(
        address indexed _from,
        bytes32 indexed _id,
        uint _value
    );

    function deposit(bytes32 _id) public payable {
        // Events are emitted using ‘emit‘, followed by
        // the name of the event and the arguments
        // (if any) in parentheses.  Any such invocation
        // (even deeply nested) can be detected from
        // the JavaScript API by filtering for ‘Deposit‘.
        emit Deposit(msg.sender, _id, msg.value);
    }
}
```

Listing 4.3. *Contract with Event derived from [28].*

```
var abi = /* abi as generated by the compiler */;
var DepositOrderReceiver = web3.eth.contract(abi);
var depositReceipt = DepositOrderReceiver.at("0x1234...ab67" /* address */)
    ;

// Pass a callback to start watching immediately
var event = depositReceipt.Deposit(function(error, result) {
    if (!error)
        console.log(result);
});
```

Listing 4.4. *Listening to Event from web applications [28].*

In Program 4.3, contract `DepositOrderReceiver` contains a function `deposit` which emit event `Deposit` when successfully executed. The event contains the Ethereum address of the sender, the identifier `_id` and the amount of deposit `value`. After being compiled and deployed to the Ethereum network, an identifier called as `abi` is generated, which can be used from outside of the Ethereum network to specify the contract. In a webpage which embed the JavaScript program 4.4, the `web3` variable is to denote the web3 web framework, which is used to interact with Ethereum network, which then specify to interact with the contract `DepositOrderReceiver`. The reference to the event `Deposit` is stored in `event` variable and then a callback function is passed into which will be invoked everytime the event is emitted.

5 APPLICATION DESIGN AND REQUIREMENTS

This chapter will describe the current system of the Blinky mobile application and important technical terms. These include: i) System overview; ii) the use cases of the application; iii) REST API; iv) the upcoming new feature requirements and v) asymptotic notation. Figures are also included to provide a visualized approach.

5.1 System overview

The Blinky mobile application is a cross-platform client written in React Native [17], a framework to develop mobile applications using the JavaScript programming language. The usage of the application enables users to register, create a Cost-Split event that happened after he/she paid for one event for everyone and need to split the cost equally and get the part of the money which others owe.

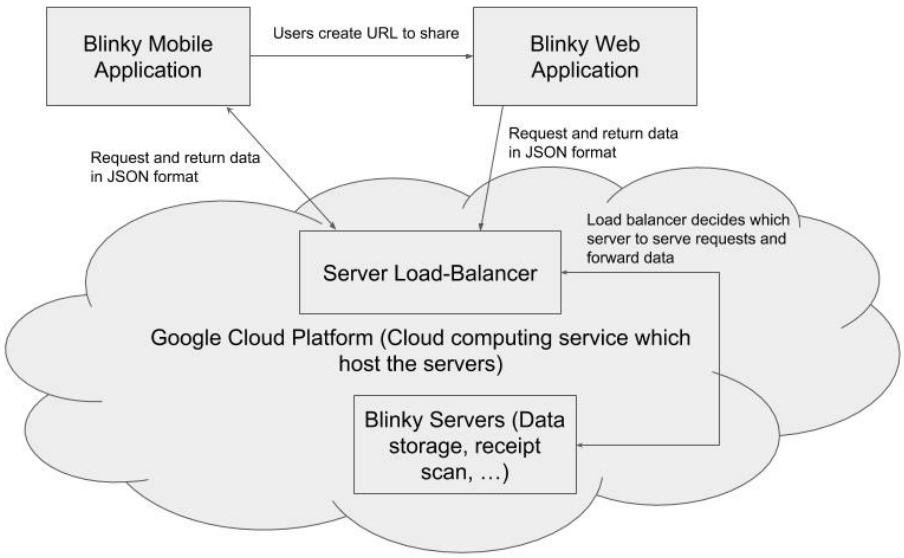

Figure 5.1. *Blinky system architecture*

Users of the Blinky Application can also generate a web URL to share that split cost to friends or relatives. The URL acts as an interface that non-Blinky users can also interact with the services even though they do not have an account. The URL leads to a web application written in React library, which helps building user interface. To provide

visualized aid, Figure 5.1 represents the system powering Blinky and where those system are deployed to.

5.2 Application use cases

The Blinky application's main goal was to help its users to calculate the shares between friends after someone has paid beforehand, to ensure a fair and equal split. The application also tries to free users from sticking to one particular payment solution, which allows users to register their preferred payment solution details and later they can share with the calculation details to their friends. After that, users are expected to arrange the payments and communication in their own preferred channel. From here until the end of this thesis, each of this series of actions from user will be referred as a "Cost-Split" event, as described in the glossary.

5.3 REST API

The clients, which include a mobile and a web version, can communicate with the back-end service with a Representational State Transfer (REST) Application Programming Interface (API) [13]. Data is represented and sent through Javascript Object Notation (JSON).

Users in the mobile application will be granted access right determined by a JSON Web Token (JWT) which can be used in the header of every request to the back-end service. The token contains the user's information which can easily be used to identify and authenticate them.

The Blinky RestAPI Service will provide an interface for the mobile application so that users can create, edit and delete Cost-Splits, and then invite participants, add expenses to split. Each time a Cost-Split, participant or expense is created on the server, an unique identifier will be generated as an integer and attached to that object. The identifier is then also be used to refer to that object in later actions of its lifetime as well as to delete.

5.4 Requirements

The requirements for integration with the Ethereum smart contract include:

- Generate a new data slot on the smart contract state when a new Cost-Split is created on the application
- Increment the data slot with new data each time a Cost-Split is edited with a new user's action such as changing expenses' detail, participants in the Cost-Split, generate an URL, or someone interacted with that URL

The requirements are chosen because they depict the common scenarios of transparency between small Cost-Split of a party, such as how much each has to pay, who did declare to have paid.

5.4.1 Generating data slot on smart contract

The desired smart contract will hold a map which contains each Cost-Split's happened event. Each Cost-Split will have its own memory space to store events in a chronological order. Whenever a new event is dispatched on the client mobile application, a function inside the smart contract should be called provided with the content of the event such as the description, time stamp, user's identifier. The function will then process the provided data and append it into that Cost-Split's memory space.

5.4.2 Provide a data-retrieval interface for user

Users of the Blinky application also have the right to request their data in a readable manner. This means that the application will need to provide a function that will represent user and request the data about the user-desired Cost-Split saved on the smart contract. The requested data should also be formatted into a readable form that they can also use as a mean of proof for the action taken by the Cost-Split owner or anyone interacted with the shared Cost-Split.

5.5 Asymptotic Notation

In order to measure the performance of an algorithm or a computer operation, asymptotic notation is used [2]. In general terms, it is a theoretical framework to measure an algorithm's operation taken to complete the task given a number of inputs. For example, if there is `n` inputs and the algorithm needs to do `n` operations on each input, the total number of operations needed to complete the task is `n * n`, which means it will be exponentially proportional to the number of inputs. In asymptotic notation, only the highest order of operations is taken. From the last example, if the number of operations needed is `n * n + 2n` then the asymptotic notation is still `n * n` since it carries order of 2.

6 IMPLEMENTATION

This chapter will describe the proposed solution for implementing a smart contract in this thesis reflecting the requirements outlined in the previous chapter. This will include explaining: 1) The chosen data structure for the smart contract; 2) The necessary functions for the smart contract; 3) Installing the Web3 framework which is used to connect the smart contract to the Blinky application and 4) The solutions to handle potential error during the lifetime of the smart contract.

6.1 Smart contract data structure

In order to write and read Cost-Split data (as described in Section 5.4), the data structure of the smart contract will need to be defined. Figure 6.1 provides a visualized data structure plan. Since each Cost-Split should have its own space to store its event data, one such option for organizing Cost-Splits could be using a `mapping` [27].

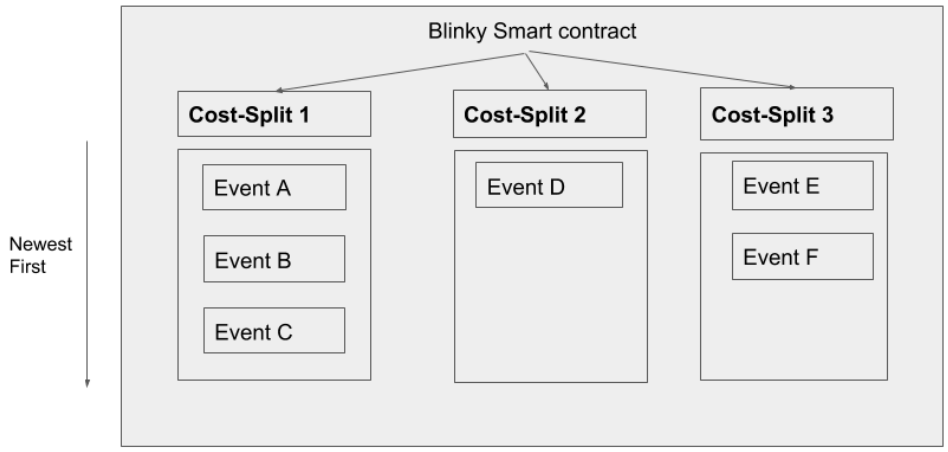

Figure 6.1. *Blinky's Cost-Split Smart contract data structure [3]*

Each Cost-Split will be identified from the key obtained from the Blinky API server, as described in Section 5.3. This will make Cost-Split fetching as quick and efficient because to find an element in a `mapping` data structure knowing the key (in this case, the Cost-

Split's identifier) only need an asymtotic performance of $\Theta(1)$.

Each event of a Cost-Split will need to carry a description, timestamp and the identifier of the person who initiates the event. Hence, `struct` data type is the only candidate for defining the data structure of a Cost-Split event. This `struct` will contain a `string` type for the description and another `string` type for the time stamp in the ISO 8601 format [21] (such as `2018-11-12T19:26:47.858Z`) since this format is compatible with various programming languages. The `struct` will also contain an `uint` field for storing the user's identifier from Blinky API Service. After this passage, the `struct` will be refered to as the `CostSplitEvent` type.

Furthermore, after being able to access the data of a Cost-Split using its key, each event should be sorted chronologically. Therefore, the smart contract will need to store the Cost-Split data as an `Array` of `CostSplitEvent`. Since each event will be recorded chronically by nature, no sorting operation is necessary on the array.

6.2 Smart contract functions

Since the smart contract will need to be able to create a new data slot for a Cost-Split, a function is needed, which can be named as `createCostSplit`. This function should accept the identifier of the Cost-Split as a parameter and should return a `boolean` type to indicate if the creation is successful or not, either `true` or `false`.

```
1
2  contract BlinkyCostSplit {
3   ...
4   function appendEvent(
5      uint costSplitId,
6      uint userId,
7      string timestamp,
8      string eventDescription) returns (bool) {
9     // The function's implementation
10    return true;
11  }
12  ...
13 }
```

Listing 6.1. *Skeleton implementation of function* `appendEvent`.

After a Cost-Split is created, users will then began to interact with it in the Blinky Application, which will incur events. Therefore, a new function is needed to start to add new events into a Cost-Split. This function can be called `appendEvent`. In order to identify the Cost-Split and to write the necessary data field of the event as specified in Section 6.1, the function will need to receive the identifier of the Cost-Split, the identifier of the Blinky user who initialized the event, the event's time stamp and the event description as the parameter. This function should also denote if the action completes successfully or not so it should return a `boolean` type like `createCostSplit`. Figure 6.1 outlined the

function's signature as described above for the purpose of demonstration.

In a normal data management software, there should be functions for creating, reading, updating and deleting data. However, in this particular smart contract, data should not be updated or deleted, since the aim of the smart contract is to embrace the transparency, which means data should be as it is from the moment of being created. Users' will to update or delete Cost-Split's events should, therefore, be added to the event list as a new event with the description provided by the user.

6.3 Web3 Framework

After defining the data structure (mentioned in Section 6.1) and the functions needed for the smart contract (as mentioned in 6.2), knowing a tool for communicating with the smart contract from the outside Internet is necessary. The Web3 Framework exists in order to solve the needs stated above. The advantage of this framework that it is written in the Javascript programming language, which is also used in the React Native framework in the Blinky mobile application. Hence, this enables fast and easy integration of the framework to both the web page and the mobile application of Blinky. In order to install and configure the Web3 Framework, the following steps are required [1]:

- React Native projects's dependencies can be managed using the `npm` package manager. Therefore, the content of the Web3 Framework can be downloaded from `npm`
- Choose a suitable Framework version that is compatible with React Native. There are two newest versions: 1.0 and 0.20.x during this time of writing this thesis. Version 1.0's content is generated dynamically, which is not compatible with React Native due to the nature of a mobile application (the code inside the app will be static after being built). Therefore, version 0.22.x is chosen to be installed.
- Pony-fill some APIs of Javascript that React Native does not support, such as the `crypto` API to perform encryption and hashing. Those can be solved by installing package `node-libs-react-native`

After following the above mentioned steps and conducting other smaller tweaks as presented in the link above, The web3 framwework can be invoked in React Native as an example in Program 6.2. First, an instance of the web3 framework is instantiated, with the default provider bundled within the framework. Then the contract address contract address and it's Application Binary Interface (ABI) is specified in line 3, 4 and then passed to the initialization function that creates an instance of `web3.eth.contract` in line 34, which represents an Ethereum smart contract. After that, the program can start calling the functions on the smart contract by calling function `methods.myMethod` on the contract instance as presented in line 36 to 38.

```
1
2  \\ Create a new web3 instance
3  const web3 = new Web3(Web3.givenProvider);
4
5  \\ The ethereum address of the contract needs to be provided
6  \\ in order to locate and interact in the blockchain
7  const contractAddress = "0x1a5c29c94D03C4c8f7414564CBD57295d61e898f";
8
9  \\ This is the metadata of the contract provided after the
10 \\ compilation.
11 const contractAbi = [
12   {
13     "constant": false,
14     "inputs": [
15       {
16         "name": "x",
17         "type": "uint256"
18       }
19     ],
20     "name": "set",
21     "outputs": [],
22     "payable": false,
23     "stateMutability": "nonpayable",
24     "type": "function"
25   },
26   {
27     "constant": true,
28     "inputs": [],
29     "name": "get",
30     "outputs": [
31       {
32         "name": "",
33         "type": "uint256"
34       }
35     ],
36     "payable": false,
37     "stateMutability": "view",
38     "type": "function"
39   }
40 ];
41
42 \\ Connect to the contract by providing the metadata
43 const myContract = web3.eth.contract(contractAbi);
44
45 \\ Example of initiating a transaction request and handling
46 \\ success callback in .then() and failure callback in .catch()
47 myContract.methods.myMethod("set")
48   .send({from: '0xde0B295669a9FD93d5F28D9Ec85E40f4cb697BAe'})
49   .then(function(receipt){})
50   .catch(function (error) {});
```

Listing 6.2. *Calling Web3 functions and making transactions on the Ethereum blockchain, with explanations [28].*

6.4 Error handling

Error handling is one of the most important aspects to consider when developing a software product, to ensure that the software works as expected and maintains a high-quality user experience. In this case of implementing the smart contract for storing Cost-Split event, there are cases which may confuse the smart contract and therefore handling the errors while developing the smart contract is a vital step.

Errors that can happen are when the smart contract tries to add an event to a non-existing Cost-Split, or in other words, the function `appendEvent` is given an invalid Cost-Split identifier. In solidity, this function invocation will result in throwing an `Exception`, all the changes to smart contract during that function call will be reverted and the failure will be notified to the caller (in this case, the Blinky Application). However, this kind of error without being handled will degrade user experience since they may not be aware of how the Blinky Application handles the event sending. Thus, a proposed solution is stated below.

The smart contract's function will use `require` function as a way to verify the validity of the data (in this case, checking the Cost-Split's identifier if it exists in the smart contract's state). If the assertion does not satisfy, the function will throw an early `Exception` to tell that the identifier is not valid. The Blinky Application can catch this Exception and display an appropiate dialog to notify user. An example of this error handling is presented in Program 6.3. From the Program, it is noted that in Solidity, the default value for a variable is 0 if it has not been set to any value yet. Hence, it is possible to check for the existence of a Cost-Split in the smart contract's state by trying to access its value inside `mapping costSplitMap`, then compare it to 0. The error message "Cost Split does not exist in the smart contract" will be thrown if the assertion fails.

```
contract BlinkyCostSplit {

 struct CostSplitEvent {
    uint userId;
    string description;
    string timestamp
 }

 mapping(uint => CostSplitEvent) costSplitMap;

 function appendEvent(
    uint costSplitId,
    uint userId,
    string timestamp,
    string eventDescription) returns (bool) {
   // The function's implementation should be in here

    require(
        // In solidity, if a value in a mapping (hash table) does
        // not exist with the provided key, the query will return 0
        costSplitMap[costSplitId] != 0,
        "Cost Split does not exist in the smart contract"
    );
   // ...  Remaining implementation for this function
   return true;
 }
 ...
}
```

Listing 6.3. *Using* `require` *function to validate input data*

7 SMART CONTRACT EVALUATION

This chapter will concentrate on evaluating the smart contract's performance with the mentioned requirement in Chapter 5.4. The evaluation consists of security, privacy, latency and troubleshooting.

7.1 Security

Since smart contract is public to the whole blockchain network, there has to be an option to define a rule to execute a function inside a smart contract. Such an option exist in the Solidity language called *function modifier* as mentioned in Section 4.3 and a ready-made example in Program 4.2. This means that the smart contract needs to define a modifier that verifies that the function caller is exactly the smart contract owner. To identify who is the owner, the smart contract's constructor need to save the Ethereum address of the caller to one of its own property for later retrieval. This solution provided by Solidity is adequate to protect the smart contract's functions from being called by the unauthorized.

7.2 Privacy

Privacy was not the goal of Ethereum network since every block made in the past can be verified and looked up by the whole network. Therefore, storing the plain data coming from the Blinky application do not provide privacy for user. In order to achieve privacy, the data being sent to smart contract will need to be encrypted. However, a new issue comes up which is who will keep the encryption key. If user is responsible for holding the key, it will require a considerable amount of technical knowledge in order to send and retrieve encrypted data correctly, which will go against the aim of the Blinky application as it tries to provide user the most user-friendly solution. If Blinky application is responsible for holding the encryption key, the aim of decentralized blockchain is also lost since the trust is now given to the Blinky server that it will keep the encryption key secure.

7.3 Latency

The Ethereum network is currently running with the Proof of Work consensus mechanism [24], which can only process 15 transactions per second on the whole network, according to the time of writing this thesis. This leads to the unavoided high latency for every time the smart contract is invoked and data is sent. Therefore, interacting with smart contract at this time will not guarantee fast transaction time.

8 CONCLUSION

Smart contract can be utilized in social finance application such as the Blinky application which involves in creating and storing event data inside the application. In this work, the Solidity programming language was used to develop a smart contract which acts as a data store for events which happens when user's Cost-Splits incur a new action as mentioned in Sub-Chapter 5.4. The language's documentation is well-written and therefore the smart contract was implemented in a short time interval. Since all the transactions happen inside the Ethereum blockchain and with the necessary security action as mentioned in Sub-Chapter 7.1, it ensures that the data will maintain its integrity, which will be a basis for a potential legal proof in the future.

By implementing a smart contract and connecting it to the Blinky application, advantages and disadvantages of blockchain can be realized. It is shown that the data state inside the smart contract can be directly requested and appended without going through the Blinky server. This improves the availability and democracy when updating data. Furthermore, it is also shown that the data update to smart contract has to pass the conditions provided in the smart contract's code in order to be accepted. The conditions are verified in every nodes joining the Ethereum network as explained in Sub-Chapter 3.1.1. However, as mentioned in Chapter 7, every transaction or data appending request has to wait for the Ethereum network until it is verified and processed. This is a potential issue that affects user experience since waiting a long time for a small transaction is discouraged. In addition, data on the blockchain does not provide users with privacy (in other words, everyone can request and access data on the blockchain). This also requires users to understand and accept that their data on the blockchain will not be private, unless there are additional encryption methods are applied.

With the advantages and disadvantages of a blockchain-based application, users need to understand what they can take advantage with what drawbacks they have to accept. For some users, this is acceptable. Therefore, it is concluded that this feature of using smart contract in a Blinky's Cost Split should not be used by default. The application should guide user of the feature and its benefits as well as drawbacks.

REFERENCES

[1] Abe. *How To: React Native with Web3 + 0xjs + 0xConnect*. URL: https://medium.com/@generalpiston/how-to-react-native-w-web3-0x-js-0xconnect-39b3d6a4dca (visited on 01/04/2019).

[2] D. Baldwin, G. Scragg and I. Books24x7. *Algorithms and Data Structures : The Science of Computing*. English. 1st. Hingham: Charles River Media, 2004. ISBN: 9781584502500;1584502509;

[3] I. Bashir. *Mastering Blockchain*. English. 1st ed. GB: Packt Publishing, 2017. ISBN: 9781787125445. URL: https://ebookcentral.proquest.com/lib/tut/detail.action?docID=4826445.

[4] M. Benchoufi, R. Porcher and P. Ravaud. Blockchain protocols in clinical trials: Transparency and traceability of consent. English. *F1000Research* 6 (2017), 66. DOI: 10.12688/f1000research.10531.4. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5676196/.

[5] V. Buterin. *Ethereum: A next-generation smart contract and decentralised application platform*. URL: https://github.com/ethereum/wiki/wiki/White-Paper (visited on 01/04/2019).

[6] E. Community. *A Next Generation Blockchain*. URL: http://ethdocs.org/en/latest/introduction/what-is-ethereum.html#a-next-generation-blockchain (visited on 01/04/2019).

[7] E. community. *Ethereum Virtual Machine*. URL: http://ethdocs.org/en/latest/introduction/what-is-ethereum.html#ethereum-virtual-machine (visited on 01/04/2019).

[8] *Crypto Commits (Past 12 months)*. URL: https://cryptomiso.com/ (visited on 12/01/2018).

[9] C. Dannen. *Introducing Ethereum and Solidity*. English. DE: Apress, 2017. ISBN: 9781484225349;1484225341;

[10] S. of the ĐApps. *State of DApps*. URL: https://www.stateofthedapps.com (visited on 01/04/2019).

[11] S. Demarinis. US Health Care Companies Exploring Blockchain Technologies. *EXPLORE* 14.6 (2018), 400–401. ISSN: 1550-8307. DOI: https://doi.org/10.1016/j.explore.2018.10.006. URL: http://www.sciencedirect.com/science/article/pii/S1550830718304208.

[12] D. Drescher. *Blockchain Basics*. English. DE: Apress, 2017. ISBN: 1484226038. URL: http://www.books24x7.com/marc.asp?bookid=128141.

[13] R. T. Fielding. Architectural Styles and the Design of Network-based Software Architectures. PhD thesis. URL: https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm.

[14] Y. Guo and C. Liang. Blockchain application and outlook in the banking industry. *Financial Innovation* 2.1 (Dec. 2016), 24. ISSN: 2199-4730. DOI: 10.1186/s40854-016-0034-9. URL: https://doi.org/10.1186/s40854-016-0034-9.

[15] D. Heaven. *Massive Facebook data breach left 50 million accounts exposed*. URL: https://www.newscientist.com/article/2181099-massive-facebook-data-breach-left-50-million-accounts-exposed/ (visited on 01/01/2019).

[16] M. Iansiti and K. R. Lakhani. The Truth About Blockchain. *Harvard Business Review* (). URL: https://hbr.org/2017/01/the-truth-about-blockchain (visited on 01/03/2019).

[17] F. Inc. *React Native*. URL: https://facebook.github.io/react-native/ (visited on 01/04/2019).

[18] P. Inc. *Paypal Homepage*. URL: https://www.paypal.com/us/home (visited on 01/01/2019).

[19] V. Inc. *Visa Homepage*. URL: https://www.visa.com (visited on 01/01/2019).

[20] L. Jutila. The blockchain technology and its applications in the financial sector. MA thesis. 2017. URL: https://aaltodoc.aalto.fi/bitstream/handle/123456789/27209/bachelor_Jutila_Laura_2017.pdf.

[21] M. Kuhn. *A summary of the international standard date and time notation*. URL: https://www.cl.cam.ac.uk/~mgk25/iso-time.html (visited on 01/04/2019).

[22] V. L. Lemieux. Trusting records: is Blockchain technology the answer?: *Records Management Journal* 26.2 (2016), 110–139. DOI: 10.1108/RMJ-12-2015-0042. eprint: https://doi.org/10.1108/RMJ-12-2015-0042. URL: https://doi.org/10.1108/RMJ-12-2015-0042.

[23] M. Morgado. *RSK: Bitcoin smart contracts (EN)*. URL: https://medium.com/coinmonks/rsk-bitcoin-smart-contracts-en-5b474ce87cd6 (visited on 01/03/2019).

[24] S. Nakamoto. *Bitcoin: A peer-to-peer electronic cash system*. 2008. URL: https://bitcoin.org/bitcoin.pdf (visited on 01/04/2019).

[25] E. Organization. *Eos blockchain website*. URL: https://eos.io (visited on 01/04/2019).

[26] E. Organization. *Ethereum Website*. URL: https://ethereum.org (visited on 01/04/2019).

[27] E. Organization. *Mapping Type, Solidity*. URL: https://solidity.readthedocs.io/en/v0.4.25/types.html#mappings (visited on 01/04/2019).

[28] E. Organization. *Solidity Documentation*. URL: https://solidity.readthedocs.io/ (visited on 01/04/2019).

[29] E. C. Organization. *Ethereum Classic Website*. URL: https://ethereumclassic.org (visited on 01/04/2019).

[30] N. Organization. *Neo - An Open Network For Smart Economy*. URL: https://neo.org (visited on 01/04/2019).

[31] A. Peyton. *India's Cosmos Bank stars in $13.5m hacking drama*. URL: https://www.bankingtech.com/2018/08/indias-cosmos-bank-stars-in-13-5m-hacking-drama/ (visited on 01/01/2019).

[32] J. Rakheja. What is blockchain? The difference between public and private blockchain. English. *PCQuest* (Aug. 2018). Copyright - Copyright 2018 Cyber Media (India) Ltd., distributed by Contify.com; Last updated - 2018-08-31. URL: https://search.proquest.com/docview/2097567733?accountid=27303.

[33] A. Reyna, C. Martín, J. Chen, E. Soler and M. Díaz. On blockchain and its integration with IoT. Challenges and opportunities. *Future Generation Computer Systems* 88 (2018), 173–190. ISSN: 0167-739X. DOI: https://doi.org/10.1016/j.future.2018.05.046. URL: http://www.sciencedirect.com/science/article/pii/S0167739X17329205.

[34] R. Ryan and M. Donohue. Securities on Blockchain. English. *Business Lawyer* 73.1 (2017), 85–108. URL: https://libproxy.tuni.fi/login?url=https://search-proquest-com.libproxy.tut.fi/docview/2049988285?accountid=14242.

[35] S. Saberi, M. Kouhizadeh, J. Sarkis and L. Shen. Blockchain technology and its relationships to sustainable supply chain management. *International Journal of Production Research* 0.0 (2018), 1–19. DOI: 10.1080/00207543.2018.1533261. eprint: https://doi.org/10.1080/00207543.2018.1533261. URL: https://doi.org/10.1080/00207543.2018.1533261.

[36] N. Singh. *NEO dApps Ecosystem: Complete List of NEO Decentralized Blockchain Applications*. URL: https://101blockchains.com/neo-dapps-ecosystem/ (visited on 01/04/2019).

[37] N. Szabo. Smart Contracts. (1994). URL: http://www.fon.hum.uva.nl/rob/Courses/InformationInSpeech/CDROM/Literature/LOTwinterschool2006/szabo.best.vwh.net/smart.contracts.html.

[38] S. Tikhomirov. Ethereum: State of Knowledge and Research Perspectives. *Foundations and Practice of Security*. Ed. by A. Imine, J. M. Fernandez, J.-Y. Marion, L. Logrippo and J. Garcia-Alfaro. ID: 10.1007/978-3-319-75650-9$_14$. Cham: Springer International Publishing, 2018, 206–221. ISBN: 9783-319756509.

[39] P. Wagenseil. *Google to Kill Google Plus Due to Possible Data Breach*. URL: https://www.tomsguide.com/us/google-plus-data-leak-shutdown,news-28259.html (visited on 01/01/2019).

[40] G. Wood. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper* 151 (2014), 1–32. URL: https://gavwood.com/paper.pdf.

[41] J. Wu and N. Tran. Application of Blockchain Technology in Sustainable Energy Systems: An Overview. English. *Sustainability* 10.9 (2018), 3067. DOI: 10.3390/su10093067. URL: https://doaj.org/article/f9337e93a561416abec4fa0d8345efe7.

[42] L. Zhou, L. Wang and Y. Sun. MIStore: a Blockchain-Based Medical Insurance Storage System. *Journal of Medical Systems* 42.8 (July 2018), 149. ISSN: 1573-689X. DOI: 10.1007/s10916-018-0996-4. URL: https://doi.org/10.1007/s10916-018-0996-4.