Building A Secure Trading Application Using Blockchain

Student Name: Alistair Madden

Supervisor Name: Dr. Ioannis Ivrissimtzis

Submitted as part of the degree of MSci Natural Sciences to the

Board of Examiners in the School of Engineering and Computing Sciences, Durham University

Abstract —

Background: Blockchain allows for consensus between parties in a decentralised network without a trusted intermediary. It does this by maintaining a distributed ledger - an eventually consistent data structure - which all nodes in the network accept through a proof of work consensus protocol. However, it is thought that many of the features of the distributed ledger are also applicable to a centralised systems. In particular, the links between transactions and assurance that they cannot be altered may be exploited to provide more accountable transactions.

Aims: This work aims to analyse the Bitcoin protocol - an existing blockchain implementation - and motivate the use of the technology in a centralised context. Further, this work intends to implement a centralised blockchain application and evaluate the features and performance of such a system.

Method: The open source blockchain framework, Openchain, is used to build a trading application. This is compared against a SQL based approach to determine the benefits and limitations of blockchain technology in a centralised context. A performance evaluation of each implementation is made using Apache JMeter to simulate the load of several user sessions. A survey is carried out in relation to user saving habits and the user experience of the software.

Results: The StockChain trading application was successfully developed using two implementations: Openchain and SQL. In the simulation of user sessions, Openchain was found to be less performant than its equivalent SQL implementation as every transaction is digitally signed. Many of Openchain's features were not used for full functionality of the application. StockChain received a positive response from survey participants with the majority of users able to trade on the platform. Issues related to the user experience of the application are also highlighted.

Conclusions: Centralised blockchain technology does not in this case offer more than a conventional SQL based system could. The idea and implementation of the StockChain application are well received, however the added value in developing based on centralised blockchain remains to be realised.

Keywords — Blockchain, Centralised System, Finance, Bitcoin, Cryptocurrency, Distributed System

I INTRODUCTION

In recent years, it has been suggested that blockchain technology will revolutionise the financial services industry (EY 2016). Blockchain - which first gained widespread attention in its implementation of the Bitcoin cryptocurrency (Nakamoto 2008) - is a combination of a data structure and consensus protocol which can be used to maintain a distributed public ledger. It allows for agreement between nodes in a network directly between parties without a trusted third party.

Cryptocurrencies in themselves are not a new idea. The main problem to overcome is known as the double spending problem. It arises due to the ability to make copies of digital data. For instance, if the user of a digital cash system receives a unit of virtual currency, they can easily duplicate it and send it on to two different recipients. It is fairly easy to design a centralised scheme to solve this problem. A central authority keeps a record of digital assets and who owns them. Whenever a transaction is made, the recipient checks with the central authority to make sure the asset has not been spent. If the asset is unspent, the recipient notifies the central authority that they are accepting the transaction and the central authority updates their records accordingly. If the central authority informs the recipient that the asset has been previously spent, the recipient rejects the transaction.

Unlike physical cash, this system isn't anonymous. However, by using cryptography, it is possible to keep the system anonymous and discourage double spending (Chaum et al. 1990). DigiCash, formed in 1989 by David Chaum, was probably the first company set up exclusively to handle online payments. In this system, clients are guaranteed anonymity so long as they don't try to cheat the system through double spending. The trade off is merchants are known to the central authority. DigiCash ultimately failed to take off due to poor support for user to user transactions (the system was mainly envisaged for user to merchant transactions) and poor adoption by merchants. An important point to note about DigiCash is that double spending is not prevented, but only able to be detected (revealing the perpetrator's identity in the process).
Further attempts have been made to create a currency backed by something other than fiat currency. E-Gold, was a centralised digital currency backed by gold. The company held gold reserves and issued digital currency based on this value. Poor security, coupled with changes in government regulation eventually forced it to be shut down by the US government (*US v. E-Gold, Ltd.* 2008).
Two more informal proposals for digital currency, namely b-money proposed by Wei-Dai and Bit Gold by Nick Szabo were never implemented directly, but come closer to the solution proposed by Nakamoto in Bitcoin. In these systems, computational puzzles similar to Adam Back's Hashcash (Back et al. 2002) for deterring denial of service attacks, and a new data structure intended for timestamping digital documents (Haber & Stornetta 1990) are used to to mint currency instead of basing it on an existing resource. Importantly however, these systems rely on a trusted, centralised timestamping service where Bitcoin does not.

Bitcoin eliminates the need for a trusted third party by maintaining a distributed, eventually consistent ledger which all nodes in the network agree upon. The exact mechanism by which this is achieved is discussed in Section II. The ability of blockchain to achieve distributed consensus has implementations far beyond currency for which it was first devised. Businesses are looking into the possibility of using the underlying blockchain technology to enforce contracts, provide clearer audit information and resolve financial disputes. It is becoming increasingly clear that blockchain will be a transformative technology in any process with a transaction based workflow (Hyperledger 2017).

There are, however, serious problems to overcome. Little is known about the nature of networks based on blockchain. This is evidenced by the wild fluctuations in valuations of cryptocurrencies. It is still unclear how secure these networks are, which as we have seen, is paramount in success of any such system. To date, only preliminary studies have been undertaken to this end (Barber et al. 2012). Bitcoin also exposes any transaction made on its network to all nodes. This

may not be desired and in some cases, would be detrimental to business operation. Nodes do not need to be vetted to join the Bitcoin network which poses a problem for businesses who have an obligation to know their customers. This is especially important for financial regulation where responsibility for improper use of a system lies with the issuing company. There is also concern about the energy consumption costs. The energy required due to the proof of work mechanism used to secure transactions on the Bitcoin network is not inconsiderable and several estimates put its consumption in the order of a few hundred megawatts (Narayanan et al. 2016). Other mechanisms such as proof of stake have been proposed that attempt to overcome these issues (King & Nadal 2012). These systems have received even less research than proof of work systems, so suffer from many of the same problems arising from their unfamiliarity.

In this work, a centralised currency exchange system based on the Openchain implementation of blockchain technology is proposed. The system acts as a mutual fund, investing users money, buying shares on their behalf. In return, the user receives Units of the fund, which can be exchanged freely with other users. Should the system be put into production, legal issues around currency and specifically money transmitters would need to be contended with. A system based on an SQL database but with otherwise the same functionality will also be implemented for comparison.

To address security concerns, the latest techniques to combat cross-site scripting (XSS) - a script injection attack - and cross-site request forgery (XSRF/CSRF) - an attack that tricks users into submitting requests - will be employed. As it is paramount to secure user data, great efforts will be taken to secure user passwords in the login database. Further, the use of HTTPS to encrypt communications between client and server will be used throughout the app as much data is highly sensitive. It is hoped by implementing this system that the merits of the existing open blockchain framework, Openchain, will be highlighted and potential drawbacks discovered.

In Section II, an examination of the Bitcoin protocol is undertaken. A high level overview of the system is given and the anatomy of transactions and blocks is studied. Their relevance to the Bitcoin network established. The understanding gained from this is linked to centralised systems using published peer-reviewed papers as far as possible. Section III gives the system architecture, motivating design decisions and implementation details. In Section IV quantitative results from comparison of the two implementations are presented. Results from a user survey of the software implementation are also given and categorised. In Section V analysis of these results is carried out and conclusions are drawn. Finally, Section VI summarises the findings of this study.

II RELATED WORK

A The Bitcoin Protocol

In the seminal paper on Bitcoin, Nakamoto (2008) proposes a peer to peer currency based on digital signatures and a distributed timestamp server. The former allows transactions to be made anonymously, whilst the latter enables a global, eventually consistent ordering of transactions which effectively solves the double-spending problem. The main breakthrough is that no third party is required to verify transactions, instead, this responsibility is spread over all ”full nodes” in the network. Anyone can become a full node by downloading the entire history of transac-

tions which is neatly contained in a novel data structure often referred to as the blockchain. Full nodes can propose groups of new transactions, known as blocks, to add to the blockchain. To do this, a computational puzzle must be solved. As each of the blocks contains a reference to the most recent previously solved block in its solution, it is proof to other nodes that work has been expended to expand the chain. As long as all transactions within the block are valid, the proposed block is accepted by other nodes as the new head of the blockchain and referenced in their solutions to the computational puzzle.

To understand exactly why this mechanism prevents double spending, how transactions are granted anonymity, and how the system can benefit from centralisation, it is necessary to study the Bitcoin protocol in detail.

Nakamoto (2008) uses data structures enabled by hash pointers in the design of Bitcoin. A hash pointer is a pointer to where data is stored along with a cryptographic hash of the data value (Narayanan et al. 2016). If the data is changed at some later point in time, this is easily detected as the hash pointer's hash value is now invalid. This property can be used to make acyclic, pointer based data structures tamper resistant (Narayanan et al. 2016). A Bitcoin transaction is an example of such a data structure and its structure is given in Figure 1.

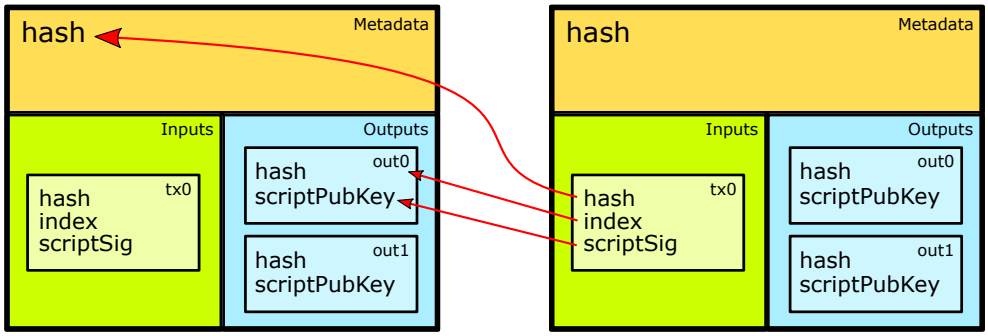

Figure 1: Two Bitcoin transactions with one input and two outputs. The red arrows show how the input of the right most (later) transaction are related to the outputs of the left most (earlier) transaction.

A transaction is comprised of three major parts: metadata, one or more output objects and one or more input objects.

- The metadata of a transaction comprises of the hash of the whole transaction as well as some other housekeeping information.
- Output objects contain the value to be output and a string representing a script written in the Bitcoin scripting language. This language can define some complicated conditions for output. It is Turing incomplete by design so that arbitrarily powerful computations can't be submitted. The simplest and most common of these scripts has the effect of being redeemable only by the owner of a public key specified by the script.
- Input objects contain the hash of a previous transaction, an output index to use from this transaction, and a digital signature which proves ownership of the public key referenced in the output script.

In the Bitcoin protocol, rather than using the whole public key in a transaction, a shorter hashed version of the public key known as an address can be used. Each public key can produce one address which is collision resistant due to the cryptographic hash used to derive it. To keep transactions anonymous, it is recommended to use a new address for each transaction. This necessitates a new key pair be created for each transaction. This complexity is hidden from the user by wallet programs which keep track of this address generation.

The transaction structure previously described has no inbuilt way to prevent double-spending. This is easily solved by introducing a third party such as a bank to check all transactions are valid. Different privacy concerns are raised with this system as the third party can track all of the transactions. This is solved by using a "blind signature" scheme (Chaum 1983) which allows some users of the system to be kept anonymous. This system still relies on a trusted third party and was primarily envisaged for transacting with merchants. As such only customer anonymity is guaranteed whilst merchants are known to the third party. Chaum et al. (1990) relaxed the conditions from preventing double-spending to detecting it, allowing for payments to made offline. If a user in this system double-spends, their identity is revealed by the bank with an overwhelmingly high probability and action can be taken.

Nakamoto (2008) proposes that proof-of-work could be used to implement a distributed timestamp server and solve the double spending problem without a third party.

A centralised timestamp server takes a cryptographic hash of a set of data with a previously published hash value and broadcasts the result (Haber & Stornetta 1990). This published timestamp can be independently verified by others by performing the same calculation. This proves the set of data must have existed at the time the broadcast was received as there is no other way the hash of the data could be found. Subsequent hash values (found by hashing some other data with this hash value) effectively render the previous hash values unalterable, as, thanks to the collision resistance property of the hash function used, no data can be changed without altering all subsequent hash values in the chain. This structure is shown in Figure 2.

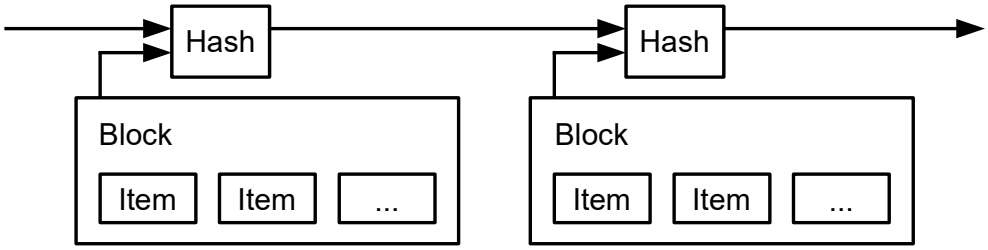

Figure 2: The structure of a centralised timestamp server (Nakamoto 2008).

To decentralise the timestamping, the cryptographic hash of the set of data is taken repeatedly, until a hash value with a predetermined number of leading zero bits is found. This, in essence, is the computational puzzle to be solved. In Bitcoin, blocks are the data structure containing transactions to be hashed. A nonce is included in the block and increased with each successive hash. This is so that each time a hash is computed, the output hash value changes and thus has a chance of beginning with the required number of zeros. If such a hash is found, the block is broadcast to all other nodes in the network. Others can verify the hash of the block begins with the required number of zero bits by performing the hash of the block themselves. They can also run the input scripts of each transaction contained in the block to verify the transactions are legitimate and no

double spending has occurred. Nodes express acceptance of the new block by using the hash of the block in any new block value proposed. This is shown in Figure 3.

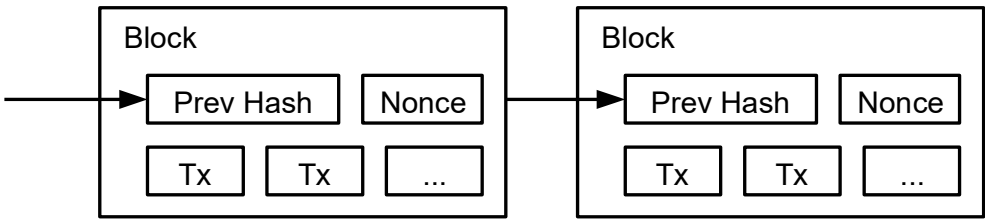

Figure 3: The block structure used in Bitcoin (Nakamoto 2008).

To verify that each transaction in a block is valid, each node maintains a database of unspent transactions. When a new block is received, each input is taken and its scriptSig is prepended to the scriptPubKey of the output it references. The script formed from the concatenation is run by the verifying node. In the vast majority of cases, the script forms checks that the signature is valid for the referenced output public key hash value. The output of the transaction is also checked against the database of unspent transactions. Double spending is prevented so long as a single node (or group of malicious nodes) does not exceed 51% of the total computational power in the network. Once all transactions in a block have been found to be valid, a node can choose to switch to referencing the block's hash its own proposed block. It is beneficial to do so, as transactions are only deemed valid if included in the longest chain of blocks. With each new block, the database of unspent transactions should also be updated to include new unspent outputs and invalidate outputs referenced by inputs to transactions in the block.

B Consequences of Blockchain and Alternate Applications

The use of public key infrastructure to identify a Bitcoin transaction's recipient affords a level of anonymity. This is the case so long as a new address is used for each transaction and no identifying information is tied to the user. However, work by Reid and Harrigan shows how public keys can be associated with each other and other information inferred due to the public nature of the Bitcoin network's history (Reid & Harrigan 2013). This has many implications for businesses and a centralised blockchain implementation. In such a system, the identity of customers is likely known, defeating any anonymity granted by blockchain between the customer and the business. Of greater concern is protecting customer identities from each other and stopping information from being inferred. There is no obligation to make the full blockchain public however, as transactions are verified by a central authority and not by a peer to peer network. In this more traditional use case, the advantage of using blockchain as a data store shifts towards auditability.

Much more can be achieved with the Bitcoin scripting language than transferring funds between parties. Smart contracts (Szabo 1997) can be implemented as transactions on the Bitcoin network. This enables enforceable contracts to be made with little trust requirements. This idea is extended further to enable contractual security and on-chain privacy (Kosba et al. 2016). In this system, transactions are private from those participating in the network and parties in the same contractual agreement are protected from each other.

Another system that expands on the Bitcoin system is Ethereum (Wood 2014). Ethereum is a platform for decentralised blockchain applications with a focus on smart contracts. It can be seen as a more general blockchain implementation, allowing a developer to easily define how their app should use a decentralised blockchain instead of trying to retrofit a solution to the Bitcoin transaction system.

Variations on the proof-of-work system have also been suggested. Proof-of-stake is a system whereby a node is allowed to add a number of blocks to the blockchain in proportion to the stake held in the system. It is thought this could improve the energy consumption of a distributed network. This is not a great concern in a centralised system however, as the central authority is not using blockchain to secure the network. Blockchain, in a centralised context is about making the history of the centralised system more auditable and transparent.

III SOLUTION

In this work, a system called StockChain is created to realise a mutual fund. At its core, StockChain is a managed portfolio of stocks. Users invest capital, and in return recieve a share of the fund proportional to their investment. This share is represented by Units of the StockChain system. StockChain reinvests users' investment in stocks, hoping to make a profit which is reflected in the price per Unit of the system.

Two functionally equivalent implementations were made, one with SQL, and one with the centralised blockchain framework Openchain. In order to minimise development and make the systems comparable, the back end of the StockChain system is decoupled from the front end as much as possible. In both implementations, the back end serves as a REST API and a web server. It is built with Node.js and the Express framework to handle HTTPS calls to the various API endpoints, as well as serving the front end AngularJS application.

The front end gives the user the functionality to check their balance (in both Units of the fund and GBP), purchase units of the fund, make transactions with other users and review their recent transactions. This functionality is maintained irrespective of the back end implementation. Figure 4 shows the overall architecture of the application.

It is worth noting that the system implemented here may be financially or legally flawed in some way. This project does not concern itself with these issues and focuses on using centralised blockchain technology to implement a realistic financial application.

A Back End SQL Implementation

To create a realistic benchmark for comparison, SQL is used as a centralised system of records. This type of implementation is completely proprietary and follows no well-defined standards other than good programming practices. A financial application requires regular auditing, so some sensible choices must be made about how the system functions. The structure of the database is shown in Figure 5.

The core of the database is the account_auth table. This stores a user's email address and hashed password. To secure the user's password against rainbow table attacks, an extra random piece of data called a salt is added to the password before it is hashed with the bcrypt hashing function (Provos & Mazieres 1999). The hashing process is repeated 10 times to make it more computationally expensive for an attacker to overcome.

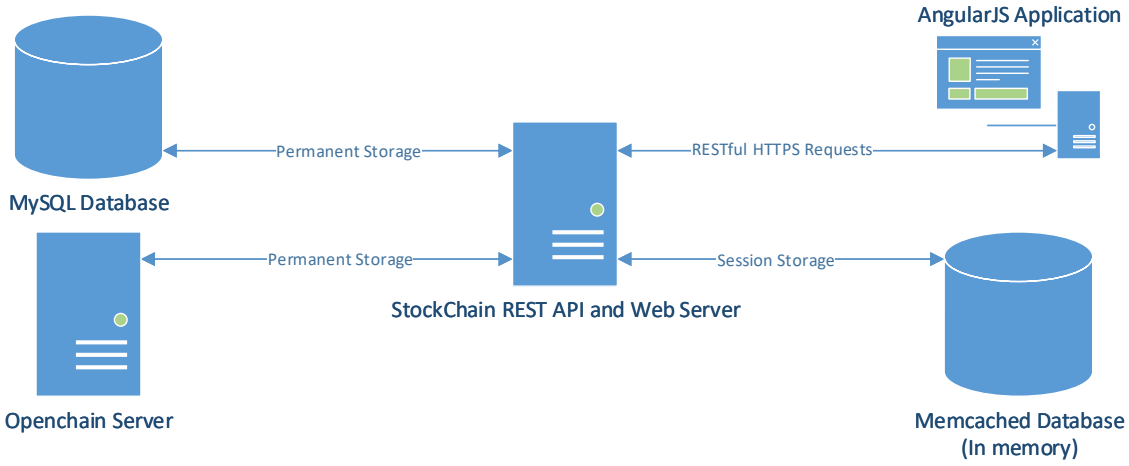

Figure 4: The architecture of the StockChain application.

Every user is given a unique integer account identification when they sign up which begins at 1. This is stored in the Account_ID field. The account identification 0 is reserved for the system. Once a user is signed up they are considered to own an account. They can sign in via the front end or REST API and check their account balance and recent transactions. For a completely new account, the balance is zero and there are no recent transactions to show.

Each time a user makes a transaction, it is stored in the account_transaction table. User account statements are calculated on the first of every month. The system uses SQL stored procedures to get each user's most recent statement and add to it transactions made in the previous month. This value is then inserted into the account_statement table.

So far, the system acts exactly as an online bank account would, however in the StockChain system, the users' capital is taken and reinvested in stocks. This is the basis of a mutual fund. This project does not concern itself with how the stock trades are made or recorded by the system itself any further than is required for operation of the client side front end.

When a user invests in the StockChain system, they are allocated Units of the fund in return. To record user investment (analogous to a bank account deposit), a credit is recorded for the user's Account_ID in the account_transaction table. This table is responsible for holding all user-user transactions as well as user-system investments. A debit for the same amount is made to the asset_cash_transaction table in accordance with double-entry bookkeeping. The asset_cash_transaction table is responsible for maintaining transactions that relate to the StockChain system itself.

To record a purchase of stock by the StockChain system, a system-system transaction is made. A credit for the value of stocks purchased is recorded in the asset_cash_transaction table. A debit is made to the asset_share_transaction table which records all the stocks held by the StockChain system and the price they were purchased at.

Like the account_transaction table, statements for the asset_share_transaction and asset_cash_transaction tables are produced. This is done on a daily basis at 16:40 after the last trades are settled on the London Stock Exchange. This enables the accurate calculation of StockChain's combined assets (stocks and capital) at this point.

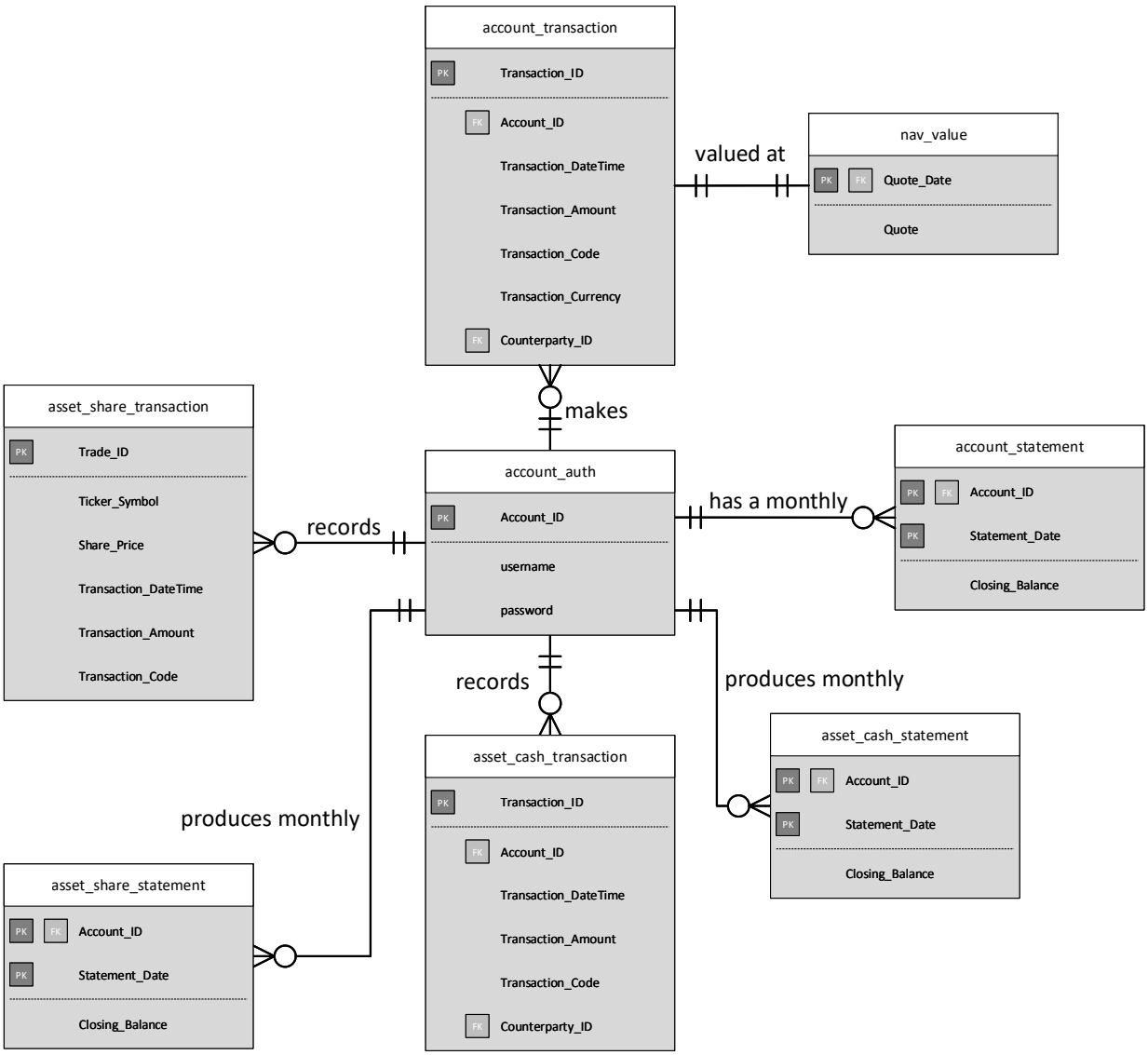

Figure 5: The structure of the Stockchain database.

The calculation of a user's balance is based on the number of Units of the system they own and the Net Asset Value (NAV) of the system. The NAV represents the price per Unit of the StockChain system. It is calculated by totaling the cash assets using the asset_cash_statement table, as well as the current value of any stock investments recorded in the asset_share_statement table and dividing by the total number of Units of the system issued to users. Initially, as the fund has issued no units, the NAV is pegged at 1. Whilst it may be preferable to update the system's NAV only occasionally so that the volatility of a user's balance remains low, mutual funds are legally required to update this value daily. In the StockChain system, the NAV is updated at 16:40 after statements of cash assets and stock assets have been produced.

To report the user's balance, the last account statement and transactions made since need to be converted into the number of Units they were worth at the time they were submitted. To do this, they are multiplied by the inverse of the appropriate NAV for the time the statement/transaction was made. If the transaction was made after 16:40, the NAV for the recorded date is used, oth-

erwise the previous day's NAV is taken. These Units are then totaled to give the user's current balance in Units. This is reconverted to GBP by multiplying by the current NAV.

In this SQL based implementation, the logic previously described is created in stored procedures on the MySQL server as opposed to in Node.js. There are two main reasons for this: Firstly, many of these operations require joins which can be performed much more efficiently on the MySQL server. Secondly, by using stored procedures, the Node.js package which connects to the database automatically sanitises variables. This defends against SQL injection attacks without additional programming.

An issue prevalent in both SQL and Openchain implementations is how to handle conversion between Units of the fund and GBP. This is a common problem in systems handling multiple currency accounts. In the SQL implementation, it is necessary to convert any values stored in GBP to Units when calculating account balances. Doing so incurs a round off error as the values computed from the NAV cannot be stored to arbitrary precision. There is no well defined standard for rounding in currency conversions. In this application, simple truncation of numbers is used.

B Back End Blockchain Implementation

A number of options were available when selecting a centralised blockchain framework to work with. Openchain was selected due to assurances that the technology works (Koroma 2016) and the availability of documentation for the software at the time of decision.

Openchain provides a Node.js package for interfacing with the Openchain server. This is reminiscent of a database management system, and on some level, blockchain can be viewed as a database designed expressly for transactions. Fundamentally, Openchain is a key value data store. It provides a specific syntax for keys and values so that they are meaningful to the framework. This allows Openchain to easily issue assets and report their balance for specific accounts without any additional work by the developer. Any additional logic beyond what Openchain provides must be written manually in the Node.js server however. For this reason, only a small portion of the SQL implementation is replaced by Openchain for this project.

In the StockChain application, Openchain replaces the account_transaction and account_statement tables. This means that user-user transactions and user-system investments are recorded using Openchain. Notably, the calculation of NAV remains the same as Section III A and is applied to the balance reported by Openchain on the Node.js server in order to determine user balances.

Openchain is set up in a closed-loop configuration allowing for stronger compliance with "Know Your Customer" (KYC) regulation. In Openchain, any change made to the ledger, including creation of user accounts and their permissions, must be made through a transaction. On creation of a user account, a private key is generated for the user and stored as an additional field in the account_auth table. The public key is derived from the private key and hashed to obtain an address. This is done in the same way as creating an address for a Bitcoin transaction would be. Unlike Bitcoin however, in the closed loop Openchain system an address is equivalent to an account as there is no need to generate new key pairs to preserve anonymity. An Openchain transaction is made that links the user's email address to their Openchain address. This enables units to be exchanged between users using email addresses instead of hard to read and remember Bitcoin addresses.

Using Openchain in this way is slightly different to the way it was designed. Instead of having the whole history of transactions available to any user on request, this functionality is restricted to administrators of the system. In this way, user details are kept private from each other, but not the central authority.

By default, this newly created account does not have the rights to send or receive funds. These permissions are granted via another Openchain transaction. After this, the account is fully set up and can send and receive Units to and from others.

All currency stored in the Openchain framework is recorded in Units of the StockChain fund instead of GBP. This makes it easier to deduce the value in GBP by applying the current NAV directly to the balance returned by Openchain. The same issues with round off errors encountered in the SQL implementation remain. Any transactions made in GBP need to be converted before being submitted to the Openchain system which necessitates truncation.

When Units of the StockChain fund are issued to a user account, an Openchain administrator account is used through the Node.js package to sign a transaction authorising the creation of new Units. This is only required when a user-system transaction is made as user-user transactions can be signed using the sending user's private key.

C Front End AngularJS Implementation

Many options were available for developing the front end of the system. Due to previous experience, AngularJS was chosen to develop the system as a web application. This gives maximum compatibility with a wide range of devices when compared to a native, platform specific application. Though web applications cannot access some of the low level features of devices, these were not needed for this project.

The web application was split up into components, each one roughly correlating to a page displayed to the user. The application was further partitioned into two so that the public website and private user account pages could be distinctly separated. A responsive approach was used on all pages so that the application would work well on screens of any size.

The public website contains a page with information directed to the trial users, a page to log into a previously created account and a page to sign up for an account. At this time, signing up for an account only requires a syntactically valid email address. This could be changed if the application were put into production, prompting the user for more information to satisfy KYC constraints and asking the user to validate their email address. Efforts were made to validate user input on the front end to indicate errors in signing up or logging in.

The user specific account section comprises of an overview page, a page to display recent transactions, and a page to add funds to the currently logged in account. A logout button is also provided to invalidate the user session and return to the website's information page.

The overview page gives a basic user profile showing the email address of the currently logged in account and its balance in both units and GBP, as well as a form for transfer of funds to another user of the system.

The transaction page shows transactions made by the account. At the time of testing, transactions were shown in chronological order with information about the sender/recipient of funds, the amount transferred and the date. This could be further improved with ways to search for

specific transactions and restrict the number of transactions displayed at a time.

Lastly, the add funds page simply adds the equivalent of £100 to the user's account when they click on a button. In production, the investment amount would be variable and user payment details would be taken.

The improvements mentioned are considered beyond the scope of this project as they do not inhibit the application's ability to test the underlying Openchain and SQL implementations. A sample of application pages viewed on a mobile device is shown in Figure 6.

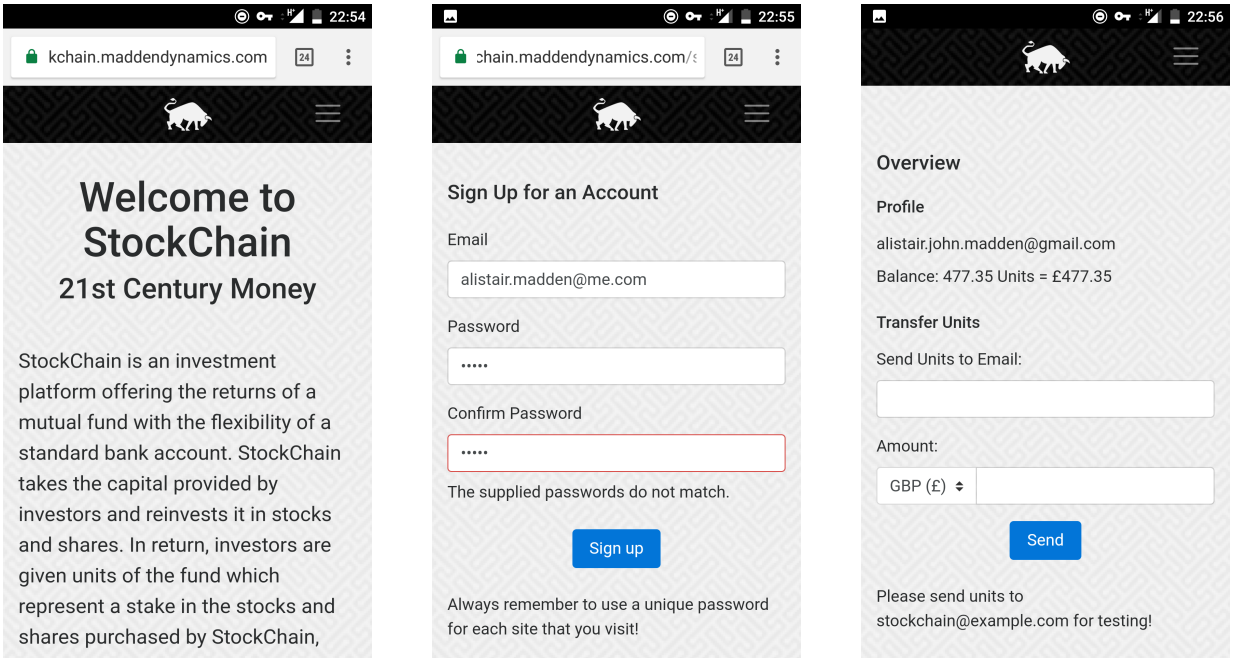

Figure 6: Website information page, sign up page and account overview page (left to right).

It is worth remarking on how the URL paths were configured in order to separate the front end from the back end.

When no path is supplied, it is assumed that the user is trying to load the about page of the application. The Node.js Express router defaults to the / path in this case. This loads the AngularJS application.

The REST API paths take the form /api/[action]. If the correct HTTP verb is used and [action] is known to Express router, the Node.js server performs the logic to complete it. Otherwise, if the HTTP verb is GET, the route is passed to AngularJS where it is treated as a 404 error from the front end. If the HTTP verb is not GET and the [action] is unknown, Express handles the request, returning a 404 from the back end.

AngularJS uses the ui-router module to handle all GET requests other than those that take the form /api/[action] (as these are handled by Node.js on the back end). If the path is known (/profile for example), the corresponding module is loaded. If not, the user is redirected to a 404 page. Additionally, restricted paths (paths which load user account modules) redirect the user to the login page if they are not signed in.

D Security Considerations

As mentioned in Section I, security has been a big factor in the demise of other centralised financial systems. As the site was to be made publicly accessible for survey participants, it was necessary to safeguard user details as much as possible. All traffic to and from the front end was encrypted before transmission with HTTPS. To make this possible, a trusted Secure Socket Layer (SSL) certificate was created with Let's Encrypt. This also marks the site as secure on most major web browsers. Users were also encouraged to use unique passwords for this application and given the option to use a fabricated email address for signing up. Validation of data was also performed extensively on the back end. This is necessary in any application to filter out incorrect and potentially malicious user input.

IV RESULTS

In this section, the performance of the SQL and Openchain implementations of the StockChain system are reported. The results from a user survey about finance and the quality of the end user application are also given.

A SQL and Openchain Performance

To measure the performance of a back end API, errors, speed and availability are usually measured.

As the system was quite simple, a custom suite of units tests was designed to test each publicly callable route. The system was tested for its response to expected submissions and also deliberate errors. The response produced by the API was matched against the expected behaviour of the system. The front end of the system was notified of errors via HTTP error status codes and custom error messages. In general, messages beginning with HTTP status code 4 are displayed to the user as these are client errors. These are expected by the system and can be solved by the user taking different actions or providing different input.

Any differences in availability between the two implementations was not tested. It was assumed that both implementations would be equally robust, at least at this development stage. In a production environment, the REST API could be periodically polled to check the paths are available and producing valid responses. It is of more interest to ask if the Openchain centralised blockchain framework offers something more in terms of functionality and what the expected performance trade off might be.

The main parameter used to test the speed of both implementations was the latency in responses to requests. To establish a baseline performance measure for each implementation, the systems were run locally with no load. This decision was initially made to reduce the number of variables such as network traffic that could impact the results. An average value over 10 requests to each route was taken. A selection of these results is given in Table 1 and Table 2.

System	/ First Access	/ Repeat Access	/api/login	/api/signup
SQL	5.769	0.759	57.760	117.228
Openchain	5.993	1.095	57.734	200.013

Table 1: Mean latency results over 10 requests to website paths. All values given in ms.

System	getAccountBalance First Access	getAccountBalance Repeat Access	addFunds	makeTransaction
SQL	3.362	1.786	4.049	7.977
Openchain	13.791	8.564	68.599	86.421

Table 2: Mean latency results over 10 requests to application paths. The preceding /api/ has been removed from paths for brevity. All values given in ms.

The first time the / path is accessed by a client (triggering the loading of AngularJS), and subsequent requests made by refreshing the web browser are given in columns 1 and 2 of Table 1.

Results for /api/login are given in column 3 of Table 1. This route generates a hash of the user's password and compares it to the one stored in the database as described in Section III A. The functionality of this route remains unchanged over the Openchain and SQL implementations.

The results for /api/signup are given in column 4 of Table 1. This route is responsible for standardising given email addresses, hashing passwords and inserting this information into the SQL database. For Openchain, the additional creation of a private key and address for the system must be made.

Table 2 shows the access times for four different api routes which formed the core of the application. For the SQL implementation, these routes call a stored procedure to carry out the action described, whereas the Openchain implementation had this logic replicated in the Node.js server.

The first access of the /api/getAccountBalance path is made as part of the user login. This is given in column 1 of Table 2. Any time the user navigates to the Overview page, their account balance is recalculated as it may have changed in the time they were browsing another part of their account. This is represented by the repeat access of the /api/getAccountBalance path in column 2 of Table 2.

The addition of funds to a user account was made via the /api/addFunds path. This is shown in column 3 of Table 2. This was the path with the biggest difference in logic between implementations. The SQL implementation solely relied on stored procedures for its logic whereas the Openchain implementation was carried out entirely on the Node.js server.

The /api/makeTransaction path was called from the Overview page to facilitate the transfer of funds between accounts. Its performance is highlighted in column 4 of Table 2.

Though these results serve as a good baseline measurement of a single route's performance, they do not take into account a situation where more load is placed on the server. In order to address this, a simulation of user load was performed. Apache JMeter was used to capture a typical session a new user may go through upon navigating to the StockChain URL in a web browser. The session involved accessing the about page, signing up for an account, browsing through the account pages and finally logging out of the application. Delays were added between each page access to model user thinking time. JMeter simulated this same flow of events for 100 concurrent users. Users were introduced gradually into the system over 50 seconds and this flow was carried out 10 times for each user. The results for the SQL and Openchain implementations are shown in Table 3 and Table 4 respectively.

Action	Average /ms	Min /ms	Max /ms	Std. Deviation /ms	Throughput /reqs^{-1}
About Page	3	1	29	2.30	1.278
Sign Up	125	119	163	4.02	1.316
Navigate Account	24	18	82	3.38	1.305
Log Out	6	0	23	1.29	1.319

Table 3: Latency results for SQL routes under load.

Action	Average /ms	Min /ms	Max /ms	Std. Deviation /ms	Throughput /reqs^{-1}
About Page	4	1	31	2.68	1.155
Sign Up	376	181	120304	3795.15	1.191
Navigate Account	86	46	261	32.52	1.183
Log Out	6	1	28	2.13	1.197

Table 4: Latency results for Openchain routes under load.

The final latency measurement was conducted for the /api/getAccountTransactions path. This route fetches all transactions made by a user account for display on the transactions page. The test was designed to discover the latency behaviour for different numbers of transactions on a user account and was conducted with a simulation of 5 users. Each user performed a loop of making 5 transactions (using the add funds page) and checking their transactions. This loop was repeated 10 times for each user. Each user loop ran sequentially so the performance would not be affected by load. The mean results for the lookup of each number of transactions are summarised in Figure 7.

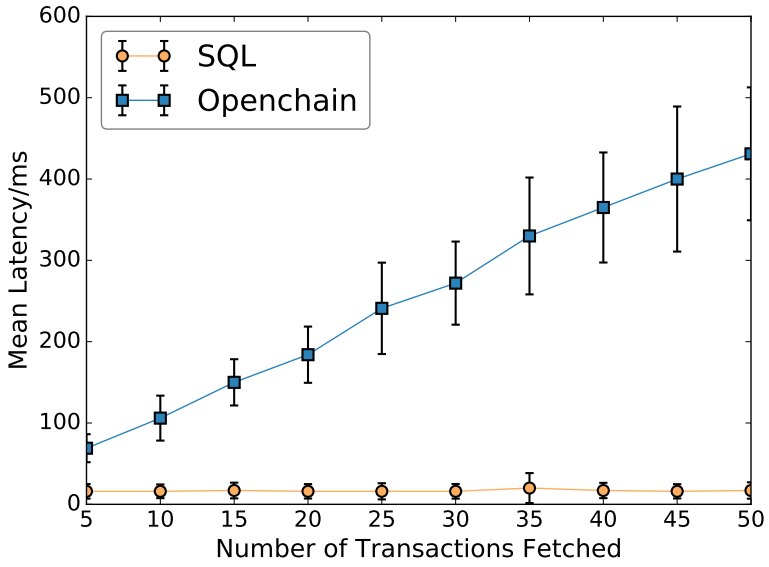

Figure 7: Mean latency for calls to /api/getAccountBalance. Error bars represent the standard deviation.

B User Survey Results

A user survey was carried out to analyse savings patterns and give feedback on the front end implementation. 29 responses were received in total.

The first part of the survey focused on the users who actively saved. This was found to be 82.8% of the respondents (24 people in total). These participants were asked to choose their most important feature when choosing a savings product (or add their own). The results are detailed in Figure 8. 25 of the 29 respondents indicated they had heard of Bitcoin and 48% of those would consider investing. All respondents who had heard of Bitcoin were asked what they believed the biggest issue with the cryptocurrency is. This is summarised in Figure 9.

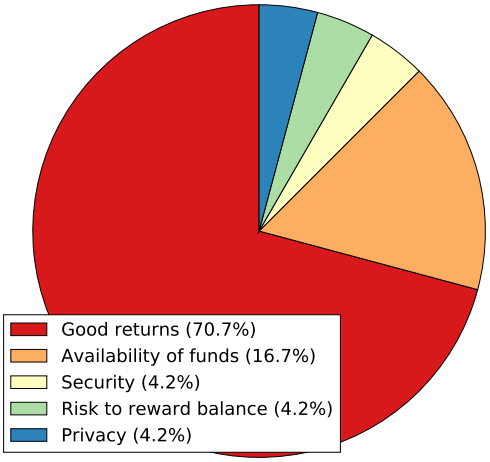

Figure 8: Features indicated to be important when looking for a savings product.

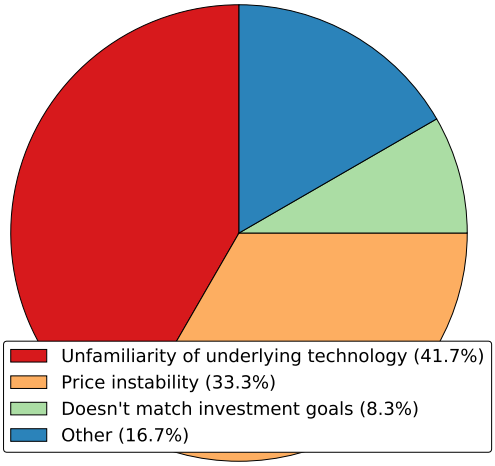

Figure 9: Biggest concerns with regard to the Bitcoin cryptocurrency.

The second section of the user survey focused on the StockChain application. Users were asked to navigate to it in a web browser of their choice and perform actions such as signing up for an account, adding funds and making transactions. The proportion of users who indicated correct functionality for different parts of the application is given in Figure 10.

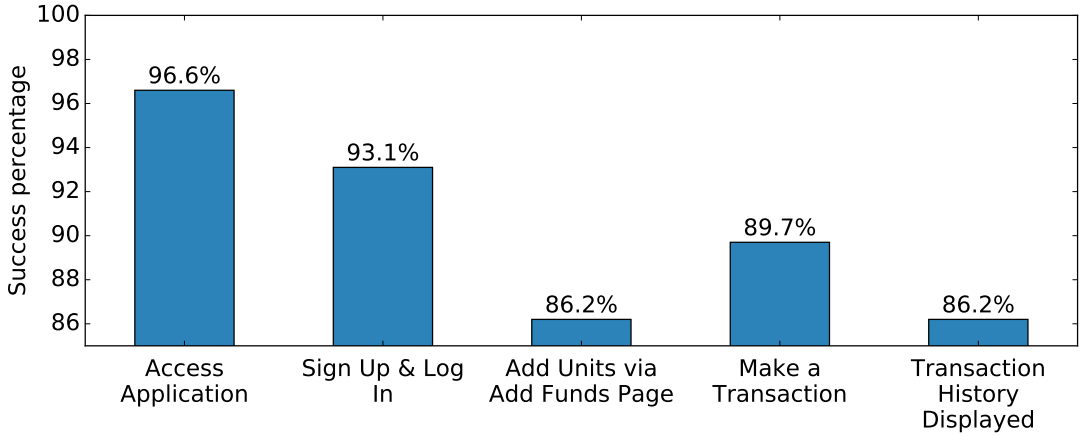

Figure 10: Proportion of responses indicating correct functionality for each action.

Respondents were also asked to give their opinion on the different sections of the application by ranking each page out of 5. The average mark for each page is presented in Figure 11. When asked to rate the entire concept in general, an average score of 3.7 was found.

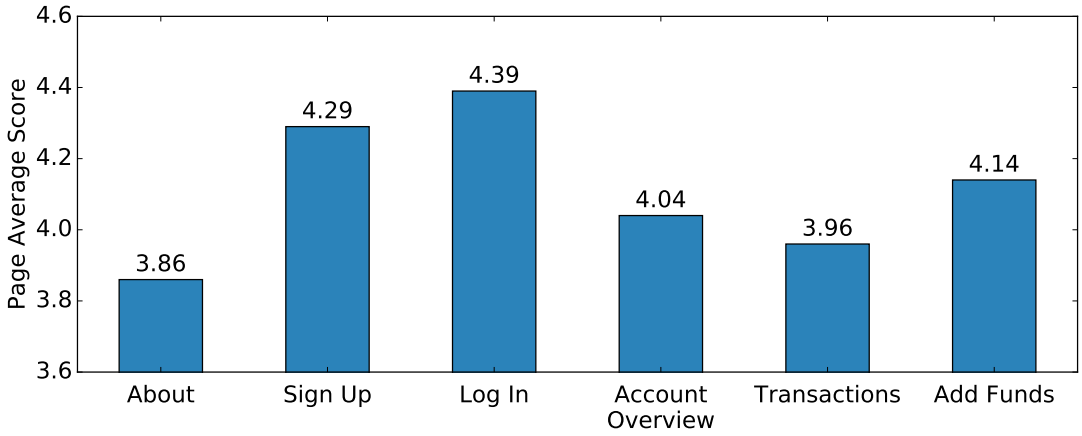

Figure 11: An average of the score given by survey participants for each page.

V EVALUATION

A Performance Comparison

It was clear that both implementations were given a level playing field during the latency tests. This is confirmed by similar latency results for implementation independent routes in Table 1.

Of more interest are the results from the /api/[action] paths as apart from /api/login, the logic of these varied over the two implementations. Openchain required additional configuration when registering each user compared with SQL. It is natural therefore that the Openchain response times shown by the /api/signup path in Table 1 should be higher. This additional configuration did not translate into lower latencies in other areas such as retrieving the account balance, adding funds or making transactions between accounts.

A similar theme was found when both implementations were tested under load. Implementation independent actions such as accessing the about page and logging out of a user account were found to be comparable. The abnormally high maximum and standard deviation for signing up an account using Openchain could be due to the single threaded architecture of Node.js. The sign up process for Openchain contains more callback functions than any other path. These functions will be called when some work is completed by another part of the system (such as the Openchain server) but cannot complete until this time. Any long running requests will have a small knock on effect on the time taken to respond to shorter requests. This could account for the marginally higher results for Openchain on implementation independent actions.

A further metric that was recorded in this test was the throughput of each system. Though the theoretical maximum for this value would be 2 requests per second, it is deceiving to compare the figures reported in Table 3 and Table 4 to this value. This is because requests were only made at the rate of 2 a second for 50 seconds. On average, the simulation of a user session would take 74.25 seconds in thinking time alone. This disparity leads to routes being busy for 50 seconds

and waiting for 24.25 seconds on average. The consequence of this is that routes must deal with 100 requests every 74.25 seconds leading to a maximum throughput of 1.35 requests per second. It can be seen that the SQL implementation achieves throughput closer to this maximum value than Openchain.

From Figure 7 it can be seen that the SQL implementation fetches all transactions tested in constant time regardless of number. Openchain exhibits a linear behaviour when performing the same action. This is expected as the Openchain API only provides a way to look up one transaction at a time. This necessitates looping over all mutations and determining their corresponding transactions before returning them. There are workarounds to this problem - the number of transactions fetched could be bounded and pagination used to load additional transactions if required. A more elaborate solution could be to use the asynchronous nature of Node.js to return each of the transactions as they are resolved. The transactions could then be displayed on the front end at different times, but the first transaction should always appear in constant time.

B Suitability of SQL and Openchain Implementations

Though the application was able to be developed with both SQL and Openchain, there are a number of compromises that each system makes.

Given the infancy of centralised blockchain frameworks, it was unlikely that Openchain would be a perfect fit for the project. The main reason for the use of public key infrastructure to digitally sign transactions was clearly inspired by Bitcoin. It is a way to keep the identity of the parties involved secret as the ledger is designed to be made public. This does not fit with the KYC constraints imposed on the StockChain application by financial regulations. If these links were used to track transactions through the network then the additional overhead would prove worthwhile. As it stands however, there is no way to do this in the Openchain API.

Through studying the codebase, Openchain was found to be based on a relational SQL database for its key-value store. From the Openchain documentation: ”Openchain is capable of immutability by committing a hash of the entire ledger (the cumulative hash) onto a non-reversible Blockchain such as Bitcoin.” This is equivalent to the scheme proposed by Haber & Stornetta (1990) using the Bitcoin blockchain as a timestamp server. This could also be realised in the SQL implementation, hashing the database and committing it periodically to the Bitcoin blockchain. This quote also suggests there is little protection at the core to guard against the centralised party responsible for the system from changing its data without relying on the Bitcoin network.

C Analysis of User Survey

The user survey highlights the knowledge and saving preferences. Less than half of respondents had heard of mutual funds. This could be why the about page received the lowest average user score in Figure 11. On review, the user score for each page should have been made out of 10. This increased precision would lead to better differentiation between the pages.

Figure 8 shows good returns were the most important factor when choosing a savings product, followed by availability of funds. Care must be taken when analysing these results as a selec-

tion of options was listed which could have biased user response. Given that StockChain was designed to provide good returns whilst still enabling user trading, the average score given to the concept in general (3.7) was expected to be higher. This provides further evidence that the core concept behind StockChain has not been communicated effectively. In launching StockChain as a production system, care would need to be taken to explain the concept thoroughly.

VI CONCLUSIONS

Blockchain technology has already been proven in a distributed environment through the Bitcoin cryptocurrency. In this scenario, all data recorded on the blockchain is made available for examination. Users of the system are granted anonymity via public key infrastructure if they use a unique address for every transaction.

This is very different from the majority of centralised cases where users of the system are not anonymous to the central party but data is not made publicly available.

Openchain is found to rely on the same anonymity and open ledger model as Bitcoin. Though the framework can be configured in a closed loop configuration for the privacy model required by the Stockchain application, many of its key features are not used. Other software frameworks will need to be explored in the future to discover if they can offer centralised immutability and greater traceability of transactions. This work recommends that this should be achieved through the development of a REST API only so that a more in depth performance evaluation of the system can be made.

References

Back, A. et al. (2002), 'Hashcash-a denial of service counter-measure'. [online]. Available at: ftp://sunsite.icm.edu.pl/site/replay.old/programs/hashcash/hashcash.pdf [Accessed 11 Apr. 2017].

Barber, S., Boyen, X., Shi, E. & Uzun, E. (2012), Bitter to better - how to make bitcoin a better currency, *in* 'International Conference on Financial Cryptography and Data Security', Springer, pp. 399–414.

Chaum, D. (1983), Blind signatures for untraceable payments, *in* 'Advances in cryptology', Springer, pp. 199–203.

Chaum, D., Fiat, A. & Naor, M. (1990), Untraceable electronic cash, *in* 'Proceedings on Advances in cryptology', Springer-Verlag New York, Inc., pp. 319–327.

EY (2016), 'Chain reaction: how blockchain technology could revolutionize the finance function'. [online]. Available at: http://www.ey.com/Publication/vwLUAssets/EY-chain-reaction/$FILE/EY-chain-reaction-how-blockchain-technology-could-revolutionize-the-finance-function.pdf [Accessed 11 Apr. 2017].

Haber, S. & Stornetta, W. S. (1990), How to time-stamp a digital document, *in* 'Conference on the Theory and Application of Cryptography', Springer, pp. 437–455.

Hyperledger (2017), 'Hyperledger whitepaper'. [online]. Available at: http://www.the-blockchain.com/docs/Hyperledger%20Whitepaper.pdf [Accessed 21 Mar. 2017].

King, S. & Nadal, S. (2012), 'Ppcoin: Peer-to-peer crypto-currency with proof-of-stake', *self-published paper, August* **19**.

Koroma, T. I. (2016), 'Analysis of blockchain centralization trends'. Master's Thesis, Durham University.

Kosba, A., Miller, A., Shi, E., Wen, Z. & Papamanthou, C. (2016), Hawk: The blockchain model of cryptography and privacy-preserving smart contracts, *in* 'Security and Privacy (SP), 2016 IEEE Symposium on', IEEE, pp. 839–858.

Nakamoto, S. (2008), 'Bitcoin: A peer-to-peer electronic cash system'. [online]. Available at: https://bitcoin.org/bitcoin.pdf [Accessed 21 Mar. 2017].

Narayanan, A., Bonneau, J., Felten, E., Miller, A. & Goldfeder, S. (2016), *Bitcoin and Cryptocurrency Technologies: A Comprehensive Introduction*, Princeton University Press, Princeton, NJ, USA.

Provos, N. & Mazieres, D. (1999), Bcrypt algorithm, USENIX.

Reid, F. & Harrigan, M. (2013), An analysis of anonymity in the bitcoin system, *in* 'Security and privacy in social networks', Springer, pp. 197–223.

Szabo, N. (1997), 'Formalizing and securing relationships on public networks', *First Monday* **2**(9). [online]. Available at: http://journals.uic.edu/ojs/index.php/fm/article/view/548 [Accessed 21 Mar. 2017].

US v. E-Gold, Ltd. (2008), Vol. 521, Court of Appeals, Dist. of Columbia Circuit.

Wood, G. (2014), 'Ethereum: A secure decentralised generalised transaction ledger', *Ethereum Project Yellow Paper* **151**. [online.] Available at: http://gavwood.com/paper.pdf [Accessed 11 Apr. 2017].