



University of Brighton

Decentralised Applications – Using Blockchain
Technology for Smart Property

Joe Shepherd

Student Number: 14809668

BSc Computer Science

Supervisor: Dr Goran Soldar

May 9th, 2018

Table of Contents

1. Project Scope	4
1.1 Aims and Objectives	4
1.2 Stakeholders	4
1.3 Measure of Success	4
1.4 Degree Relevance	5
2. Background Research.....	6
2.1 Existing research	6
2.2 Concepts of Blockchain.....	6
2.2.1 Defining Blockchain	6
2.2.2 Structure of a Blockchain	7
2.2.3 Mining – Proof of work	8
2.2.4 Benefits and Limitations	10
2.3 Ethereum	10
2.3.1 Ethereum concepts	10
2.3.2 Smart contracts	11
2.3.3 Smart property	11
2.4 DVLA Logbook	11
3. Specification and Planning.....	12
3.1 Deliverables	12
3.2 Risk Analysis.....	13
3.3 Methodology	14
4. Development	15
4.1 Application Architecture	15
4.1.1 Client-Side	15
4.1.2 Server-Side.....	16
4.2 Requirements	16
4.2.1 Functional Requirements:	17
4.2.2 Non-Functional:	17
4.3 Smart Contracts	18
4.3.1 Remix IDE	18
4.3.2 Contract Architecture.....	19

4.3.3 First Contract	19
4.3.4 Second Contract.....	20
4.3.5 Third Contract.....	20
4.3.6 Problems Encountered	21
4.4 Front-End Application.....	22
4.4.1 Design	22
4.4.2 Application Code	23
4.4.3 Problems Encountered	25
4.5 Technologies Used	25
5. Testing.....	26
5.1 Unit Tests	26
5.1.1 Solidity Tests	26
5.1.2 JavaScript Unit Tests	27
5.2 Black Box Testing	28
5.2.1 Problems Encountered	31
6. Limitations and Improvements	32
6.1 Blockchain.....	32
6.2 Application	33
7. Review and Reflection	33
7.1 Background research	33
7.2 Planning and Development	33
7.3 Conclusion.....	34
References	35
Appendix A: Gannt Chart.....	38

1. Project Scope

1.1 Aims and Objectives

The aim of this project is to investigate the use of blockchain technology and analyse how it can be used in future applications, specifically in smart property. To achieve this aim, the objectives are as follows:

- Research blockchain architecture to understand how this technology is implemented.
- Research how assets can be programmed onto a blockchain as smart property contracts to use as a tool for managing property rights.
- Develop smart contracts written in solidity to store data about a car logbook and deploy them onto a blockchain.
- Develop a prototype web application using HTML and JavaScript to interact with the blockchain to allow users to manage property rights of a car logbook.

1.2 Stakeholders

Stakeholders of this project include myself, my supervisor and second reader. The methods of communication that will be used to contact my supervisor will mainly be through emails and meetings. A set time and place were agreed upon to conduct these meetings and confirmed through emails.

1.3 Measure of Success

To measure the success of this project, the main factors will be on the creation of the products identified in the Gantt chart, whether it was created within the time frame and to a good quality, as well as the use of feedback from my supervisor on the deliverables produced. If deliverables are created according to plan, it will give a good measure of the success of the project as it will show that a comprehensive amount of background research has been conducted to gain enough understanding of the problem area, to recognise how the application can be implemented, as well as its complexity to estimate a correct schedule of activities. However, the final working product will be the ultimate measure of success on its ability to view, edit or change ownership of a logbook, and whether the application implements the user requirements of the system.

1.4 Degree Relevance

The knowledge and skills gained throughout all years of my degree will be used as the base to carry out this project. The front-end product will be a web application that uses HTML5, CSS and JavaScript, which relates to my degree as one of the modules carried out in first year was on web development, so the skills learnt here will be required to build the application. The back-end product will be the smart contracts written in the Solidity language. I have not done any modules on blockchain or learnt this language before, however it relates to my degree as Solidity has the same principles as an object-oriented language, such as Java. Throughout the course I have completed many modules that use Java programming and learnt about object-oriented design principles, which will be used to design and program this part of the project. Lastly, I have studied modules that helped me gain the skills to project manage this process. By choosing to do this project, I think it has a good balance between the skills I have learnt in previous modules throughout the course, and new skills that I will need to learn independently. I wanted to challenge myself to learn new and developing technologies by expanding and applying previously learnt skills and knowledge.

2. Background Research

2.1 Existing research

Blockchain technology is a new technological innovation, with the literature on blockchain proving this. According to Yli-Huumo et al (2016), most of the literature on blockchain is from 2012 onwards, with current research largely related to identifying core concepts, improving or resolving issues the technology has, and how it has been applied within the area of digital currency, namely Bitcoin. Research on Bitcoin focuses on mining and protocols (Dev, 2014; Gervais et al, 2014), and predominantly in the business management/finance area. ‘Almost every major financial institution in the world is doing blockchain research at the moment, and 15% of banks are expected to be using blockchain in 2017’ (Gupta, 2017).

Blockchains interest is increasingly expanding to a wide range of stakeholders from a diverse span of industries, resulting in smaller amounts of research focusing on the blockchain technology behind Bitcoin. They discuss the current and future applications of blockchain, including some limitations of the current techniques it uses. (Swan, 2015; Gov, 2016; Kishigami et al, 2015). Further research is being undertaken on public and private blockchains (Bussmann, 2017; Swan, 2016), and new software like Ethereum is being developed with a Turing complete programming language to help bring about the next generation of blockchain applications (Buterin, 2014).

2.2 Concepts of Blockchain

2.2.1 Defining Blockchain

Blockchain technology was first introduced by Nakamoto (2008) for the use of Bitcoin and is defined as a distributed data structure that ‘relies on a P2P network of computers, where all the members of the network maintain a distributed, shared, trusted, public ledger of transactions which everyone can inspect, but that no single user controls.’ (Shermin, 2017 p.500). It allows for two non-trusting users to interact with each other across the distributed peer to peer network without the presence of an intermediary party. Figure 2.1 demonstrates how a blockchain network is structured. Every node is connected to all other nodes, and each one has an exact copy of the blockchain, meaning every transaction is recorded on all nodes in the network. New transactions must be verified by all nodes before it is permanently added to ensure the data integrity of the blockchain.

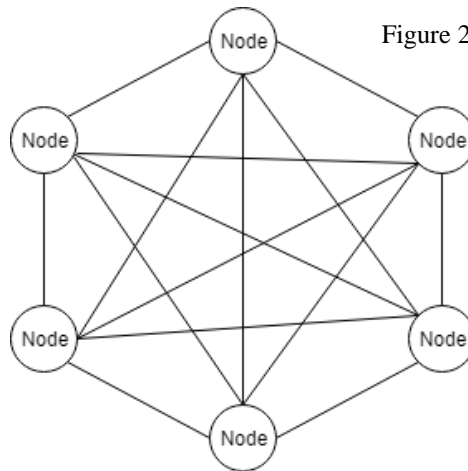
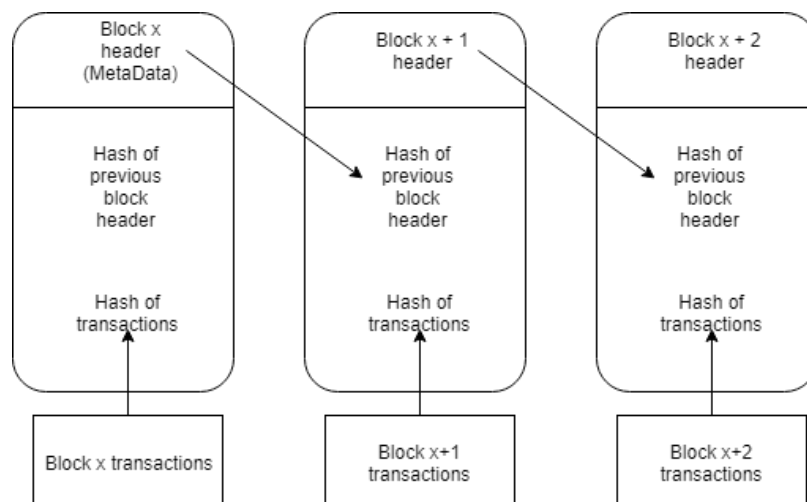


Figure 2.1 -Blockchain network

2.2.2 Structure of a Blockchain

The data structure comprises of blocks containing records of transactions carried out on the network. Cryptographic hashing of the metadata in a block is used to chain them together. The previous block's hash is assigned to the next block which keeps the order of the chain, similar to a linked list data structure. The hash of a block is partly determined by the preceding block, as this block's hash is part of the data used to generate the next block's hash (Prusty, 2017). Figure 2.2 demonstrates how each block is structured. This function is one of the blockchain's main strengths, as it prevents attempts to falsify data in the blockchain by making it tremendously problematic to fabricate new blocks or to change existing blocks. The hash acts as a digital fingerprint to prove the data has not been tampered with, as if data in one block is changed its hash value would change, causing all the following blocks' hash to change, so other nodes on the network would see the blockchain has been edited. To successfully do so every block in the blockchain would need to be changed simultaneously on all nodes in the network, proving almost impossible (White, 2017).

Figure 2.2 -Blockchain structure

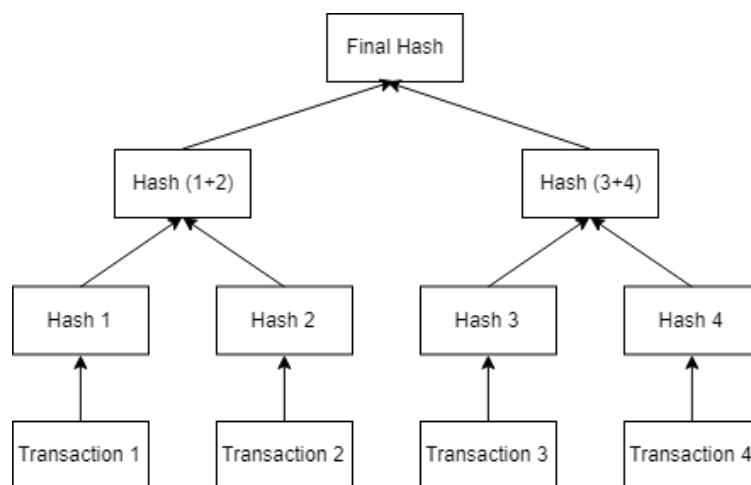


Building further from Shermin (2017), Christidis and Devetsikiotis (2016), transactions are recorded on the blockchain between two parties and validated through a consensus mechanism by other peers on the network. Members interact with the chain through public and private keys. A public key makes users accessible on the network for other users to transfer coins or other data to their account. The private key is used for a user to sign a transaction and keeps their account secure as only the user knows their own private key. A significant innovation to the blockchain are multi-signature transactions. These transactions allow two parties to trade autonomously without the need of a trusting third party. Essentially operating as an escrow service (Cuccuru, 2017).

2.2.3 Mining – Proof of work

On the majority of blockchains, including Bitcoin, the consensus on the network is achieved through miners using a method named proof-of-work. Miners compete with each other through computational power to ‘mine’ new blocks to the chain. The SHA-256 (Secure Hash Algorithm) hash function is used to provide a solution to the difficult mathematical hash value that creates a new block. The blocks hash value is created by computing every transactions hash, then these are hashed together multiple times to create one hash value, called a Merkle root, demonstrated at figure 2.3. This is then hashed again with the previous blocks hash, a ‘nonce’ value, which is a random number, a timestamp, and a level of difficulty of the hash algorithm (Morabito, 2017).

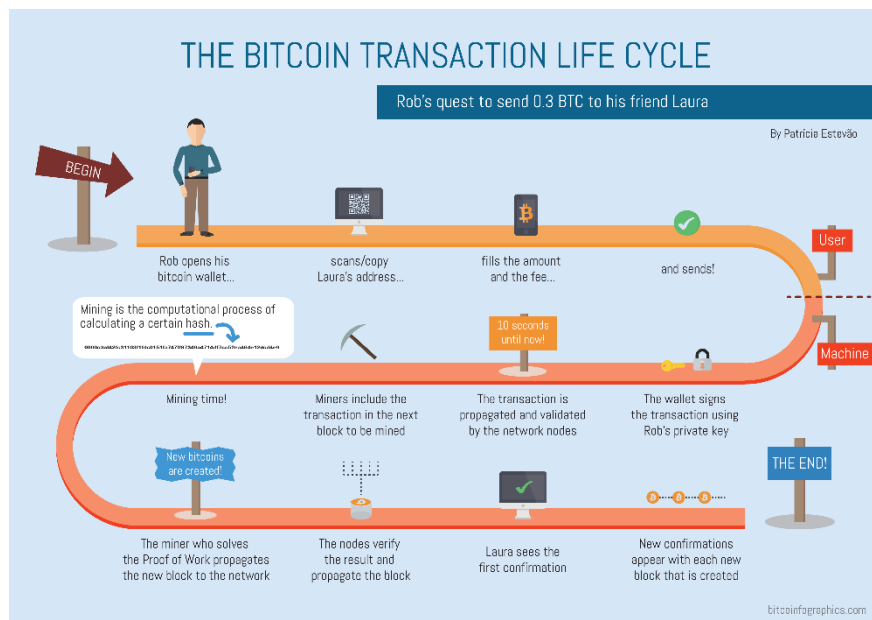
Figure 2.3 -Merkel tree of hash values



Miners then work out the solution of the algorithm through trial and error of different nonce values until it matches the hash value of the block. Prior to this a short verification process is carried out by the miners. The first step is to verify transactions by confirming the coins have not been spent and are signed by the senders' private key. Approved transactions are assigned to a non-verified block, then the hash value can be calculated. The final step of mining is to attach the verified block to the blockchain and wait for all other nodes to verify its authenticity by using the solution the miner provides to compute the hash value (Sammons, 2015). The whole mining process is shown at figure 4.4 (Estevão, 2015).

This is very costly in terms of computing power; therefore, miners are rewarded with cryptocurrencies every time they verify a hash value is correct. This algorithm becomes increasingly more complex as the blockchain grows, limiting the pace of blocks being added (Chuen, 2015). The mining reward comes from processing other people's transactions. A transaction fee, termed gas in Ethereum, is charged to each user and is given to the miner that adds the users transaction to a block. In Bitcoin and Ethereum, miners are also reward with Bitcoin or Ether respectively for each new block they add, as blocks contain the data on the coins, each block holds a certain amount of new coins which the miner then owns. (Barski and Wilmer, 2014).

Figure 2.4 – Bitcoin transaction lifecycle



2.2.4 Benefits and Limitations

By using replication, cryptographic hashing and irreversibility, the blockchain has become a very powerful tool to secure data due to its immutability and enables a trust-less network to be established for people to transact with each other through its use of transparency. Furthermore, it provides flexibility, cost savings and is efficient in certain aspects compared to other database record keeping technologies (Beck et al, 2016). However, on the other hand, blockchains like Bitcoin and Ethereum can take a long-time processing a transaction, consequential of its distributed and cryptographic nature. Furthermore, many blockchains are not regulated currently, which lead to Bitcoin becoming widely used in illegal markets and has resulted in millions being robbed from investors from scams. The computational power required and its rise in popularity also has detrimental effects on the environment, as the electricity demand to run mining rigs is more than what some countries use. Finally, due to its complexity it is difficult for end users to understand how to use blockchain applications, so it still has a long way to go before being fully adopted (Marr, 2018).

2.3 Ethereum

2.3.1 Ethereum concepts

Ethereum is a blockchain like Bitcoin that improves functionality with the use of a turning complete programming language, named Solidity. Unlike Bitcoin, Ethereum can store and execute code on the network (Buterin, 2014). Solidity is a higher level, object-oriented language similar to that of java, however objects are named contracts. It is statically typed with inheritance, libraries and user defined types (Solidity, 2018). It is compiled into bytecode by the Ethereum Virtual Machine (EVM) prior to being added to a block. The EVM acts as a large decentralised computer for the network that is designed to be completely isolated from the blockchain environment. To deploy contracts to the Ethereum blockchain, the node deploying it has to pay gas like mentioned before, as contracts still go through the verification process in the same way transactions are added to a block (Garcia-Alfaro et al., 2017).

Instead of being just purely a list of transactions, Ethereum's basic unit is an account. Contracts are like their own private account on the network, governed by its own code, and they can be accessed through sending a transaction to the account that holds the contract. Externally owned accounts are what users use. These are controlled by private keys and are accessible through public keys. Contract accounts only perform operations when an externally owned account sends a transaction or calls the contract to retrieve data (EthDocs, 2018).

2.3.2 Smart contracts

This functionality introduced smart contracts to the blockchain. The ledger deploys a smart contract to trigger transactions automatically when a condition is met (Buterin, 2014). This allowed the blockchain to evolve past the cryptocurrency area into many others. Szabo (1994) wrote a definition of a smart contract as ‘a computerized transaction protocol that executes the terms on a contract’. They act as a trusted automatic third party between transactions agreed upon by two parties (Christidis and Devetsikiotis, 2016), that eliminate delays without the use of intermediaries (Cuccuru, 2017). A smart contracts main function is to govern blockchain transactions (Kim and Laskowski, 2017).

2.3.3 Smart property

Smart contracts allowed for the creation of smart property. Drawing from Swan (2015) and Cuccuru (2017), the blockchain can be used for the transactions of all types of property. Tangible objects can be programmed and represented by a blockchain and secured. The ownership of these objects can then be controlled and accessed using the smart contract code at a specific Ethereum address. Users could trade assets by accessing the assets from their private key on the network, then send ownership of the assets to a user’s public key. This user then owns that object and it can only be accessed through their private key. A smart contract deployed on the network would ensure that the assets are not transferred until conditions are met, for example the buyer receives a product then the smart contract sends the funds.

Cuccuru (2017) points out a few concerns on the use of smart contracts, such as the code written for the contract not being understood by a non-technical person could cause confusion, or the fact that as smart contracts are automatic and cannot be changed once deployed, incorrect use of them could cause damage on the blockchain. Shermin (2017) points out that a benefit of smart contracts is that they would reduce transaction costs of the two parties making an agreement as no middle man would need to be paid, and it shortens the transaction process greatly. However, risks and benefits are mainly theoretical, as not much practical research on smart contracts has been conducted. (Gov, 2016; Yli-Huumo, 2016).

2.4 DVLA Logbook

As part of this project is to create a digital system for a V5C vehicle logbook document, some background research on logbooks was conducted. When a person buys a brand-new car, it can take up to 6 weeks to receive a paper V5C document. Furthermore, to change details owners need to send the logbook off to the DVLA and get a new logbook sent back, which is also lengthy (Gov, 2017). From this brief analysis, the system proposed in this project could provide ways to improve this process using smart property, as it would remove the paper V5C document, and as stated above, can reduce operational costs of a business as well as the amount of

time to process logbooks significantly, all whilst keeping the data very secure and tamperproof.

3. Specification and Planning

3.1 Deliverables

After conducting more research on blockchain and Ethereum, some changes have been made to the original project plan. I have more understanding of how applications in Ethereum and solidity can be programmed, as well as connecting the front-end application to the blockchain. Therefore, the order of deliverables has been changed slightly as some incorrect assumptions were made during the interim planning. The estimated schedule of these deliverables can be seen in a Gantt chart from Appendix A.

As seen from the Gantt chart, the first set of the applications deliverables that will be completed are the smart contracts written in Solidity. As I will be learning a new programming language, this will most likely take a great deal of time to understand. Furthermore, as I gathered from my research, the smart contracts can be tested separately from the front-end application. Due to these reasons I decided to develop this part first. The smart contracts will be used to store data in the blockchain about a car logbook and implement the methods that will be used to manage ownership rights. If any major changes need to be made to any other deliverables to adapt to changes in the smart contract design, this will be carried out in an agile, iterative manner.

The next deliverables for the application will be the design and development of the front-end application. The development of this will be split into two sections. The HTML and CSS code will be developed and tested together iteratively. Finally, the JavaScript code can be developed. This is also fairly new to me, and Ethereum provides a library called Web3 to interact with the blockchain. This will also take time to learn and implement, however will be developed last as this section brings the whole application together and cannot be implemented without the previous deliverables being completed first.

3.2 Risk Analysis

Risk	Probability (1-5)	Impact (1-5)	Mitigation	Contingency
Incorrect Estimate times for deliverables	3	3 – may result in deliverables being of low standard or incomplete final product.	Ensure enough research has been carried out to gain better understanding of how long timings could take. Reserve time for items that may fall behind schedule.	Make scope of project smaller.
Learning new programming language could prove difficult in time frame	2	5 – final product will be incomplete.	Ensure enough time has been reserved to read documentation and practice with a small sample program	Try to complete the simpler requirements of the system so final product has some functionality.
Development framework could be to challenging	2	3 – deliverables will be behind on time schedule	Research development framework to ensure implementation can be done.	Change implementation approach
Requirements analysis incorrect	3	3 – result in functionality of final product being incorrect or deliverables not being on time	Get a good grasp of the subject area before creating the requirements	Go back and change the requirements if there is enough time.
Research of problem area could be low quality	2	4 – result in weak understanding of project and result in a low-quality product being made	Carry out a large amount of research on blockchain to fully understand the technology and how it can be used/implemented.	Do more research throughout development process.
Data loss	1	5	Keep backups of project	Recover from backups
Injury/Illness of myself or supervisor	3	3- deliverables being behind schedule	Ensure extra time for deliverable completion in case of illness	Can go to second reader for feedback on project if need be.

3.3 Methodology

For this project, I will adopt an agile approach. This allows room for change, is value driven, and reduces complexity of a project. Work throughout will be defined in timeboxes. A certain number of deliverables will be produced in a time box frame (time between meetings with my supervisor). This ensures each deliverable is of acceptable quality before moving on to the next stage and improves the efficiency of the iterative development process. If the deliverables are concluded to not be of an acceptable quality, another cycle of that timebox frame will be conducted.

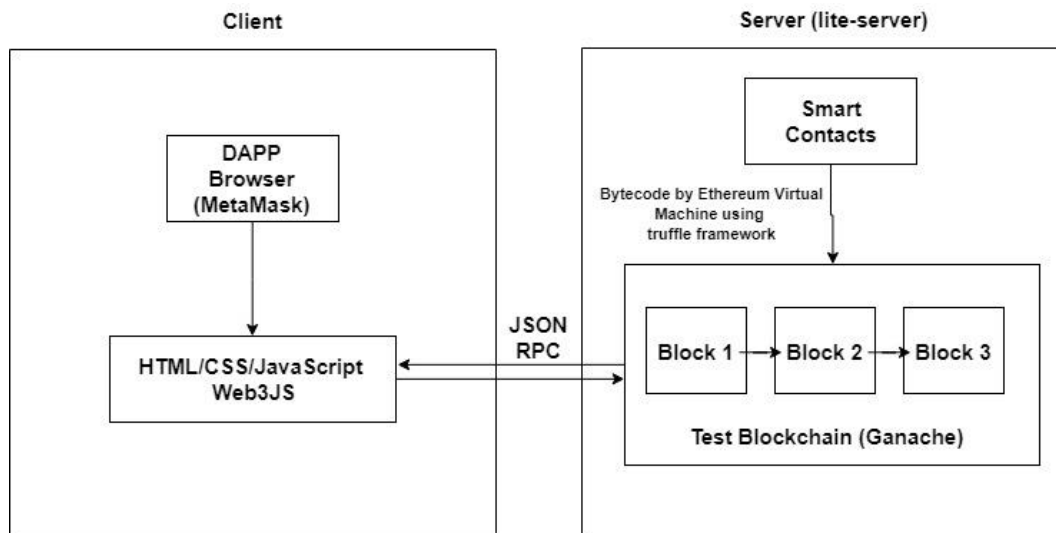
This was chosen rather than the waterfall model as this approach lacks flexibility and adaptability throughout each stage of the project lifecycle, due to its linear step by step approach. The requirements are defined at the start of the project, then the rest of the project is developed according to these requirements. As this project is not fully understood yet and requires a great deal of learning, the agile approach is much more suited as requirements are likely to change throughout the process once more understanding of the technology is gained, therefore the software will need to be adapted accordingly. Furthermore, using the waterfall approach, working software isn't produced until the end of the project which would be risky as I could make some incorrect assumptions as to how the application could be implemented, resulting in the whole project planning process being incorrect.

Additionally, this approach allows testing to be carried out after each deliverable has been created to ensure that no major problems will arise in the final working product. Test cases will be written to test the code to ensure the correct functionality is produced. Once the testing is complete and I am confident the code works, the next iteration of deliverables can be produced and tested. This can also be applied to the designing process. Feedback from a small sample will be collected to ensure the design allows for the correct functionality to be implemented. If any stages do not meet an acceptable standard and require previous sections to be re done, the agile approach allows for this adaptability.

4. Development

4.1 Application Architecture

Figure 4.1 – Architecture diagram of DAPP



To begin with, an architectural diagram was created to demonstrate the graphical representation of the decentralised application, detailing the components and principles of the architecture.

4.1.1 Client-Side

By looking at figure 4.1, the client-side to this application includes the front-end application written in HTML, CSS and JavaScript, along with a Dapp browser. Google Chrome and Firefox has an extension called MetaMask that gives the normal web browser the functionality to interact with the Ethereum blockchain (Metamask, 2018). Because of this, the application will only run on these web browsers with this extension installed. This makes it much less complicated for users to run a Dapp, as they can connect to the Ethereum blockchain and interact with it without having to download the whole blockchain and run a node themselves, which is a very daunting task. However, a downside to this is that it makes this application less accessible to a wider span of users, as many people use web browsers such as Safari or Internet Explorer.

The front-end application uses the Web3 JavaScript library to connect to the blockchain using the JSON-RPC data-interchange format. This will be the bridge between the client and server side. Contract functions will be accessed through this library to change or retrieve the ownership or details of a logbook. There are many different implementations of the Web3 library for different programming

languages, for example Java, however the JavaScript library was chosen as creating a web application seemed like the right platform for this type of application. Using a web app makes it much more easily accessible to users and provides a simple and easy way to interact with the blockchain with minimal understanding of the technology itself. The JavaScript library is also much less complicated than the Java library to implement and has more functionality included in the API, making the development process easier (Web3js, 2018).

4.1.2 Server-Side

The server side of this application, again from figure 4.1, includes the smart contracts and the Ethereum blockchain. The blockchain will hold all the data about the logbooks and which Ethereum account owns each one through the private keys. Instead of having a new contract for each logbook, one contract account will be used to store all the logbooks onto the chain, so only one contract needs to be accessed to manage the logbooks. Truffle have developed a test blockchain application called Ganache (Truffle, 2018). This software creates a simulated blockchain on the computer that MetaMask can be connected to that allows the application to be run without deploying it onto a live network. This tool will come in very handy for development and testing purposes, as deploying to a live network requires Ether which would end up being a very costly process, and as transactions may take a while to complete on a live network, would slow down the testing process substantially. Furthermore, if deployed onto a live network and there is a bug, it would require a great deal of time to keep deploying to the network every time the code has been changed, hence the decision to use a simulated blockchain.

4.2 Requirements

To gather the requirements for this application, I used my background research and further reading of the code documentations to gain a good understanding of the problem area, which helped me to discover what functionality could and could not be implemented using the Solidity programming language and the Web3 library. As the stakeholders are only myself, my supervisor and second reader, no primary research was conducted when planning the requirements of the system. Additionally, due to this being a prototype, the requirements do not reflect what would be of a professional level application. Further requirements to this system could be added, however the technology is still in its infant stage, limiting its potentials. Some of these requirements have changed throughout the process of this project, due to the volatile nature of this project and its planning/implementation.

4.2.1 Functional Requirements:

1. A user should be able to access the logbooks they own through their MetaMask account.
2. Upon opening the webpage, a user's logbooks should be automatically listed.
 - 2.1. If an error occurs whilst retrieving the logbooks, an error message should popup on the webpage detailing what went wrong.
3. A user should only be able to view their own logbooks, no one else's.
4. A user should have the ability to open up the details about the logbook to view the details of the owner of the car, along with all the car details.
5. A user should be able to change the ownership details of a logbook that they own
 - 5.1. An edit button should be available next to each logbook that opens up a popup page allowing a user to change the ownership details.
 - 5.2. Only ownership details should be allowed to be edited, not the car details.
 - 5.3. Upon pressing the edit button in the popup page, a MetaMask popup should appear with the transaction details, detailing the price of ether required and an option to accept or decline the transaction.
6. A user should be able to transfer the logbook to another person's public key on the blockchain.
 - 6.1. A transfer button should be available next to each logbook that opens up a popup page allowing the user to enter someone's public key address.
 - 6.2. Upon pressing the transfer button in the popup page, a MetaMask popup should appear with the transaction details, detailing the price of ether required and an option to accept or decline the transaction.
 - 6.3. Once ownership has been transferred, the webpage should refresh automatically, removing the logbook that has just been transferred.

4.2.2 Non-Functional:

1. The web application should be able to run on only the newest versions of Google Chrome and Mozilla Firefox with MetaMask installed.
2. Application should be designed so that not much understanding of the blockchain is necessary to operate, only some understanding of how to use MetaMask.

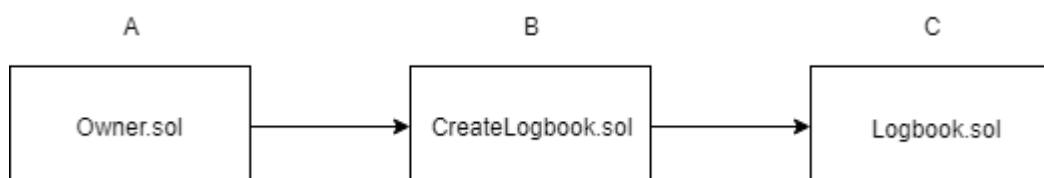
3. Some security measures should be in place so that no one can access other people's logbooks and change ownership details.
4. All users should have a unique MetaMask account with a unique public and private key to access their logbooks.
5. No user should be able to create a new logbook onto the blockchain.
6. Users should not have to login to an account when accessing the web application, as all this functionality is handled through MetaMask.

4.3 Smart Contracts

4.3.1 Remix IDE

Before beginning to program the smart contracts, a few code editors were looked into to discover which one would provide the most productivity. Visual Studio Code has support for HTML, CSS and JavaScript, but does not have Solidity language functionality built in. However, it does allow for open source addons to be installed that offers features like syntax highlighting, code completion, linting and compilation of contracts. In practise, there were some glitches with the linting functionality. This made it extremely difficult to find errors in the code as some errors should have not been identified. Additionally, this addon does not allow debugging of code, therefore an editor called Remix was used. This editor was developed for the Solidity language, containing all the functionality of the addon previously mentioned, as well as code formatting and debugging options. For these reasons, Remix was used to develop the smart contracts, and then transferred into Visual Studio Code for the rest of the development.

Figure 4.2 – Smart contract structure



4.3.2 Contract Architecture

As mentioned in 4.1.2, one contract account will be used on the blockchain. To keep the code organised and more readable, the functionality implemented is broken down into three separate contracts that inherit from the contract created before it. This creates a linear structure of inheritance demonstrated in figure 4.2, where C inherits from B, and B inherits from A. Once compiled into bytecode and deployed onto a blockchain, a single contract is created instead of three separate contracts, allowing the code to be compartmentalised into smaller, more manageable pieces without the added complexity of accessing multiple contract locations on the blockchain. Furthermore, by developing the contracts this way, it makes it easier to test the contracts functionality and identify which contract is causing errors, as each one can be tested separately (Solidity, 2018).

4.3.3 First Contract

To learn how to implement the contracts the Solidity documentation was used. The first contract created is only a few lines of code but is vital to the security of the blockchain application. When smart contracts are deployed to a blockchain, anyone can inspect the contract code and call its functions that are declared public. Due to this app being able to create logbooks that prove ownership rights of a car, it would be a major security concern if anyone could just call the contract function that creates a new logbook. In the real world, this process is controlled by the DVLA and all paper logbooks have a unique serial number and a watermark to prove its authenticity. To overcome this issue, the first smart contract implements some functions to only allow the owner of the contract to call certain functions, which would be the DVLA. To implement this, the contract stores a state variable of type address (20-byte value of an Ethereum accounts public key) that is set from the contracts constructor method as the Ethereum account address that deployed the contract (Solidity, 2018).

Figure 4.3 – Function modifier

```
modifier onlyOwner() {  
    require(msg.sender == owner);  
    _;  
}  
  
**  
* @dev Allows the current owner to transfer ownership of the contract  
* @param newOwner The address to transfer ownership to.  
*/  
function transferContract(address newOwner) public onlyOwner {  
    // Transfer ownership to newOwner  
}
```

To ensure only the owner can call certain functions, a special function called a function modifier is used, shown in figure 4.3. A modifier is written once and can be stated in any number of function declarations to easily change the behaviour of the function. If stated, the code declared in the modifier is executed before the function code. This modifier was implemented using a require statement. A require statement is like an if statement that reverts the code back to its original state if the conditions are not correct. In this case, the require statement checks if the person calling the method is the owner address of the contract, if not the function code does not get executed (Solidity, 2018).

4.3.4 Second Contract

The second smart contract implements the functions to create a new logbook and has state variables to keep track of how many logbooks a person owns, as well as who owns each logbook. As mentioned above the create logbook function uses a modifier to check if the person calling it is the owner of the contract. To store the data about each logbook, a new data type was created using a struct. This is a special data type that allows for the creation of new user defined types which can hold many different variable types inside (Solidity, 2018).

Once a logbook is created, it is stored inside an array that is the same data type as the struct that was created. To keep track of who owns the logbook, a mapping is used that links the array index of the logbook to an Ethereum account address, shown in figure 4.4. A mapping in Solidity is like a hash table that creates an association between a key and a value. The key is not actually stored in the mapping though, only a hash used to look up the value is stored. Another mapping is used to keep track of how many logbooks a person owns that is used later to improve the transaction costs of a function call, explained in 4.3.5 (Solidity, 2018).

Figure 4.4 – Mappings

```
Logbook[] logbooks;  
mapping(uint => address) logbookToOwner;  
mapping(address => uint) logbookCount;
```

4.3.5 Third Contract

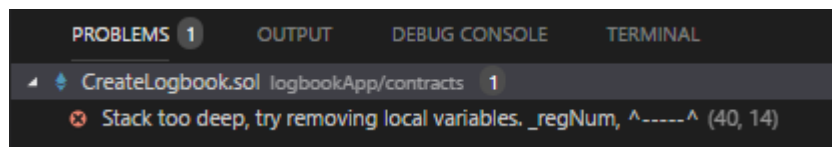
The last smart contract implements the functionality that the web application will call to change or retrieve details of a logbook, or to transfer ownership. To retrieve all the logbooks a user owns, the mapping of account to the number of logbooks is accessed to create a fixed length array of integers that will store the array indexes of the logbooks. This reduces the amount of memory needed for the function to execute rather than if the array length was not specified, as dynamically sized arrays use more memory than fixed sized arrays, improving performance and reducing the

gas cost to execute. The array of logbooks is then looped through to check if the array index is mapped to the address of the person calling this function and then added to the array of integers. When retrieving or changing the information for the logbooks the array index is passed through and the logbook at that position in the array is accessed. A require statement was used in these functions that checks if the person trying to retrieve information is the owner, by using their address and checking it against the corresponding mapping. This adds some more security to the application as it prevents other users viewing or changing the logbook details of a logbook they do not own, which helps to secure the data residing on the chain (Solidity, 2018).

4.3.6 Problems Encountered

Originally to hold the details of the logbook, the struct used strings to store data. However, when the function to create a new logbook was developed, compiling issues stating ‘stack too deep’ appeared and did not allow the smart contracts to be compiled, shown in figure 4.5. This error occurs as the Ethereum memory stack is not that large, so it can only hold roughly 16 variables in a function depending on how it is used and what variable types are being used (Solidity, 2018). Logbooks have a lot of details to store, so this was very hard to try and solve without losing vital information. Additionally, strings use up more space on the memory stack than other variables due to it being a reference type that is dynamically sized.

Figure 4.5 – Stack too deep error



To overcome this issue the variable type `byte32` is used that holds an array of 32 bytes. Strings can be passed into a `byte32` variable which allows the details for the logbook to be entered into the struct, however to retrieve the information to display on the web application, a function that converts bytes to strings needed to be implemented. A benefit to this is that it allowed more variables to be stored inside the struct, so more information about the logbook could be added as byte variables use up less memory on the stack. Furthermore, `byte32` also reduced the amount of gas needed for transactions to occur due to the lower memory usage, in turn making transferring ownership or changing details cheaper for users.

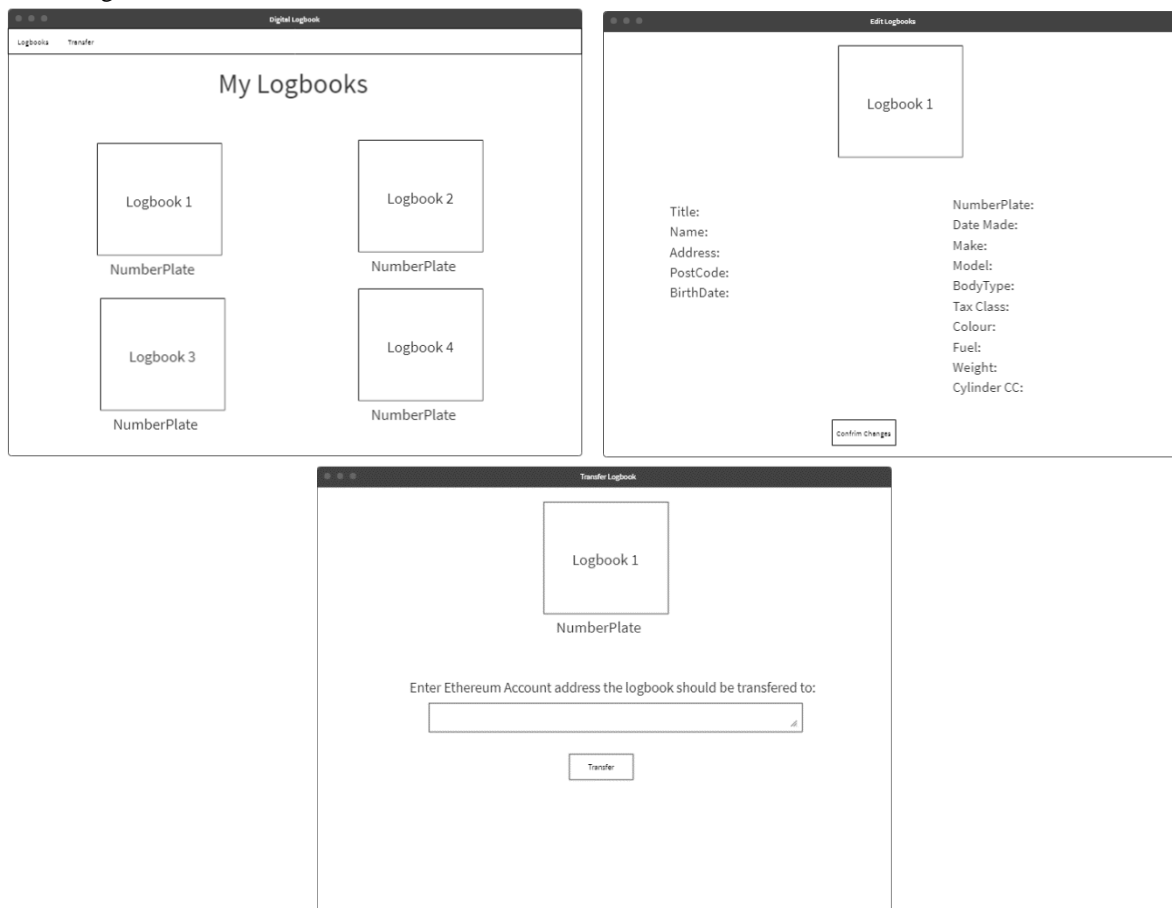
On the other hand, the Ethereum stack too deep problem limits how much data can be stored in object types like structs, so if other applications wanted to store smart property onto a blockchain, this would have to be taken into consideration which may not make it very feasible.

4.4 Front-End Application

4.4.1 Design

The design of the front-end application was not of main focus throughout this project as most of the time was required to learn Solidity and Blockchain technology. For these reasons the design was kept simple, however some time still has to be allocated as the application front-end highlights the sole purpose of this project. Initial designs are available in figure 4.6.

Figure 4.6 – Wireframes

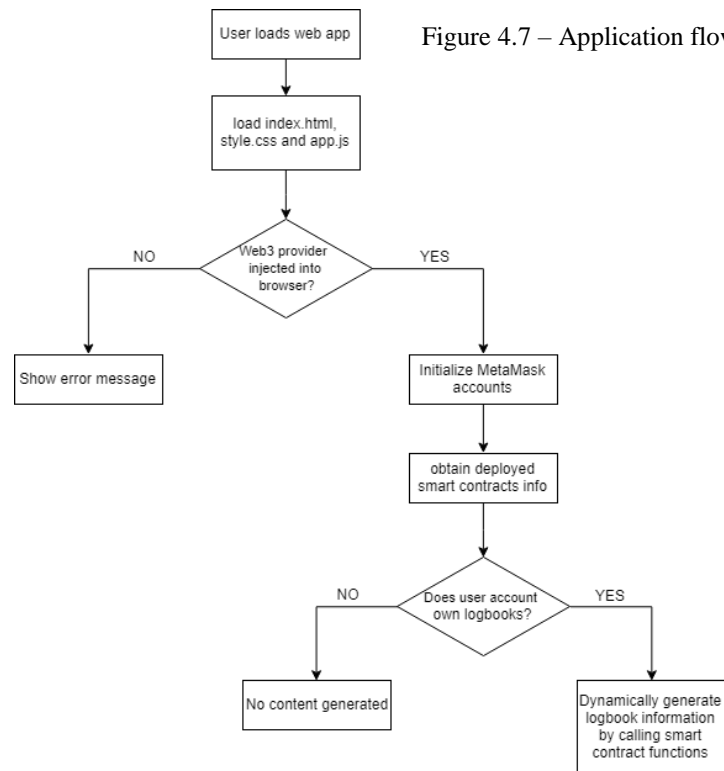


A very small sample of friends and people around university were asked some questions to give feedback on the design. As mentioned, not much time was allocated so did not allow me to get a larger sample. However, some suggestions were made to make the design more user friendly. Instead of having three separate webpages to view, edit and transfer logbooks, people suggested that the edit and transfer button be placed on the same page, and an overlay popup appears when pressed that allows a user to do the functionality requested. I took this feedback on board, as this would make programming the web application much less complicated as well as making it more user friendly and improve the user experience. Only one HTML, CSS and JavaScript file is needed with this design instead of three, resulting in less code. The final designs implemented with CSS are available at section 5.

4.4.2 Application Code

Before writing the HTML and CSS code, frameworks such as bootstrap were looked into as these provide very easy methods to create a professional looking website in a short period of time. However, as the final design was simple enough, this was not necessary. To develop the JavaScript code, the jQuery library was used as this provides added functionality to normal JavaScript that makes it much easier to manipulate the webpage. Additionally, the Web3js library was used, however not the original library provided by Ethereum. Truffle provide a modified version of this library that makes it much easier to access the smart contracts on the blockchain, which makes code shorter and less complicated. This library provides a collection of libraries that allow a webpage to interact with an Ethereum node, using a HTTP or IPC connection.

When a user opens up the webpage, the application checks if a Web3 provider has been injected to the browser. In this case, the Web3 provider is MetaMask. Initialising the accounts is the next step. The application retrieves the accounts from the Web3 provider, which would be the users Ethereum account they are logged into on MetaMask. Once this is complete the smart contracts deployed onto the blockchain need to be initialised. When being deployed, a .json file of the compiled contract gets placed into a build folder in the applications directory. A jQuery statement retrieves this json file of the contract that has information about its location on the blockchain to be able to call its functions and stores it in a variable. A diagram demonstrating the applications flow when a user first opens the web application is shown in figure 4.7 (Web3js, 2018).



To load the user’s logbooks, this is dynamically generated at runtime to make the user experience better to use. The location of the smart contract is accessed and requests a call to the blockchain to retrieve the logbooks a person owns. It does this by using promises. Promises execute the function code and wait for the transaction call to be completed before any other code is executed, which is useful for blockchain applications as the transactions may take time to complete, so no code is executed until the transaction data is retrieved. jQuery statements then manipulate the HTML code to show the logbooks on the webpage (Web3js, 2018).

When a user needs to send a transaction instead of a call, such as changing logbook details or transferring ownership, the same process is carried out. A call is when a user does not need to change data in the blockchain, and a transaction is one that does. Because of this a call does not require gas, however when a transaction is sent, a MetaMask popup appears asking a user to confirm the transaction and gives an estimate of the max transaction cost of the operation. Once a transaction is sent a user has to wait for the miner to verify this onto a block. As I used a test blockchain application it automatically mines new blocks to the chain so transactions were verified straight away (Web3js, 2018).

4.4.3 Problems Encountered

Before I used the truffle implementation of the web3 library, the original was used. This was very hard to setup as the contracts address on the blockchain has to be manually retrieved and entered into the JavaScript code, and writing the code to call a smart contract is much lengthier than the truffle library. This caused many problems in the application that I could not get working correctly. None of the user's logbooks were getting retrieved from the blockchain and it was very hard to debug why this was happening. After a few days of trial and error I found the truffle library which made it very easy to call the blockchain as it automatically gets the smart contract location when initialising the contract, resolving the issues that I had before.

Furthermore, I did not fully understand how to use promises when calling contract functions as were new to me. When writing long chains of promises, the code started to look quite messy which made identifying errors more complicated. The Web3 library doesn't have any way of identifying errors in code before it is run. To overcome some of the errors, which was mainly due to syntax errors or incorrect smart contract function name, a catch block is used at the end of each promise with the error message being logged to the console to identify which part of the code was not executing. This made it a little easier to debug however still was a very daunting task.

4.5 Technologies Used

Many different software and libraries was used to make this software possible, as well as other code provided by the development frameworks. These are as follows:

NPM and Node.js – used to install the required libraries and development frameworks. This has to be installed before the application can be run on a computer as it provides the command line tools to execute the program. This creates the node module folder in the project directory as lite-server was added from npm to run the web server.

Truffle – development framework used as the backbone for this application. Has to be installed before application can be run as it also provides the command line functions to execute. This was also used to setup the file directory of the application. When it creates a project, some files are created. These are shown below:

- Migrations.sol – used for the deployment of the smart contracts onto the blockchain.
- Migrations folder containing 1_initial_migration.js – also used for managing the deployment of the smart contract.
- bs-config.json – contains the information on where to find the deployed smart contracts json files.

- `package-lock.json` – contains the dependencies that were installed from npm with `lite-server` and others.
- `package.json` – metadata about project like version number, `lite-server` etc.
- `truffle-config.js` – sets up the network that is used for the development.
- `truffle-contract.js` – truffles version of the Web3 library used to interact with the blockchain.

MetaMask – chrome extension used to run a Dapp on a normal browser.

Lite-Server – used to create a server that the web application is run on (Papa, 2017).

Ganache – test blockchain used when deploying smart contracts.

jQuery - JavaScript library used to in the front-end as it makes manipulating the webpage easier.

Image of wheel – Royalty free image used in the design of front end (Pixabay, 2018).

5. Testing

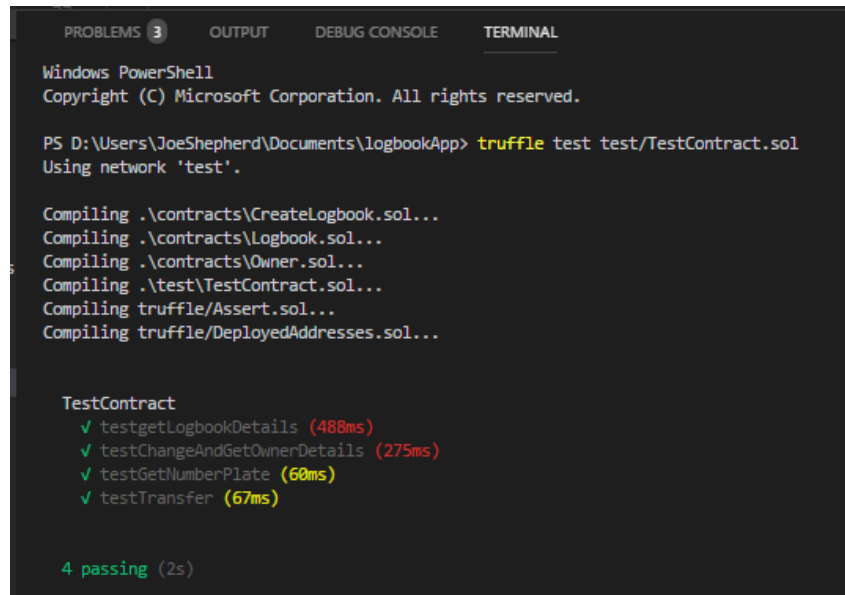
5.1 Unit Tests

5.1.1 Solidity Tests

After the solidity contracts were created, unit tests were written in the Solidity language to test that the core functionality works correctly. This was done to ensure that they did not return incorrect data when the contracts are connected to the front-end application with the Web3 library. The functions that create a new logbook, retrieve logbook details, change logbook details and transfer ownership were all tested to see if the values returned from the functions were consistent with the values entered into the logbook. When testing this way, the modifier that uses a `require` statement to confirm only the owner of the contract could call this method had to be removed from the create logbook function, as the test contracts cannot check which account the function is being called from, therefore kept returning errors. This meant that the security of the contracts could not be tested initially, however was tested after with unit tests written in JavaScript.

Unit tests in solidity work similar to other types of unit tests. Before the test are run, a logbook is created containing some dummy data to be interacted with. The solidity functions are then called, changing the state or retrieving the data from the logbook. After this, the `assert` statement checks the values returned from function calls is the correct data.

Figure 5.1 – Running the unit tests in Solidity



```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

PS D:\Users\JoeShepherd\Documents\logbookApp> truffle test test/TestContract.sol
Using network 'test'.

Compiling .\contracts\CreateLogbook.sol...
Compiling .\contracts\Logbook.sol...
Compiling .\contracts\Owner.sol...
Compiling .\test\TestContract.sol...
Compiling truffle\Assert.sol...
Compiling truffle\DeployedAddresses.sol...

TestContract
✓ testgetLogbookDetails (488ms)
✓ testChangeAndGetOwnerDetails (275ms)
✓ testGetNumberPlate (60ms)
✓ testTransfer (67ms)

4 passing (2s)
```

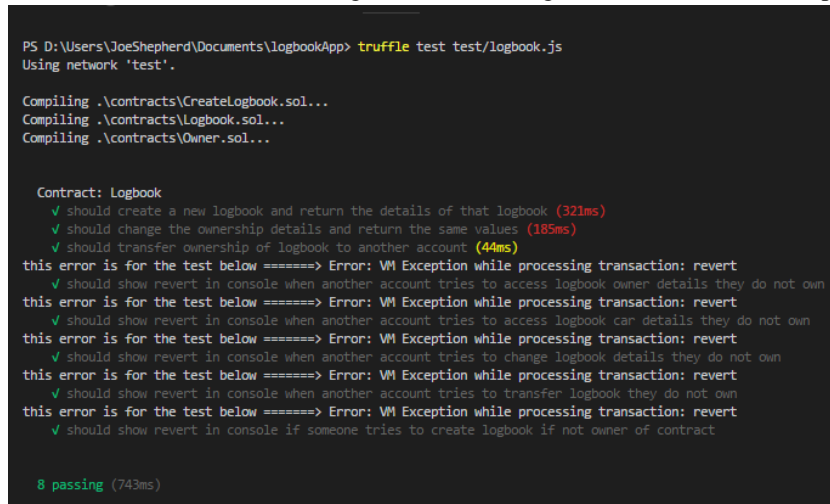
Figure 4.3 shows the test contracts being run. The first and third test checks that all the details in the logbook are the same as the data entered when the logbook is created. This tests all the retrieve functions in one test instead of having a separate test for a single retrieve function, as this would make the test cases extremely long and more complicated to read. As there are many assert statements inside this test, it takes quite a large amount of time to run, hence the colour being in red. However, the time it takes for the tests to run are of no significance for these tests, as the speed at which the functions are called in the actual program depend on the blockchain network and its miners. The second test changes the details of the logbook and then retrieves them, checking the returned values are the same as the ones entered into the function. This also tests multiple functionalities in the smart contract at once to keep things simpler. The last test checks that transferring the logbook to another account works correctly. To implement this test, a function inside the contracts were created that returns the address of the owner of the logbook, so it can be checked against the address the logbook is sent too. These tests prove that the contracts functionality works as intended to as no errors were returned.

5.1.2 JavaScript Unit Tests

Once all the solidity tests were completed and run without any errors, the development of the front-end was carried out. As this was being developed, JavaScript unit tests were written to test that the correct values were being returned from the smart contracts when interacting with the Web3 library, exactly the same

as the previous tests. Further tests were created to test the security of the contracts by ensuring that an error is thrown if the revert statements are invalid. Truffle uses the Mocha framework along with the Chai framework for the assertion libraries to test the JavaScript code, which runs on Node.js in the browser and makes it very easy to test that the front-end code implements the correct functionality, showing clear indications of what went wrong if a test fails.

Figure 5.2 – Running the unit tests in JavaScript



```
PS D:\Users\JoeShepherd\Documents\logbookApp> truffle test test/logbook.js
Using network 'test'.

Compiling .\contracts\CreateLogbook.sol...
Compiling .\contracts\Logbook.sol...
Compiling .\contracts\Owner.sol...

Contract: Logbook
  ✓ should create a new logbook and return the details of that logbook (321ms)
  ✓ should change the ownership details and return the same values (185ms)
  ✓ should transfer ownership of logbook to another account (44ms)
  this error is for the test below =====> Error: VM Exception while processing transaction: revert
  ✓ should show revert in console when another account tries to access logbook owner details they do not own
  this error is for the test below =====> Error: VM Exception while processing transaction: revert
  ✓ should show revert in console when another account tries to access logbook car details they do not own
  this error is for the test below =====> Error: VM Exception while processing transaction: revert
  ✓ should show revert in console when another account tries to change logbook details they do not own
  this error is for the test below =====> Error: VM Exception while processing transaction: revert
  ✓ should show revert in console when another account tries to transfer logbook they do not own
  this error is for the test below =====> Error: VM Exception while processing transaction: revert
  ✓ should show revert in console if someone tries to create logbook if not owner of contract

8 passing (743ms)
```

Figure 4.4 shows the tests being run in JavaScript. The first three are very similar to the Solidity test, checking that the smart contract functionality works as expected when the web3 functions call them. The last five tests check the security of the smart contracts, to ensure only owners can change, view or transfer their logbooks, and that only the owner of the smart contracts can create new logbooks. Testing these parts of the code is vital to the application as without it the application could easily be changed by programmers and steal or create new logbooks, which would make the program worthless.

5.2 Black Box Testing

Once most of the development was completed and the unit tests ensured the core functionality was in working order, black box testing was carried out for the rest of the process to test if the application fulfils the software requirements and that there were no bugs when using the program. By adding catch blocks in the code to logs the errors in the console, the errors could be viewed from a web browser using the inspect function. A file named seed.js is run once the smart contracts have been deployed onto the network that adds multiple contracts to the blockchain to the initial contract owner account.

Figure 5.3 – Initial screen

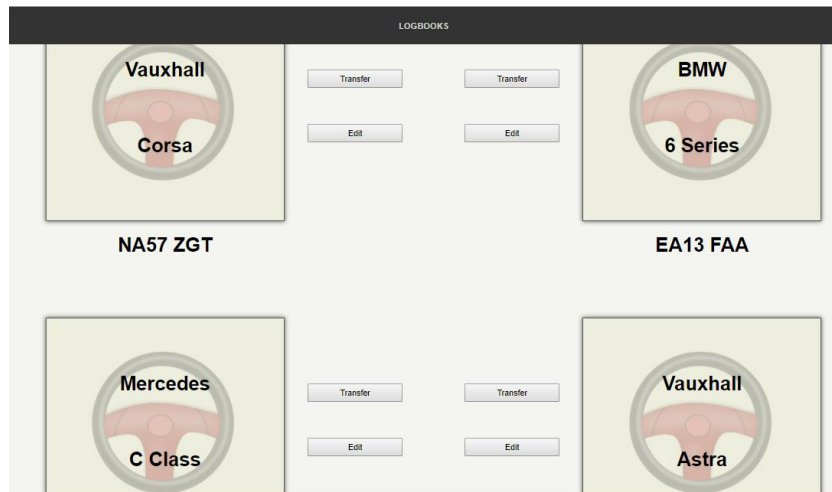


Figure 5.3 shows the initial screen upon opening the webpage when logged into the first account provided by Ganache (owner of contracts). The correct logbooks show on screen with the correct details.

Figure 5.4 – view/edit logbook details

Figure 5.4 demonstrates the logbook details that show up upon pressing the edit button(left) to change logbook details and the logbook picture(right) to show the logbook details. When initially creating the logbooks empty owner details was entered, however both screens show the correct logbook details appearing under these sections. Figure 5.5 demonstrates the logbook details after the edit button was pressed with some owner details entered into the text boxes. The MetaMask popup details the transaction being confirmed and added onto the blockchain, with the price of the transaction in ether on the right. The transaction cost to change details essentially didn't cost anything at all.

Figure 5.5 – edit logbook details

The image shows a 'Logbook Details' form on the left and an 'Account 1' summary on the right. The form has two sections: personal details and vehicle details.

Logbook Details	
Title:	Mr
First Name:	John
Last Name:	Smith
Birth Date:	00/00/00
Address:	someAddress
Postcode:	somePostcode
Registration:	NA57 ZGT
Date Made:	04/06/1997
Make:	Vauxhall
Model:	Corsa
Body Type:	Hatchback
Tax Class:	V149
Colour:	
Fuel:	
Weight:	

Account 1
Ox62730...
99.597 ETH
73949.37 USD
[BUY] [SEND]

SENT | **TOKENS**

8 May 09 2018 02:18
Ox345cA3e0...3e10
[Icon] 0 ETH

Figure 5.6 – transfer ownership

The image shows a 'CONFIRM TRANSACTION' screen on the left and a 'Transfer Ownership' modal on the right.

CONFIRM TRANSACTION Private Network

Account 1
627306...EF57
99.606 ETH
73852.07 USD

Amount: 0.00 ETH / 0.00 USD

Gas Limit: 90000 UNITS

Gas Price: 100 GWEI

Max Transaction Fee: 0.009000 ETH / 6.67 USD

Max Total: 0.009000 ETH / 6.67 USD

Data included: 68 bytes

[RESET] [SUBMIT] [REJECT]

Transfer Ownership

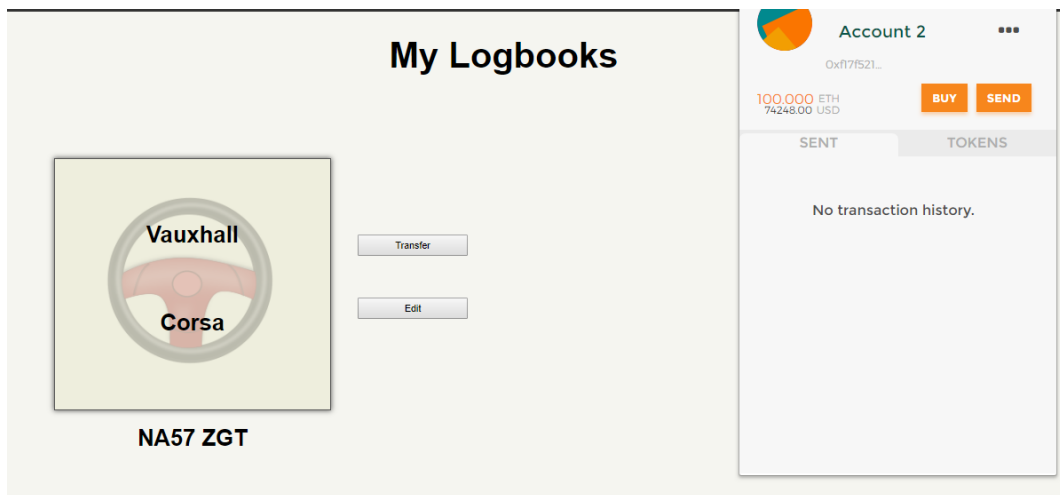
Number Plate: NA57 ZGT

To (Wallet address key): 0xf17f52151EbEF6C7334FAD080c5704D

[Transfer]

Figure 5.6 and 5.7 demonstrates the transferring of logbooks. The logbook is sent from the first account provided by Ganache to the second account. The max transaction fee seems quite high initially, although when the transaction is actually carried out it hardly cost anything at all. Figure 5.7 shows the second account being logged into MetaMask, who now owns the logbook with the number plate 'NA57 ZGT'.

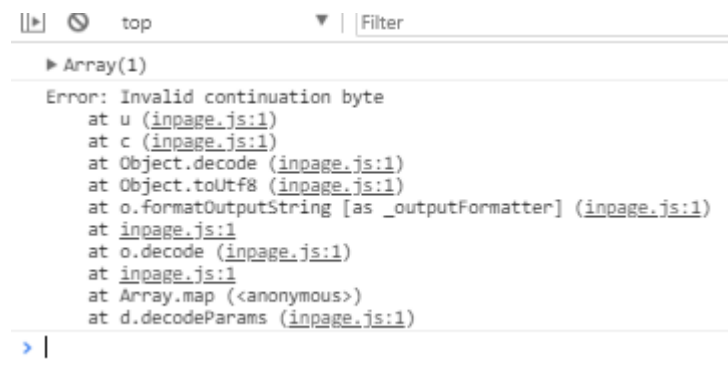
Figure 5.7 – transferring ownership



5.2.1 Problems Encountered

One of the major issues that I encountered when black box testing can be seen in figure 5.4. The last four logbook details appear blank when trying to access them. This was due to the error which can be seen in figure 5.8. For some reason the function to retrieve these details did not work and I could not debug the reason for this as there is hardly any documentation on these kinds of errors for the Web3 library. The functions to retrieve details had to be split due to the stack too deep error mentioned in 4.3.6.

Figure 5.8 – Error occurred



From the black box testing, it shows that the application works as specified from the software requirements created, apart from the errors that I could not figure out how to resolve. However, all the main functionality works correctly to a good enough standard. The security of the smart contracts could not be thoroughly tested through this method, as only an owners' logbooks is shown to them for

viewing/editing, although proves that some security of the system still works correctly due to this reason. The unit tests should have been enough testing for the security of the system, however if this was to be deployed onto a real network, more testing would need to be carried out.

6. Limitations and Improvements

6.1 Blockchain

As this technology is still not fully developed yet, and due to not enough time being able to learn all there is on blockchain, there are a few limitations to point out from this application. Firstly, when a user wishes to transfer ownership of their logbook to another Ethereum account, they have to enter this by text into the dialog provided. In Solidity there is no way of checking if an Ethereum account is valid, therefore if a user writes an incorrect address the logbook would be lost forever and could never be accessed again. This would be a major concern and make the application obsolete, as users would be losing their proof of ownership of a car, which would be very costly for them. To overcome this issue, the application could have users create an account that is linked to their MetaMask account. To transfer a logbook, they could then search for the account they wish to send it to, and upon selecting them their public key Ethereum address would be automatically retrieved by the application, so that the user is not responsible for ensuring the account address is correct.

Another major concern with this application, is that because the Ethereum blockchain is public and duplicated across many nodes, programmers have access to all variables and functions declared in a smart contract. Although they would not be able to steal or create logbooks, they would be able to read the data inside the variables. As logbooks contain private user information, this would breach data confidentiality resulting in a loss of trust for the users, not to mention that it would also be illegal. Some ideas that may solve this issue, would be to deploy this application onto a private network. Not much research was done on this, however private networks are partly controlled by a centralised party, who give users access to certain information. The users' private data could be kept private this way. Another idea could be that the data could be encrypted before being put onto the blockchain, with the decryption key being off-chain. Users could have another private key they can decrypt their private information with. More research is needed on using blockchain for smart property to overcome some of these issues identified.

6.2 Application

Further improvements could be carried out on the application code. As JavaScript and Solidity was new to me, I think that the code could be cleaned up a little and better practise could be implemented to improve performance, security, code standards etc. The CSS code could also be improved to make the design of the application look more professional. As not much time was allocated, a more thoughtful approach to the design would ensure that it looks better.

7. Review and Reflection

7.1 Background research

The background research conducted prior to this project being carried out greatly influenced the way this application was implemented, as well as the development frameworks and libraries that were used. I gained a considerable amount of knowledge about blockchain technology and its concepts to gain enough understanding of how to implement this project. Reading the documentation for Solidity, Web3 and Ethereum taught me how to program this application and defiantly helped influence the way it was developed. It has helped me further improve my programming skills and has widened my knowledge on writing in different languages. I think more research on private blockchains and cryptography would have been helpful to find out if some of the issues discussed previously could have been solved, as even though a high amount of research was conducted, not everything about blockchain was fully understood, resulting in some of the issues raised being figured out later in development.

7.2 Planning and Development

Throughout the development of this project, an agile approach was maintained and deliverables were produced in time box frames, tested, and re done if necessary. Some parts of the development required more time boxes than other parts, practically when writing the JavaScript code, as this is where most of the application errors occurred. This slightly pushed back the time that some deliverables were produced from the original project plan, however enough time was allocated to account for this so all the deliverables were still produced on time for the deadline. Throughout the project some sections were revisited, such as the requirements when the design feedback was obtained, as the system requirements needed to be adapted

to work with the new design. As I chose an agile approach this adaptability and volatility was expected, therefore this methodology worked well with this project and ensured the application was to a good standard.

7.3 Conclusion

From conducting this project, I think that blockchain technology can be a very powerful tool for securing and protecting smart property to manage ownership rights. Although there are some major concerns that need to be addressed, with more time and research I believe this can be resolved. If this was to be implemented in the real world, it would provide many benefits over the current V5C paper logbooks. The time it takes to edit details and transfer ownership would be cut down dramatically, and the cost of doing so would also be decreased. Therefore, this could provide businesses with ways to decrease their operational costs, and also make the cost to customers cheaper as well, making it much more simpler and a better experience for everyone.

To conclude I think this project overall went successfully. Although some deliverables were pushed back in the timeline, all the deliverables were still produced to a good quality. This was a result from gaining enough understanding before starting the project through the research, and also learning the Solidity language enough before starting the programming. The tests produced ensured that all the functionality was implemented correctly and that the security functions created worked well with the application. When conducting the black box testing, all the requirements created were successfully implemented, with the expectation of one error not showing all the logbook details, which shows that the application is of good standard and went according to plan. This project has been very challenging and has given me much more technical understanding in this field and I think the whole process has gone well and I am proud to have completed it.

References

- Barski, C. and Wilmer, C., (2014). *Bitcoin for the Befuddled*. No starch press.
- Beck, R., Czepluch, J.S., Lollike, N. and Malone, S., (2016), May. Blockchain-the Gateway to Trust-Free Cryptographic Transactions. In *ECIS* (p. ResearchPaper153).
- Bussmann, O. (2017). *A public or private blockchain? New Ethereum project could mean both*. [online] Available at: <https://www.americanbanker.com/opinion/a-public-or-private-blockchain-new-ethereum-project-could-mean-both> [Accessed 22 Nov. 2017].
- Buterin, V. (2014). A next-generation smart contract and decentralized application platform. *white paper*.
- Christidis, K. and Devetsikiotis, M. (2016). Blockchains and Smart Contracts for the Internet of Things. *IEEE Access*, 4, pp.2292-2303.
- Chuen, D.L.K. ed., (2015). *Handbook of digital currency: Bitcoin, innovation, financial instruments, and big data*. Academic Press.
- Cuccuru, P. (2017). Beyond bitcoin: an early overview on smart contracts. *International Journal of Law and Information Technology*, 25(3), pp.179-195.
- Dev, J.A., (2014), May. Bitcoin mining acceleration and performance quantification. In *Electrical and Computer Engineering (CCECE), 2014 IEEE 27th Canadian Conference on* (pp. 1-6). IEEE.
- Estevão, P. (2015). *The Bitcoin Transaction Life Cycle*. [online] Imgur. Available at: <https://imgur.com/a/BCvZr> [Accessed 27 Apr. 2018].
- EthDocs. (2018). *Ethereum Homestead Documentation — Ethereum Homestead 0.1 documentation*. [online] Available at: <http://www.ethdocs.org/en/latest/> [Accessed 9 Apr. 2018].
- Gervais, A., Karame, G., Capkun, V. and Capkun, S. (2014). Is Bitcoin a Decentralized Currency?. *IEEE Security & Privacy*, 12(3), pp.54-60.
- Gov (2016). *Distributed Ledger Technology - Beyond Blockchain*. [online] Gov.uk. Available at: https://www.gov.uk/government/uploads/system/uploads/attachment_data/file/492972/gs-16-1-distributed-ledger-technology.pdf [Accessed 20 Nov. 2017].
- Gov (2017). *Vehicle registration: New and used vehicles - GOV.UK*. [online] Available at: <https://www.gov.uk/vehicle-registration/new-and-used-vehicles> [Accessed 22 Nov. 2017].

Gupta (2017). A Brief History of Blockchain. [online] Harvard Business Review. Available at: <https://hbr.org/2017/02/a-brief-history-of-blockchain> [Accessed 17 Oct. 2017].

Kim, H. and Laskowski, M. (2017). *A Perspective on Blockchain Smart Contracts: Reducing Uncertainty and Complexity in Value Exchange*.

Kishigami, J., Fujimura, S., Watanabe, H., Nakadaira, A. and Akutsu, A. (2015). The Blockchain-Based Digital Content Distribution System. *2015 IEEE Fifth International Conference on Big Data and Cloud Computing*.

Marr, B. (2018). *Forbes Welcome*. [online] Forbes.com. Available at: <https://www.forbes.com/sites/bernardmarr/2018/02/19/the-5-big-problems-with-blockchain-everyone-should-be-aware-of/#505cf89d1670> [Accessed 20 Apr. 2018].

Metamask. (2018). *MetaMask*. [online] Available at: <https://metamask.io/> [Accessed 9 Apr. 2018].

Morabito, V., (2017). Business Innovation Through Blockchain. *Cham: Springer International Publishing*.

Nakamoto, S., (2008). Bitcoin: A peer-to-peer electronic cash system.

Papa, J. (2017). *lite-server*. [online] npm. Available at: <https://www.npmjs.com/package/lite-server> [Accessed 9 May 2018].

Pixabay. (2018). *Free Image on Pixabay - Steering Wheel, Steering, Wheel*. [online] Available at: <https://pixabay.com/en/steering-wheel-steering-wheel-150137/> [Accessed 9 Feb. 2018].

Prusty, N., (2017). *Building Blockchain Projects*. Packt Publishing Ltd.

Sammons, J. ed., (2015). *Digital Forensics: Threatscape and Best Practices*. Syngress.

Shermin, V. (2017). Disrupting governance with blockchains and smart contracts. *Strategic Change*, 26(5), pp.499-509.

Solidity. (2018). *Solidity — Solidity 0.4.23 documentation*. [online] Available at: <https://solidity.readthedocs.io/en/v0.4.23/> [Accessed 2 Mar. 2018].

Swan, M. (2016). *Blockchains may replace the institutions that safeguard commercial activities*. [online] LSE Business Review. Available at: <http://blogs.lse.ac.uk/businessreview/2016/03/31/blockchains-may-replace-the-institutions-that-safeguard-commercial-activities/> [Accessed 19 Nov. 2017].

Swan, M., (2015). *Blockchain: Blueprint for a new economy*. " O'Reilly Media, Inc."

Szabo, N. (1994). Smart Contracts [online] Available at: <http://szabo.best.vwh.net/smart.contracts.html>

Truffle. (2018). *Documentation / Truffle Suite*. [online] Available at: <http://truffleframework.com/docs/> [Accessed 9 Apr. 2018].

Web3js. (2018). *web3.eth — web3.js 1.0.0 documentation*. [online] Available at: <https://web3js.readthedocs.io/en/1.0/web3-eth.html> [Accessed 2 Apr. 2018].

White, G. (2017). Future applications of blockchain in business and management: A Delphi study. *Strategic Change*, 26(5), pp.439-451.

Yli-Huumo, J., Ko, D., Choi, S., Park, S. and Smolander, K. (2016). Where Is Current Research on Blockchain Technology?—A Systematic Review. *PLOS ONE*, 11(10), p.e0163477.

Appendix A: Gantt Chart

