

Newcastle University

Storing and managing user privacy preferences using blockchain

Joel Gibson (B19204600)

May 2022

BSc Computer Science

Supervisor – Dr Charles Varlei Neu

Word Count: 14,995 (excluding headings)

Abstract

Smart buildings represent a fast-growing market, but with their increasing prevalence comes privacy concerns for their occupants. The vast array of sensors commonly found within these buildings means more data is being collected about occupants whether they know it or not. In particular they can be used to gather data about occupants that they may not feel comfortable disclosing, like - for example - details of their daily routine or occupancy reports for their office.

This dissertation looks at the application of blockchain technology to help preserve user privacy within smart buildings. It produces a system which provides smart building occupants a way to correctly specify and manage their privacy preferences, and control with whom they are shared. It is intended to be linked to smart building management systems, with the goal being that when privacy preferences are shared with smart building management systems, the smart building will ensure all data collected by sensors respects these preferences defined by occupants.

In this dissertation a summary of relevant research is presented, followed by the development of a decentralised application which allows users to specify and share their privacy preferences which are stored securely on a blockchain. To conclude an evaluation of the produced system is performed to assess its capability in helping preserve the privacy of smart building occupants, and future work is proposed to improve the system.

Declaration

"I declare that this document represents my own work except where otherwise stated"

Joel Gibson

Acknowledgments

I would like to thank my supervisor Charles Neu for the inspiration for this project, as well as help and support throughout.

Table of Contents

1. Introduction.....	8
1.1 Context and Motivation	8
1.2 Aim and Objectives	8
1.3 Paper Structure	9
2. Background.....	12
2.1 Privacy	12
2.2 Smart Buildings and Sensors	13
2.3 Blockchain	13
2.3.1 Smart Contracts	14
2.3.2 Decentralised Applications	14
2.4 Similar Works	14
2.5 Cryptography	15
3. Methodology	16
4. Design.....	17
4.1 Overall System Model	17
4.2 Smart Contract Design	18
4.3 JavaScript Methods Design	20
4.4 DApp Design.....	22
5. Technology and Tools	25
5.1 Truffle Suite.....	25
5.2 MetaMask	25
5.3 Solidity.....	25
5.4 JavaScript	26
5.5 React	26
5.6 Visual Studio.....	26
5.7 GitHub	27
6. Implementation	28
6.1 Smart Contract	28
6.1.1 setPreferences and getPreferences 1.0	28
6.1.2 addApprovedAddress and removeApprovedAddress	30
6.1.3 getPreferences 2.0	32
6.1.5 getApprovedAddresses	33
6.1.4 deletePreferences and deleteAllPreferences	33
6.1.6 Conclusion	34

6.2 Cryptographic Methods	35
6.2.1 Encrypt and Decrypt.....	35
6.2.2 Generate Key.....	37
6.2.3 Hash Key	38
6.3 DApp.....	38
6.3.1 UI 1.0	39
6.3.2 UI 2.0	40
6.3.3 UI 3.0	42
6.3.4 UI 3.1	42
6.3.5 UI 3.2	44
7. Testing.....	45
7.1 Smart Contract Testing	45
7.2 UI 1.0 Testing	46
7.2 UI 3.2 Testing	47
7.3 Improvements to Testing Strategy	47
8. Results and Evaluation	48
8.1 Resulting System	48
8.2 Evaluation of Results.....	53
8.2.1 Smart Contract	53
8.2.2 Cryptographic Methods	54
8.2.3 DApp.....	55
8.2.4 Documentation	59
8.3 Satisfaction of Objectives.....	60
8.3.1 Objective 1	60
8.3.2 Objective 2	60
8.3.3 Objective 3	60
8.3.4 Objective 4	61
8.3.5 Objective 5	61
9. Conclusions	62
9.1 Reflection	62
9.2 Future Work	63
9.2.1 Re-design the DApp UI	63
9.2.2 API	63
9.2.3 Deployment.....	63
9.2.4 Smart Contract Security Analysis	63
9.2.5 Smart Contract Cost Analysis	63

9.2.6 Documentation and Guides	64
10. References	65

Table Of Figures

Figure 1: Feature-Driven Development Overview Source: [28]	16
Figure 2: Overall System Model Diagram	17
Figure 3: Pseudocode mappings to store user preferences, and approved addresses.....	18
Figure 4: getPreferences Function Pseudocode	19
Figure 5: addApprovedAddress Function Pseudocode.....	20
Figure 6: removeApprovedAddress Function Pseudocode	20
Figure 7: encryptPreferences Function Pseudocode.....	21
Figure 8: decryptPreferences Function Pseudocode.....	21
Figure 9: genKey Function Pseudocode.....	22
Figure 10: hashKey Function Pseudocode	22
Figure 11: DApp UI Wireframe	23
Figure 12: Initial Implementations of setPreferences and getPreferences.....	28
Figure 13: setPreferences and getPreferences using address + key	29
Figure 14: setPreferences and getPreferences BEFORE approved addresses.....	29
Figure 15: keyInUse helper function.....	30
Figure 16: usedKeys mapping	30
Figure 17: Initial implementation of addApprovedAddress Function	30
Figure 18: Finished implementation of addApprovedAddress Function.....	31
Figure 19: approvedAddressExists helper function.....	31
Figure 20: Finished implementation of removeApprovedAddress function	32
Figure 21: Finished implementation of getPreferences function.....	32
Figure 22: Finished getApprovedAddresses function	33
Figure 23: Finished implementation of deletePreferences function.....	33
Figure 24: Finished deleteAllPreferences function.....	34
Figure 25: crypto-js Encrypt and Decrypt functions	35
Figure 26: node-forge Encrypt and Decrypt functions	36
Figure 27: aes-js Encrypt and Decrypt Functions.....	37
Figure 28: Finished implementation of genKey function	37
Figure 29: Finished implementation of hashKey function.....	38
Figure 30: DApp Wireframe split into React Components	38
Figure 31: YourAccount Component Pseudocode.....	39
Figure 32: First implementation of YourAccount Component	40
Figure 33: DApp UI 1.0	40
Figure 34: Navigation bar before and after Bootstrap	41
Figure 35: Preferences Form before and after Bootstrap	41
Figure 36: DApp UI 2.0	41
Figure 37: DApp UI 3.0.....	42
Figure 38: window.confirm example with DApp output	42
Figure 39: window.alert example with DApp output	43
Figure 40: Confirmation Modal Example	43
Figure 41: Failure Modal Example	43

Figure 42: DApp UI 3.1	44
Figure 43: DApp UI 3.2	44
Figure 44: Test Logs for setPreferences Function.....	45
Figure 45: Test Logs for UI 1.0 Retrieve Button	46
Figure 46: Test Logs for UI 3.2 Retrieve Button	47
Figure 47: MetaMask login screen.....	48
Figure 48: Key entered into Key Input Box	49
Figure 49: Using the form to define preferences	49
Figure 50: Adding an approved address	50
Figure 51: Approved Address dropdown after adding an address.....	50
Figure 52: User's blockchain address as displayed on the DApp.....	50
Figure 53: Console command to retrieve user's preferences.....	51
Figure 54: Returned encrypted preferences	51
Figure 55: Error when trying to getPreferences from non-approved address.....	52
Figure 56: Your Account Component.....	55
Figure 57: Secret Key Management Component.....	55
Figure 58: Approved Addresses Component	56
Figure 59: Add New Address Error Messages.....	56
Figure 60: Add/Remove Address Confirmation Modals	57
Figure 61: Preferences Form Component	58
Figure 62: Delete All Preferences Component	58
Figure 63: Delete All Preferences Confirmation Modal.....	59
Figure 64: Excerpt from README.md	59

Glossary

DApp - Ethereum Docs describes decentralised applications as follows: “A decentralised application (or dapp) is an application built on a decentralized network that combines a smart contract and a frontend user interface. A dapp has its backend code running on a decentralized peer-to-peer network. Contrast this with an app where the backend code is running on centralized servers” [1].
See background for more detail.

Crypto Wallet – Coinbase defines crypto wallets like so: “Crypto wallets store your private keys, keeping your crypto safe and accessible. They also allow you to send, receive, and spend cryptocurrencies like Bitcoin and Ethereum” [2].

1. Introduction

1.1 Context and Motivation

Many in the computing world have stressed the ‘growing concern’ around maintaining user privacy in an increasingly data-driven society [3]. Thus, enabling users to manage their privacy preferences and correctly specify how they’d like their data to be used is an important issue [4]. Smart buildings are an area where such concerns for privacy are being raised [5].

Smart buildings are a fast-growing market, and their growing prevalence means more data is being collected about people whether they know it or not [6]. This of course raises many privacy concerns, particularly regarding the disclosure of data that a user may not feel comfortable disclosing [6]. A wide variety of sensors and systems found in smart buildings could be used to track the location or behaviours of a person, including smartphone usage, occupancy detection, CO2 monitoring, light-level monitoring, temperature monitoring, and even smart-meter readings [5][6]. Consequently, we must look to build smart building systems in such a way that they meet the privacy preferences of their occupants, allowing the occupants control over the data collected about them [6].

This project looks to address such privacy concerns surrounding smart buildings by producing a system which enables users to easily specify their privacy preferences and share them with smart building systems. This will provide users fine-grain control over their privacy within these smart buildings and puts the control back in their hands regarding what data is collected about them. These privacy preferences will themselves also be kept private, to alleviate any fears users may have of being discriminated against because of their choices. For example, an employee working in a smart building may not feel comfortable with regular occupancy readings for their office but may feel too intimidated by their employer to set their preferences as such. Preferences are thus kept private, and the intention is that when preferences are shared with smart building management systems they are simply used to ensure data collected by sensors within the smart building respects the preferences.

1.2 Aim and Objectives

This project aims to produce:

A system using blockchain, which will allow users to store and manage their privacy preferences.

This is a modification of the original project aim which was simply to produce “*a system using blockchain to store user privacy preferences*”. As the nature of the project became clearer, it became necessary to add management to the aim, as the user should be able to modify, control access to, and delete their preferences; these aspects were not adequately represented by the phrase “*to store user privacy preferences*”. It is intended for the system to be linked to smart building management systems, so that any data collected by the smart building respects the privacy preferences of its occupants, which they define through this project’s application.

The aim was split into the following objectives, which are accompanied by a brief description of how they contribute towards fulfilling the aim:

1. Research and explore blockchain technology and its potential uses in preserving user privacy.

This objective looks to develop an understanding of the technologies and concepts involved in the project, and how they can be applied to preserve privacy, in order to establish a knowledge base for the project.

2. Investigate and learn about implementing blockchains, decentralised applications, and smart contracts. Explore the tools and processes involved.

To be able to implement the system, one must learn how to convert knowledge of the technologies into actual implementations. This includes learning about the technologies themselves, as well as tools, platforms, development practices, and so on.

3. Use the knowledge gained to produce an overview of the system.

This encompasses the planning stage: producing requirements and designing the system. This ensures a clear picture is formed of the system, including all its constituent parts, laying out what needs to be developed.

4. Produce the system outlined using the technologies and processes learned about.

This represents the implementation stage, where the system overview produced in objective 3 is converted into a working system.

5. Discuss the ways the system could be expanded, and its potential role as part of a bigger picture in maintaining user privacy preferences in smart buildings.

The purpose and use of this project needs to be clear and understandable, with its role in conjunction with smart building management systems emphasised. This paper looks to achieve this objective.

1.3 Paper Structure

This document is divided like so:

Chapter 1 – Introduction

An introduction to the dissertation explaining the context and motivation for the project, as well as its aims and objectives. It also provides an outline of the paper's structure to aid the reader in understanding and navigating it.

- Context and Motivation
- Aim and Objectives
- Paper Structure

Chapter 2 – Background

Here necessary background for the project is set out, covering the concepts and technologies involved.

- Privacy
- Smart Buildings and Sensors
- Blockchain
- Similar Works
- Cryptography

Chapter 3 – Methodology

This chapter details the chosen methodology and approach taken to developing the system.

Chapter 4 – Design

This is the first step of development, planning the implementation of the system. This includes producing smart contract requirements, DApp requirements, and UI designs.

- Overall System Model
- Smart Contract Design
- JavaScript Methods Design
- DApp Design

Chapter 5 – Technology and Tools

A discussion of the technology and tools chosen for this project, and why they were chosen.

- Truffle Suite
- MetaMask
- Solidity
- JavaScript
- React
- Visual Studio
- GitHub

Chapter 6 – Implementation

An account of the development process for the system.

- Smart Contract
- Cryptographic Methods
- DApp

Chapter 7 – Testing

A discussion of the role of testing, and how it was approached in the project.

- Smart Contract Testing
- UI 1.0 Testing
- UI 3.2 Testing
- Improvements to Testing Strategy

Chapter 8 – Results and Evaluation

A discussion of the produced system, and evaluation of how successfully the project's objectives were fulfilled.

- Resulting System
- Evaluation of Results
- Satisfaction of Objectives

Chapter 9 – Conclusions

A detailed reflection on the project, and a discussion of future work that can be done surrounding it.

- Reflection
- Future Work

2. Background

This chapter covers the necessary background for the project. Research involved using a range of materials to learn about the concepts and technologies for the project, from published research to more informal sources like forums and tutorials.

2.1 Privacy

This paper heavily relies on *The SITA principle for Location Privacy – Conceptual Model and Architecture* [7], with the privacy preference options presented to the user within the application following the model in this paper.

The paper discusses the issue of privacy within location-based services (LBSs) like Facebook and Google. It raises the concern that for most popular LBSs, the user has a lack of privacy options and thus is pushed towards either enabling or disabling full access to their location. The paper goes on to produce its own conceptual model to address the lack of control users have over their location privacy.

The model breaks down location data into 4 dimensions:

- **Spatial:** Where? The location.
- **Identity:** Who? The identity of the user.
- **Temporal:** When? The time the data was recorded.
- **Activity:** What? What the user was doing at the time.

It then divides these dimensions into 5 levels, which represent sharing varying amounts of data for that dimension. The levels are:

- **0:** No Information.
- **1:** Aggregation.
- **2:** Obfuscation.
- **3:** Regulation.
- **4:** Full Information.

Levels 0 and 4 are self-explanatory, with either none or all data recorded for that dimension being shared. Aggregation refers to grouping the data, for example if level 1 was chosen for the temporal dimension the time attached to the user's location data may be rounded to the hour, or possibly even the day. Obfuscation sees randomness added to the data, for example when chosen for the identity dimension, the user's details may be slightly changed. Say their age was modified by a couple of years to make them less identifiable. Regulation applies a set of rules to the data, for example when chosen for the spatial dimension location data may not be recorded within a certain distance of the user's home or place of work.

Being able to choose the level of data collected for each individual dimension provides the user fine-grain control of the location data collected about them. The privacy preferences referred to in the title, aim, and objectives are in the form of these dimensions, with this project's application allowing the user to choose and manage their preferred levels for these dimensions and share them with smart building systems as they see fit.

2.2 Smart Buildings and Sensors

A smart building is defined as a building that “uses technology to enable efficient and economical use of resources, while creating a safe and comfortable environment for occupants” [8]. It is a fast-growing industry, and thus – increasingly - more people are using smart buildings on a daily basis. Concerns however have been raised about the privacy of occupants, with smart buildings being fitted with a range of sensors from CO₂, temperature, and light-level monitoring, to occupancy detection [9]. Such sensors can be used to collect information about users that they may not feel comfortable disclosing [6]. Using sensor data, a user's daily routine could be monitored like when they arrive in the office, what time they normally leave for lunch, when they leave to go home, how much time they spend at their desk etc [10]. The idea of being able to extrapolate all this data isn't even that far-fetched, with papers even being published about such potential privacy issues with smart meters [5]. Many smart buildings nowadays come equipped with a complex array of these sensors, and thus it is a serious data privacy issue, and one that only continues to grow with the industry.

2.3 Blockchain

Blockchain is a distributed ledger which works on a peer-to-peer network, so that records are not in control of a centralised organisation (like a bank or social media company) [11]. It is essentially a decentralised database keeping a record of transactions, managed by a large network of computers [12]. Blockchain features a number of key characteristics, including:

- **Decentralisation:** In blockchain, control is no longer in the hands of a central authority but is shared across a distributed network. No longer needing to put your trust in a central authority creates what is known as a ‘trustless environment’. Each member maintains their own copy of the blockchain, meaning alterations must be approved by a consensus of the members [13]. This also prevents there being a single point of failure [11], a problem faced by many centralised systems.
- **Persistency:** Blockchain transactions can be quickly validated and invalid transactions will not be admitted to the blockchain by honest parties [14]. Once admitted it is nearly impossible to alter or delete a transaction (immutability/tamper-proof) [12].
- **Anonymity:** Users interact with blockchain using a generated address, which does not reveal anything about the user's real identity. Technically users are pseudo-anonymous, as transactions could be traced back to a user, thus linking their identity to their address [3].
- **Transparency:** Records are open for anybody on the network to read [11].
- **Auditability:** As all users have access to transaction records, it is straightforward to track and audit transactions [11].

Blockchains can be broken down into two main categories: permissionless and permissioned. Permissionless (public) blockchains are open to the public, and anyone can join them. Bitcoin and Ethereum are examples of permissionless blockchains. In permissioned blockchains (such as private/consortium blockchains) only known, trusted parties can join. They can be used by organisations to have their own private blockchain where only members of the organisation can join [11] [12]. As they are fully controlled by the organisation, they can be configured to meet their specific needs, for example eliminating the transaction fees associated with public blockchains like Ethereum or increasing throughput [15].

2.3.1 Smart Contracts

IBM defines smart contracts as “programs stored on a blockchain that run when predetermined conditions are met” [16]. They “facilitate, execute, and enforce the terms of an agreement” [17], whilst not requiring a trusted third party or middleman to facilitate like more traditional contracts. Despite being the first application of blockchain [11] Bitcoin isn’t the best platform for smart contract development [13], with newer blockchains built to support smart contracts. For example, Ethereum has a built-in Turing-complete programming language [18], with code being executed in the Ethereum Virtual Machine [13]. This has made Ethereum the most popular platform for smart contracts [17], with high-level languages such as Solidity emerging to support the writing of smart contracts [13].

2.3.2 Decentralised Applications

Ethereum Docs describes decentralised applications as follows: “an application built on a decentralized network that combines a smart contract and a frontend user interface (UI). A Dapp has its backend code running on a decentralized peer-to-peer network. Contrast this with an app where the backend code is running on centralized servers” [1]. A DApp is simply an application where the code runs on the blockchain via a smart contract.

There is plenty of discussion surrounding DApps and the use of blockchain in applications, particularly regarding weighing up the need for blockchain in your application. Some applications may benefit from blockchain’s properties such as being trustless, tamperproof, anonymous, and auditable, but this of course comes at the cost of the extra complexity associated with blockchain [19]. Thus, it is necessary to not only discuss if your system needs blockchain [20] but also HOW much blockchain your system needs [21].

2.4 Similar Works

Decentralizing Privacy: Using Blockchain to Protect Personal Data is a paper which addresses privacy concerns arising due to the increasing amount of user data stored by both public and private companies [3]. It combines blockchain with non-blockchain technologies to produce a system looking to overcome some common privacy issues. Firstly, it ensures that users have ownership over their own data. With a centralised database, the service in control of the database has full ownership and unlimited access to the data. Blockchain’s decentralised nature places ownership back into the user’s hands, and they get to delegate access permissions to services as they see fit. This is a crucial part of what this project’s system looks to achieve as well, giving the user control of their privacy preferences and who they share them with. Secondly, the paper embraces data transparency and auditability. With

a centralised database it can be unclear what data is being collected about the user, how it is being used, and by who. Blockchain is transparent by nature, with transaction logs being open for everyone to view. This makes it easily visible who is accessing a user's data, when they accessed it, and how. Once again this is a core value of this project's system, being able to see who is accessing and using your privacy preferences, with changes to them being clearly visible and traceable. This paper influenced the thinking behind this project, which also looked to produce a blockchain system which can be linked to off-blockchain systems (smart building management systems). It also reinforced how blockchain can help to protect personal data, and thus why it is being used in this project

A Blockchain-based approach for matching desired and real privacy settings of social network users [22] is another useful paper, and one with a similar goal to this project. It again reinforces the ability for blockchain's decentralised nature to place control back into the hands of users. In this case it is applied to privacy settings for social media. What stops a social media platform from manipulating your privacy settings for their own gain? They store your settings, and if they were to change them, you'd have no way to prove they did. Perhaps you changed them and simply forgot? With settings back in the hands of the user, only they can change them, with the auditability of blockchain allowing you to track when changes were made. This paper presents a model for interaction between a social media and a user via a smart contract, and inspired the model produced for this project.

Where my project differs from these papers is that it looks to apply blockchain to preserve privacy in smart buildings, because - as mentioned in the introduction - smart buildings are a growing market with serious implications on user privacy. These papers clearly highlight that blockchain and decentralisation can help to protect the privacy of users and give them control of their own data. Interestingly, however, the transparency of blockchain proves a challenge to this project's proposed system. Although the ability to view and audit changes is crucial, the privacy preferences the system will deal with must be private themselves. Only you and whoever you choose to share your preferences with should know what they are, but blockchain transparency allows anyone to read your transactions and extract your preferences. Reading these papers highlighted this major problem and demonstrated that some encryption would be required for the project.

2.5 Cryptography

Cryptography is the study and implementation of processes, which manipulate data for the purpose of hiding and authenticating information [23]. As discussed in the previous section it became clear that the project would require some use of cryptography to overcome blockchain's transparency without sacrificing auditability. Thus, encryption is necessary so the user's privacy preferences are unreadable when viewing transactions but can be decrypted by anyone with authorised access.

At the time of development AES encryption is the most widely used symmetric encryption scheme. It is both generally faster than counterparts, as well as being the most secure with the largest key size [24] [25]. AES being symmetric means the same key can be used to encrypt and decrypt the data. This is useful for this project, as sharing the key is an easy way to allow another party to decrypt and read your preferences.

The most secure hashing algorithm at the time is SHA-3, released in 2015 [26]. Hash functions transform data (plaintext) into a fixed-length output (hash value) which is unrecognisable from the input. Hence, they are another way of hiding information, like making the user's secret key unreadable in transaction logs. Hash functions are irreversible, their result can't be used to get the original value back.

3. Methodology

As the project was going to require learning new technologies the requirements were likely to change as the project progressed, so it was clear the flexibility and adaptability offered by agile development methodologies [27] would be crucial. Therefore, the methodology used during the development of this project was heavily based on Feature-Driven Development (FDD). FDD is an agile methodology which uses iterative development cycles focused on individual features [28].

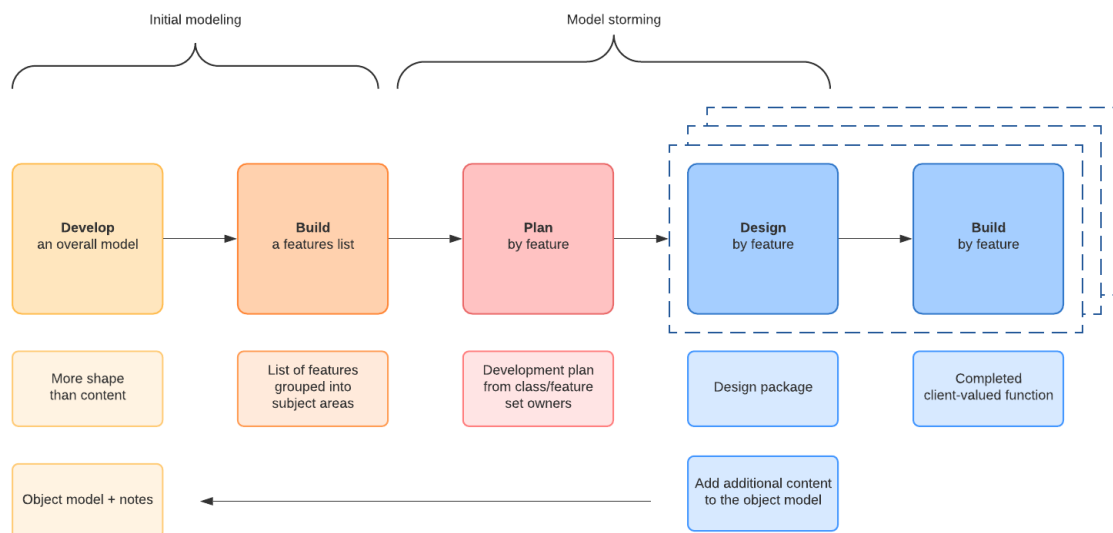


Figure 1: Feature-Driven Development Overview Source: [29]

As shown in Figure 1 FDD has 5 key stages [28]:

1. Develop an overall model.
2. Build a features list.
3. Plan by feature.
4. Design by feature.
5. Build by feature.

Stages 3-5 are repeatedly iterated for each feature until all features are implemented and your overall system has been developed. Despite not being intended for single developers [29], FDD aligned really well with the approach set out for this project. For example, objective 3 (“Use the knowledge gained to produce an overview of the system.”) perfectly matches stage 1 of FDD, so continuing with FDD’s other stages was a logical progression. Breaking down the system into smaller features makes development much easier to coordinate, and the time spent individually planning each feature makes the approach extremely thorough and leads to less unforeseen bugs and problems.

4. Design

This chapter encompasses stages 1-3 of Feature-Driven Development (FDD): develop an overall model, build a features list, and plan by feature.

4.1 Overall System Model

The overall model for this project was developed by considering the interactions between different parts of the system and the different parties that would be using it.

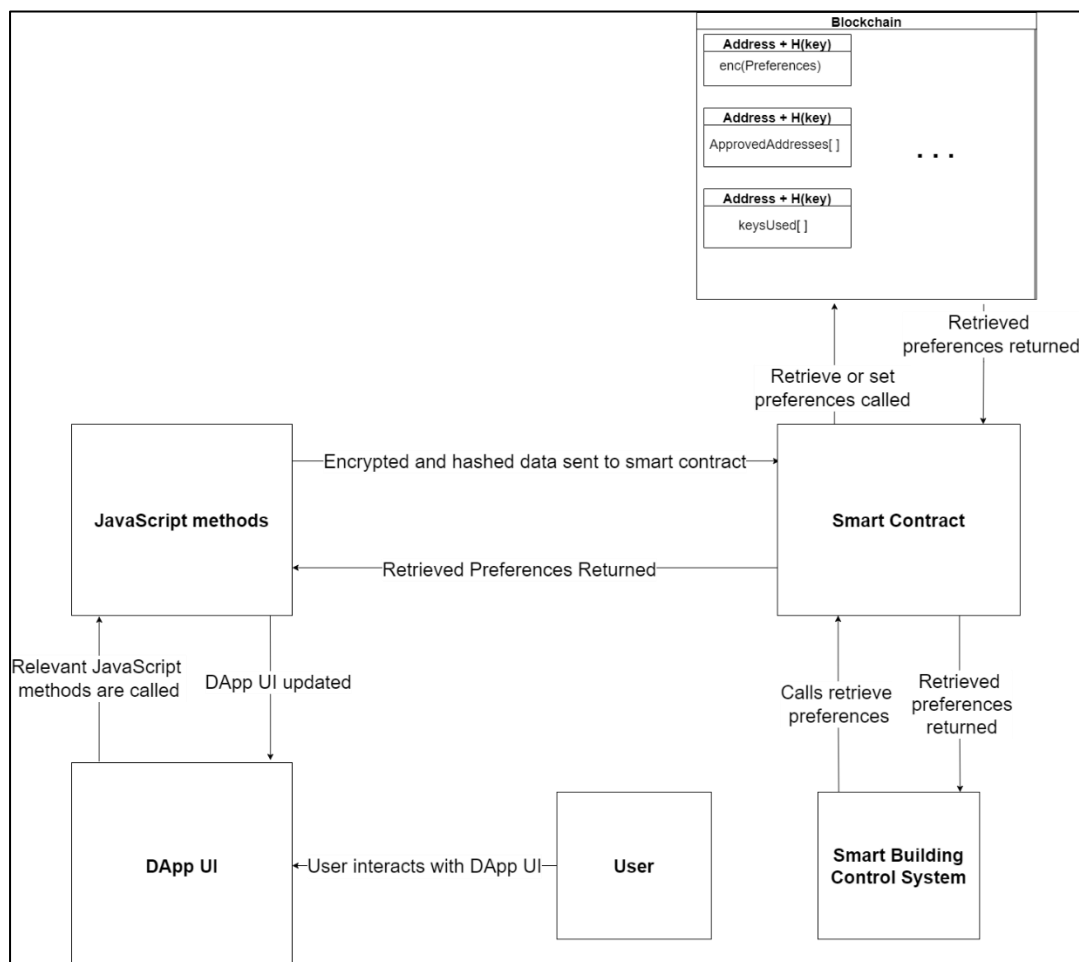


Figure 2: Overall System Model Diagram

Figure 2 is the resulting diagram of the overall system model. It breaks down the system into 4 key parts, and 2 parties that will use it. The key parts are the DApp UI, JavaScript methods, the smart contract, and the blockchain itself. The 2 parties are the ordinary user who defines and manages their privacy preferences using the DApp, and the smart building control system which interacts directly with the smart contract to retrieve the privacy preferences shared by its occupants.

The smart contract handles all interaction with data stored on the blockchain, whether that is retrieving preferences, storing or modifying preferences, or managing access to preferences. As mentioned above smart building control systems will interact with this contract directly by calling the relevant commands. The access of preferences by smart building systems is controlled by the user and owner of those preferences. Users can manage a list of 'approved addresses' which are allowed to retrieve their preferences from the blockchain. The smart contract will allow only these approved

addresses to retrieve a user's preferences, ensuring that the user has full control over who can access them.

It is important to mention that users can manage as many different sets of preferences as they want, each with its own secret key. So – for example - a user can define and manage some privacy preferences under one key which they share with a certain smart building, and under another key they manage different preferences which they share with another smart building. This allows the user to share different preferences with different smart buildings, or even with different rooms in the same building. Regular users don't ever directly interact with the contract, instead they make inputs via the DApp UI, and JavaScript methods work as an intermediary calling the necessary commands on behalf of the user.

The diagram does an excellent job of breaking down the system into the key parts that need to be considered and developed. The blockchain is handled by Ganache, meaning with just one click a blockchain can be initialised, leaving just the 3 other key parts to develop. In the remainder of this chapter these parts are broken down into features, and those features planned (FDD stages 2 and 3).

4.2 Smart Contract Design

The role of the smart contract in the system was to retrieve and modify data stored on the blockchain. This mostly involves the encrypted privacy preferences of users, but also includes lists of approved addresses as discussed above.

To continue following FDD the smart contract is broken down into a list of features, which in this case refers to individual functions. These features were planned by writing basic pseudocode implementations. The following are required features for the smart contract:

Set Preferences Function:

This function is called to store a set of privacy preferences on the blockchain. It is called both the first-time a set of preferences are stored, as well as when modifying already existing preferences. Preferences are stored in a mapping (dictionary) in which the key is the combination of the user's address and the hash of the secret key, and the value stored is the encrypted privacy preferences of the user (this can be seen in line 1 of Figure 3, more discussion of encryption and hashing can be found in the JavaScript methods design section). Only the user can define and modify their own preferences, and if the same key was used by another user they would simply be modifying their own set of preferences under this key.

1. `mapping(address + H(key) => enc(preferences)) userPreferences;`
2. `mapping(address + H(key) => address[]) approvedAddresses;`

Figure 3: Pseudocode mappings to store user preferences, and approved addresses.

This was a very-high priority function to be developed, as it is absolutely necessary for the system to work.

Get Preferences Function:

This function attempts to retrieve a set of preferences. This function can be used to retrieve the preferences of any user, but only approved addresses will be successful and have the encrypted preferences returned.

```
Function getPreferences(address userAddress, string memory hashKey) public  
returns(string encPreferences, string[] approvedaddresses){  
    IF msg.sender IN approvedAddresses[userAddress + hashKey]:  
        IF hashKey IN usedKeys[userAddress]:  
            IF userAddress = msg.sender:  
                return( userPreferences[ userAddresss + hashKey],  
                    approvedAddresses[userAddressss + hashKey]);  
            ELSE:  
                return( userPreferences[ userAddressss + hashKey]);  
        ELSE:  
            ERROR (KEY NOT FOUND)  
    ELSE:  
        ERROR CODE (READ PERMISSION NOT GRANTED)
```

Figure 4: getPreferences Function Pseudocode

Figure 4 showcases a pseudocode outline for the getPreferences function. It takes a user's address and secret key hash as parameters, which in combination should produce a mapping key as seen in line 1 of Figure 3. The address that called the function (msg.sender) will be checked to see if it is an approved address for these preferences, and the encrypted preferences will be returned only if it is.

This was also a very-high priority function to be developed, as it too is absolutely necessary for the system to work.

Delete Preferences Function:

The delete preferences function is necessary to allow the user to delete a certain set of privacy preferences. It is a self-explanatory function, which can only be called by the user to delete preferences of their own.

This was a medium priority function to be developed, not strictly necessary but it would be frequently used if implemented.

Delete All Preferences Function:

This is the 'nuclear option' allowing the user to delete all preferences stored under every one of their keys. It is an emergency measure only intended to be used if the user has forgotten their secret key and wishes to prevent their privacy preferences from being accessed anymore. Normally the user could just remove addresses from the approved addresses list that they no longer wanted to give access to, but this wouldn't be possible if the user has forgotten their secret key.

This was a low priority function to be developed, not necessary but useful to have.

Add Approved Address Function:

This function adds an approved address for a certain key (see Figure 5), to allow that address to retrieve the preferences stored under the key. This is how the user can manage who they share their preferences with. Much like with setPreferences only the user can manage the approved addresses for their preferences.

```
Function addApprovedAddress(address approvedAddress, string hashKey)
ACCESSMODIFIER return(bool success){
    approvedAddresses[msg.sender + hashKey] += approvedAddress
}
```

Figure 5: addApprovedAddress Function Pseudocode

This is a high priority function, a key part in ensuring users have control over who can read their preferences.

Remove Approved Address Function:

This removes an approved address for a certain key (see Figure 6), preventing that address from being able to retrieve those preferences anymore. Just like in addApprovedAddress, approved addresses can only be removed by the user.

```
Function removeApprovedAddress(address approvedAddress, string hashKey)
ACCESSMODIFIER return(bool success){
    approvedAddresses[msg.sender + hashKey] -= approvedAddress
}
```

Figure 6: removeApprovedAddress Function Pseudocode

This is also a high priority function, as once again it plays a key part in controlling who can read preferences.

4.3 JavaScript Methods Design

As discussed previously the JavaScript methods act as an intermediary between the DApp and the smart contract, responsible for calling the contract functions set out above on behalf of the user. Much of the JavaScript for the project would actually be a part of the DApp code, thus it wasn't explicitly planned as many questions would remain unanswered until after the DApp was implemented. For example, when calling setPreferences where would the preferences be obtained from, and how? The answer to such a question was dependent on the UI implementation, which at this point of planning was unclear.

Cryptographic methods, however, were planned as their functionality was clear regardless of UI implementation. The system required methods to encrypt and decrypt preferences, generate secret keys, and hash the secret key. The plans for these methods are discussed below.

Encrypt Preferences:

The system requires the preferences to be encrypted to overcome blockchain's transparency (as discussed earlier). If the setPreferences contract function was called with unencrypted preferences, the user's preferences would be on full display to anyone viewing the transaction logs on the blockchain. This would compromise the privacy of the user's preferences, hence the preferences must be encrypted.

A symmetric encryption scheme was chosen as users will need to share the encryption key with smart building systems. Consequently, asymmetric encryption is unsuitable, as in asymmetric schemes the private key must not be shared with others, but key sharing is a crucial part of this project's system. Knowledge of the key is required to access the user's preferences, as they are stored mapped under a composite key of the user's address and the hash of the key (as described in smart contract design). Once returned, the preferences will then be decrypted with the same key allowing them to be read and utilised by a smart building system.

```
Function encryptPreferences(preferences, secretKey){  
    return( encsecretKey(preferences));  
}
```

Figure 7: encryptPreferences Function Pseudocode

The pseudocode for this function (Figure 7) is straightforward, requiring the preferences and the secret key as parameters and simply returning the encryption of the preferences. AES was the chosen symmetric encryption scheme, as it was the most secure symmetric scheme at the time [24].

This function will be called automatically when the user wants to set/modify their preferences, and the encrypted result is what will be passed to the smart contract setPreferences function for storage. This is a necessity for the system to work as intended, as without it the user's preferences are not kept private and the system is effectively pointless.

Decryption Function:

As privacy preferences are stored encrypted on the blockchain, when retrieved they will still be encrypted. Hence, a decryption function is necessary so the preferences can be read.

```
Function decryptPreferences( encPreferences, secretKey){  
    return( decsecretKey(encPreferences));  
}
```

Figure 8: decryptPreferences Function Pseudocode

This function will simply AES decrypt the encrypted preferences to return the original plaintext preferences (see Figure 8). This function will be called automatically after preferences are retrieved. This too is a necessity for the system.

Generate Key Function:

As seen in Figure 9 this function produces an AES key used to encrypt preferences, which will also be hashed for use as part of the composite key in the smart contract.

```
Function genKey()  
    return(generateKey())  
}
```

Figure 9: genKey Function Pseudocode

This function is required to enable AES encryption and decryption, and thus is just as necessary.

Hash Key Function:

As mentioned previously, encrypted preferences are stored on the blockchain under a composite key which includes the hash of the secret key. Therefore, it is necessary for there to be a function to perform this hashing. Much like with the preferences, the key would appear within the transaction logs visible to everyone, so it must be hashed to overcome blockchain transparency.

```
Function hashKey(secretKey){  
    return( H(secretKey)  
}
```

Figure 10: hashKey Function Pseudocode

Much like the other cryptographic methods the pseudocode (Figure 10) for hashing the key is extremely simple, we feed in the secret key as a parameter and receive its hash in return. The most secure hashing standard for the time was chosen, SHA-3 [26].

4.4 DApp Design

Design of the DApp required identifying the necessary UI features to allow the user to manage their privacy preferences. The means the UI needed to provide a way for the user to do all the things the smart contract offers from getting and setting preferences, to adding and removing approved addresses.

This led to the production of a wireframe to showcase the simplest DApp UI implementation (Figure 11). It is the most bare-bones UI, providing access to all the system's features albeit in an unattractive way.

Figure 11: DApp UI Wireframe

This UI could then be broken down into all the necessary features for the DApp UI, discussed below.

1. Your Account:

A section which displays the blockchain address the user is signed into MetaMask with.

2. Secret Key Input Box:

An input box for the secret key. User can type in an existing key, or have a new key generated to fill the box.

3. Retrieve Button:

When pressed the smart contract `getPreferences` function will be called with the user's address and hash of the secret key in the input box. Successful retrieval will have the preferences form filled with the retrieved preferences, and the approved addresses drop-down filled with all approved addresses for those preferences.

4. Get New Key Button:

When pressed this button will call the `genKey` JavaScript function to generate a new AES key, which will fill the secret key input box.

5. Delete These Preferences Button:

When pressed the smart contract deletePreferences function will be called with the user's address and hash of the secret key from the input box.

6. Preferences Form:

Features one drop-down selection box for each SITA dimension, allowing the user to define their chosen level for each dimension.

7. Submit Button

When pressed this button will call the smart contract setPreferences function from the user's address, with the preferences from the form encrypted, and the secret key from the input box hashed.

8. Approved Addresses Drop-down:

Drop-down selection displaying all the approved addresses for this preferences set. This selection can be used to choose an approved address to delete by pressing the cross (approved address remove button).

9. Approved Address Remove Button:

When pressed this will call the smart contract removeApprovedAddress function on the selected approved address.

10. New Approved Address Input Box:

Input box for new approved addresses.

11. Approved Address Add Button:

When pressed this button will call the smart contract addApprovedAddress function on the approved address in the input box.

5. Technology and Tools

Choosing the correct technologies and tools is an important part in enabling the development of the system designed in the previous chapter. This section introduces the technologies and tools used in the final system, and why they were chosen.

5.1 Truffle Suite

The Truffle Suite consists of Truffle, Ganache, and Drizzle. Truffle is a development environment, asset pipeline, and testing framework; Ganache allows the user to run a local Ethereum blockchain for testing and development; and Drizzle is a collection of front-end libraries to aid in DApp front-end development [30]. As a whole the Truffle Suite provides fantastic all-round support for creating a DApp.

Many alternatives exist to the Truffle Suite, like Hardhat [31], Embark [32], and Quorum [33]. Quorum is a blockchain platform developed by JP Morgan [34], and it was seriously considered as an option. In particular its support of private transactions [35] was tempting as an alternative to encryption, however after consideration private transactions compromise traceability/auditability by hiding certain transactions from logs. Thus it wasn't used, as these are key properties of this project's system.

Ultimately, the quality and breadth of documentation, guides, and tutorials Truffle Suite offers made it the most appealing choice. Furthermore, it offers a range of 'Truffle boxes' containing boilerplate code to help you start your Dapp [36], making it even easier to start developing with Truffle. The choice of Truffle influenced many of the other technology and tool decisions, as it requires Node Package Manager (NPM) [37], and many of the tutorial Truffle boxes make use of MetaMask, Solidity, and JavaScript.

5.2 MetaMask

Truffle recommends the MetaMask browser extension - available for Chrome and Firefox - to interact with DApps from your browser. It allows users to manage their blockchain accounts and can act as a crypto wallet [38]. It works with any Ethereum-compatible blockchain, such as Ganache [39], and is by far the most popular tool of its kind [40].

MetaMask was chosen for a few key reasons: its ease of use, its compatibility with Ganache, and its use in many Truffle Suite tutorials. It suits the needs of the project and is compatible with the other technologies being used, making it the obvious choice.

5.3 Solidity

Solidity, a high-level object-oriented language with a similar syntax to JavaScript, was one of the first programming languages for blockchain [41]. It is the primary language of Ethereum [42], and thus there exists a massive amount of documentation, guides, and forums to consult and learn from.

Other languages like Vyper [43], and Rust [44] are frequently used for blockchain development, but the choice for this project was Solidity. The system would be using Ganache to host an Ethereum blockchain, and thus it seemed the obvious choice to use Ethereum's most common language.

Furthermore, the tutorials provided by Truffle all use Solidity, so it seemed sensible to continue using it for this project.

5.4 JavaScript

JavaScript is a lightweight multi-paradigm programming language [45]. It is the most used programming language in the world [46], being heavily used in web development.

One of the requirements of the Truffle Suite is NodeJS, an “asynchronous event-driven JavaScript runtime” which is “designed to build scalable network applications” [47]. In short it enables development of web servers using JavaScript [48]. As NodeJS is a requirement for Truffle to work, Truffle and JavaScript are heavily interlinked. JavaScript is mostly used within the front-end of the DApp for UI and web server configuration, and Truffle boxes provide boilerplate code supporting many front-end frameworks/libraries like React.js [49] and Vue.js [50]. JavaScript – consequently - is practically built into the DApp framework provided by Truffle.

5.5 React

React.js [49] is the chosen front-end library for the system. It breaks down the application into components, making use of JavaScript XML (JSX) [51] to combine HTML with the scripting power of JavaScript. The Truffle ‘react’ box [52] used in this project provides boilerplate code for creating a DApp using React.

React wasn’t originally chosen for the front-end. Originally lite-server, which is a lightweight development only server [53], was used alongside a simple HTML page. It was provided in the tutorial Truffle Box ‘Pet Shop’ which many third-party DApp tutorials like that of Dapp University [54] build their DApp from. The reason React was chosen over lite-server is because imports did not work within the boilerplate code of ‘Pet Shop’ meaning necessary libraries – like those needed for the cryptographic methods – could not be imported. This meant the requirements for the system could not be met while using lite-server, thus an alternative was required in the form of React.js.

The project styles the HTML page produced through React using custom CSS, as well as Bootstrap [55] (using a library called React-Bootstrap [56]).

5.6 Visual Studio

Visual Studio Code is an integrated development environment (IDE) used to write, edit, build, run, and debug code [57]. It supports a wide selection of languages out of the box including – crucially for this application – JavaScript, HTML, and CSS [58]. Furthermore, support for other languages, or extra features, can be added to Visual Studio by installing extensions from the ever-growing selection on the community marketplace [59].

Visual Studio Code was the IDE of choice due to its support for most of the languages necessary for this project out of the box, but also for its look and layout. It is very intuitive to use, and any extra features needed can easily be installed from the marketplace. For example, Solidity support was necessary for this project, and this was added in just a few clicks via the marketplace. It provides everything a programmer could need, whilst being an incredibly convenient tool to use.

5.7 GitHub

GitHub is the world's largest code host [60], and the most popular version control system. Version control is the practice of tracking and managing changes to code and is an invaluable tool to help developers keep their codebase organised and trace changes [61].

The GitHub Desktop application [62] was used for this project, as it provides an application through which to manage your GitHub repositories. It is considerably easier to use than manually typing commands into the terminal like with Git Bash [63], and thus makes managing your repositories much simpler.

6. Implementation

Implementation is the stage after design in which the system is actually built. Implementation for this project involved the development of the required features identified and planned in the design stage. The completed versions of these features then combine to make up the full prototype system.

Implementation took the form of 3 clear phases, each dedicated to an individual part of the system. These phases were smart contract development, cryptographic methods development, and DApp development, and they took place in that order. As discussed in chapter 3, the project looked to use a development methodology inspired by FDD, but implementation deviated somewhat from FDD's normal structure. Typically, stages 4 and 5 of FDD (design and build) would be completed for each feature once, resulting in a finished feature. However, during development some features required multiple iterations of design and building. For example, the `getPreferences` method was designed and built multiple times to keep up with the evolution of the smart contract. The DApp UI also went through multiple stages of design and build as it improved. The 3 phases of development are discussed in more detail in the rest of this chapter.

6.1 Smart Contract

Smart contract development required converting the pseudocode plans for functions into working Solidity functions. The process wasn't quite that simple, and as mentioned the project diverged from the approach of FDD a little during this process, as some of the functions within the smart contract required multiple iterations before they were finalised.

Functions were developed in order of how crucial they were to the system, so `set` and `get` preferences first, and `delete` all last. Function implementations are briefly discussed below.

6.1.1 `setPreferences` and `getPreferences` 1.0

Basic implementations of `setPreferences` and `getPreferences` were produced first, allowing unrestricted storage and retrieval of preferences on the blockchain.

```
mapping(address => uint[4]) userpreferences;

constructor() public{}

function setPreferences(uint spatial, uint identity, uint temporal, uint activity) public returns(bool success){
    // Require statement here to ensure all dimensions are 0-4
    userpreferences[msg.sender] = [spatial,identity,temporal,activity];
    return(true); // Need to return upon success of setting, but need require statement
}

function getPreferences() public returns(uint[4] memory preferenceslist){
    return(userpreferences[msg.sender]);
}
```

Figure 12: Initial Implementations of `setPreferences` and `getPreferences`

In initial designs, user preferences were stored as an array of integers representing the SITA dimensions. They were stored in a mapping with the key being the user's address (see Figure 12 line

1), so at this early stage a user could only manage one set of preferences. This was resolved by the next iteration which stored preferences under a composite key of user address + key (see Figure 13).

```
mapping(bytes => uint[4]) userPreferences; // Mapping bytes (address + key) as you cant use struct or array as mapping keys
// Need to use bytes to be able to concatenate the address and key, should change later to a particular size bytes when we know address and key combined size

constructor() public{}

function setPreferences(uint spatial, uint identity, uint temporal, uint activity, string memory key) public returns(bool success){
    // Require statement here to ensure all dimensions are 0-4
    bytes memory keyBytes = abi.encodePacked(key); // Convert string parameter for the secret key to bytes
    userPreferences[abi.encodePacked(msg.sender, keyBytes)] = [spatial,identity,temporal,activity]; // Store the preference array mapped under address + secret key
    return(true); // Need to return upon success of setting, but need require statement
}

function getPreferences(string memory key) public returns(uint[4] memory preferencesList){
    bytes memory keyBytes = abi.encodePacked(key); // Convert string parameter for the secret key to bytes
    return(userPreferences[abi.encodePacked(msg.sender, keyBytes)]);
}

}
```

Figure 13: *setPreferences and getPreferences using address + key*

Several more iterations culminated in the 1.0 versions of setPreferences and getPreferences, the pre-approved addresses implementations (see Figure 14).

```
mapping(bytes => string) private userPreferences; // Mapping bytes (address + key) as you cant use struct or array as mapping keys
// Need to use bytes to be able to concatenate the address and key, should change later to a particular size bytes when we know address and key combined size

mapping(address => string[]) private usedKeys; // String array of the H(key) in use for a user

constructor(){}
```

```
function setPreferences(string memory preferences, string memory key) public returns(bool success){
    // Require statement here to ensure all dimensions are 0-4

    if (keyInUse(msg.sender,key) == false){ // If the key isn't already in use it needs to be added to the array in the usedKey mapping
        usedKeys[msg.sender].push(key);
    }

    bytes memory keyBytes = abi.encodePacked(key); // Convert string parameter for the secret key to bytes
    userPreferences[abi.encodePacked(msg.sender, keyBytes)] = preferences; // Store the preference array mapped under address + secret key as the mapping key
    return(true); // Need to return upon success of setting, but need require statement
}

function getPreferences(address userAddress,string memory key) public returns(string memory preferences){
    bytes memory keyBytes = abi.encodePacked(key); // Convert string parameter for the secret key to bytes

    if(keyInUse(userAddress, key)){
        return(userPreferences[abi.encodePacked(userAddress, keyBytes)]);
    }
    else{
        revert PreferencesNotFound(userAddress, key);
    }
}

}
```

Figure 14: *setPreferences and getPreferences BEFORE approved addresses*

These versions allowed the user to manage multiple different sets of preferences by using different keys and allowed retrieval of other user's preferences using getPreferences. A helper function keyInUse was also developed (Figure 15).

```

function keyInUse(address userAddress, string memory keyHash) private returns(bool keyExists){ // Helper function to check if a key already exists in keysUsed
    keyExists = false; // Boolean value for whether the key exists yet or not

    for(uint i; i < usedKeys[userAddress].length; i++){ // Iterate through the user's keys
        if (keccak256(bytes(usedKeys[userAddress][i])) == keccak256(bytes(keyHash))){ // Compare the key in usedKeys array and the key the user has entered
            keyExists = true;
        }
    }

    return(keyExists);
}

```

Figure 15: *keyInUse helper function*

This function wasn't part of the requirements, but a need for it became apparent during development. It uses a third mapping called `usedKeys` (Figure 16), which maps a user's address to a dynamic array of all keys the user has preferences stored under.

```

mapping(address => string[]) private usedKeys; // String array of the H(key) in use for a user

```

Figure 16: *usedKeys mapping*

6.1.2 addApprovedAddress and removeApprovedAddress

Both adding and removing approved addresses involves the modification of a mapping called `approvedAddresses`. It uses the same composite key as `userPreferences`, a combination of the user's address and secret key. This composite key maps to a dynamic array of addresses (see Figure 17 line 1), which approved addresses are added to and removed from.

```

mapping(bytes => address[]) private approvedAddresses; // Mapping linking address + H(key) to approved addresses

function addApprovedAddress(address approvedAddress, string memory keyHash) public returns(bool success){
    if(keyInUse(msg.sender, keyHash)){
        bytes memory keyBytes = abi.encodePacked(keyHash);
        approvedAddresses[abi.encodePacked(msg.sender, keyBytes)].push(approvedAddress);
        return (true);
    }
    else{
        return (false);
    }
}

```

Figure 17: *Initial implementation of addApprovedAddress Function*

The implementation of `addApprovedAddress` changed very little from start to finish. The only major change was checking whether an address was already approved before adding it to avoid duplicate addresses.

```

function addApprovedAddress(address approvedAddress, string memory keyHash) public returns(bool success){
    // Attempts to add an approved address for the provided address + H(key) combination, only callable on own preferences

    if(keyInUse(msg.sender,keyHash)){
        // Only add approved address if the preferences exist
        if (approvedAddressExists(msg.sender, keyHash,approvedAddress)){
            // Prevents duplicate entries in approvedAddresses, returns error
            return(true); // Success as address already approved
        }
        // If approved address isn't already present
        bytes memory keyBytes = abi.encodePacked(keyHash); // Convert string parameter for the secret key to bytes
        approvedAddresses[abi.encodePacked(msg.sender, keyBytes)].push(approvedAddress); // Add approved address to list
        return (true);
    }
    else{
        // Returns error if key doesn't exist
        revert KeyNotInUse(msg.sender, keyHash);
    }
}
}

```

Figure 18: Finished implementation of addApprovedAddress Function

The finished version (Figure 18) makes use of another helper function approvedAddressExists (Figure 19), which once again was not defined within the original requirements but emerged as a useful addition to aid other functions. It checks whether an address is already approved for a particular preferences set

```

function approvedAddressExists(address userAddress, string memory keyHash, address approvedAddress) private view returns (bool success){
    // Helper function which checks if an address exists in approved addresses for address + H(key) combination

    if(keyInUse(userAddress,keyHash)){ // Don't waste computing power (and thus Ethereum) looping through if the key isn't being used
        bytes memory keyBytes = abi.encodePacked(keyHash);
        address[] memory approvedAddressList = approvedAddresses[abi.encodePacked(userAddress, keyBytes)];
        for(uint i; i < approvedAddressList.length; i++){ // Iterates through the users approved addresses
            if (approvedAddressList[i] == approvedAddress){ // Checks if the address exists in approvedAddresses
                return(true);
            }
        }
        return(false);
    }
    else{
        // Provide Approved Address not found error instead of key not in use error to prevent leaking whether a user is using
        // a certain key (necessary as this function is called in getPreferences, which can be called by not just the preferences owner)
        revert ApprovedAddressNotFound(approvedAddress,userAddress,keyHash);
    }
}
}

```

Figure 19: approvedAddressExists helper function

The removeApprovedAddress function (Figure 20), is slightly more complicated than addApprovedAddress.

```

function removeApprovedAddress(address removeAddress, string memory keyHash) public returns (bool success){
    // Removes an approved address for address + H(key) combination, only callable on own preferences

    if(removeAddress == msg.sender){
        // User not allowed to remove their own address
        revert CantRemoveOwnAddress(msg.sender, keyHash);
    }

    if(keyInUse(msg.sender, keyHash)){ // Don't waste computing power (and thus Ethereum) looping through if the key isn't being used
        bytes memory keyBytes = abi.encodePacked(keyHash);
        address[] memory approvedAddressList = approvedAddresses[abi.encodePacked(msg.sender, keyBytes)]; // Gets the list to reduce the complexity of the for loop
        for(uint i; i < approvedAddressList.length; i++){ // Iterates through the users approved addresses
            if (approvedAddressList[i] == removeAddress){ // Attempts to find the address to be removed
                approvedAddresses[abi.encodePacked(msg.sender, keyBytes)][i] = approvedAddresses[abi.encodePacked(msg.sender, keyBytes)][approvedAddressList.length - 1];
                approvedAddresses[abi.encodePacked(msg.sender, keyBytes)].pop(); // Pops the last address in the array, the last address now occupies [i]
                return (true);
            }
        }
        // If we loop through all approved addresses without finding it return error
        revert ApprovedAddressNotFound(removeAddress, msg.sender, keyHash);
    }
    else{
        // Returns error if key doesn't exist
        revert KeyNotInUse(msg.sender, keyHash);
    }
}

```

Figure 20: Finished implementation of removeApprovedAddress function

Solidity does not have a keyword or inbuilt function to check whether an array contains a value, like Python's `in` [64] or Java's `.contains()` [65], hence why both `removeApprovedAddress` and `approvedAddressExists` use a for loop to check the value of every item in the array. Solidity also lacks an in-built function to delete an item at a specific array index. As a work-around `removeApprovedAddress` instead copies the last item in the array into the index of the item being deleted so it is overwritten, and then deletes the last item in the array using `.pop()`. This is how an approved address is removed.

6.1.3 getPreferences 2.0

With the implementation of approved addresses, `getPreferences` needed re-designed to account for calls from approved versus non-approved addresses.

```

function getPreferences(address userAddress, string memory key) public view returns(string memory preferences){
    // Attempts to retrieve preferences stored under the provided address + H(key) combination
    bytes memory keyBytes = abi.encodePacked(key); // Convert string parameter for the secret key to bytes

    if( (msg.sender == userAddress) || (approvedAddressExists(userAddress, key, msg.sender)) ){
        // Only addresses approved by the user are allowed to retrieve the encrypted preferences
        if(keyInUse(userAddress, key)){
            // Make sure the key is in use (and thus the preferences attempting to be retrieved exist)
            return(userPreferences[abi.encodePacked(userAddress, keyBytes)]);
        }
        else{
            // Return error indicating this preferences set isn't found
            revert PreferencesNotFound(userAddress, key);
        }
    }
    else{
        // Return error indicating the user isn't approved to retrieve these preferences
        revert ApprovedAddressNotFound(msg.sender, userAddress, key);
    }
}

```

Figure 21: Finished implementation of getPreferences function

The finished version of `getPreferences` will only return the preferences if the caller is either the owner of the preferences OR is an approved address for them (Figure 21 line 5). This version makes use of the helper function seen earlier `approvedAddressExists`.

6.1.5 `getApprovedAddresses`

This is quite a crucial function that was overlooked whilst designing the system. It is used to obtain all approved addresses for a preferences set, so they can be displayed in the DApp (see Figure 22).

```
function getApprovedAddresses(string memory keyHash) public view returns(address[] memory addresses){
    // Gets list of approved addresses for address + H(key) combination, only callable on own preferences
    if(keyInUse(msg.sender,keyHash)){
        // Only retrieve approved addresses if the key is in use
        bytes memory keyBytes = abi.encodePacked(keyHash);
        return(approvedAddresses[abi.encodePacked(msg.sender, keyBytes)]);
    }
    else{
        // Returns error if key doesn't exist
        revert KeyNotInUse(msg.sender, keyHash);
    }
}
```

Figure 22: Finished `getApprovedAddresses` function

It simply returns the array of approved addresses for the preferences stored in the `approvedAddresses` mapping seen in Figure 17.

6.1.4 `deletePreferences` and `deleteAllPreferences`

As the lowest priority these functions were developed last, with `deletePreferences` implemented first out of the two (Figure 23).

```
function deletePreferences(string memory keyHash) public returns(bool success){
    // Attempts to delete a user's preferences stored under the provided H(key), only callable on your own preferences
    bytes memory keyBytes = abi.encodePacked(keyHash); // Convert string parameter for the secret key to bytes

    // Delete just leaves all the entries under these keys as the default ('' for strings, and empty arrays)
    // getPreferences, getApprovedAddresses etc don't work as the key is no longer in use
    if(keyInUse(msg.sender,keyHash)){
        // Only delete preferences if the key is use
        delete approvedAddresses[abi.encodePacked(msg.sender, keyBytes)];
        delete userPreferences[abi.encodePacked(msg.sender, keyBytes)];

        for(uint i; i < usedKeys[msg.sender].length; i++){ // Iterate through the user's keys
            if (keccak256(bytes(usedKeys[msg.sender][i])) == keccak256(bytes(keyHash))){ // Compare the key in usedKeys array and the key the user has entered
                usedKeys[msg.sender][i] = usedKeys[msg.sender][(usedKeys[msg.sender].length) - 1]; // Set the key to be deleted equal to the last key in the array
                usedKeys[msg.sender].pop(); // Remove the last key in the array
                return(true);
            }
        }
    }
    else{
        // Return error indicating the key isn't in use
        revert KeyNotInUse(msg.sender, keyHash);
    }
}
```

Figure 23: Finished implementation of `deletePreferences` function

Once again it uses the `keyInUse` helper function to check if the requested key is actually in use before performing any operations, if not the transaction is reverted. The array of approved addresses for the preferences set is cleared, and then so are the preferences themselves in the `userPreferences` mapping. Deleting from the `usedKeys` mapping was more difficult. The array must be searched through using a `for` loop, checking the value at each index. As the keys are stored as bytes, we must use `keccak256` hashing [66] to check if the key in the array matches the key being deleted. Once matched the key is deleted from the array, but – as discussed already– you can't delete a specific index from a Solidity array, so the same method as used in `removeApprovedAddress` is used to re-arrange the array and pop the last value.

The `deleteAllPreferences` function uses the `usedKeys` mapping to iterate through all the keys the user has, and calls `deletePreferences` on them all one by one (see Figure 24).

```
function deleteAllPreferences() public returns (bool success){
    // The nuclear option. Allows user to delete all preference sets stored for them without providing the key.
    // For use in case a user wants to delete a preference set but has forgotten the key

    string[] memory keysInUse = usedKeys[msg.sender]; // Gets all the keys the user has

    if(keysInUse.length == 0){ // Returns false if there are no preferences to delete
        revert KeyNotInUse(msg.sender, "any");
    }

    for(uint i; i < keysInUse.length; i++){ // Iterates through all the keys the user has
        deletePreferences(keysInUse[i]); // Deletes the preference set under this key
    }

    return(true);
}
```

Figure 24: Finished deleteAllPreferences function

6.1.6 Conclusion

The planning done within the design stage made producing a smart contract which satisfies all the requirements fairly straightforward, as most of the functions already had pseudocode plans to start from. The resulting smart contract is based around three mappings which store all the relevant information for the system, with all contract functions interacting with them in some way. A few extra functions like `keyInUse` and `approvedAddressExists` were developed to aid the key functions, but `getApprovedAddresses` was a core function that was missed during the design stage. This doesn't necessarily mean the planning was bad as requirements often evolve throughout a project as understanding of the system develops.

It is worth mentioning that the functions developed work regardless of encryption. Whether the key is hashed and the preferences encrypted or not is irrelevant to the smart contract. Ensuring that they are is a job for the DApp. Furthermore, the Solidity keyword **revert** [67] is used frequently throughout the contract code. `Revert` undoes all state changes made within that function, reversing any changes made [68]. In this system it is used to reverse changes at times when an error would occur to ensure that no changes are committed to the blockchain. When a `revert` occurs the user is refunded the cost of the transaction; more discussion of costs can be found in chapters 8 and 9.

6.2 Cryptographic Methods

The development of the cryptographic methods also saw deviation from the structure of FDD. In particular, the implementations of encrypt and decrypt required multiple different iterations until a suitable solution was found.

6.2.1 Encrypt and Decrypt

The encrypt and decrypt functions looked to use AES, as discussed in design. The first iteration of encrypt and decrypt used AES encryption provided by the library crypto-js [69].

```
export function encryptPreferences(pref, secretKey){
  return AES.encrypt(pref,secretKey);
}

export function decryptPreferences(encPref, secretKey){
  return AES.decrypt(encPref,secretKey);
}
```

Figure 25: crypto-js Encrypt and Decrypt functions

This iteration (seen in Figure 25) worked at first, preferences would be encrypted with the provided key, and could be decrypted again using the same key. However, these implementations didn't work when combined with smart contract storage. The encryptPreferences method did not return a string for the ciphertext, but rather a CipherParams object. This same CipherParams object was needed by the decryptPreferences function to be able to get the plaintext back, but the smart contract was unable to store such an object. As a result, a different AES library was needed.

The second iteration (Figure 26) used AES encryption provided by the node-forge library [70].

```
export function encryptPreferences(pref, secretKey){
  var combined = Buffer.from(secretKey, 'hex').toString();

  console.log("combined = ", combined);

  // Splits the secret key back into key and IV
  var key = combined.substring(0,32)
  console.log("substring key = ", key);
  var iv = combined.substring(32)
  console.log("substring iv = ", iv);

  var cipher = forge.cipher.createCipher('AES-CBC', forge.util.createBuffer(key)); // Forge uses its own buffer format so create buffer to avoid errors
  cipher.start({iv: forge.util.createBuffer(iv)}); // Forge uses its own buffer format so create buffer to avoid errors
  cipher.update(forge.util.createBuffer(pref));
  cipher.finish();
  var encrypted = cipher.output;
  // outputs encrypted hex
  console.log("encrypted = ", encrypted);
  console.log(encrypted.toHex());

  decryptPreferences(encrypted, secretKey);
}

export function decryptPreferences(encPref, secretKey){
  var combined = Buffer.from(secretKey, 'hex').toString();

  // Splits the secret key back into key and IV
  var key = combined.substring(0,32)
  console.log("substring key = ", key);
  var iv = combined.substring(32)
  console.log("substring iv = ", iv);

  var decipher = forge.cipher.createDecipher('AES-CBC', forge.util.createBuffer(key)); // Forge uses its own buffer format so create buffer to avoid errors
  decipher.start({iv: forge.util.createBuffer(iv)}); // Forge uses its own buffer format so create buffer to avoid errors
  decipher.update(encPref);
  var result = decipher.finish(); // check 'result' for true/false
  // outputs decrypted hex
  console.log(decipher.output);
  console.log(decipher.output.toHex());
  var result = Buffer.from(decipher.output.toHex(), 'hex').toString();
  console.log("result = ", result);
}
```

Figure 26: node-forge Encrypt and Decrypt functions

Unfortunately, node-forge produced a similar problem to crypto-js. The result of encryption this time was a `ByteBuffer` object, and decryption required this same object. Once again, the smart contract was unable to store such an object.

After these failed implementations it was clear a library was needed which output a string when encrypting rather than an object, and could decrypt using this string. Thus, the third iteration used the library `aes-js` [71].

```
export function encryptPreferences(pref, secretKey){
  var key = aesjs.utils.hex.toBytes(secretKey); // Convert hex secret key to bytes
  var prefBytes = aesjs.utils.utf8.toBytes(pref); // Convert preferences string to bytes

  var aesCtr = new aesjs.ModeOfOperation.ctr(key, new aesjs.Counter(5));
  var encryptedBytes = aesCtr.encrypt(prefBytes); // Encrypt preferences

  var encryptedHex = aesjs.utils.hex.fromBytes(encryptedBytes); // Convert encrypted preferences to hex for storage

  return(encryptedHex);
}

export function decryptPreferences(encPref, secretKey){
  var key = aesjs.utils.hex.toBytes(secretKey); // Convert hex secret key to bytes
  var encPrefBytes = aesjs.utils.hex.toBytes(encPref); // Convert encrypted preferences from hex to bytes

  var aesCtr = new aesjs.ModeOfOperation.ctr(key, new aesjs.Counter(5));
  var decryptedBytes = aesCtr.decrypt(encPrefBytes); // Decrypt preferences

  var decryptedText = aesjs.utils.utf8.fromBytes(decryptedBytes); // Converts decrypted bytes to UTF-8 text
  return(decryptedText);
}
```

Figure 27: aes-js Encrypt and Decrypt Functions

The final iteration using `aes-js` (Figure 27) worked perfectly. As encryption with `aes-js` returned a ciphertext string, the encrypted preferences could be stored on the blockchain using `setPreferences`, retrieved using `getPreferences` and successfully decrypted to give back the original plaintext.

The amount of iteration for the encryption and decryption functions was unplanned and was motivated entirely by problems with certain implementations.

6.2.2 Generate Key

Generate key (Figure 28) was simpler to implement. It uses the `crypto` library [72] which comes with `Node.js`.

```
export function genKey(){
  var key = crypto.randomBytes(32);
  var hexKey = aesjs.utils.hex.fromBytes(key);
  return (hexKey);
}
```

Figure 28: Finished implementation of genKey function

The 32-byte key is generated using the `randomBytes` method provided by `crypto`. The key is 32-bytes which is the key-length for AES-256, the largest and most secure version of AES [73]. The randomly generated key is converted into hexadecimal form to reduce the number of symbols within it.

6.2.3 Hash Key

The hash key function (Figure 29) was also simple to implement, with the js-sha3 library [74] providing SHA-3 hashing.

```
export function hashKey(secretKey){  
  return(sha3_512(secretKey)); // Returns the SHA3-512 hash of the key as a string  
}
```

Figure 29: Finished implementation of hashKey function

The function just simply calls the sha3_512 method on the received secret key and returns the resulting hash. SHA3-512 was chosen as it is the largest available version of the most secure hashing algorithm SHA-3.

6.3 DApp

Once again DApp development strayed from the general structure of FDD requiring iteration. With all the contract and cryptographic functions developed, a DApp was the last thing to implement to provide a front-end for the user to interact with these functions through.

As recommended by the React docs, the first step to developing a UI was to break down the draft UI into components [75]. This was conducted on the DApp wireframe produced in the design stage (see Figure 30).

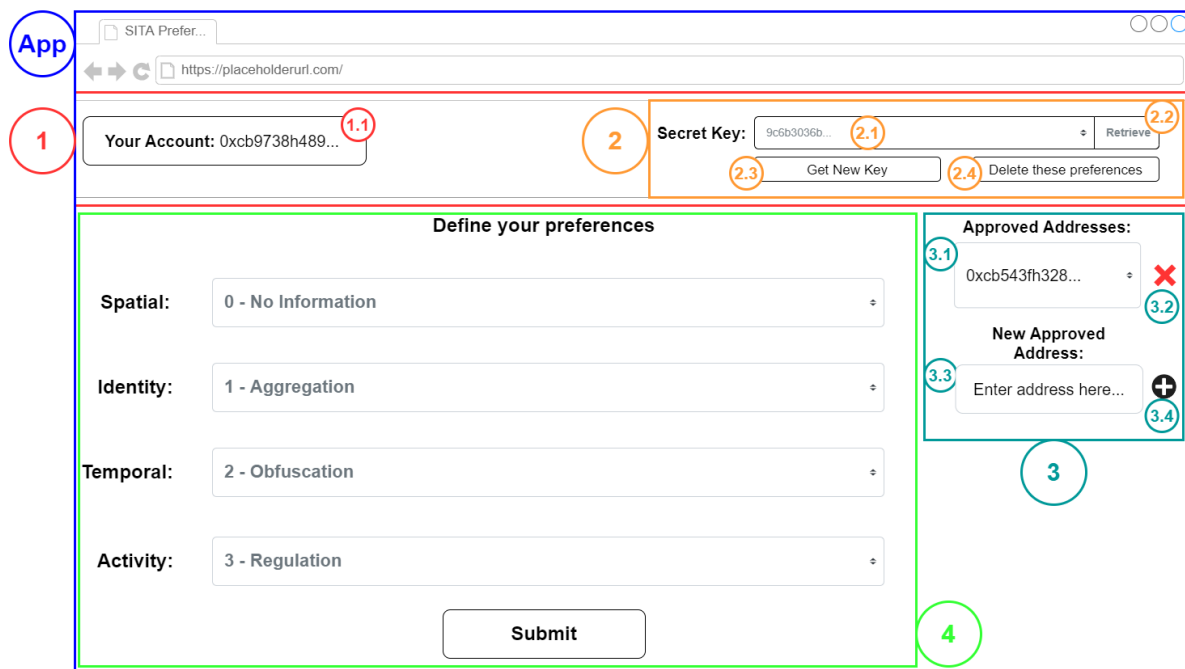


Figure 30: DApp Wireframe split into React Components

Five components were identified:

- App: The entire app page, handling initialisation of the page, linking of MetaMask accounts etc.
- (1) Your Account: Container to display the user's blockchain address.

- (2) Secret Key Management: Section including the input box for the secret key, retrieve button, get new key button, and delete these preferences button.
- (3) Approved Addresses: Management of approved addresses, including a drop-down of approved addresses, delete button, input box, and add button.
- (4) Preferences Form: Form including dropdowns for each dimension and a submit button.

6.3.1 UI 1.0

The first UI implementation followed the structure of FDD steps 4 and 5. Each component had a pseudocode implementation designed, and then this was built with React code. The only exception to this was the App component, which was mostly provided by the 'react' Truffle Box. Generally, modifications to App were made to facilitate the other components.

1. Your Account:

```
class YourAccount extends Component{
  constructor(props){
    super(props);
    this.state = {address: ""};

    web3.eth.getCoinbase(function(err,account){
      if(err === null){
        this.props.onGetAddress(this.state.address)
        this.SetState(address = account) // However you do this to pass up to parent
      }
    });
  }

  render(){
    const address = this.state.address;
    return(
      <p> Your Account: {address} </p>
    );
  }
}
```

Figure 31: YourAccount Component Pseudocode

Figure 31 shows an example of the pseudocode plan for the YourAccount component. It was the simplest component and looked to just render a paragraph containing "Your Account: blockchain address".

```

class YourAccount extends Component{
  constructor(props){
    super(props);
    this.state = {address: this.props.address};
  }

  render(){
    const address = this.state.address;
    return(
      <p> Your Account: {address} </p>
    );
  }
}

```

Figure 32: First implementation of YourAccount Component

The first version of YourAccount (Figure 32) works very similar to the plan, but instead of trying to obtain the user's address itself the App component gets the user's address when the page starts and passes it to the YourAccount component via properties (props). Props in React are similar to function parameters, they are values passed to the component and can be variables or functions [76]. The YourAccount component stores the address passed to it (this.props.address) in its own local state, as properties in React can't be modified. This is actually unnecessary, as the component never looks to modify the value of address. This would be resolved in later iterations.

One-by-one all the other components had pseudocode designs produced and were implemented. The result of completing all the components was DApp UI 1.0 (Figure 33), the most basic UI for the application. It provided access to all the features of the application, however it was not aesthetically pleasing.

Figure 33: DApp UI 1.0

6.3.2 UI 2.0

To improve the appearance of the UI, Bootstrap was used. Bootstrap is “the world’s most popular front-end open source toolkit” [55]. To use Bootstrap alongside React the library React-Bootstrap [56] was used, which provided React versions of Bootstrap components. For example, the coloured div

used for the top navigation bar of the page was replaced with a Bootstrap NavBar component (see Figure 34).

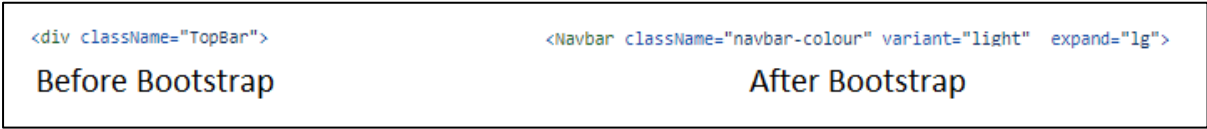


Figure 34: Navigation bar before and after Bootstrap

Each feature was iterated through again, replacing HTML components like buttons and forms with Bootstrap equivalents (see Figure 35).



Figure 35: Preferences Form before and after Bootstrap

Using Bootstrap components allowed the use of pre-defined CSS styles for the components which eliminated the need to write detailed custom CSS. For example, in Figure 34 the NavBar uses the variant “light” to use the pre-defined light NavBar style. The culmination of updating all components to use Bootstrap was UI 2.0, seen in Figure 36.

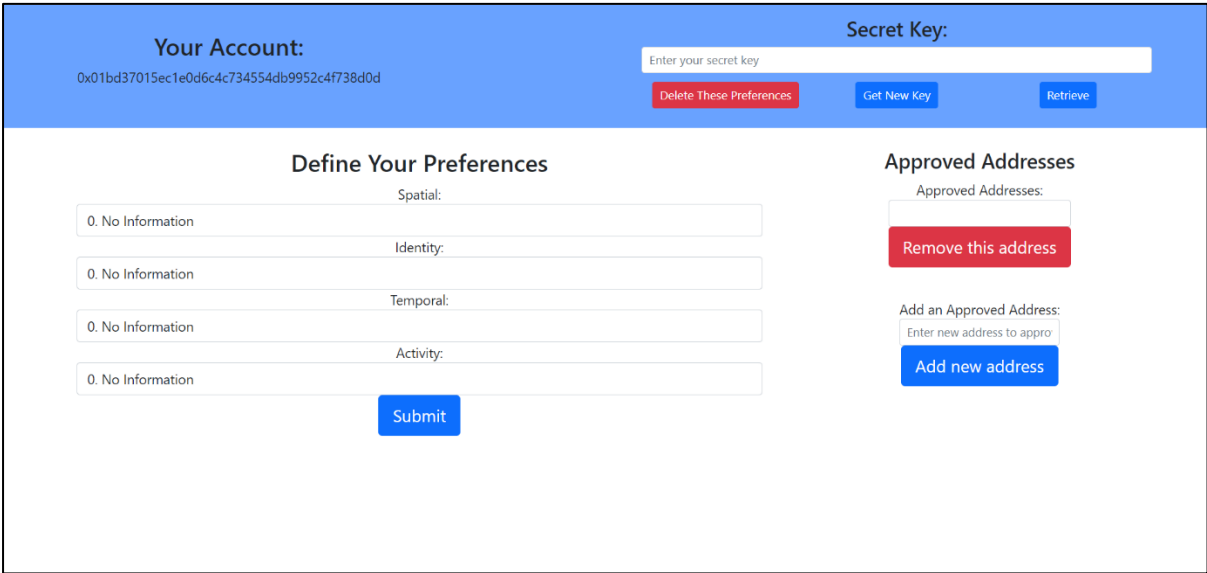


Figure 36: DApp UI 2.0

UI 2.0 represented a major improvement in appearance, but there were still improvements that could be made.

6.3.3 UI 3.0

Extra styling was added to the application using CSS. Most of the changes focused on improving the look and layout of the page. Buttons, input boxes, drop-downs etc were spaced better, and had shadows added. Some colours were changed.

The screenshot shows the DApp UI 3.0 interface. At the top, there's a light blue header bar. On the left, it says "Your Account:" followed by a long hexadecimal address. On the right, it says "Secret Key:" followed by an input field and three buttons: "Delete These Preferences", "Get New Key", and "Retrieve". Below the header, the page is divided into two main sections. The left section is titled "Define Your Preferences" and contains four dropdown menus labeled "Spatial:", "Identity:", "Temporal:", and "Activity:". Each dropdown menu currently shows "0. No Information". At the bottom of this section is a blue "Submit" button. The right section is titled "Approved Addresses" and contains an "Approved Addresses:" dropdown menu. Below this is a red button labeled "Remove this address". Further down is a section labeled "Add an Approved Address:" which includes an input field with the placeholder text "Enter new address to approve" and a blue button labeled "Add new address".

Figure 37: DApp UI 3.0

UI 3.0 (Figure 37) once again was a considerable upgrade on its predecessors and was a successful iteration.

6.3.4 UI 3.1

The first minor UI change was to replace the pop-ups for the application. Pop-ups were used throughout to provide warning messages, to confirm success or failure, and to get user confirmation for actions like generating a new key. These pop-ups used the JavaScript methods **window.alert** [77] and **window.confirm** [78].

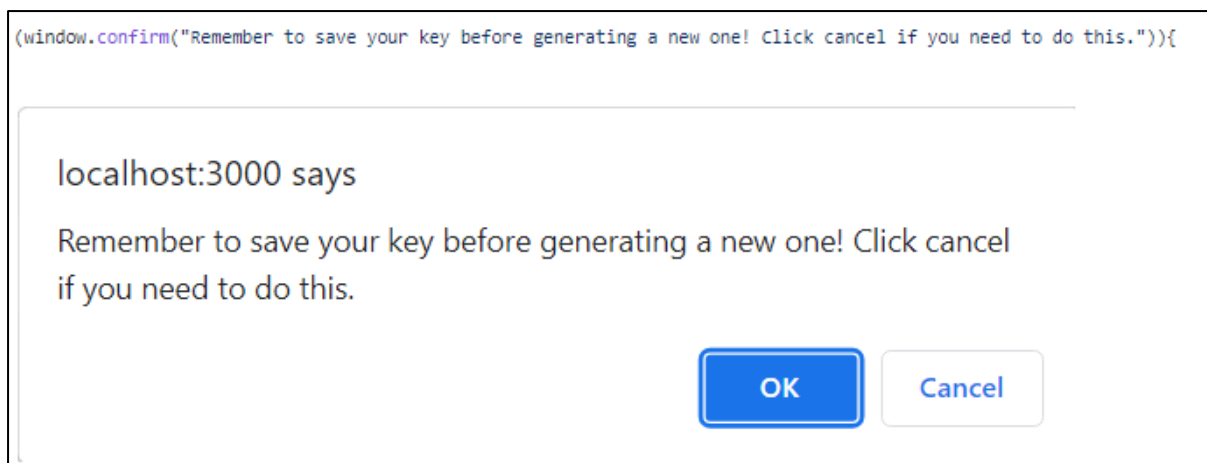


Figure 38: window.confirm example with DApp output

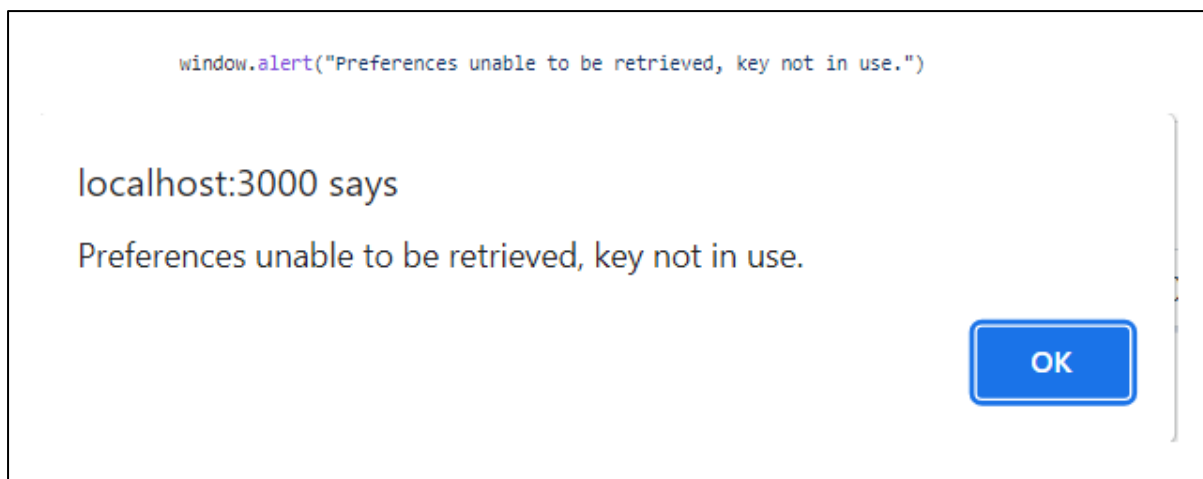


Figure 39: *window.alert* example with DApp output

As showcased by Figures 38 and 39, the pop-ups were very simplistic and did not match the DApp's look anymore. Thus, they were replaced by Bootstrap modals [79], which allowed much more control over the styling of the pop-ups.

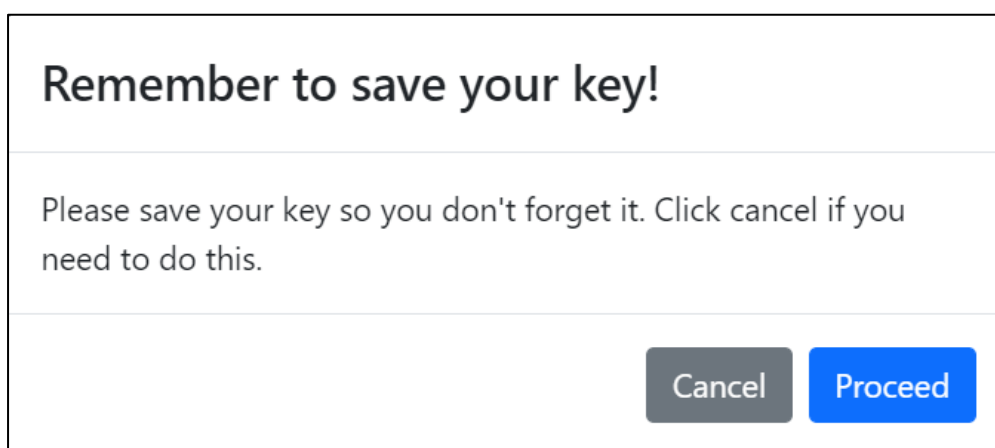


Figure 40: *Confirmation Modal Example*

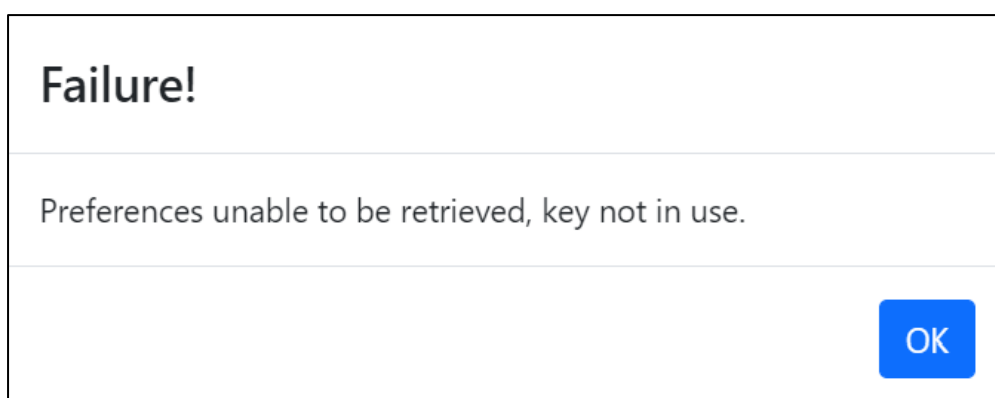


Figure 41: *Failure Modal Example*

Figures 40 and 41 show the equivalent pop-ups to Figures 38 and 39 but using modals. UI 3.1 (Figure 42) ultimately was a minor update to the previous UI, just re-styling the DApp's pop-ups.

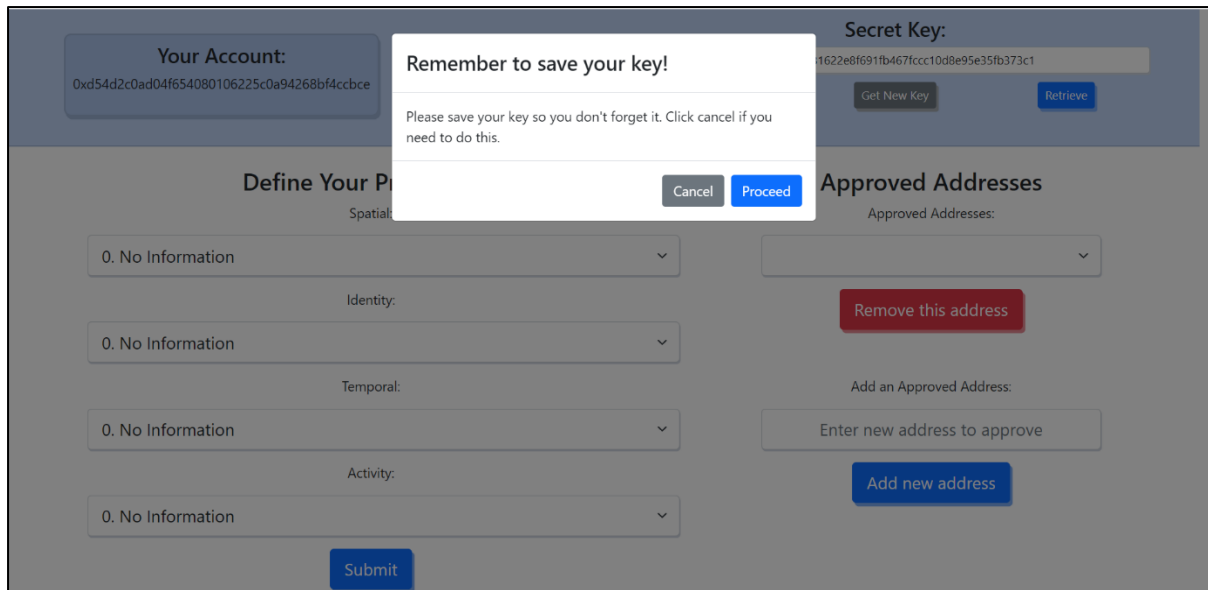


Figure 42: DApp UI 3.1

6.3.5 UI 3.2

The final UI iteration added a 6th component, the delete all preferences button. As it was the lowest priority feature the delete all preferences button wasn't included in the UI designs, as it was unclear whether there would be time to add it. As it turns out there was plenty of time to add this component, so the final iteration did just that.

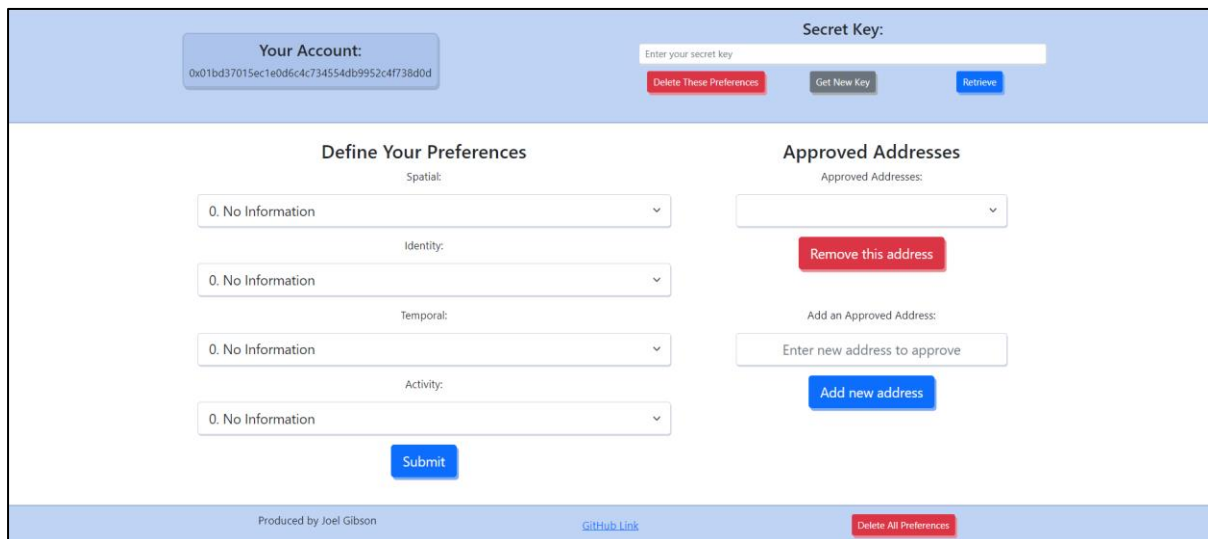


Figure 43: DApp UI 3.2

Another Bootstrap Navbar was added to the bottom of the page to hold the delete all preferences button, and some extra info like the author and a link to the GitHub page for the code were added as well (see Figure 43). With the completion of this final DApp iteration, the system as a whole was completed.

7. Testing

The goal of testing an application is to ensure that the system's requirements are met, and to uncover cases in which the system behaves unexpectedly or incorrectly in order to resolve these problems [80].

The use of an agile methodology meant testing for this project wasn't restricted to after the system was developed. Ad-hoc testing [81] was carried out throughout the development of the system, ensuring new code worked as intended. This was mostly in the form of white-box testing, with specific focus on ensuring the internal behaviour of the code being tested was as expected. This was supplemented by manual unit testing [82] conducted at major milestones during the project. These unit tests were designed to test all possible circumstances individual parts of the system could face, and the results were documented along with any necessary action taken.

7.1 Smart Contract Testing

Upon completion of the smart contract, each smart contract function (excluding the helper functions) was thoroughly tested to ensure that for every possible scenario the function produced the expected result. This type of testing was more akin to black-box testing [81], as all that mattered was checking that certain inputs yielded the correct output.

setPreferences function:					
No.	Test	Description	Expected Result	Actual Result	Actions Taken
1	Call setPreferences from an invalid address.	Call setPreferences with an address not on the Ganache network.	Error message. Nothing stored in contract.	Error message. Nothing stored in contract.	None necessary.
2	Call setPreferences from valid address and provide valid key and preferences.	Call setPreferences with an address on the Ganache network, providing a string key and string preferences.	Preferences stored in the contract under address + key.	Preferences stored in the contract under address + key.	None necessary.
3	Call setPreferences from valid address and provide valid key BUT invalid preferences.	Call setPreferences with an address on the Ganache network, providing a string key BUT non-string preferences.	Error message. Nothing stored in contract.	" (blank string) stored in contract under the key.	Added require statement to prevent " being a valid input for preferences (entering integer or other type would result in " also). Test passes with this change.
4	Call setPreferences from valid address and provide valid preferences BUT invalid key.	Call setPreferences with an address on the Ganache network, providing string preferences BUT a non-string key.	Error message. Nothing stored in contract.	Preferences stored in contract under the key " (blank string).	Added require statement to prevent " being a valid input for key (entering integer or other type would result in " also). Test passes with this change.

Figure 44: Test Logs for setPreferences Function

Figure 44 shows the testing logs for the setPreferences contract function. There are 4 possible scenarios in which setPreferences is called, and notably for test 3 and 4, the result was not what was anticipated. Actions were taken to address this, and upon running the tests again the actual result matched the expected result.

As mentioned, this process was carried out for every contract function. The thoroughness of the testing was extremely helpful in highlighting any unexpected behaviour, but also once completed affirmed that the smart contract behaved as intended, meeting its requirements.

7.2 UI 1.0 Testing

Similar testing was carried out after the completion of the first DApp implementation. Every component of the UI was tested under every possible scenario it could face.

Retrieve Button:					
No.	Test	Description	Expected Result	Actual Result	Actions Taken
1	Retrieve without a key.	Click the retrieve button without entering anything into the Secret Key input box.	Error message prompting the user to enter a key before trying to <u>retrieve</u> .	"Preferences unable to be retrieved" error message.	Added an error message that states "Key can't be blank".
2	Retrieve with a valid key that is in use.	Click the retrieve button with a valid key that has preferences stored for it in the Secret Key input box.	Preferences retrieved. Prompt confirms successful retrieval, the dimension drop-downs are filled with the retrieved preferences, and the approved addresses drop-down is filled.	The dimension drop-downs are filled with the retrieved preferences, and the approved addresses drop-down is filled. No prompt confirming successful retrieval <u>however</u> .	Added a prompt after successful retrieval stating "Preferences successfully retrieved".
3	Retrieve with a valid key that isn't in use.	Click the retrieve button with a valid key that has NO preferences stored for it in the Secret Key input box.	Error message informing the user the key isn't in use.	"Preferences unable to be retrieved" error message.	Modified message to say "Preferences unable to be retrieved. Key not in use", as after the modifications made in other tests this is the only reason preferences retrieval should fail.
4	Retrieve with an invalid key.	Click the retrieve button with an invalid key (not 64 hexadecimal characters) in the Secret Key input box.	Error message informing the user the key is of incorrect format.	"Preferences unable to be retrieved" error message.	Error messages added for keys not exactly 64 characters <u>in</u> length, or not hexadecimal.

Figure 45: Test Logs for UI 1.0 Retrieve Button

Testing of the initial UI revealed many oversights and lots of unexpected behaviour which needed addressed. For example, as seen in Figure 45, testing of the retrieve button yielded unexpected results in every scenario. Many of the other components also had problems, but the thorough manual unit testing was invaluable in highlighting these unforeseen issues which once uncovered could be quickly resolved.

7.2 UI 3.2 Testing

The final round of testing for the system was carried out after completion of the DApp. The same tests as used for UI 1.0 were applied to the completed UI (example shown in Figure 46).

Retrieve Button:					
No.	Test	Description	Expected Result	Actual Result	Actions Taken
1	Retrieve without a key.	Click the retrieve button without entering anything into the Secret Key input box.	Warning message "Key can't be blank".	Warning message "Key can't be blank".	None necessary.
2	Retrieve with a valid key that is in use.	Click the retrieve button with a valid key that has preferences stored for it in the Secret Key input box.	Preferences retrieved. Modal confirms successful retrieval, the dimension drop-downs are filled with the retrieved preferences, and the approved addresses drop-down is filled.	Preferences retrieved. Modal confirms successful retrieval, the dimension drop-downs are filled with the retrieved preferences, and the approved addresses drop-down is filled.	None necessary.
3	Retrieve with a valid key that isn't in use.	Click the retrieve button with a valid key that has NO preferences stored for it in the Secret Key input box.	Modal message "Preferences unable to be retrieved, key not in use".	Modal message "Preferences unable to be retrieved, key not in use".	None necessary.
4	Retrieve with key not 64 characters long.	Click the retrieve button with an invalid key (not 64 characters long) in the Secret Key input box.	Warning message "Key must be 64 characters long".	Warning message "Key must be 64 characters long".	None necessary.
5	Retrieve with key not hexadecimal.	Click the retrieve button with an invalid key (not 64 hexadecimal characters) in the Secret Key input box.	Warning message "Key must be hexadecimal".	Warning message "Key must be hexadecimal".	None necessary.

Figure 46: Test Logs for UI 3.2 Retrieve Button

Testing of the completed UI was considerably more successful, with next to no unexpected results. This confirmed that the system behaved as intended, and thus that the requirements were met.

7.3 Improvements to Testing Strategy

Writing of automated tests most likely would've saved time during this project. Truffle Suite provides an automated testing framework for testing your smart contract, and writing automatic tests would certainly help the upkeep of the system. A set of automated tests would allow quick and automated confirmation that the functions still work, and they could be run whenever modifications were made to the contract.

React can also be automatically tested with libraries such as Jest [83] and React Testing Library [84]. It is less clear how well automated tests would've worked for the DApp UI, as it is possible the change to Bootstrap in UI 2.0 onwards might've invalidated any tests written for the original UI. However, a set of automated tests for the final UI would certainly aid maintenance of the system.

Lastly, the testing for this project was carried out by the sole-developer, so tests are unlikely to be exhaustive. It is possible certain inconsistencies and bugs have slipped through the cracks, and a more formal testing strategy may help to alleviate such concerns in future.

8. Results and Evaluation

This chapter presents and evaluates the results of this project.

8.1 Resulting System

The completion of the 3 core parts of the system produced a working prototype which met all the requirements defined in the design stage. The contract is deployed to a test Ethereum blockchain using Ganache, and the DApp is hosted locally on localhost:3000 using NPM. Consequently, the current version of the system can only be used on the local machine it is hosted on. Below is a walkthrough of the system, acting as a sort of functionality test [85] to display that all parts of the system work as intended and work together to allow the user to store and manage their privacy preferences with blockchain.

A user accesses the DApp through their web browser, signing in using the MetaMask extension (see Figure 47).

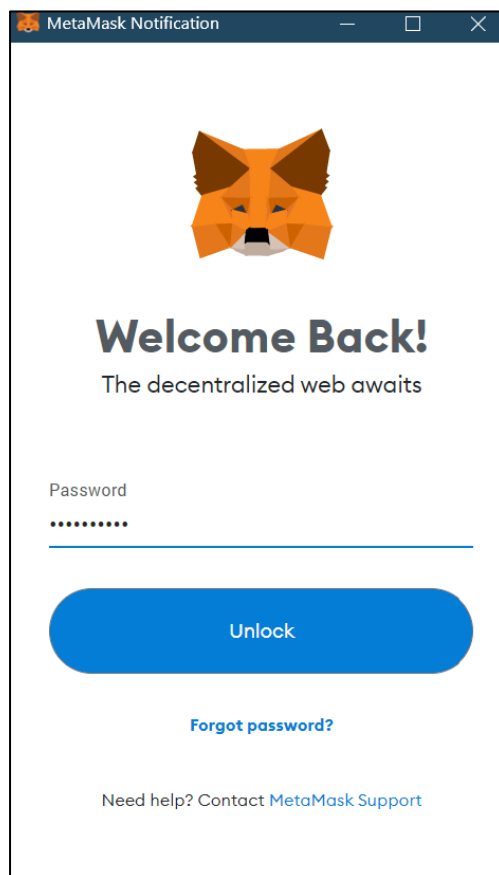


Figure 47: MetaMask login screen

The user is able to enter a pre-existing key in the secret key input box or generate a new key (see Figure 48).

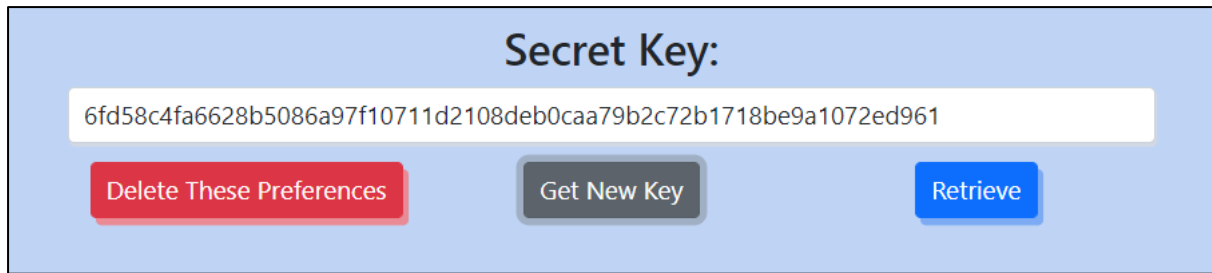


Figure 48: Key entered into Key Input Box

Clicking retrieve will populate the SITA dimensions on the form with the retrieved preferences if any are stored for that key. The user can then use the form (seen in Figure 49) to define their desired SITA levels.

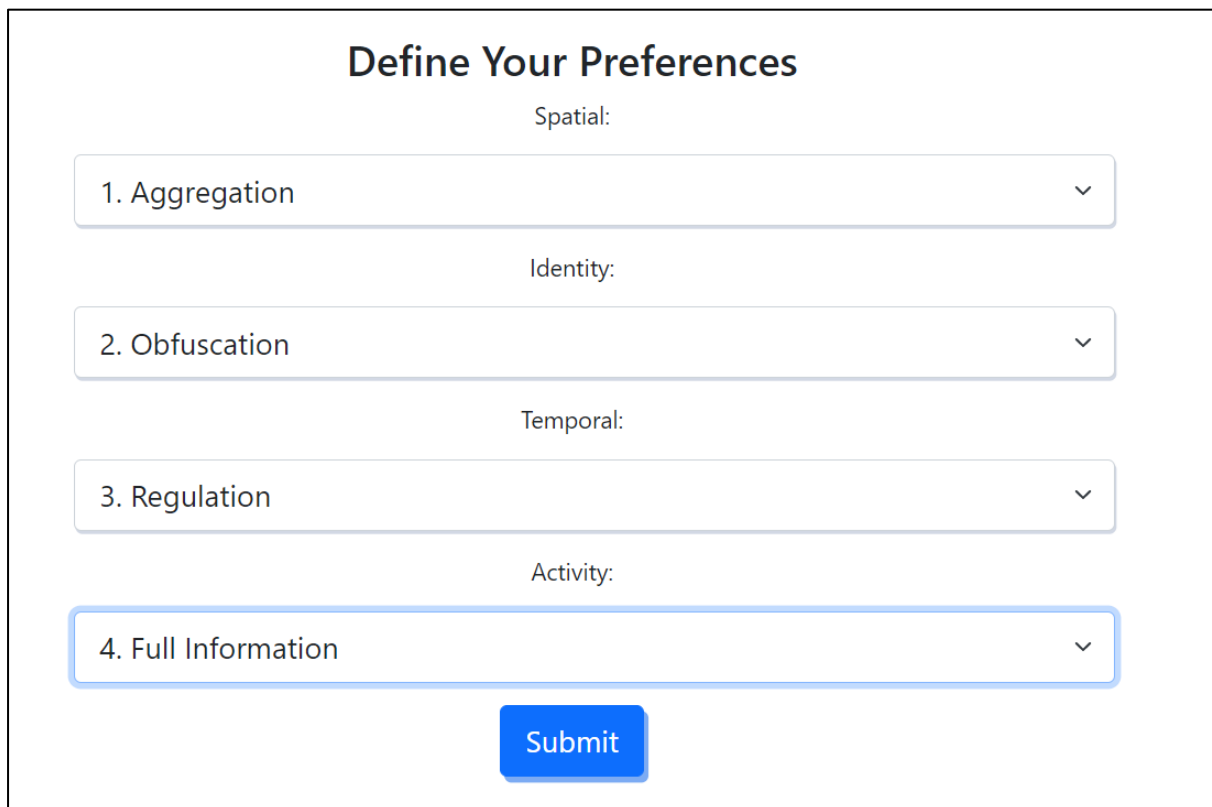
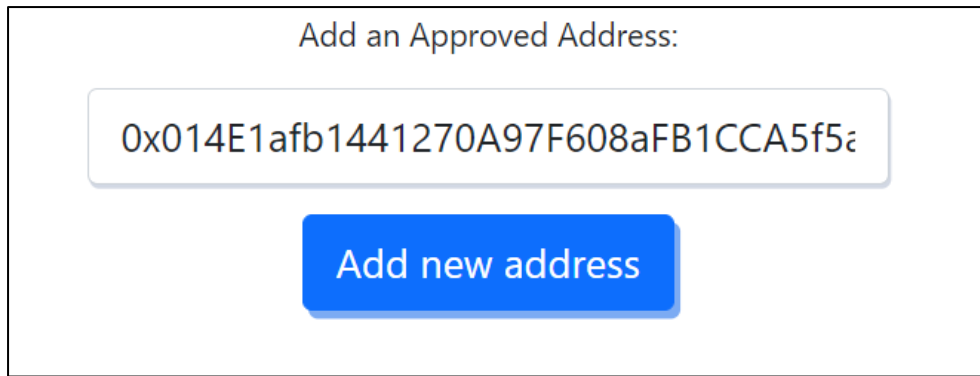


Figure 49: Using the form to define preferences

Clicking submit will have these preferences combined into a string, for the example in Figure 49 this would be “1234”, which would then be encrypted. Encryption is necessary to ensure that the user’s preferences aren’t clearly visible in transaction logs when making setPreferences calls to the contract, thus ensuring they stay private. Using the setPreferences method encrypted preferences are then stored on the blockchain, using the mappings discussed within implementation. The user can then specify who else can retrieve these preferences by adding approved addresses. To approve a smart building the user would have to find their blockchain address and add it. They do this by inputting the address into the input box seen in Figure 50, then clicking add new address. Ensuring only approved addresses can retrieve a user’s privacy preferences is how the system keeps control of the preferences within the user’s hands.



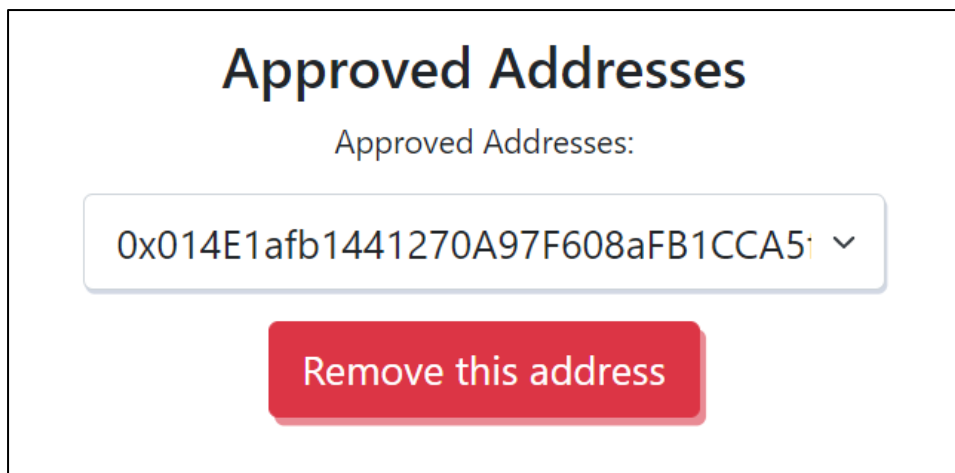
Add an Approved Address:

0x014E1afb1441270A97F608aFB1CCA5f5a

Add new address

Figure 50: Adding an approved address

After clicking add new address this address will be approved to retrieve the user's preferences and it will appear in the approved addresses dropdown (see Figure 51).



Approved Addresses

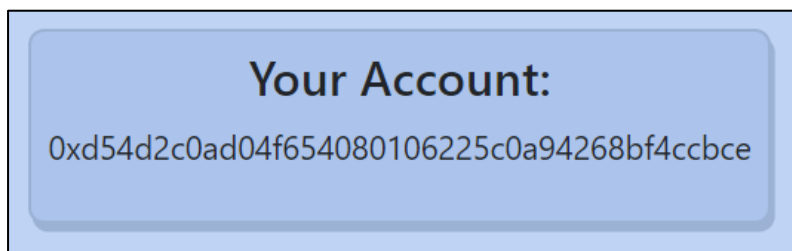
Approved Addresses:

0x014E1afb1441270A97F608aFB1CCA5f5a ▼

Remove this address

Figure 51: Approved Address dropdown after adding an address

The user would then need to share their blockchain address and secret key with the smart building system. The user's blockchain address is conveniently displayed in the top-bar of the DApp (see Figure 52).



Your Account:

0xd54d2c0ad04f654080106225c0a94268bf4ccbce

Figure 52: User's blockchain address as displayed on the DApp

For the smart building system to then access the preferences the user defined, first they'd need to SHA-3 hash the secret key, as the preferences are stored under a composite key using the hash of the secret key. This is to prevent leakage of the secret key within transaction logs, instead only the hash of the key can be seen. They would then call the contract `getPreferences` method through the console with the user's address and the key hash, like shown in Figure 53:

```
results = await app.getPreferences.call("0xd54d2c0ad04f654080106225c0a94268bf4ccbce",
"d0eeb3b13b634d27e08c6a9c6f336f50462cdf19df92353c45e48392dd2cafc48c794e110bfc06692345b0593f38820f71c6965028c999cd28d78abe57edf256",
{from: "0x014E1afb1441270A97F608aF81CCa5f5a8Cf8b3A"}))
```

← User Address
← Secret Key Hash
← Smart building/requester's address

Figure 53: Console command to retrieve user's preferences

As the address calling `getPreferences` is an approved address, the returned results variable will contain the user's encrypted preferences (Figure 54).

```
truffle(ganache)> results
'c94058d8'
```

Figure 54: Returned encrypted preferences

The smart building system could then AES decrypt these preferences using the secret key to receive the SITA dimensions the user defined (in this case "1234"). This process of hashing, retrieval, and decrypting is handled automatically for users when using the DApp, but isn't for smart buildings.

If the caller of `getPreferences` hadn't been an approved address, they would have received an error preventing them from retrieving the preferences. The error (Figure 55) is quite long, but unfortunately not too informative as Truffle console errors don't provide custom error messages, even when one is defined.

[illegible]

Figure 55: Error when trying to getPreferences from non-approved address

If a user wanted to prevent an address from retrieving their preferences again, they could remove them from the approved addresses by selecting the address in the dropdown and using the “Remove this address button” (as seen in Figure 51). Alternatively, the user could delete the preferences set altogether if they wished using the “Delete These Preferences” button seen in Figure 48.

This walkthrough showcases how both an ordinary user would define and share their preferences, and a smart building would attempt to retrieve them. If the user were to modify these privacy preferences or define a new set with a different key the process would be the same.

8.2 Evaluation of Results

8.2.1 Smart Contract

The smart contract met all the requirements set out for it within the design stage, but it is important to discuss to what extent and quality they were met.

All of the functions perform what is expected of them, but there is a discussion to be had surrounding their cost. As the system uses Ethereum there is a cost associated with actions that need to write to the blockchain. This includes setting/modifying preferences, adding/removing approved addresses, and deleting preferences. This means the vast majority of actions that a user would carry out when using the system cost Ether. Thus, firstly, it is worth considering if the smart contract functions are working efficiently, to use as little gas as possible. The contract implementation is heavily reliant on writing to and reading from its three mappings:

- `userPreferences`: Storing the encrypted user preferences.
- `usedKeys`: Storing all keys in use for each user.
- `approvedAddresses`: Storing the addresses approved to read each preferences set.

The `userPreferences` mapping is essential, as the preferences themselves must be stored decentralised to ensure the user has full ownership of them. It could be argued that `approvedAddresses` and `usedKeys`, however, don't necessarily need to be stored on the blockchain. The `usedKeys` mapping is mostly used to avoid errors by preventing operations on keys that do not exist, therefore, it probably could be removed and replaced with a different form of error-handling. Try and catch statements as well as `revert` could be used to catch and handle errors, rather than avoid them outright by checking if the key exists. The only functionality this change would sacrifice would be `deleteAllPreferences`, which relies on the `usedKeys` mapping. If cost did prove to be a problem for the system removing `usedKeys` and losing `deleteAllPreferences` may be a reasonable sacrifice, or perhaps an alternative and less costly version of `deleteAllPreferences` could be developed.

The `approvedAddresses` mapping could also be removed to reduce costs, however this may compromise the system too much to be worth it. One could argue that if the user keeps their keys secret as they're supposed to that there should be no need for approved addresses, as only parties the user shares their secret key with should know the key and thus be able to retrieve their preferences. However, blockchain's transparency means that anybody can read `getPreferences` calls to the contract in the transaction logs, and without approved addresses they could then make the same call and be returned the preferences. Technically, this user still wouldn't be able to decrypt the preferences retrieved as they don't have the key, just the hash of the key, however it is not good practice to allow the collection of encrypted data just because malicious parties won't be able to read it. It would be possible for parties to collect data from the system, and keep it until either the hashing algorithm, or encryption algorithm are broken [86]. Admittedly, this is a very specific danger to combat, and it may not be worth the extra cost associated. More reading and consulting the opinions of experts would be required to identify whether the risk justifies the cost.

Secondly, the cost of the contract raises questions of who will pay the costs. In the paper *A Blockchain-based approach for matching desired and real privacy settings of social network users* [22] they suggest deploying the contract to the Ethereum mainnet, and having the social media bear the costs of user transactions. Having this project's system deployed to the mainnet is certainly an option, and the

system could be extended to record transaction costs so that smart building systems can reimburse their users. This however relies on those in charge of the smart building management systems accepting these extra costs, as it is doubtful that users would want to bear the costs of managing their preferences. In fact, this could discourage users from setting and managing their preferences accurately for fear of incurring costs. Therefore, a reluctance from both parties to pay may require the contract to be deployed on a private or semi-private blockchain [12]. This would require smart buildings to host their own blockchain where they deploy the contract and allow users of the smart building to register and manage their preferences. Such a solution would allow the waiving of any transaction fees, eliminating any concerns regarding transaction costs. This would unfortunately come at the cost of the transferability available within the base system. With smart buildings running their own blockchain networks, users would no longer be able to easily share their preferences between smart buildings like the base system offers. There may also be some concerns regarding how decentralised the system actually is if the blockchain is hosted by the smart building that will use the preferences.

Thirdly, and importantly, the smart contract has not been checked to see if it is secure. Security of smart contracts is of great importance, particularly when handling sensitive data [87] such as the privacy preferences stored in this system. Securing against potential smart contract vulnerabilities such as: re-entrancy vulnerability, transaction-ordering dependence, or timestamp dependence [88] [17] was never considered during the development of the smart contract. Tools exist to detect vulnerabilities within smart contract bytecode [88], and ensuring your smart contract is secure is a necessity before deploying it for use.

Lastly, the smart contract isn't easy to connect to for smart building systems. As connecting this project's system to smart buildings is a core part, the lack of support for such a connection is not great. For example, with the current system, smart buildings must implement their own systems to SHA-3 hash the secret key provided by the user so that they can retrieve preferences, and to AES decrypt the retrieved preferences. Development of an API would be the ideal solution, providing an interface between smart building systems and the smart contract, rather than them interacting directly. It would allow the smart building system to simply feed the address and secret key to the API, and the API would handle all hashing, retrieval, and subsequent decryption. The smart building system would just receive the decrypted preferences in return. Such an abstraction would greatly simplify the connection between smart building systems and this project's system. Furthermore, Solidity events could be added to the smart contract to alert smart buildings when a preferences set they are interested in has been modified, so that they know to retrieve the updated preferences. The only reason such an API and events system wasn't developed for this project was time-constraints.

8.2.2 Cryptographic Methods

The cryptographic methods perform the tasks set out in design, however their security has not been verified. Firstly, the aes-js library used for encryption, and decryption, was produced by a developer in Canada not affiliated with any notable development firms [89]. This doesn't mean that the library is insecure, in fact it is used and trusted by over 150,000 projects on GitHub [90], but caution should certainly be taken. The crypto library used for generating a key is a little more trustworthy, as it is provided as part of Node.js. Node.js is open-source and widely used [91], thus it is highly unlikely that insecure cryptographic functions would be released to the public. Lastly, the js-sha3 library used to hash the secret key is produced by an unknown developer going by emn178 [92]. Once again though their library is heavily used, with over 240,000 GitHub repositories including it [93]. None of these

libraries shown any signs of being a security risk to the project and are often recommended in forums and guides. Nevertheless, a proper analysis of libraries should be done in future before using them, particularly where sensitive data may be involved.

8.2.3 DApp

The final version of the DApp provides the user access to all the necessary features required to utilise the system. As the DApp was produced in components, each individual component will be discussed and evaluated.

Your Account:

This component is quite simple, displaying the user's blockchain address to them much like many websites display the username when signed in (see Figure 56).

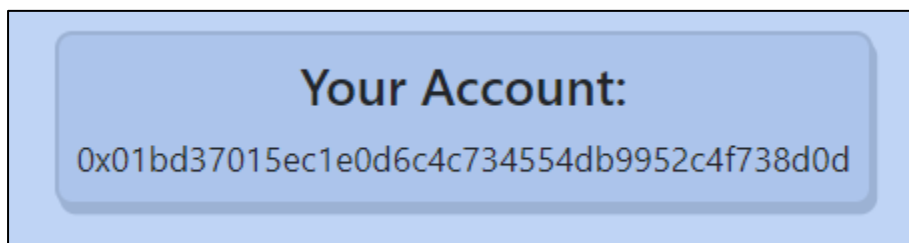


Figure 56: Your Account Component

The title and address are contained within a simple rounded container, the styling of which could certainly be improved. Displaying the blockchain address maybe isn't the most intuitive feature and displaying a username would certainly be simpler and easier for users to recognise. This would however require an accounts system to be developed, so that users can provide a username to attach to their blockchain address. This could be something to develop in the future and is the first example that the DApp design is not the most intuitive for an average user.

Secret Key Management:

Secret key management refers to the other section of the top-bar where the user enters the secret key and can retrieve and delete preferences (see Figure 57). There is not too much reasoning behind why this component was placed in the top-bar. It could be argued in combination with the user's address – also displayed in the top-bar – the key creates your 'login' of sorts, this is how you specify which preferences set you'd like to work with.

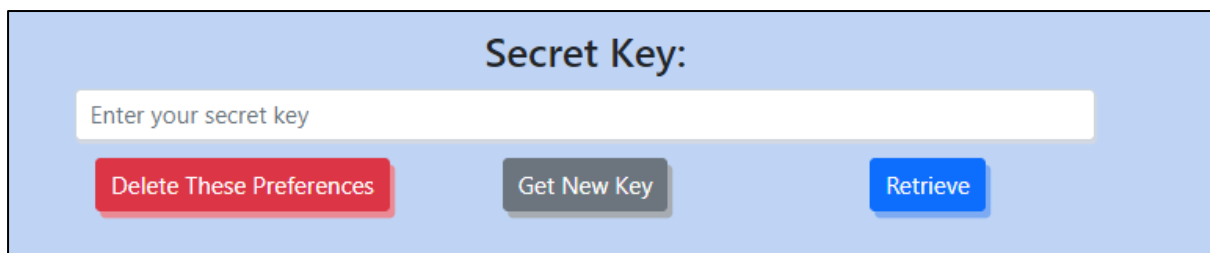
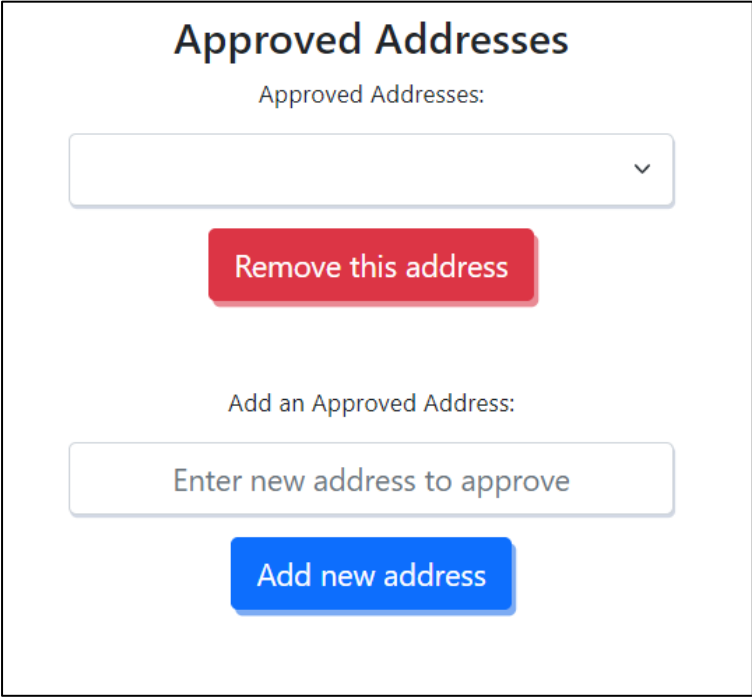


Figure 57: Secret Key Management Component

Once again, the look of this component is incredibly simple, and could be improved, but functionally the component works great. The secret key can be entered into the input box or clicking the get new key button will auto-fill the box with a key. All the buttons produce modals to warn the user or to get confirmation before performing the action (like deleting preferences). Clarity may be an issue however, as a new user to the system will be unlikely to understand what is meant by a secret key, or what exactly these buttons will do. This is a continuation of the design not catering to the average user, it is not intuitive what this component does.

Approved Addresses Management:

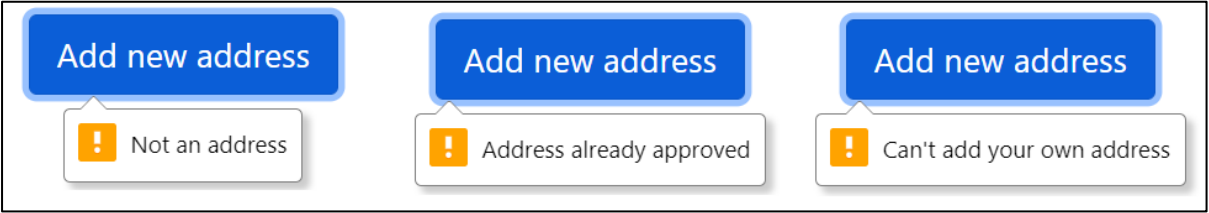
This component refers to the section of the UI dedicated to approved addresses. It is split into 2 sections: viewing/removing approved addresses and adding approved addresses (see Figure 58).



The image shows a UI component titled "Approved Addresses". It contains two main sections. The first section, labeled "Approved Addresses:", features a white drop-down menu with a downward arrow. Below it is a red button with the text "Remove this address". The second section, labeled "Add an Approved Address:", features a white text input box with the placeholder text "Enter new address to approve". Below the input box is a blue button with the text "Add new address".

Figure 58: Approved Addresses Component

Approved addresses are available for the user to view via the drop-down selection. This also doubles up as a selection for the remove this address button; the address selected in the drop-down is the one that will be removed when clicking the button. A text input box allows the user to enter a new address to be added, and errors messages are provided to inform the user if the address they entered is not valid (see Figure 59).



The image shows three blue buttons, each with the text "Add new address". Below each button is a white error message box with an orange exclamation mark icon. The error messages are: "Not an address", "Address already approved", and "Can't add your own address".

Figure 59: Add New Address Error Messages

Pressing the add new address button with a valid address in the input box will result in the address being added to approved addresses, and it will be displayed in the drop-down. The component is fully functional but could certainly look a lot better. More importantly this component once again isn't intuitive, it is unclear what approved addresses are and what adding or removing them does. This is only explained via pop-ups when the user tries to add or delete an approved address (shown in Figure 60). This isn't user-friendly and the whole design lacks clarity surrounding what it does.

<p>Are you sure you want to remove this approved address?</p> <p>Once removed this address will no longer be able to retrieve these preferences.</p> <p><input type="button" value="Cancel"/> <input type="button" value="Proceed"/></p>	<p>Are you sure you want to add this approved address?</p> <p>Adding this approved address will allow these preferences to be retrieved by the user at this address.</p> <p><input type="button" value="Cancel"/> <input type="button" value="Proceed"/></p>
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Figure 60: Add/Remove Address Confirmation Modals

This problem is potentially worsened by a user's lack of knowledge of blockchain, they may not understand what a blockchain address is, and thus may find it difficult to enter addresses to approve. Furthermore, it is quite difficult to remember who/what each address refers to. A resolution to this problem would be to add a nickname system, where the user can attach a name to each blockchain address. For example, one address may be labelled 'My office', or another 'Conference Room 1'. This would make it significantly clearer with who/where the preferences are being shared, and in turn makes access much easier to manage.

Preferences Form:

The preferences form provides a way for the user to define their SITA privacy preferences. It is a very simple form with four drop-down selections - one for each SITA dimension – for the user to choose their chosen privacy level (see Figure 61). Much like the rest of the DApp this is perfectly functional, but is rather simplistic in looks, and could be improved in that area.

Define Your Preferences

Spatial:

0. No Information ▼

Identity:

0. No Information ▼

Temporal:

0. No Information ▼

Activity:

0. No Information ▼

Submit

Figure 61: Preferences Form Component

Clarity, however, is a more pressing matter for the preferences form. Just like the rest of the DApp UI, it is never explained what this component is doing or what it means. There is no explanation as to what the 4 SITA dimensions are and what they mean, and it is also not explained what the different level options for each dimension mean. It cannot be expected that an ordinary user has read and understood *The SITA principle for Location Privacy – Conceptual Model and Architecture* to be able to use the application, particularly as it is intended to be used by regular occupants of smart buildings. Thus figuring out a way to easily, and simply, get across the ideas of SITA to an ordinary user would be extremely helpful to the clarity and usability of the DApp. This is certainly something to work on in the future.

Delete All Preferences Button:

Perhaps the most intuitive component in the DApp, the delete all preferences button does just that. This button (shown in Figure 62) is intentionally positioned out of the way in the bottom-bar of the application, because it is intended to only be used in emergency situations in which the user has lost a key/keys and really feels the need to prevent these preferences from being accessed anymore.

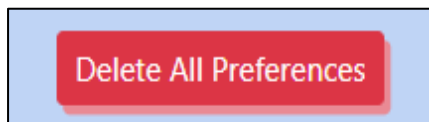


Figure 62: Delete All Preferences Component

Much like the rest of the UI, the styling could certainly be improved, but it is at least clearer what this component does. The confirmation modal that pops-up really ensures that it is obvious what clicking this button does (see Figure 63).

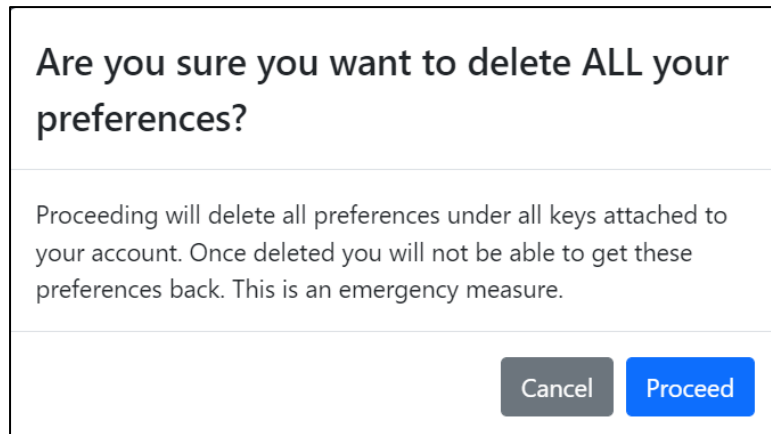


Figure 63: Delete All Preferences Confirmation Modal

8.2.4 Documentation

Very little documentation was produced for this project. The GitHub repository for the project does have a fairly detailed README file (Figure 64) explaining how to setup and run the project for yourself, but this is a technical document aimed at other developers. Similarly, the code is commented throughout to try aid developers to use and modify it.

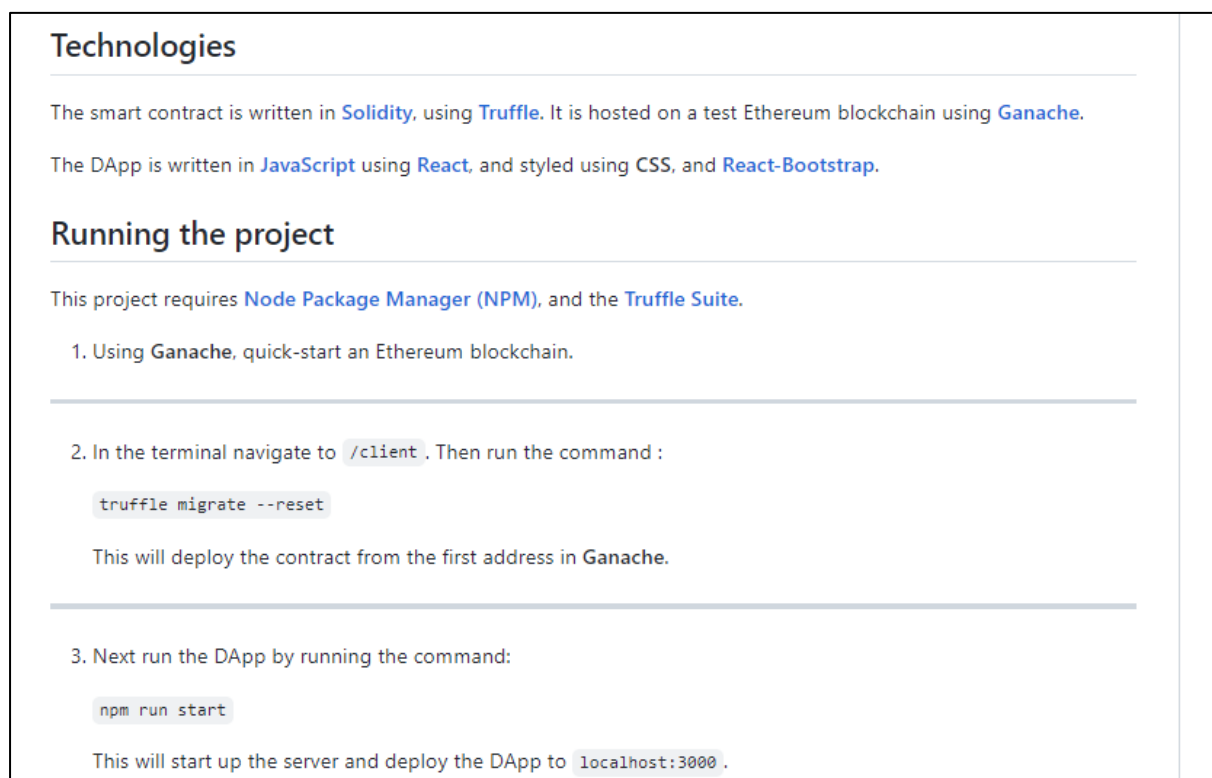


Figure 64: Excerpt from README.md

Consequently, the documentation for the project lacks the clarity problem suffered by much of the DApp, it provides no help to an ordinary user. There is no guide to help users utilise the system, and the DApp itself provides no help either. There is no explanation as to what a secret key is, what it does, what an approved address is etc. This would certainly be a focus for future development, as although

the project produced a working prototype, what does it matter if very few intended users would be able to use it.

Similarly, there is a lack of documentation for smart building systems. Outside of reading the code and the comments within it, there is no explanation of how a smart building system can connect to this system in order to retrieve the privacy preferences of their users. A proper set of documentation and guides should be developed alongside any future work on an API, as people need to be able to understand how to link their own system to this one.

8.3 Satisfaction of Objectives

The aim for this project was *“to produce a system using blockchain, which will allow users to store and manage their privacy preferences”*. This section looks back at the objectives set out to try and achieve this aim and looks to evaluate to what extent they have been fulfilled.

8.3.1 Objective 1

Research and explore blockchain technology and its potential uses in preserving user privacy.

This objective was satisfied by the research discussed in chapter 2 (background). Through a literature review it was learnt the key properties of blockchain, and similar works demonstrated how they can be applied to help preserve user privacy. A deeper knowledge of blockchain certainly could’ve been acquired with further research, in particular learning more about the challenges associated with blockchain. However, the timeline for the project most likely couldn’t stretch to include more research, so the outcome for this objective is satisfactory.

8.3.2 Objective 2

Investigate and learn about implementing blockchains, decentralised applications, and smart contracts. Explore the tools and processes involved.

This objective was fulfilled. Through a combination of the literature review, as well as research into technologies (see chapter 5), a solid understanding of implementing blockchains, decentralised applications, and smart contracts was garnered. There is – of course – plenty of further learning that can be done on the topic, particularly surrounding some of the more advanced features and good development practices. Overall though the objective itself was satisfied, as an understanding was gained, and tools and technologies were even identified which would be used in the system.

8.3.3 Objective 3

Use the knowledge gained to produce an overview of the system.

This objective was completely fulfilled. With the newly acquired understanding of DApp infrastructure, a full model was developed for the system (see chapter 4). This model showcased all the different parts necessary for the system, and these constituent parts were further broken down to produce requirements. This design process was quite successful, as the completed system barely strayed from the designs made, with just a few minor oversights having to be added during development.

8.3.4 Objective 4

Produce the system outlined using the technologies and processes learned about.

This is the most difficult objective to evaluate as although the system was implemented, as discussed a lot earlier in this chapter, there is plenty of room for improvement. A system was produced using the outline, but it is a fairly basic realisation of the established requirements. We will consider the working prototype produced a success but will not hide the areas in which it is lacking.

8.3.5 Objective 5

Discuss the ways the system could be expanded, and its potential role as part of a bigger picture in maintaining user privacy preferences in smart buildings.

This objective is also somewhat hard to evaluate, because – as discussed within the introduction (chapter 1) – this objective looks to be fulfilled within this dissertation. Explicit discussions regarding the bigger picture are mostly constrained to the first 2 chapters, however plenty of references to this system's involvement with smart buildings are made throughout. Consequently, we will tentatively say this objective was achieved as the discussion has been had, albeit maybe not quite to the full extent desired.

9. Conclusions

9.1 Reflection

I feel that the project's original aim was fulfilled, as the system produced for this dissertation uses blockchain to allow users to store and manage their privacy preferences. I would consider the system as a prototype with many areas that could be improved upon; there is more discussion of this within the next section on future work. The system acts more as a proof of concept as it has never been deployed outside of my local machine. It would likely be easy to reproduce on other machines but deploying it as an actual organisation - looking to use it in conjunction with your smart building management system - would probably be a step too far for the current state of the project.

Throughout the project I have learned a lot. Firstly, through background reading and literature review for this project I have learned about blockchain, and I am now quite interested in the application of blockchain, and its continuing growth. I also learned about privacy, in particular the concerns around user privacy as every aspect of our lives becomes dominated by technology. This extends to smart buildings, where I found out just how significant the data collected by sensors can be. Even just a smart meter can reveal so much about a person's life, and they may not even realise it [5]. Through all this reading I also learnt how to interact with the computer science knowledge base. I had never really interacted with computer science academic literature previously, but now I am equipped with the skills to know how to find and use the literature to learn.

Secondly, I learnt a lot of new technologies and skills. JavaScript and Solidity were brand new languages to me, but throughout this project I have grown more comfortable with them. JavaScript was fairly straightforward to learn, with plenty of guides and documentation. The only real problems presented were to do with finding the right libraries, as discussed in implementation (chapter 6). Solidity was the more difficult of the two to learn, requiring some problem solving for problems which other high-level languages would provide in-built keywords or methods for. For example, Solidity's lack of an `in` keyword, or the inability to store objects in a mapping. Regardless I would happily use both languages in future. I also learnt about using React and Bootstrap to produce UI for applications and would consider them when producing JavaScript front ends in future. I would also look to use the Truffle Suite again for any DApp development, as it worked well throughout and provides a great basis to start building from.

Lastly, I learnt a lot about project management. This project was the most structured project I have ever undertaken, and the first where I have really tried to follow a development methodology. Research and learning at the start of the project really ensured I had a good understanding of the technologies and concepts I'd need for my system, rather than trying to learn everything during development. System design was invaluable to the success of this project, and it is one of the first times I have ever thoroughly planned my system before starting development. Creating a detailed plan for my system and breaking it down into individual features really helped me to understand what I needed to build to accomplish my aim, as well as providing a great way to track progress as the project went on. The quality of my designs and planning streamlined implementation as I already had an outline of each feature, and there were very few surprises during development as a consequence. My development approach and methodology weren't without their problems, however. Although the project was broken down into parts, and then features, a detailed time plan was never created. Progress was clear as more and more features were completed, but this never accounted for how much time each feature would take to complete. Furthermore, features weren't developed in sprints [94] like they would be during actual FDD, with features just being developed until completion,

however long that took. Luckily, this never became a problem for this project with development being completed in good time. But, a more detailed time plan for the development of this project might've been beneficial. Allocating an amount of development time to each feature and performing a sprint on them would've created a well-established timetable for the development, ensuring it would be completed on time. Thus – despite this being the most structured project I have ever developed – I still have areas in which to improve my development practices in future.

9.2 Future Work

As touched on within the results and evaluation chapter there are several areas in which the project could be improved and extended, they are briefly detailed here.

9.2.1 Re-design the DApp UI

The DApp UI is fairly basic and could certainly be made more aesthetically pleasing. A major part of the re-design would be to improve the clarity, and usability for ordinary users, as the current UI is not very intuitive.

9.2.2 API

The main extension to the system is to enable its main purpose, being linked to smart building management systems. This would require the development of an API, to provide an easy way to connect this project to smart building management systems. This would include adding events to the smart contract to alert the smart building system when preferences they are using have been updated.

9.2.3 Deployment

An investigation into how best to deploy this system would be useful also. This includes hosting of the DApp, but also whether the smart contract should be deployed to a public blockchain (like the Ethereum Mainnet) or to a private/corporate blockchain. There is some discussion to this affect within the results section of this paper, but a more detailed study would be good.

9.2.4 Smart Contract Security Analysis

There are many security concerns regarding smart contracts. This system's contract would be handling sensitive privacy preferences, so it needs to be secure. A proper analysis to ensure the contract is secure and bug-free would be important before it could be deployed in real world scenarios.

9.2.5 Smart Contract Cost Analysis

Utilising the system to store and manage privacy preferences costs Ether. You would want the system's operations to cost users as little as possible to encourage them to correctly specify and update their privacy preferences without concern for the cost. Therefore, an analysis and modification of the contract functions to ensure they use as little gas as possible would be useful.

9.2.6 Documentation and Guides

This project lacks good documentation and guides, and thus it isn't as easy to use as it should be. This goes hand-in-hand with the re-designing of the UI, as clarity for ordinary users is a major concern. However, there is also no documentation for smart building systems on how to use the system either. Good documentation to this affect should be produced alongside the aforementioned API. Finally, documentation would need to be produced for those deploying the system (assuming it isn't deployed publicly), this would be a product of the investigation into deployment.

10. References

- [1] “Introduction to dapps,” *ethereum.org*, Apr. 11, 2022.
<https://ethereum.org/en/developers/docs/dapps/> (accessed May 03, 2022).
- [2] “Crypto basics - What is a crypto wallet?,” *www.coinbase.com*.
<https://www.coinbase.com/learn/crypto-basics/what-is-a-crypto-wallet/> (accessed May 01, 2022).
- [3] G. Zyskind, O. Nathan, and A. S. Pentland, “Decentralizing privacy: Using blockchain to protect personal data,” in *2015 IEEE CS Security and Privacy Workshops*, 2015, pp. 180–184. doi: 10.1109/SPW.2015.27.
- [4] M. Rudolph, S. Polst, and J. Doerr, “Enabling users to specify correct privacy requirements,” in *International Working Conference on Requirements Engineering: Foundation for Software Quality*, Mar. 2019, pp. 39–54. doi: 10.1007/978-3-030-15538-4_3.
- [5] S. Cejka, F. Knorr, and F. Kintzler, “PRIVACY ISSUES IN SMART BUILDINGS BY EXAMPLES IN SMART METERING,” Jun. 2019.
- [6] P. Pappachan *et al.*, “Towards privacy-aware smart buildings: Capturing, communicating, and enforcing privacy policies and preferences,” Jun. 2017.
- [7] M. Andersen, M. Kjærgaard, and K. Grønbæk, “The SITA principle for location privacy — Conceptual model and architecture,” in *2013 International Conference on Privacy and Security in Mobile Systems (PRISMS)*, Jun. 2013, pp. 1–8. doi: 10.1109/PRISMS.2013.6927184.
- [8] T. Bell, “What is a Smart Building?,” *www.trueoccupancy.com*, Jul. 05, 2021.
<https://www.trueoccupancy.com/blog/what-is-a-smart-building>
- [9] K. Weekly *et al.*, “Building-in-briefcase: A rapidly-deployable environmental sensor suite for the smart building,” *Sensors*, vol. 18, Art. no. 5, 2018, doi: 10.3390/s18051381.
- [10] M. Jin, N. Bekiaris-Liberis, K. Weekly, C. J. Spanos, and Bayen, Alexandre M, “Occupancy detection via environmental sensing,” *IEEE Transactions on Automation Science and Engineering*, vol. 15, no. 2, Art. no. 2, 2018, doi: 10.1109/TASE.2016.2619720.
- [11] M. Ahmed, I. Elahi, A. Muhammad, U. Aslam, I. Khalid, and M. A. Habib, “Understanding blockchain: Platforms, applications and implementation challenges,” in *ICFNDS '19: Proceedings of the 3rd International Conference on Future Networks and Distributed Systems*, Jul. 2019, pp. 1–8. doi: 10.1145/3341325.3342033.
- [12] S. S. Sarmah, “Understanding blockchain technology,” vol. 8, pp. 23–29, Aug. 2018, doi: 10.5923/j.computer.20180802.02.
- [13] Z. Zheng, S. Xie, H. Dai, X. Chen, and H. Wang, “An overview of blockchain technology: Architecture, consensus, and future trends,” in *2017 IEEE International Congress on Big Data (BigData Congress)*, 2017, pp. 557–564. doi: 10.1109/BigDataCongress.2017.85.
- [14] “What is Decentralization?,” *Amazon Web Services, Inc.* <https://aws.amazon.com/blockchain/decentralization-in-blockchain/>

- [15] Roben Castagna Lunardi, Henry Cabral Nunes, S. Branco, Bruno Hugentobler Lipper, Charles Varlei Neu, and Avelino Francisco Zorzo, "Performance and cost evaluation of smart contracts in collaborative health care environments," *CoRR*, vol. abs/1912.09773, 2019, [Online]. Available: <http://arxiv.org/abs/1912.09773>
- [16] IBM, "What are smart contracts on blockchain?," *www.ibm.com*.
<https://www.ibm.com/topics/smart-contracts>
- [17] Maher Alharby and Aad van Moorsel, "Blockchain-based smart contracts: A systematic mapping study," *CoRR*, vol. abs/1710.06372, 2017, [Online]. Available: <http://arxiv.org/abs/1710.06372>
- [18] Vitalik Buterin, "A NEXT GENERATION SMART CONTRACT & DECENTRALIZED APPLICATION PLATFORM," 2015.
- [19] S. Porru, A. Pinna, M. Marchesi, and R. Tonelli, "Blockchain-oriented software engineering: Challenges and new directions," in *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)*, 2017, pp. 169–171. doi: 10.1109/ICSE-C.2017.142.
- [20] M. E. Peck, "Blockchain world - Do you need a blockchain? This chart will tell you if the technology can solve your problem," *IEEE Spectrum*, vol. 54, no. 10, Art. no. 10, 2017, doi: 10.1109/MSPEC.2017.8048838.
- [21] F. Wessling, C. Ehmke, M. Hesenius, and V. Gruhn, "How much blockchain do you need? Towards a concept for building hybrid DApp architectures," 2018, pp. 44–47.
- [22] G. Lax, A. Russo, and Lara Saidia Fasci, "A Blockchain-based approach for matching desired and real privacy settings of social network users," *Information Sciences*, vol. 557, pp. 220–235, 2021, doi: <https://doi.org/10.1016/j.ins.2021.01.004>.
- [23] "IBM i: Cryptography concepts," *www.ibm.com*, Sep. 08, 2021.
<https://www.ibm.com/docs/en/i/7.4?topic=cryptography-concepts> (accessed May 01, 2022).
- [24] D. P. Mahajan and A. Sachdeva, "A Study of Encryption Algorithms AES, DES and RSA for Security," *Global Journal of Computer Science and Technology*, vol. 13, no. 15, Dec. 2013, [Online]. Available: <https://computerresearch.org/index.php/computer/article/view/272/272>
- [25] R. Stubbs, "Symmetric Encryption Algorithms - Their Strengths and Weaknesses, and the Need for Crypto-Agility," *www.cryptomathic.com*, Mar. 12, 2019. <https://www.cryptomathic.com/news-events/blog/symmetric-encryption-algorithms-their-strengths-and-weaknesses-and-the-need-for-crypto-agility#:~:text=AES%20is%20the%20symmetric%20algorithm>
- [26] M. J. Dworkin, "SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions," Jul. 2015, doi: 10.6028/nist.fips.202.
- [27] Agile Alliance, "What is Agile Software Development?," *Agile Alliance*, Jan. 16, 2019.
<https://www.agilealliance.org/agile101/>
- [28] R. Lynn, "What is FDD in Agile?," *Planview*. <https://www.planview.com/resources/articles/fdd-agile/>

- [29] Lucid Content Team, "Why (and How) You Should Use Feature-Driven Development," *www.lucidchart.com*, Dec. 04, 2019. <https://www.lucidchart.com/blog/why-use-feature-driven-development>
- [30] "Sweet Tools for Smart Contracts," *Truffle Suite*. <https://trufflesuite.com/>
- [31] "Ethereum development environment for professionals by Nomic Labs," *hardhat.org*. <https://hardhat.org/>
- [32] "Embark into the Ether. | Embark," *framework.embarklabs.io*. <https://framework.embarklabs.io/> (accessed May 01, 2022).
- [33] "ConsenSys Quorum," *ConsenSys*. <https://consensys.net/quorum/>
- [34] A. Irrera, "ConsenSys acquires JPMorgan's blockchain platform Quorum," *Reuters*, Aug. 25, 2020. [Online]. Available: <https://www.reuters.com/article/us-jpmorgan-consensys-quorum-idUSKBN25L1MR>
- [35] "Public and private transactions - GoQuorum," *consensys.net*, Mar. 10, 2022. <https://consensys.net/docs/goquorum/en/latest/concepts/privacy/private-and-public/> (accessed May 01, 2022).
- [36] "Truffle Suite - Truffle Boxes," *trufflesuite.com*. <https://trufflesuite.com/boxes/>
- [37] "npm | build amazing things," *Npmjs.com*, 2019. <https://www.npmjs.com/>
- [38] "MetaMask," *Metamask.io*, 2019. <https://metamask.io/>
- [39] "Introduction | MetaMask Docs," *docs.metamask.io*. <https://docs.metamask.io/guide/#account-management>
- [40] ConsenSys, "MetaMask Surpasses 10 Million MAUs, Making It the World's Leading Non-Custodial Crypto Wallet," *ConsenSys*, Aug. 31, 2021. <https://consensys.net/blog/press-release/metamask-surpasses-10-million-maus-making-it-the-worlds-leading-non-custodial-crypto-wallet/>
- [41] T. T. Lee, "Comparison of The Top 10 Smart Contract Programming Languages in 2021," *pontem.network*, Sep. 24, 2021. <https://pontem.network/posts/comparison-of-the-top-10-smart-contract-programming-languages-in-2021>
- [42] I. Nikolic, A. Kolluri, Ilya Sergey, P. Saxena, and Aquinas Hobor, "Finding the greedy, prodigal, and suicidal contracts at scale," *CoRR*, vol. abs/1802.06038, 2018, [Online]. Available: <http://arxiv.org/abs/1802.06038>
- [43] "Vyper — Vyper documentation," *vyper.readthedocs.io*. <https://vyper.readthedocs.io/en/stable/> (accessed May 01, 2022).
- [44] "Rust Programming Language," *Rust-lang.org*, 2018. <https://www.rust-lang.org/>
- [45] MDN Contributors, "JavaScript," *MDN Web Docs*, Jul. 19, 2019. <https://developer.mozilla.org/en-US/docs/Web/javascript>

- [46] "Stack Overflow Developer Survey 2020," *Stack Overflow*.
<https://insights.stackoverflow.com/survey/2020#technology-programming-scripting-and-markup-languages-professional-developers>
- [47] Node.js Foundation, "About | Node.js," *Node.js*, 2014. <https://nodejs.org/en/about/>
- [48] P. Teixeira, *Professional Node.js building JavaScript-based scalable software*. Indianapolis, Ind. Wiley, Wrox, 2013.
- [49] Facebook, "React – A JavaScript library for building user interfaces," *Reactjs.org*.
<https://reactjs.org/>
- [50] E. You, "Vue.js," *Vuejs.org*, 2000. <https://vuejs.org/>
- [51] "Introducing JSX – React," *Reactjs.org*, 2019. <https://reactjs.org/docs/introducing-jsx.html>
- [52] "React Truffle Box - Truffle Suite," *trufflesuite.com*. <https://trufflesuite.com/boxes/react/>
- [53] J. Papa, "lite-server," *npm*. <https://www.npmjs.com/package/lite-server> (accessed May 02, 2022).
- [54] G. McCubbin, "The Ultimate Ethereum Dapp Tutorial (How to Build a Full Stack Decentralized Application Step-By-Step) | Dapp University," *Dapp University*, Jan. 09, 2020.
<https://www.dappuniversity.com/articles/the-ultimate-ethereum-dapp-tutorial>
- [55] M. Otto, "Bootstrap," *Getbootstrap.com*, 2000. <https://getbootstrap.com/>
- [56] "React Bootstrap," *Github.io*, 2020. <https://react-bootstrap.github.io/>
- [57] Microsoft, "Visual Studio Code," *Visualstudio.com*, Apr. 14, 2016.
<https://code.visualstudio.com/>
- [58] "Language Support in Visual Studio Code," *code.visualstudio.com*, Mar. 30, 2022.
<https://code.visualstudio.com/docs/languages/overview>
- [59] "Managing Extensions in Visual Studio Code," *code.visualstudio.com*, Mar. 30, 2022.
<https://code.visualstudio.com/docs/editor/extension-marketplace>
- [60] "Github Language Stats," *madnight.github.io*.
https://madnight.github.io/github/#/pull_requests/2021/4
- [61] Atlassian, "What is version control," *Atlassian*, 2019.
<https://www.atlassian.com/git/tutorials/what-is-version-control>
- [62] "GitHub Desktop," *GitHub Desktop*, 2019. <https://desktop.github.com/>
- [63] "Git - Downloads," *git-scm.com*. <https://git-scm.com/downloads>
- [64] "Python in Keyword," *www.w3schools.com*.
https://www.w3schools.com/python/ref_keyword_in.asp (accessed May 02, 2022).

- [65] "Java String contains() Method," *www.w3schools.com*.
https://www.w3schools.com/java/ref_string_contains.asp (accessed May 02, 2022).
- [66] "Keccak Team," *keccak.team*. <https://keccak.team/keccak.html>
- [67] "Expressions and Control Structures — Solidity 0.4.24 documentation," *docs.soliditylang.org*.
<https://docs.soliditylang.org/en/v0.4.24/control-structures.html> (accessed May 02, 2022).
- [68] S. McKie, "Solidity Learning: Revert(), Assert(), and Require() in Solidity, and the New REVERT Opcode in the...," *BlockChannel*, Sep. 30, 2017. <https://medium.com/blockchannel/the-use-of-revert-assert-and-require-in-solidity-and-the-new-revert-opcode-in-the-evm-1a3a7990e06e> (accessed May 02, 2022).
- [69] E. Vosberg, "crypto-js," *npm*, Dec. 14, 2016. <https://www.npmjs.com/package/crypto-js>
- [70] D. Lehn, M. Sporny, and D. Longley, "node-forge," *npm*.
<https://www.npmjs.com/package/node-forge>
- [71] R. Moore, "aes-js," *npm*. <https://www.npmjs.com/package/aes-js> (accessed May 02, 2022).
- [72] "Crypto | Node.js v14.5.0 Documentation," *nodejs.org*. <https://nodejs.org/api/crypto.html>
- [73] N-able, "Advanced Encryption Standard: Understanding AES 256," *N-able*, Jul. 29, 2019.
<https://www.n-able.com/blog/aes-256-encryption-algorithm#:~:text=AES%20128%20uses%2010%20rounds> (accessed May 02, 2022).
- [74] emn178, "js-sha3," *npm*. <https://www.npmjs.com/package/js-sha3> (accessed May 02, 2022).
- [75] "Thinking in React – React," *reactjs.org*. <https://reactjs.org/docs/thinking-in-react.html>
- [76] "Components and Props – React," *Reactjs.org*, 2019. <https://reactjs.org/docs/components-and-props.html>
- [77] "Window.alert()," *MDN Web Docs*. <https://developer.mozilla.org/en-US/docs/Web/API/Window/alert>
- [78] "Window.confirm() - Web APIs | MDN," *developer.mozilla.org*.
<https://developer.mozilla.org/en-US/docs/Web/API/Window/confirm>
- [79] M. O. contributors Jacob Thornton, and Bootstrap, "Modal," *getbootstrap.com*.
<https://getbootstrap.com/docs/4.0/components/modal/>
- [80] I Sommerville, *Software engineering*, 9th ed. Boston: Pearson/Addison-Wesley, 2011.
- [81] "Types of Software Testing: Different Testing Types with Details," *Softwaretestinghelp.com*, Apr. 07, 2017. <https://www.softwaretestinghelp.com/types-of-software-testing/>
- [82] Atlassian, "The different types of testing in Software | Atlassian," *Atlassian*, 2019.
<https://www.atlassian.com/continuous-delivery/software-testing/types-of-software-testing>
- [83] "Jest · Delightful JavaScript Testing," *Jestjs.io*, 2017. <https://jestjs.io/>

- [84] N. McCurdy, "React Testing Library | Testing Library," *testing-library.com*, Jul. 21, 2021. <https://testing-library.com/docs/react-testing-library/intro/> (accessed May 02, 2022).
- [85] "What is Functional Testing? Types & Examples | Micro Focus," *Microfocus.com*, 2022. <https://www.microfocus.com/en-us/what-is/functional-testing#:~:text=Functional%20testing%20is%20a%20type,with%20the%20end%20user's%20expectations.> (accessed May 12, 2022).
- [86] Patrick Howell O'Neill, "The US is worried that hackers are stealing data today so quantum computers can crack it in a decade," *MIT Technology Review*, Nov. 03, 2021. <https://www.technologyreview.com/2021/11/03/1039171/hackers-quantum-computers-us-homeland-security-cryptography/> (accessed May 09, 2022).
- [87] W. Zou *et al.*, "Smart contract development: Challenges and opportunities," *IEEE Transactions on Software Engineering*, vol. 47, no. 10, Art. no. 10, 2021, doi: 10.1109/TSE.2019.2942301.
- [88] J. Xu, F. Dang, X. Ding, and M. Zhou, "A survey on vulnerability detection tools of smart contract bytecode," in *2020 IEEE 3rd International Conference on Information Systems and Computer Aided Education (ICISCAE)*, 2020, pp. 94–98. doi: 10.1109/ICISCAE51034.2020.9236931.
- [89] "ricmoo - Overview," *GitHub*. <https://github.com/ricmoo> (accessed May 03, 2022).
- [90] "Network Dependents · ricmoo/aes-js," *GitHub*. https://github.com/ricmoo/aes-js/network/dependents?package_id=UGFja2FnZS0xODU3ODlyNw%3D%3D (accessed May 03, 2022).
- [91] <https://www.facebook.com/nodejsfoundation>, "Introduction to Node.js," *Introduction to Node.js*, 2019. <https://nodejs.dev/>
- [92] "emn178 - Overview," *GitHub*. <https://github.com/emn178> (accessed May 03, 2022).
- [93] "Network Dependents · emn178/js-sha3," *GitHub*. https://github.com/emn178/js-sha3/network/dependents?package_id=UGFja2FnZS0xNTE3OTY2MQ%3D%3D
- [94] Atlassian, "Sprints | Atlassian," *Atlassian*, 2022. <https://www.atlassian.com/agile/scrum/sprints> (accessed May 04, 2022).