Aditya Subramanian *et al*, International Journal of Computer Science and Mobile Applications,
Vol.6 Issue. 4, April- 2018, pg. 19-26 **ISSN: 2321-8363**
Impact Factor: 5.515

Floor Planning Application using Smart Contracts

Aditya Subramanian[1], Sanjeev Rao[2], Mr. Muthamilselvan S[3]
[1]*Student, Department of CSE, SRM University, Ramapuram, aditya21subu@gmail.com*
[2]*Student, Department of CSE, SRM University, Ramapuram, rao.sanjeev0@gmail.com*
[3]*Professor, Department of CSE, SRM University, Ramapuram*

Abstract

A blockchain is an interesting recent innovation, which, at its core, is essentially a distributed, decentralized public ledger; a few cryptocurrencies (which make use of a blockchain at its core) have had the feature of smart contracts added to them recently. Smart contracts are pieces of code, a binding contract, which runs on the network; these can be used to store, retrieve and perform other operations on the blockchain. Due to the distributed nature of the blockchain, the possibilities of smart contracts are only beginning to be realized, with many potential applications for the future. This project is a floor planning application that makes use of the preceding smart contracts to store data on the Ethereum network. Floor planning applications are commonly used in construction, and by designers, to represent the overhead structure of a room or set of rooms. Typically, a user would create a plan and save it on their computer, or to cloud storage platforms, and possibly send it to another user later, for review or other purposes. While data loss is less of an issue nowadays, especially with the rising popularity of cloud-based storage solutions which automatically manage storage to minimize risk of data loss, it is not unheard of for disasters to occur which may affect a portion of users. By using the Ethereum blockchain to store data, it is possible to avoid costs associated with that of storage on servers, and in theory, can prevent data loss, or reduce the risk of it occurring by a large amount, due to the redundant storage available due to the distributed nature of the chain. Furthermore, using a blockchain has additional advantages: by its nature, it has the feature of immutability, and by extension, nonrepudiation; since data on a blockchain is public by default, it is also possible to build a "map" of plans which have been stored, allowing for greater collaboration between various interested parties.

Keywords: Smart contracts, block chain, application, data loss, servers, data.

1. Introduction

Smart contracts are pieces of code, a binding contract, which runs on the network; these can be used to store, retrieve and perform other operations on the blockchain. Floor planning applications are commonly used in construction, and by designers, to represent the overhead structure of a room or set of rooms. Typically, a user would create a plan and save it on their computer, or to cloud storage platforms, and possibly send it to another user later, for review or other purposes. By using the Ethereum blockchain to store data, it is possible to avoid costs associated with that of storage on servers, and in theory, can prevent data loss, or reduce the risk of it occurring by a large amount, due to the redundant storage available due to the distributed nature of the chain. Furthermore, using a blockchain has additional advantages: by its nature, it has the feature of immutability, and by extension, nonrepudiation; since data on a blockchain is public by default, it is also possible to build a "map" of plans which have been stored, allowing for greater collaboration between various interested parties.

A blockchain is an interesting recent innovation, which, at its core, is essentially a distributed, decentralized public ledger; a few cryptocurrencies (which make use of a blockchain at its core) have had the feature of smart contracts added to them recently. Smart contracts are pieces of code, a binding contract, which runs on the network; these can be used to store, retrieve and perform other operations on the blockchain. Due to the

distributed nature of the blockchain, the possibilities of smart contracts are only beginning to be realized, with many potential applications for the future. This project is a floor planning application that makes use of the preceding smart contracts to store data on the Ethereum network. Floor planning applications are commonly used in construction, and by designers, to represent the overhead structure of a room or set of rooms. Typically, a user would create a plan and save it on their computer, or to cloud storage platforms, and possibly send it to another user later, for review or other purposes. While data loss is less of an issue nowadays, especially with the rising popularity of cloud-based storage solutions which automatically manage storage to minimize risk of data loss, it is not unheard of for disasters to occur which may affect a portion of users.

Section 2 of this paper explains the existing system present. Section 3 is a brief survey of the issues with the existing system; Section 4 then explains the system in detail, with the modules and the overarching description and working. Section 5 describes how the implementation is made. Finally, section 6 (conclusion & future work) concludes the paper.

2. Existing System

Currently, floor plans are made with specialized floor planning software. Floor planning software is software that is like other designing software, but with a specific focus on the creation of floor plans: while a regular designing application allows for generic items to be used with customized shapes being drawn, with layer effects and other filters, there is no such feature or feature set available for a floor planning application. Instead, what it offers in exchange for reduced functionality is that of enhanced features in the field specific to architects and designers: the ability of dimensioning (precise sizes for items), a wide range of items for creating a floor plan, 3D view of the building automatically generated for the floor plan, amongst others, features which would otherwise not be available for the generic design software. However, there are a few drawbacks or deficiencies that are present in the current system that are better addressed by this project.

Data storage in general is limited to the local user's hard disk, or their flash-disks and other backup drives; in the recent past, the rise of cloud computing means that there has been a higher focus on storing data offsite, to reduce the number of accidents that could occur which lead to the loss of data. However, it is believed that this measure is also insufficient.

Most planning software is part of a suite of other design applications; they may offer their own cloud-based storage to the user. However as has been described earlier in the previous sections, it is not quite 'a free lunch' – the user must pay a (usually recurring) cost to maintain their files online; if they do not pay for the service they can no longer store files to the cloud, and worse, risk having their data deleted. Such pricing schemes are usually costly, with 'personal' to 'enterprise' licenses costing anywhere between $100 to $3000 or more! Even a casual user would face difficulty facing these costs and acting as a barrier to entry (because the option of using the software usually requires them to register for their service as well.)

Other systems have been proposed which aim to use the blockchain in a similar manner, however they seem to deal with different use-cases – in particular, applications from storing healthcare data [3] to generalized data have been explored; however, they have their drawbacks and/or are not quite formalized yet.

3. Issues with existing system

As mentioned earlier, services exist for storing data on the cloud, providing an extra layer of data redundancy and security. However, it is believed that this approach is not enough by itself; as mentioned earlier, there have been cases where the cloud-based storage has known to fail. Despite having high availability and low failure rates, there have still been cases where users have found out that their cloud-storage-based data has been erased as a result of an accident or some other cause; while this is much less the case for paid cloud storage solutions, which also have extra insurances versus free storage, the paid storage solutions are still high in cost, and as such, few users opt to use those.

Further is the issue of immutability. Cloud storage, and indeed any generic storage, is mutable, which means that it can be changed once written to; however, this is not always a good thing. Viruses, ransomware and other

malware can all affect storage on the user's end; some 'smarter' viruses have even been able to access the user's cloud storage data if they were logged in accidentally at the time! By making specific data sets immutable (as the user decides), it is possible to have a solution that is not at risk for being modified inadvertently, or by third parties. Lack of modification ability by a third party also lends itself to the property of nonrepudiation [6]; no party can claim that the result was not what was intended – for example, in the existing system, any party can modify their copy of a file (in this case, a plan) and claim to have received it as is, and neither party can prove that their version is the original (file creation & modification dates are an improvement, but not fool proof as they can be spoofed). Distributed applications which make use of smart contracts are often run autonomously, and without direct control by any singular party, over the data and operations carried out by the smart contract; even though there is a lack of standardization of these terms [2], the consensus is that it is an incorruptible and autonomous system, which cannot be stopped.

With ever-growing interest in the blockchain, more possibilities for its usage, along with related technology like smart contracts, are being realized. By distributing data on the blockchain, it is possible to realize many of the benefits earlier seen which are otherwise not easily available on traditional systems. However, it carries with it its own set of challenges, especially due to the widespread damage caused by earlier accidents [5]; despite this, it looks to be a solid path for the future, especially with more research being conducted into new methods which can be used to prevent or mitigate the damage caused by a system of formal verification [1], which should help ease the transition for future applications.

4. Method & System Design

The following section aims to describe in detail the design considerations that were taken in the construction of the system, along with their accompanying flowcharts and system descriptions. The overall working of the system has also been described in high level terms.

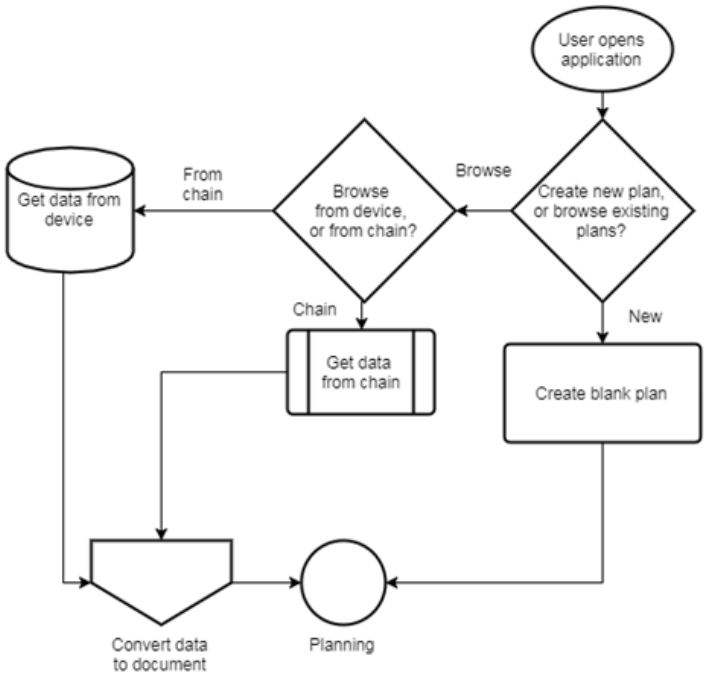

Figure 1: Architecture diagram of system

The phases are:

User Interface: The user interface module is the front-facing module which is visible to the user; it is the application frontend that serves to let the user create, edit, and browse for floor plans via a graphical user interface. Unlike a purely command-line based application, the graphical user interface is a lot easier to use for

most end users; the drawback of this is that it cannot be automated and is not particularly accessibility-aware. However, as discussed in the prior sections, it is a limitation that has been considered assuming that most users do not fall under these categories.

The user interface consists of a menu, a design section and the store/load section. As stated before, these are just the front ends for the other parts of the module (which will be discussed soon below); as such, they do little to no computational work themselves – apart from creating the floor plan – as they delegate the tasks to other modules instead.

The menu allows the user to pick the choice that they want (create new, browse existing from disk or from the chain, or exit); the design part allows the user to add parts to the layout, either by clicking or dragging the various parts. These are then later scanned by the serialization and deserialization module; hence, the two are dependent on each other (although it is possible to reduce coupling by introducing another abstraction layer, this approach was chosen for efficiency.)

Each screen is a "panel". A panel is nothing but a user interface which can be docked to a different window, to allow for a fluid workflow. By using a panel layout scheme, users can customize layouts to their preferences, and hence optimize their workflow.

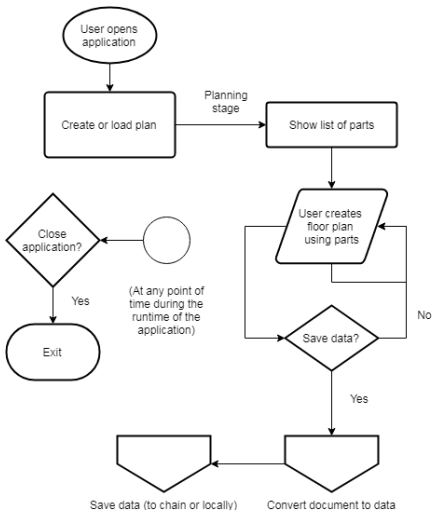

Figure 2: *User Interface Module*

Serialization & deserialization: The serialization and deserialization module are responsible for converting the floor plan into a series of bytes / raw data, and for converting a sequence of raw data into a floor plan.

Serialization is the process of converting a set of information into raw data; this is performed when a floor plan is sent to it – i.e. after the user decides to save the floor plan they made via the user interface – and the module scans all the items in the floor plan, along with the layout information. It then writes data in a fixed-format sequence, with the following format (except where mentioned otherwise, all types are 4-byte unsigned integers):

- ITEM_COUNT
- LAYOUT_WIDTH
- LAYOUT_HEIGHT
- For every item in layout:
  - X

  - Y
  - WIDTH
  - HEIGHT
  - ITEM_TYPE
- (blank delimiter)
- For every item in layout:
  - NAME (UTF-8)

Finally, the entire byte-array is compressed via the Lempel-Ziv-Markov-Chain (LZMA) algorithm, to reduce the number of bytes stored on the chain (and hence the overall transaction cost.)

Deserialization is the process of converting raw data into meaningful information (in this case, a floor plan); when deserializing raw data into a floor plan, it is decompressed and read the opposite way: The first 4 bytes read indicates the number of items present; the next 8 indicate the size of the total layout; and then a sequence of 4,4,4,4,4 bytes are read for each (number of items) until the total number of items is reached, and finally the names of each item are set while reading a UTF string from the data.

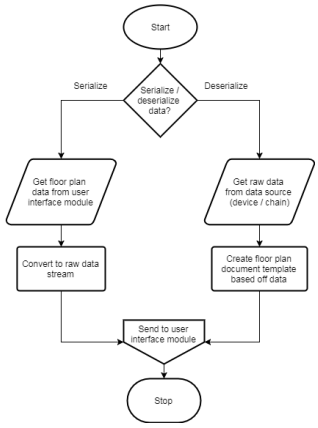

Figure 3: Serialization & Deserialization Module

External interface module: The external interface module is the module responsible for receiving and sending data from and to the smart contract. In effect, it is the realization of the web3 library. The external interface module can receive data from the application – more specifically, the serialization and deserialization module – and send it to the smart contract. A basic sequence of steps is outlined in the flowchart above, which is shown below in more detail; at its core, the external interface is a wrapper for the web3 module along with convenience functions for interfacing with the rest of the application as well.

Sending data from the application to the chain:

1. The application sends a message to the serialization and deserialization module to convert the floor plan into raw data.
2. The data is received by the external interface module
3. It calls the predefined web3 functions to load the smart contract
4. The smart contract stores the data on the blockchain.

Receiving data from the chain to the application:

1. The application sends a message to the external interface module to load the data from the chain
2. The external interface module calls the appropriate web3 functions
3. The smart contract returns the data to the interface.
4. The interface sends the raw data to the serialization and deserialization module to convert to a floor plan.

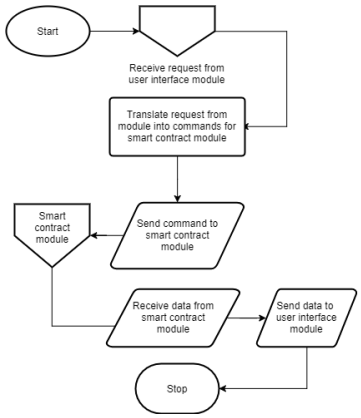

Figure 4: External Interface Module

Smart Contract: The smart contract module is the module that runs on the Ethereum virtual machine; this is described in the system architecture and deployment diagrams as shown earlier. The flowchart above describes the overall sequence of actions which occur when an external event occurs which triggers the working of the smart contract module.

Note: it is important to note that the external event that occurs is from the view of the smart contract module. Since the smart contract module runs on the Ethereum VM, it is technically external to the rest of the system; in other words, an external event which occurs from the perspective of the smart contract module is an internal event which occurs in the application. Similarly, any internal event from the smart contract module is an event external with respect to the application.

The smart contract module receives a request from the external interface module to execute some function: either receiving or sending data from or to the chain; if receiving data from the chain, then the user is authenticated if they are attempting to access private data; otherwise the data is directly received if it is public. When storing data, the data is directly stored; the user must choose in the application – whether it is public or private data, depending on the capabilities supported in the application.

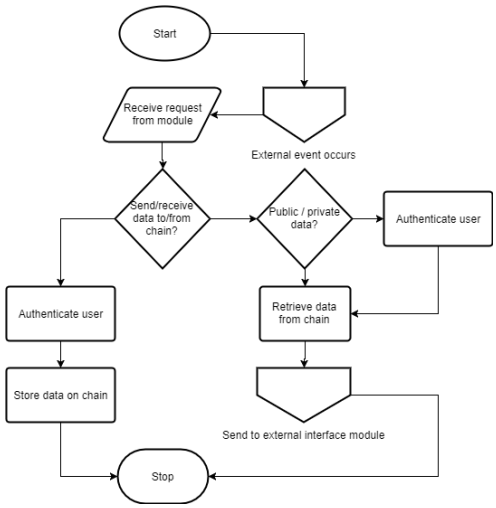

Figure 5: Smart Contract Module

5. Implementation Details

In this section, the hardware and the software used to implement the system will be listed along with a brief reason for their usage.

5.1 Hardware used

The hardware which is required by the application is that which most of the population can be safely assumed to have: a mouse, a monitor and, optionally a keyboard. While a keyboard is recommended, it is optional; unlike CLI (command line interface) applications which require only a keyboard, the application needs only a mouse to interact, as it was primarily a GUI (graphical user interface) based application. A keyboard can be used to create more precise and detailed plans; however, the opposite case holds, too: without a mouse, the application cannot be used, as the keyboard is used only for alteration of existing items in the plan, and so the full feature set of the application is not available without the use of a mouse.

As for the underlying system hardware required to run the application, any computer based on the x86/x86-64 architecture can be assumed to meet the requirement for running the software; the memory requirements are rather low, although older devices with less than 256 MB of free RAM may encounter troubles when running the application.

Software Used

Since the main application is made using the Adobe AIR framework, it guarantees multiplatforming capability; however, it is unlike Java in that it is a bit more restricted as far as operating systems are concerned – as of this writing, Adobe AIR currently runs only on Windows and MacOS operating systems (support for Linux had been dropped in the past due to various issues). As a result, those users running Linux may not be able to use the application without switching to another OS, either in a VM or otherwise.

To store and retrieve data to and from the Ethereum network, the application must interface with the smart contract running on the network, or on enabled nodes; to communicate with the smart contract running on the network, the Web3 framework is used, which offers an API to do so easily. This runs in a web browser which is integrated into the application, offering seamless back-and-forth communication.

Note that while Adobe AIR is not supported on Linux, since the features regarding the smart contract make use of the JavaScript-based Web3 framework, it is possible to make an alternative client which translates calls from and to Web3.

6. Conclusion & Future Work

In this paper a method has been discussed which allows users to create, modify and store floor plans on the Ethereum blockchain has been designed using the appropriate tools. By shifting some of the data storage onto the blockchain, it is expected that it will inherit the distributed redundant storage and non-repudiable properties of the blockchain.

While the current system implementation has fulfilled the basic expectations for such a system, which is to allow users to create and store floor plans on the blockchain, there are still quite a lot of features that can be added or improved upon, which if implemented, then the user quality-of-life would be drastically improved; such features which aim to bring it up to par with other commercial offerings include more fine-grained controls, support for dimensioning, and other, similar features; also, full multiplatforming capability is not available due to the limitations of Adobe AIR – that is, Linux support was removed after version 2.6. A possible solution is discussed below, along with others.

Fine-grained controls such as auto-resizing, snapping-to-grid, non-overlapping and overlapping constraints, along with limited boundaries and greater furniture types would make it much easier to use for both professionals and beginners alike; however, there is a trade-off between floor plan complexity and the resulting data size. For example, on average a given basic floor plan is about 140 bytes at maximum; by adding extra features, it is likely that it will increase to at least twice that. 140 bytes does not seem like much, but by storing

data on the blockchain, there is an extra cost incurred, with the cost increasing along with the size. Hence, it is in everyone's best interests to minimize file size, and yet achieve a large feature set (which seems like a conflicting set of requirements, unless proper measures for compression are taken.) Other methods are suggested which can help improve compression efficiency of not only the smart-contract bytecode, but the resulting data stored too (as it is dependent on the smart contract) which can increase efficiency by up to 75% [7]

Adobe AIR is not supported on Linux, which remains a deficiency in the application; however, the highly modular nature of the application (interfacing with web3 to store data via an external interface) means that creating an alternate interface which runs on Linux (or any other target platform) is easier than the alternative, i.e. recreating it from scratch.

Encryption of plans is also another possible issue: each plan is by default unencrypted, and storage on the blockchain means that anyone with the application installed can browse and view the data (and hence the resulting plan) if they so want to. This is both a good and a bad thing, that is, having plans be open means that collaboration is aided, as any user or set of users can be working on the same plan. However, it also means that any user who wants their plan to be private, but reap the benefits at the same time, has no solution at hand. For this, encryption can be used, but it goes against the spirit of the project. However, for those who want it as a feature, it is possible to get the equivalent functionality by installing an encryption module which intercepts data from the frontend (after it is serialized and before it is sent to the web3 module) and then encrypts it, before finally sending it to the web3 interface for deployment on the chain; the reverse process could be carried out before deserializing the plan data into a floor plan.

References

[1] Abdellatif, T., & Brousmiche, K. L. (2018). Formal Verification of Smart Contracts Based on Users and Blockchain Behaviors Models. IFIP International Conference on New Technologies, Mobility and Security (NTMS), (pp. 1-5). Paris, France.

[2] Bracamonte, V., & Okada, H. (2017). The issue of user trust in decentralized applications running on blockchain platforms. IEEE International Symposium on Technology and Society (ISTAS), (pp. 2017, pp. 1-4.). Sydney, Australia.

[3] Esposito, C., Santis, A. D., Tortora, G., Chang, H., & Choo, K. K. (2018). Blockchain: A Panacea for Healthcare Cloud-Based Data Security and Privacy? IEEE Cloud Computing, 31-37.

[4] Magrahi, H., Omrane, N., Senot, O., & Jaziri, R. (2018). NFB: A Protocol for Notarizing Files over the Blockchain. IFIP International Conference on New Technologies, Mobility and Security (NTMS), Paris, France, (pp. 1-4). Paris, France.

[5] Marchesi, M. (2018). Why blockchain is important for software developers, and why software engineering is important for blockchain software (Keynote). International Workshop on Blockchain Oriented Software Engineering (IWBOSE), (pp. 1-1). Campobasso, Italy.

[6] Nayak, A., & Dutta, K. (2017). Blockchain: The perfect data protection tool. International Conference on Intelligent Computing and Control (I2C2), (pp. 1-3). Coimbatore, India.

[7] Pontiveros, B. B., Norvill, R., & State, R. (2018). Recycling Smart Contracts: Compression of the Ethereum Blockchain. IFIP International Conference on New Technologies, Mobility and Security (NTMS), Paris, France, 2018, (pp. 1-5). Paris, France.

