San Jose State University

SJSU ScholarWorks

Master's Projects Master's Theses and Graduate Research

Spring 5-14-2020

Land Registry on Blockchain

Mugdha Patil
San Jose State University

Follow this and additional works at: https://scholarworks.sjsu.edu/etd_projects

Part of the Other Computer Sciences Commons

Recommended Citation
Patil, Mugdha, "Land Registry on Blockchain" (2020). *Master's Projects*. 912.
DOI: https://doi.org/10.31979/etd.2cc7-a5nd
https://scholarworks.sjsu.edu/etd_projects/912

This Master's Project is brought to you for free and open access by the Master's Theses and Graduate Research at SJSU ScholarWorks. It has been accepted for inclusion in Master's Projects by an authorized administrator of SJSU ScholarWorks. For more information, please contact scholarworks@sjsu.edu.

Land Registry on Blockchain

A Project

Presented to

The Faculty of the Department of Computer Science

San José State University

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

by

Mugdha Patil

May 2020

© 2020

Mugdha Patil

ALL RIGHTS RESERVED

The Designated Project Committee Approves the Project Titled

Land Registry on Blockchain

by

Mugdha Patil

APPROVED FOR THE DEPARTMENT OF COMPUTER SCIENCE

SAN JOSÉ STATE UNIVERSITY

May 2020

Dr. Thomas Austin	Department of Computer Science
Dr. Robert Chun	Department of Computer Science
Dr. Ching-seh (Mike) Wu	Department of Computer Science

ABSTRACT

Land Registry on Blockchain

by Mugdha Patil

The commercial real estate market is a significant part of the global economy, currently dominated by a small set of firms and organizations that lack transparency. The process of property transfers also requires third party intervention which is expensive. In many countries, the process of title transfers is problematic. We are still in the initial steps of digitization, due to the improvement required in terms of use of technology to represent assets in digital forms. Increase in liquidity of investments and purchases, proper management, documentation as well as ease of access is the future of real estate. Blockchain technologies have the potential to drive these changes as explained in Chapter 1.

Blockchain technologies like Ethereum[1] include asset tokenization, and act as immutable and decentralized transaction ledgers. Tokens on the ledger can represent the real estate assets. Particulary, the non-fungible tokens on Ethereum can serve as a representation of transfer of resources. Ethereum grants trusted and distributed smart contracts for token operations. My project is a system for real estate cadastral record keeping and title transfers that uses the ERC-721 specification[2] related to the non-fungible tokens.

Testing of the implementation is done using Government records from District of Columbia[3]. Ethereum is a natural choice for this project due to its evolution, an active development community, and many supporting languages and tools that facilitate use of smart contracts. This paper gives the background, implementation details and significance of such a system. Some cost-related and defensive mechanisms offered by the system are discussed later in the report.

ACKNOWLEDGEMENTS

I am very grateful to Dr. Thomas Austin for his continuous support and guidance throughout the project. He allowed the freedom of choice, discussion and implementation, gave deeper insights at the same time and had faith in me. I would like to thank the committee members, Dr. Ching-seh (Mike) Wu and Dr. Robert Chun for being a part of the committee, for giving their valuable time and providing feedback. Lastly, I am very thankful to my family and friends whose love and support encouraged me in this wonderful journey of the Masters' program.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

Real Estate and Blockchain

1.1 Introduction

Offering to buy and sell properties in more granular pieces is one of the benefits of blockchain, and security tokens in particular. For instance, individual investment units can be identified and represented using a security token, i.e ERC-20 standard[5], ERC-721 standard[2] or its variant. For trade to happen in regulatory secondary markets, tokens will provide a method for transaction processing, as well as be used as ownership and property identifiers.

Blockchain acts like a shared and distributed ledger. The transactions are bundled in blocks post processing, while the blocks are cryptographically linked that form a chain. No single centralized authority decides the fate of transactions. Rather, a network of public or private nodes will operate based on a consensus mechanism. This enables decentralization of authority when each node validates and maintains copies of ledger verified by their peers. Thus, a chronological "chain" of data is formed which cannot be changed by any of the parties. At the same time, its integrity is verifiable using cryptography. Replication of data allows protecting transaction records as well as multiple verifiable sources of truth at different nodes. If the system is able to track transactions in real time without a centralised sole source of authority, this will prove to be of a high value to the real estate industry[6].

In the further sections, the paper describes history of real estate tracking systems as well as systems that have been devised with blockchain using non-fungible tokens in recent years. This project undertakes a use case of real estate by simulating Land Registries and Cadastrals. Ethereum, non-fungible tokens, smart contracts are explained in the next chapter. ERC-721 token implementation is discussed in detail. The following chapters describe the system implementation and workflow.

Later, feasiblity analysis and experiements done are elaborated and compared with the traditional systems. The report concludes with the discussion of future work.

1.2 History and Background

A new standard for Ethereum called ERC-721[2] was proposed by a Vancouver-based startup called Axiom Zen. They launched a blockchain-based collection game called Cryptokitties[7] in November 2017. Unique traits are given by the original smart contract, and can be used by the users to trade or breed digital cats. This smart contract issues a unique cat every 15 minutes, which is then bought by a buyer in the form of a token. This is analogous to buying a bitcoin having special properties. If two such cats breed, it creates a unique cat with DNA inherited from its parents, and blockchain stores all tokens linked to the cats. Blockchain stores the ownership proof while Axiom Zen maintains and stores the cryptokitty in a private centralized server.

Cryptokitties has gained much success and raised considerable awareness about non-fungible tokens. This is the first successful consumer use-case of a blockchain without requiring a commodity token. "Left Gallery" is a German digital art gallery[8] which offers purchase and management of multimedia art using a blockchain. Asset classes like "crypto-composables" are also emerging, like the KittyHats[9] allowing users to link hats to Cryptokitty(the hat is an ERC-20 token linked to cryptokitty). DADA[10] is another social network where users communicate with crypto-art. The art market is provided with a decentralised ledger for tracking art pieces, jewellery, watches, etc, and also partnering with auction houses. Super Rare[11] and Open Sea[12] bypass the traditional auctions and historical art galleries by providing a decentralised distribution system. This digital art scene is giving a new direction to the use of non-fungible tokens by providing a new business model.

All this was so far focused on digital art, but this could very well be extended to physical assets due to the innovation provided by ERC-721 in the form of a distributed ledger system for non-fungible tokens. A blockchain is said to be valuable if users are dealing with assets which need to be exchanged or the complicated transactions require third-party intervention. For example, selling a house requires physical or legal facilitation of transactions by third parties. In case of art pieces, it requires experts to authenticate the art which is crucial for system's viability. But third parties pose a major downside for majority users by keeping them out of the market. Additional necessities and higher transaction costs also reduce market liquidity. Top players like auction houses charge heavy commission to provide liquidity to the market. A blockchain startup called Harbor[13] aims at tokenizing real world assets through blockchain, which are otherwise expensive to track and hard to trade. Examples include company shares, real estate or art. Thus blockchain can address this issue with a unified shared data layer, while ensuring compliance and authority using smart contracts[14].

1.3 Use Case: Land Registries and Cadastrals

It has been historically a challenge to access the land registry information stored offline by the regional authorities in most countries. As discussed before, with the blockchain potential to shorten a tedious process of maintaining and recording titles, offering transparency in business, and elimination of a central party, the land registers should be moved on-chain. This way, the land registers will be preserved on the network. Visibility, availability, verifiability and sustainability in digital form is possible for present and future documents[15].

Titling and land ownership claims can hinder many development projects in many countries. After the titles are verified and their integrity proof stored in the blockchain registry, there is a greater chance of ownership assurance by relying on the immutable transaction record. Many countries are moving their land registries on blockchain networks. Good examples of this are places like Vermont city in United States and Ukraine[15]. Use of non-fungible tokens for representing real estate properties now seems like a logical and almost inevitable step.

1.4 Why Ethereum

Bitcoin[16] was the first digital currency on the blockchain. Therefore, blockchain came to be mostly identified with Bitcoin. In 2011, Ethereum creator Vitalik Buterin made his attempts to build applications on Bitcoin blockchain but with no success. He realized that each application will require a separate original blockchain which is highly complicated. This gave rise to Ethereum by changing the core blockchain elements with the specifications in 2014[17] and launching the Blockchain in 2015[18].

Ethereum is a decentralized platform having all the advantages of blockchain technology and one that also allows creation of decentralized applications (DApps)[6]. Ethereum has its own currency called "Ether" that can be traded like Bitcoin. Both have different goals and capabilities even if they look similar. Bitcoin, as an example, was created with an aim of replacing conventional fiat currency with digital currency. On the contrast, Ether does more than acting as decentralized currency by enabling developers to create DApps using smart contracts.

Some of the reasons[15] why Ethereum is the choice for developers, builders and entrepreneurs is-

- The change to proof-of-stake will only increase the already high degree of decentralization. Proof-of-stake is the power of mining or validating transactions based on the number of coins one holds.
- An active developer community of about 200,000 developers working on live products that support many important tools / frameworks.

- Continuous development by the open-source community, recent example is the upcoming Constantinople upgrade[19].
- The Ethereum core developer team aims to make transactions more efficient by implementing a proof-of-stake based consensus system called the "Casper" protocol. This proves to be better than the currently used proof-of-work protocol[20].
- As per Coindesk data, by January 2018, the world market capitalization of Ethereum was over 130 billion USD[21]. That is when Ethereum became the 2nd largest blockchain network.

1.5 Blockchain Challenges: Issues with Adoption and use

1.5.1 Conceptual challenges

Blockchain comprises of numerous new terms like proof-of-stake, decentralized storage, wallets, immutable ledger, mining and so on. It takes time to understand these terms for any developer or user due to the conceptual leap from the current technologies[22].

Similarly, people faced challenges in early days of the internet while understanding concepts like card payment or URL. This implies that the current blockchain challenges are not impossible to overcome. They just need some time and experimentation to improve the processes for doing things differently.

1.5.2 Trust issues

Converting fiat cash into cryptocurrency and trading in this form is a trust challenge. People get stuck when converting their hard earned money into something like tokens which sounds foreign to them. Also, fraud is still possible around edge cases like introducing land or any other property into the system, that has originally been acquired with fraud. Private keys could also be stolen that gives unauthorized parties a chance to transact properties. Improved safer interfaces will help alleviate these challenges.

1.5.3 Regulatory challenges

Token offerers need to abide by security regulations of the state. Custodial arrangements might be limited to certain entities like registered dealers, brokers, banks, etc. Such hurdles may put up barriers in the adoption at various levels.

1.5.4 Complexities

Changing the way of handling transactions is difficult due to the complexities involved in the regulations and dealings of property data.

1.5.5 Volatility and Market Risks

Tokenization of fractional property will increase the number of active real estate investors. Trading tokens in minutes will also increase potential for large swings, reduce transaction friction and increase speed of transactions dramatically.

1.6 Existing implementation

As a real estate marketplace worldwide, Propy[23] serves as a decentralized property registry. They aim to solve the problems for global real estate industry by allowing buyers, sellers, title agents to utilize a suite of smart contracts for real estate online purchases. A digital ownership transfer is ensured when trading property on Propy registry.

When the idea of this system was introduced, Propy was in a development phase pertaining to the transaction platform for brokers and agents. The word "cadastre" means a public record that documents the value, extent and ownership of land for taxation purpose. Example of this is the Doomsday book in early England (1086 A.D). Blockchain, for this use-case, will provide high value to the Government sectors through unique identification of property as well as security.

CHAPTER 2

Ethereum and Smart Contracts

Improved transaction processing and securing real estate property using blockchain will ease the customer's efforts in exchanging properties or their shares, maintaining, and perfecting property transactions as well as processing revenue. Due to this, new investors and more number of parties can engage in ownership, liquidity, and take risks effectively[15].

As a global and decentralized platform, Ethereum has emerged as a successful technology for Blockchain. A lack of centralized authority does not hinder agreement implementation of mutually untrusted parties. Constraints by different parties are implemented by Ethereum by taking advantage of the blockchain features. The different parties can now sign an agreement/contract with complete transparency. Therefore, the software code running on Ethereum is known as "Smart Contract".

This section will provide a technical background of the system followed by the implementation details[24].

2.1 Ethereum

An embedded quasi-Turing complete computing machine facilitates computer program execution on the Ethereum nodes in the blockchain network[16]. Quasi-Turing completeness is a system of data manipulation rules and such a system can mathematically have the capability of performing any possible computation. A valid and correct block sequence with all the information can be obtained from nodes. These nodes manage transactions by interacting among themselves. Ethereum accounts have blockchain addresses which are the identities managed by a public-private key pair. Ether cryptocurrency, as mentioned in Section 1.4, acts as an incentive to the miners.

For a distributed peer to peer network, mining is participation for consensus. Mining activity requires time and processing power due to being computationally intensive. The currency issuers do not have huge processing capabilities, so the role of a miner is crucial. He invests resources for submitting transaction blocks and verification, and in return the currency issuer offers rewards in the form of digital coins.

The accounts can use Ether for storage and exchange. Along with crypto values, the account can also contain code. They are called "smart contracts" and are associated with a blockchain address determined at the time of contract creation[25].

Transactions ensure a block sequence representing change of states in the ledger. These transactions are made by exchanging messages between accounts, which contain a payload (essentially binary data), and some amount of Ether.

2.2 Account Types in Ethereum

Both account types hosted in Ethereum are managed by the Ethereum Virtual Machine (EVM) in a similar way[1].

- Externally Owned Accounts(EOAs) - These accounts have a unique address. They hold balance in the form of Ether and are managed using public-private key pairs. Humans using the Ethereum platform correspond to these accounts. Code is not bound to these accounts. Ethereum's ecosystem has API available for writing EOA's initial transactions[26]. Programs are started by issuing transaction in two ways -
  - Deploying a new smart contract instance.
  - A message sent to a smart contract function at the provided address.
- Contract Accounts - These are managed by the stored code i.e the smart contracts.

2.3 Smart Contracts

Solidity is a programming language developed from C++, JavaScript and Python. It is a mature and high-level language that allows running decentralized application programs over blockchain. Smart contracts are written in Solidity[27]. Its code compilation produces a bytecode that runs on Ethereum Virtual Machine (EVM). Due to the blockchain specific features, Ethereum smart contracts are highly successful and thus have bought with them a new set of applications. Hence, it becomes necessary to fully exploit the capabilities of Solidity as a programming language.

2.4 Basic Execution Workflow

The smart contracts on the network nodes receive the broadcast requests to execute their stored code. This activates the transactions using the pairs of public-private keys. The input data payload received via messages is used to execute the stored code by the smart contract accounts. Transactions consume "gas units" to avoid infinite computations and this is paid in Ether. Message and delegate calls are the other types of interactions. They have payload input for invocation, a source address of sender EOA, a target address and gas as a fee to reward the miner nodes. They also return data and amount in Ether consumed as a form of payment. Executing smart contract code in the calling program context is called as "delegate calls"[28].

2.4.1 Ethereum Virtual Machine

A virtual machine is a level of abstraction between the executing machine and executing code, that improves software portability and ensures separation of different applications as well as from the host. For Ethereum, the Ethereum Virtual Machine (EVM) executes the contract bytecode. A virtual stack is encapsulated in every Ethereum node. EVM bytecode is a result of compilation of the contracts written in high-level languages. Every Ethereum node running on EVM instance agrees on the execution of same set of instructions. Also, the EVM bytecode is completely in isolation from the host processes, the network and the filesystem. EVM is a system that can perform any logical computational function steps. EVM implementations are available in Java, C++, JavaScript, Python, Ruby, etc. For the Ethereum protocol, the EVM plays a key role as the consensus engine of the Ethereum system, as well as guaranteeing full deterministic programs. Any instruction running on EVM has an associated cost given in gas units by a system that tracks the execution cost[6].

2.4.2 Ethereum Request for Comments

The world has become very active in blockchain usage with heterogeneous typologies and an active developer community. One of the interesting findings is that the developer community has surpassed the first era of smart contracts, which was comprised of concepts like "parties" or "agreements". Code reuse, existing smart contract enhancements, efforts to improve efficiency and security of Solidity is strongly evident. The developer community has created several other typologies from games to utility tokens. One of the interesting typologies is the development of ERC-721, which is a token standard in Ethereum for developing applications that can transact non-fungible assets using blockchain.

Before diving into ERC-721 protocol, the concept of Ethereum Request for Comments (ERCs) need explanation. Smart contracts refer to these as technical documents. A set of rules is required by the Ethereum ecosystem for the implementation of tokens. Usually written by developers, these documents contain descriptions of the contract and protocol specifications. ERC is revised, commented upon and only then accepted by the community through an Ethereum Improvement Proposal (EIP)[2] before becoming a standard. In this way, ERC can be called as a specific type of EIP, may be of different types like token, registration name, URI schemes, packets, libraries, etc.

and regarded as the application-level standards or conventions. There are four states of an EIP[2]:

- draft - opened for consideration, such as the ERC-721 non-fungible token standard.
- accepted - planned for immediate adoption.
- final - This is an implemented EIP, like the ERC-20 token standard.
- deferred - This means dismissed for now and may be considered in the future.

CHAPTER 3

Standards and Technologies

This chapter discusses the various tools, technologies and standards used to build the project. The tools are chosen considering their pros and cons, project scalability concerns and sometimes adhering to governmental requirements. Some of the tools are from new startups and under active development. This underlines the need to introduce them in this chapter.

3.1 Solidity language

A type of software programs called smart contracts written in Solidity language are considered in this analysis. Solidity is specially designed for executing code on the Ethereum Virtual Machine(EVM). Like object-oriented languages, Solidity contracts contain functions and state variables. It refers to currently executing contract instance using keyword "this". Messages can be sent to other contract or the current contract by the contract function, also with some gas fee and some amount of virtual money. An implicit variable called "msg" holds details of the current message call. The information includes caller address (i.e msg.sender), amount of money sent(i.e msg.value), etc. A simple bank implemented using Solidity smart contract is given in Listing 3.1.

Listing 3.1: Solidity smart contract implementing a simple bank[29]

```
contract Bank {
    mapping (address => uint) amounts;
    function withdraw (uint n) {
        require (amounts [msg.sender] >= n);
        amounts [msg. sender] -= n;
        msg.sender.transfer (n);
    }

    function deposit() payable {
        amounts [msg.sender] += msg.value ;
    }
}
```

3.1.1 Non-Fungible token standards and implementation

Before we understand non-fungible tokens, it is important to understand the difference between fungible and non-fungible tokens. Both are fundamentally different. Fungible tokens are interchangeable like fiat currency. They are analogous to a dollar bill, which does not make any difference even if exchanged with a holder for the same amount. This makes fungible tokens uniform in nature. These are also divisible into smaller amounts. The ERC-20 is a well-known standard for issuance of fungible tokens.

In contrast, non-fungible tokens are non-interchangeable. They are similar to how a birth certificate of one person cannot be exchanged with another person. This is due to the uniqueness each non-fungible token has, unlike the fungible tokens. Non-fungible tokens cannot be divided and token is the only elementary unit. ERC-721 is a new standard on Ethereum blockchain to issue non-fungible tokens.

ERC-721 is still in the draft state but people have already started using it. A new smart contract standard called ERC-721 was proposed on September 20, 2017, to create and issue non-fungible tokens(NFTs). Games like CryptoCup, CryptoFighters and CryptoKitties[30] use ERC-721 as virtual collectibles. Their value is obtained from their scarcity, and NFT application to the real world is an active research domain. ERC-721 tokens must implement the proposed ERC-165 interface reviewed in Section 3.1.3. This standard allows the detection of interfaces implemented by a contract. The token implements ERC-165 to make the implementing code to interact and comply with itself.

3.1.2 Interfaces for non-fungible token implementation

Trade, management and ownership of unique tokens with other developers' contracts (DApp, website, other software, etc) requires implementation of a minimum interface by the smart contract. This is defined by ERC-721. The interfaces ERC-721 and ERC-165 are mainly implemented by all the ERC-721 contracts. With these guidelines, other programmers can write code to interact with our code without knowing the entire codebase. The following section minimally describes some of the important method signatures of ERC-721 interface[31]. Other supporting interfaces are briefly discussed. They will be explained in detail in the subsequent sections.

3.1.3 ERC-165

The ERC-165 standard is just a way of checking if your contract's fingerprints match the fingerprint of any given interface. The XOR of all function selectors in the interface forms the interface Id. This interface is to verify that our contract has used the ERC-721 interface. The two ways to get a function selector are-

Listing 3.2: function-selector-methods[32]

```
   this.balanceOf.selector //or
   bytes4(keccak256("balanceOf(address)"))
```

The function selector cares only about the parameter types, the function name and nothing else, so there is no need to implement all the functions.

Listing 3.3: ERC-165 interface[32]

```
interface ERC165 {
    /// @notice Query if a contract implements an interface
           function supportsInterface(bytes4 interfaceID) external
              ↪ view returns (bool);
}
```

3.1.4 ER721TokenReceiver

Listing 3.4: ERC721TokenReceiver interface[32]

```
interface ERC721TokenReceiver {
           /// @notice Handle the receipt of an NFT
           function onERC721Received(address _operator, address _from,
              ↪  uint256 _tokenId, bytes
           _data) external returns(bytes4);
        }
```

As the name implies, it's an interface for contracts that can receive ERC-

721 tokens. A valid implementation of ERC721TokenReceiver interface will return the function selector bytes4(keccak256("onERC721Received(address, address, uint256,bytes)"))[32].

contract TokenERC721 is ERC721, CheckERC165... (where contract CheckERC165 is ERC165 ...)

3.1.5 ERC-721 interface

Listing 3.5: ERC-721 interface[32]

```
interface ERC721 /* is ERC165 */ {
    event Transfer(address indexed _from, address indexed _to, uint256
        ↪  indexed _tokenId);
    event Approval(address indexed _owner, address indexed _approved,
        ↪ uint256 indexed _tokenId);
    event ApprovalForAll(address indexed _owner, address indexed
        ↪ _operator, bool _approved);

    /// @notice Count all NFTs assigned to an owner
    function balanceOf(address _owner) external view returns (uint256)
        ↪ ;

    /// @notice Find the owner of an NFT
    function ownerOf(uint256 _tokenId) external view returns (address)
        ↪ ;

    /// @notice Transfers the ownership of an NFT from one address to
        ↪ another address
    function safeTransferFrom(address _from, address _to, uint256
        ↪ _tokenId, bytes data) external payable;

    /// @notice Transfers the ownership of an NFT from one address to
        ↪ another address
    function safeTransferFrom(address _from, address _to, uint256
        ↪ _tokenId) external payable;

    /// @notice Transfer ownership of an NFT -- THE CALLER IS
        ↪ RESPONSIBLE TO CONFIRM THAT `_to` IS CAPABLE OF RECEIVING
        ↪ NFTS OR ELSE THEY MAY BE PERMANENTLY LOST
    function transferFrom(address _from, address _to, uint256 _tokenId
        ↪ ) external payable;

    /// @notice Set or reaffirm the approved address for an NFT
    function approve(address _approved, uint256 _tokenId) external
        ↪ payable;

    /// @notice Enable or disable approval for a third party ("
        ↪ operator") to manage
    /// all of `msg.sender`'s assets.
    function setApprovalForAll(address _operator, bool _approved)
        ↪ external;
        /// @notice Get the approved address for a single NFT
        function getApproved(uint256 _tokenId) external view returns (
            ↪ address);

        /// @notice Query if an address is an authorized operator for
            ↪ another address
        function isApprovedForAll(address _owner, address _operator)
            ↪ external view returns (bool);
    }
```

ERC-20[5] has almost become the standard for comparing other token proposals, so will be taken reference of to explain ERC-721 from Open Zeppelin implementations of ERC standards. Open Zeppelin offers tools for writing and operating decentralized applications. Following are the categories of token standards - Ownership, creation, transfer, allowance for transfer and burning tokens. The parent in this context is ERC721BasicToken.sol and is inherited by ERC721Token.sol referred to as child[33]. They are explained below.

3.1.5.1 Ownership of token

- ERC-20 has mapping of tokens to respective owners' addresses using -

Listing 3.6: ERC-20 mapping for value[32]

```
mapping (address => uint256) balances
```

- The available balance can simply be verified against this mapping during transfer of tokens. But for ERC-721, each token is unique i.e token of same class can hold different values. Since each token must be tracked, ERC-20 mapping does not work. Instead, each NFT is identified through an uint256 Id. This is an array of tokenIds throughout the contract, and each token has its own index in the context of all available ERC-721 tokens.

Listing 3.7: ERC-721 mapping for value[32]

```
uint256[] internal allTokens  - (1)
```

- Each individual address also keeps track of the owned tokens indexes from allTokens array since one address can hold more than one tokens. In ERC-20 we check only against balance.

Listing 3.8: ownedTokens mapping[32]

```
mapping (address => uint256[]) internal ownedTokens      - (2)
```

- Now to avoid iterating over the tokenId indexes array for finding the owner of a token, each tokenId is also mapped to its respective owner like an inverted index.

Listing 3.9: tokenOwner mapping[32]

```
mapping (uint256 => address) internal tokenOwner     - (3)
```

- This is also required while deleting tokens because in Solidity, deletion of an element in an array does not completely delete the element but replaces it with 0. So, during deletion of tokens, rather than deleting the element, we rearrange the array.
- To check how many tokens are owned by a specific address, we maintain-

Listing 3.10: ownedTokensCount mapping[32]

```
mapping (address => uint256) internal ownedTokensCount   - (4)
```

- This is updated when tokens are owned, transferred, purchased or burnt. The motivation behind this data structure is verification.
- Following data structures are maintained since they are used while burning tokens.

Listing 3.11: ownedTokensIndex mapping[32]

```
// Mapping from token ID to index of the ownedTokens from (2)
mapping (uint256 => uint256) internal ownedTokensIndex; - (5)
```

- Following data structure is updated during minting.

Listing 3.12: allTokensIndex[32]

```
// Mapping from token ID to position in the allTokens array from (1).
mapping (uint256 => uint256) internal allTokensIndex; - (6)
```

3.1.5.2 Creation of token

- Token creation is very easy in ERC-20 in which we can add to the "totalSupply" variable. For ERC-721, updation of individual data structures is necessary[33]. All global data structures or variables are updated in these two functions-
- addTokenTo(receiverAddress, tokenId) - Used to add tokens to owner ledger. It has full implementation which calls basic implementation. "tokenId" is the unique Id chosen mostly by the owner or the one allowed to call this function. Basic implementation checks that tokenId is not already owned and data structures "tokenOwner" and "ownedTokensCount" are updated. Full implementation

updates data structures from Listing 3.11 by adding token to the end of the ownedTokens array. The owner address is stored at an index in the array.

Listing 3.13: addTokenTo(..) parent implementation[32]

```
function addTokenTo(address to, uint256 tokenId) internal {
    require(tokenOwner[tokenId] == address(0));
    tokenOwner[tokenId] = to;
    ownedTokensCount[to] = ownedTokensCount[to].add(1);
}
```

- mint(receiverAddress, tokenId) - To update allTokens array, mint() is used. Child implementation performs a check by calling parent implementation to ensure we are not minting to address 0. Parent then calls child implementation of addTokenTo(to, tokenId). After parent implementation is complete, we add tokenId to allTokenIndex mapping in Listing 3.12 and allTokens array in Listing 3.7. Thus, for the extra check while maintaining all information in an ERC-721 contract, we make use of mint() function even if we can call addTokenTo(to, tokenId) directly.

Listing 3.14: mint(...) parent implementation[32]

```
function mint(address to, uint256 tokenId) internal {
    require(to != address(0));
    addTokenTo(to, tokenId);
    emit Transfer(address(0), to, tokenId);
}
```

3.1.5.3 A note about metadata(optional)

Listing 3.15: tokenURIs[32]

```
// Optional mapping for token URIs
mapping(uint256 => string) internal tokenURIs;    - (7)
```

This is set using setTokenURI(..) where string holds metadata URI. Using a structure with variables is far cheaper than creating smart contract per asset. Prior to assigning data, it also checks if the tokenId exists.

3.1.5.4 Allowance

In games, escrows or auctions, we may need to delegate the job of token transfer to other address or contract. There is a method available for the owner to approve another address to allow spending tokens on his behalf. The approved address allowance to spend the tokens is checked by another transfer function[33]. In ERC-20, the approved spender calls transferFrom(..) function (discussed later) to check if msg.sender is allowed to transfer tokens and the owner has sufficient funds. But in case of ERC-721, we approve and transfer tokenIds. Following structure maintains tokenIds with their approved address / approved Id.

Listing 3.16: tokenApprovals mapping[32]

```
tokenApprovals(tokenId => approvedAddress/approved ID)  - (8)
```

- approve (receiverAddress, tokenId) - Approve msg.sender for one tokenId. It first checks for the ownership in tokenApprovals or if the msg.sender is approved to spend all tokens for a particular owner address. This is done using isApprovedForAll().

Listing 3.17: tokenApprovals mapping[32]

```
function approve(address _to, uint256 _tokenId) public {
   address owner = ownerOf(_tokenId);
   require(_to != owner);
   require(msg.sender == owner || isApprovedForAll(owner, msg.sender))
       ↪ ;

  if (getApproved(_tokenId) != address(0) || _to != address(0)) {
      tokenApprovals[_tokenId] = _to;
      emit Approval(owner, _to, _tokenId);
    }
}
```

Listing 3.18: operatorApprovals mapping[32]

```
operatorApprovals(ownerAddress => approvedSpenderAddress => Bool) -
    ↪ (9)
```

- isApprovedForAll (ownerAddress, operatorAddress) - This is assisted by a global variable as above.
- Spender can assign additional spend capabilities if approved for all the tokens. Also, methods like getApproved(tokenId) ensures address(0) is not granted approval. In this way, the desired address mapping to tokenApprovals is complete.

Listing 3.19: isApprovedForAll(..) implementation[32]

```
function isApprovedForAll(address owner, address operator) external
   ↪ view
   returns (bool) {
      return _operatorApprovals[owner][operator];
     }
```

Listing 3.20: setApprovedForAll(..) implementation[32]

```
function setApprovalForAll(address operator, bool approved) external {
   _operatorApprovals[msg.sender][operator] = approved;
   emit ApprovalForAll(msg.sender, operator, approved);
}
```

3.1.5.5 Transfer

Transfer is done through two different functions-

1) transferFrom(senderAddress, receiverAddress, tokenId)

- This method is discouraged since no additional data is provided for verification. A modifier canTransfer() checks approval of msg.sender to own or transfer tokens.
- After validating addresses of sender and receiver, clearApproval() will remove the approval given to senderAddress to transfer tokens any further[34].
- Next, removeTokenFrom(from, tokenId) function in parent is invoked from the child's full implementation(similar to addTokenTo(to, tokenId)).

- Entry of the token is discarded from ownedTokensCount mapping and tokenOwner mapping.
- The index of the token being transferred is now occupied by the last token of the ownedTokens array. The array is then shortened by 1.
- The tokenId is associated with its new owner by using the addTokenTo(to, tokenId) function.

Listing 3.21: transferFrom(..) implementation[32]

```
function transferFrom(address _from, address _to, uint256 _tokenId)
    ↪ public canTransfer(_tokenId){
   require(_from != address(0));
   require(_to != address(0));

   clearApproval(_from, _tokenId);
   removeTokenFrom(_from, _tokenId);
   addTokenTo(_to, _tokenId);
   emit Transfer(_from, _to, _tokenId);
}
```

2) safeTransferFrom(senderAddress, receiverAddress, tokenId)

This function comes in two forms - safeTransferFrom parameters with and without a bytes parameter.

Listing 3.22: safeTransferFrom(..) implementation[32]

```
function safeTransferFrom(address _from,address _to,uint256 _tokenId,
   ↪ bytes _data)
  transferFrom(_from, _to, _tokenId);
  require(checkAndCallSafeTransfer(_from, _to, _tokenId, _data));
}
```

Sending token to a contract without appropriate functions will result in losing the tokens permanently. To prevent such erroneous transfers, the safeTransferFrom(..) function checks for prior standard interface implementation before token transfer. ERC721_RECEIVED, also called the magic value, is the function signature of the onERC721Received() function. It ensures that token is sent to a valid method. The checkAndCallSafeTransfer(..)[34] satisfies an additional requirement for checking whether "to" address is a contract. After verification, onERC721Received(..) will return the same function signature as expected from a standard interface. Otherwise, transferFrom(..) function is rolled back because the "to" address does not seem to implement the expected interface.

Listing 3.23: checkAndCallSafeTransfer(..) implementation[32]

```
function checkAndCallSafeTransfer(address _from,address _to,uint256
   ↪ _tokenId,bytes data)internal returns (bool){
    if (!_to.isContract()) {
        return true;
}
    bytes4 retval = ERC721Receiver(_to).onERC721Received(_from,
       ↪ _tokenId, _data);
    return (retval == ERC721_RECEIVED);
}}
```

3.1.5.6 Burning Tokens

For transferring/removing ownership/burning tokens we use ownedTokensIndex in Listing 3.11 and Listing 3.12.

- burn(ownerAddress, tokenId) - Use value variable to specify number of tokens for burning. Update the balances of msg.senderwhen tokens are burnt and reduce totalSupply of tokens.
- A specific token at an index has to be removed for ERC-721 tokens. It is similar to functions mint(..), addTokenTo(..) in which parent implementation of ERC-721 is called using "super" keyword. Following are the steps to burn a token -
  - Use method clearApproval(..). After this, the use removeTokenFrom(..) to remove the token from thw ownership of the owner. Alert this change to the frontend using an event.
  - Discard the mapping to the token index which will remove the metadata

corresponding to the token.

  - Rearrange allTokens array similar to removing tokens from ownership, by replacing the burnt token index with the last token and reducing the array size by 1[33].

Listing 3.24: burn(..) implementation[32]

```
function _burn (address owner, uint256 tokenId) internal {
    require(ownerOf(tokenId) == owner, "ERC721: burn of token that is
       ↪ not own");
    _clearApproval(tokenId);
    _ownedTokensCount[owner].decrement();
    _tokenOwner[tokenId] = address(0);
    emit Transfer(owner, address(0), tokenId);
}
```

3.2 Tools and technologies

This section introduces all the tools that are used for building the system.

3.2.1 Node.js

Node.js[35] is an open-source environment that helps in building scalable and fast server-side web applications. It is a cross-platform environment that sits on top of the V8 JavaScript engine by Chrome. Real-time connections between web applications can facilitate initiating communication by either the client or the server. Node.js is efficient and lightweight due to its event-driven and non-blocking I/O model. Web applications that use Node.js can have real-time connections that allow both the client and server to initiate communication. This is in contrast with the traditional client-server paradigm. Thus, free exchange of data is possible by both the parties. Open web stack technologies like HTML, CSS and JavaScript running on server standard port 80 also forms the basis on Node.js[35].

3.2.2 npm: Node Package Manager

Package management is by default offered with Node.js as a built-in support. This is called the "npm" tool[36]. Using npm, downloading a complete list of packaged modules is possible. The module ecosystem is freely available.

3.2.3 Express framework

Web and mobile applications can be supported by a set of features provided by Express.js[37]. It is a web application framework that has varied HTTP utility methods, middleware, and its flexibility is evident from popular frameworks based on Express.js.

3.2.4 Backend using TypeScript

TypeScript[38] is mainly developed by Microsoft and is a superset of the JavaScript language. It aims to catch mistakes early through a type system and this makes JavaScript development more efficient. The code is now faster to implement, easy to understand, refactor and merge with less bugs or boilerplate tests. This helps developers in having a correct workflow.

3.2.5 Testnets

Ethereum Test Network aka Testnets help the developers in testing their code on a dummy chain. For this chain, free Ether or tokens are obtained with no real world value. They are useful for blockchain system development and testing before launching the system on the real blockchain. The reason for choosing Rinkeby over other test networks like Ropsten is that Rinkeby is supported by the Bitski wallet at its present stage of development. Rinkeby[39], started by the Geth team is a proof-of-authority blockchain where Ether is only requested and not earned through mining.

3.2.6 Bitski

App wallet is meant for blockchain applications for execution of programmatic transactions. Bitski[40] is a digital wallet used for many Ethereum DApps. Its backend SDK helps the application to submit transactions via a standard web3 API from the wallet belonging to the backend. Wallets like Metamask can be used only if the API calls are made from the frontend. Bitski collaborates with the 0xcert framework and so is used in this project along with 0xcert API at the backend.

3.2.7 0xcert framework and API

Decentralized applications can now be built with 0xcert(pronounced [zeer-oh-eks-surt]), which is a tool-set consisting of open-source JavaScript libraries, some closed source APIs to provide powerful decentralized applications (DApps)[4]. It reduces the development time, risks and costs by providing a set of functions for deployment, management, verification and certification of assets. They can also store cryptographic representation of data objects and selectively disclosed information can be verified by the third parties. Solidity code is abstracted using 0xcert framework. The framework is available as an npm module and handles the underlying complexity of writing Solidity smart contracts. It helps in asset management like any other set of APIs. 0xcert provides a decentralized API for enterprise and its use is encouraged since the interaction is super simple compared to the 0xcert Framework. But 0xcert API is a paid feature. As an alternative, 0xcert framework can also be used since it is open-source, but then the developer has to perform build everything from scratch.

3.2.8 Frontend

The frontend is not of major focus here. The idea is to demonstrate blockchain as a strong candidate in maintaining property-related information. Figure 1 has visualized the use of a frontend only for explanatory purposes.

3.2.9 MySQL

MySQL is a relational database management system. It is based on Structured Query Language(SQL). This is used for relational database that stores administrative details as well as asset-related information. It is easy to setup, scalable, fast, and reliable. Additionally, it is most widely used throughout the world and actively supported by an international developer team.

CHAPTER 4
Implementation

This chapter discusses the technical aspects of the project like the system design, the workflow and the backend API. The system design is done by me as part of the project. Some of the components like Bitski wallet, 0xcert dashboard, Rinkeby test network, MySQL database, http-server are the supporting tools that help achieve the objective of the system use-case, while the backend components, MySQL database schema, static-server file structures are part of the system implementation. The frontend is a future possibility which can be customized for different stakeholders. To simulate the function calls for the purpose of this project, a Postman API client helps in making calls to the backend. Supporting backend libraries are Express.js, 0xcert libraries, bitski-node, mysql, etc. Figure 2 to Figure 9 provide snippets of supporting tools that are integrated with the system. They reflect results of the interaction of the backend with the Ethereum blockchain.

4.1 System Design

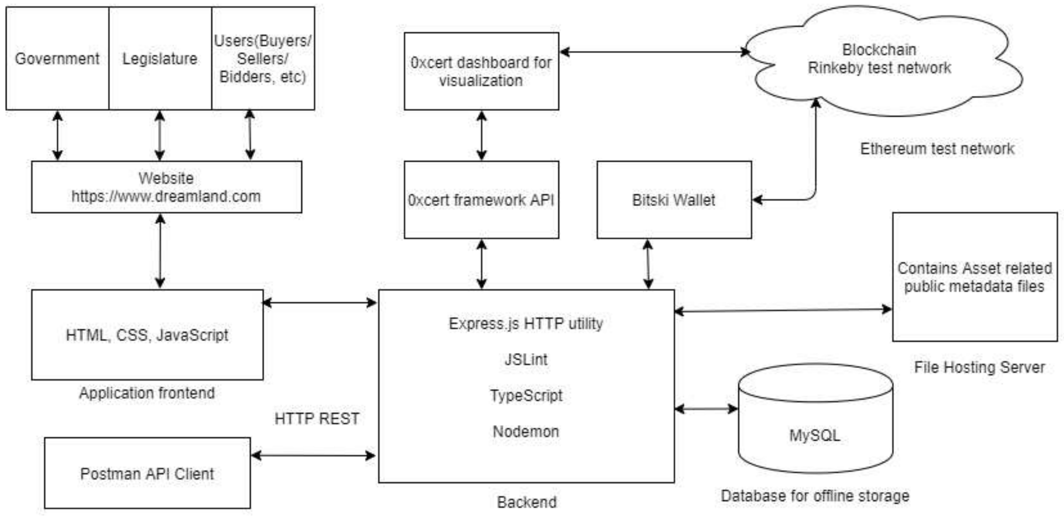


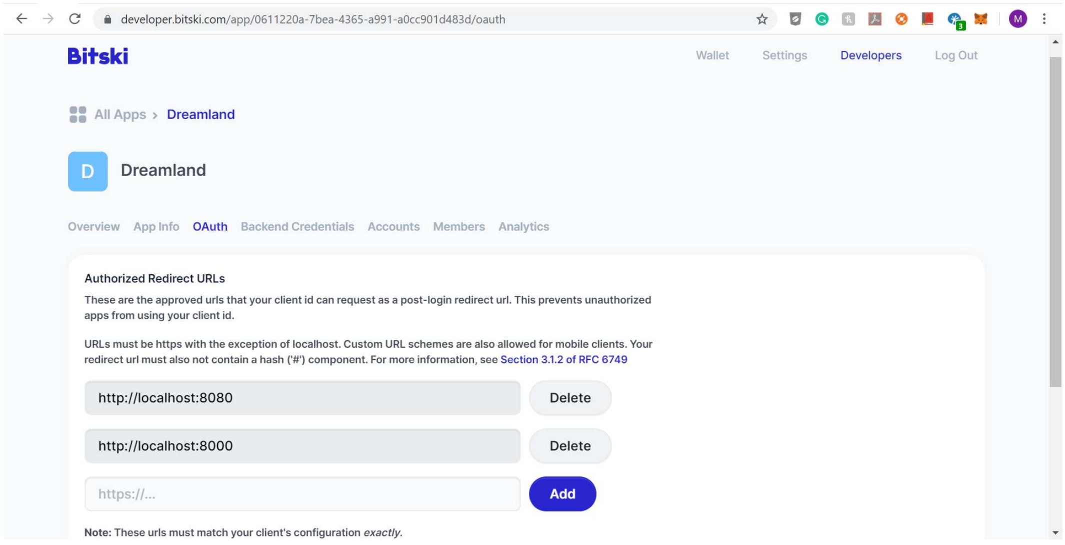
Figure 1: System Architecture for DApp

Figure 1 above describes different components of the system developed by me.

The users can be the Government, legislative agencies, dealers, brokers, buyers or sellers of the property. They will access the system through a hosted website. The request will come to the deployed application frontend hosted on a web server, that relays requests to the backend server. Note that, currently, backend API is accessed using Postman API client in the absence of a frontend.

The backend will communicate to the Ethereum blockchain via 0xcert framework. This framework has plugins like Rinkeby test network for development, and main Rinkeby network for real world transactions on actual Ethereum blockchain. All smart contract related operations will be taken care by the 0xcert framework.

The Bitski wallet is integrated with the backend by specifying the connect URL on Bitski platform as shown in Figure 2. It also stores client configuration and Ethereum addresses. The configuration is discussed in Section 4.3.


developer.bitski.com/app/0611220a-7bea-4365-a991-a0cc901d483d/oauth

Bitski

Wallet Settings Developers Log Out

All Apps > Dreamland

D Dreamland

Overview App Info OAuth Backend Credentials Accounts Members Analytics

Authorized Redirect URLs

These are the approved urls that your client id can request as a post-login redirect url. This prevents unauthorized apps from using your client id.

URLs must be https with the exception of localhost. Custom URL schemes are also allowed for mobile clients. Your redirect url must also not contain a hash ('#') component. For more information, see Section 3.1.2 of RFC 6749

http://localhost:8080 Delete

http://localhost:8000 Delete

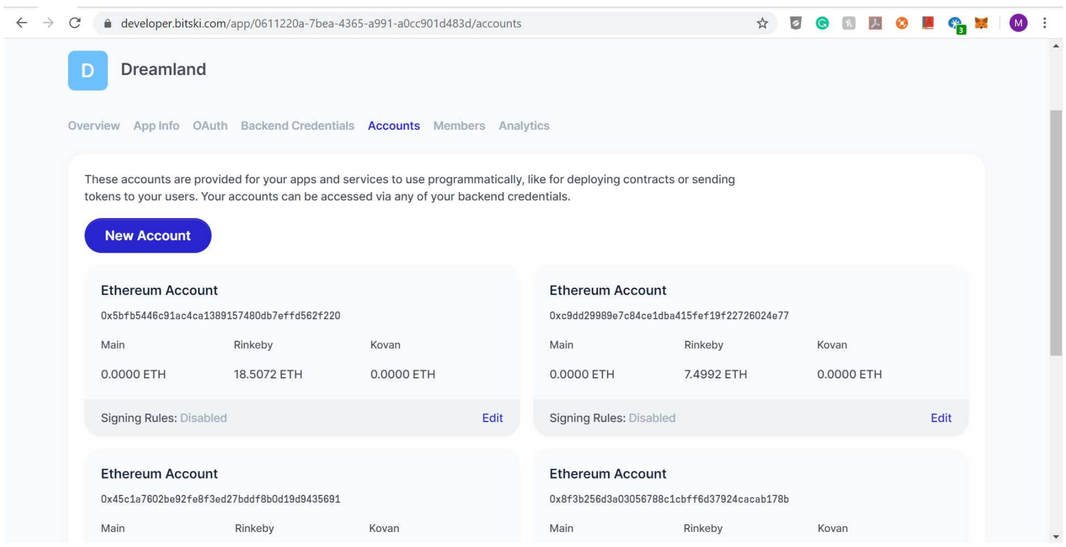
https://... Add

Note: These urls must match your client's configuration *exactly*.

Figure 2: Bitski Connect URL with the backend

Bitski can also hold more than one apps, each app having a collection of addresses as shown in Figure 3. Different types of property transactions can be segregated as separate applications in the wallet.

0xcert also provides a dashboard to keep track of the assets, ledgers, transfers


developer.bitski.com/app/0611220a-7bea-4365-a991-a0cc901d483d/accounts

Dreamland

These accounts are provided for your apps and services to use programmatically, like for deploying contracts or sending tokens to your users. Your accounts can be accessed via any of your backend credentials.

Ethereum Account
0x5bfb5446c91ac4ca1389157480db7effd562f220

Main	Rinkeby	Kovan
0.0000 ETH	18.5072 ETH	0.0000 ETH

Signing Rules: Disabled

Ethereum Account
0xc9dd29989e7c84ce1dba415fef19f22726024e77

Main	Rinkeby	Kovan
0.0000 ETH	7.4992 ETH	0.0000 ETH

Signing Rules: Disabled

Ethereum Account
0x45c1a7602be92fe8f3ed27bddf8b0d19d9435691

Main Rinkeby Kovan

Ethereum Account
0x8f3b256d3a03056788c1cbff6d37924cacab178b

Main Rinkeby Kovan

Figure 3: Bitski holding apps

and the activity as shown in Figure 4. There are differences between this dashboard and the application frontend. The frontend will serve the land cadastral use-cases and other activities like user management or communication with the backend. In contrast, the 0xcert dashboard will be used to display ledger and asset details.

The database can be used for storing offline records like user management details for DApp backend, asset evidence and metadata file links for easy reference, complex user operation details and the real world property attributes provided by the user. Currently, it is used to store all the information related to the 0xcert assets, ledgers and clients. The backend plays a key role in binding the storage, visualization, frontend and the blockchain together. The Bitski wallet and 0xcert dashboard are connected to the Rinkeby network. Any activity on the network will be reflected on these platforms. Following steps provide the workflow of the implementation and can also be regarded as setup guidelines for user onboarding -

- Create an application in Bitski wallet.
- Generate client configuration for the application. Specify backend URL to connect.

- Add associated Ethereum addresses to the account.
- Login on Land Registry application.
- API call - Initialize client by providing Bitski application configuration. This returns a client signature.
- Using the client signature, login to access the 0xcert dashboard.
- Refill credits on the 0xcert dashboard for the Ethereum address.
- API - Get existing ledger information from blockchain. If ledger is not available, use developed API to deploy(create) a new ledger on the client.
- API - Get information about existing asset. If asset is not available, use developed API to create a new asset.
- API - Transfer an asset - This is possible directly with a single signer if the transfer is initiated by the owner, otherwise he will require privileges. Multiple stakeholders can be involved in one transfer by providing their approval using signatures.
- Grant privileges to the receiver by creating value approval and asset approval for further transfer of assets.
- Store user management details, operational details, asset metadata file paths and real world asset information in offline storage.
- Host asset-related files on a server.
- Rinkeby test network - View the underlying smart contract related details on Rinkeby transaction URL.

4.2 Explaining the workflow

The 0xcert ecosystem employs a variety of modules to connect with various platforms since it is platform agnostic. Connection to the Ethereum blockchain and performing queries is done via the Bitski provider. First, we create an account under

Bitski developer options, then add application and addresses to it. We also generate configuration for on the application for access from the backend.

The 0xcert API as described in Section 4.3 will help in connecting to the Bitski application using this configuration. The initialization of the client can be simulated when a user logs in to the frontend. The client signature obtained during initialization is then used to login to the 0xcert dashboard portal as shown in figure 4. The dashboard This provides information about user credits, his API activity, ledgers and asset related information including transfers, in which the logged in address is involved. Credits are fungible tokens used by 0xcert and are discussed in detail in Section 5.3.

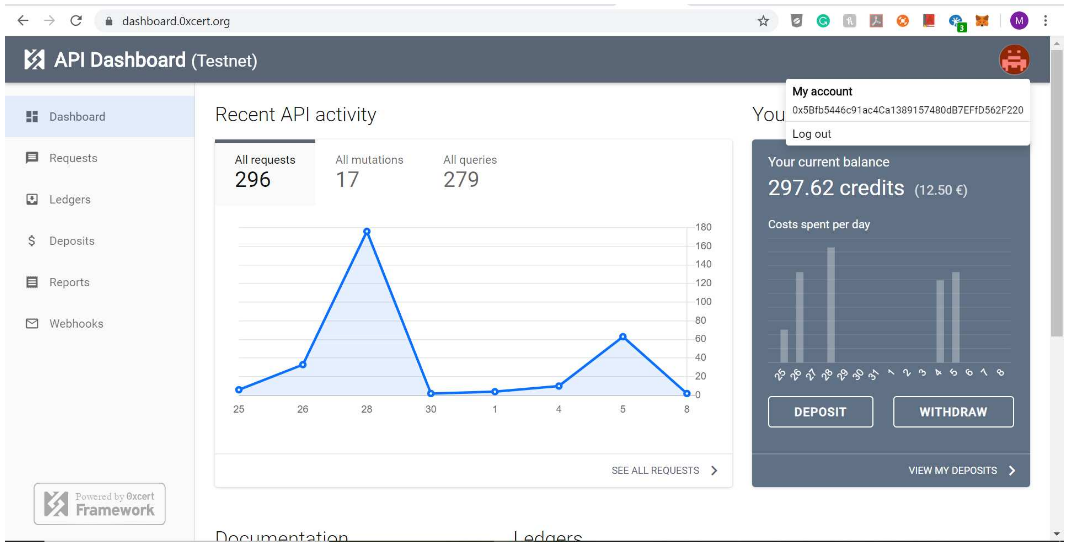
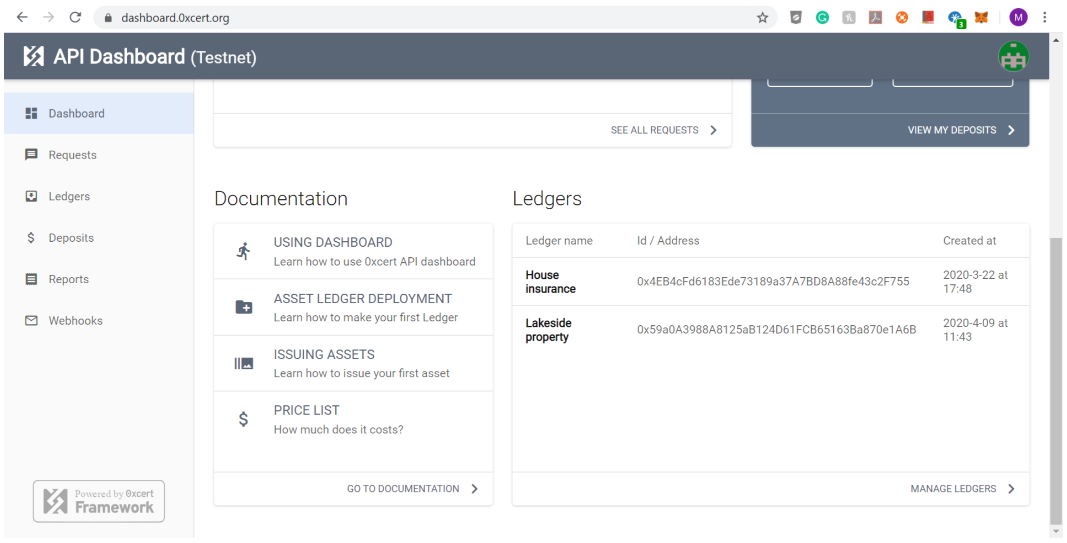


Figure 4: 0xcert Credits and Activity

The next thing is creation of an asset ledger. Any operations on the asset ledger are reflected on the blockchain, since it represents an 0xcert smart contract on the blockchain. An asset created on the ledger directly translates to an asset on the underlying smart contract. The ledgers can be seen on 0xcert dashboard as in Figure 5.

Thus, the asset ledger acts as a container to define the structure of the assets inside it. An asset is a digital representation of any item on the ledger and follows a specific schema structure of the ledger. It can be viewed as a folder, one that holds together specific assets of an issuer and its related owners[4].



dashboard.0xcert.org

API Dashboard (Testnet)

- Dashboard
- Requests
- Ledgers
- Deposits
- Reports
- Webhooks

SEE ALL REQUESTS

VIEW MY DEPOSITS

Documentation

- USING DASHBOARD — Learn how to use 0xcert API dashboard
- ASSET LEDGER DEPLOYMENT — Learn how to make your first Ledger
- ISSUING ASSETS — Learn how to issue your first asset
- PRICE LIST — How much does it costs?

GO TO DOCUMENTATION

Ledgers

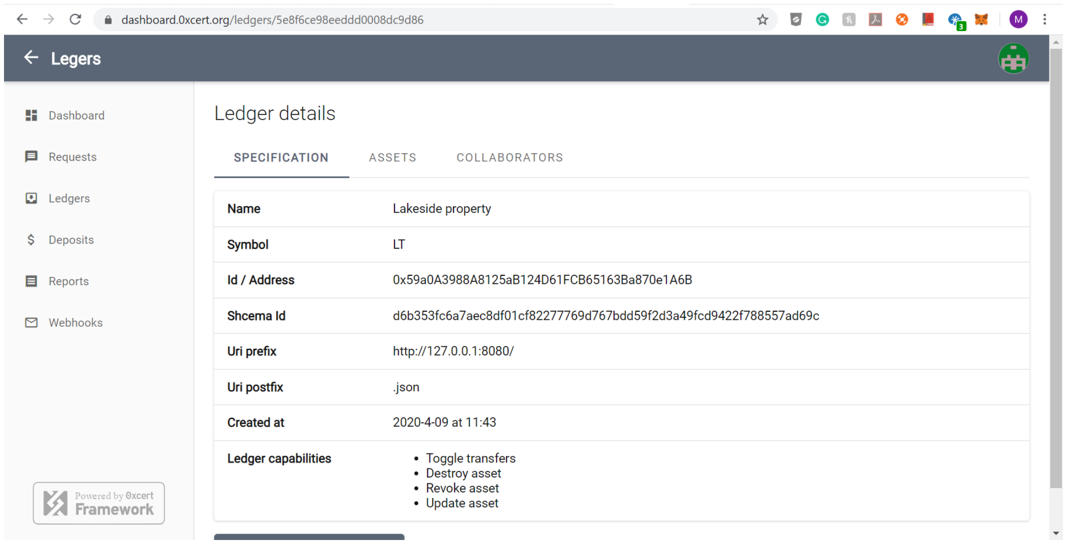
Ledger name	Id / Address	Created at
House insurance	0x4EB4cFd6183Ede73189a37A7BD8A88fe43c2F755	2020-3-22 at 17:48
Lakeside property	0x59a0A3988A8125aB124D61FCB65163Ba870e1A6B	2020-4-09 at 11:43

MANAGE LEDGERS

Powered by 0xcert Framework

Figure 5: Listed Ledgers

We can also get the details of that ledger using its reference Id as in the Figure 6. We can then create assets in that ledger. For example, a separate ledger can help in segregating the property records based on zip-code. Asset under a ledger can be seen in Figure 7.



dashboard.0xcert.org/ledgers/5e8f6ce98eeddd0008dc9d86

Legers

- Dashboard
- Requests
- Ledgers
- Deposits
- Reports
- Webhooks

Ledger details

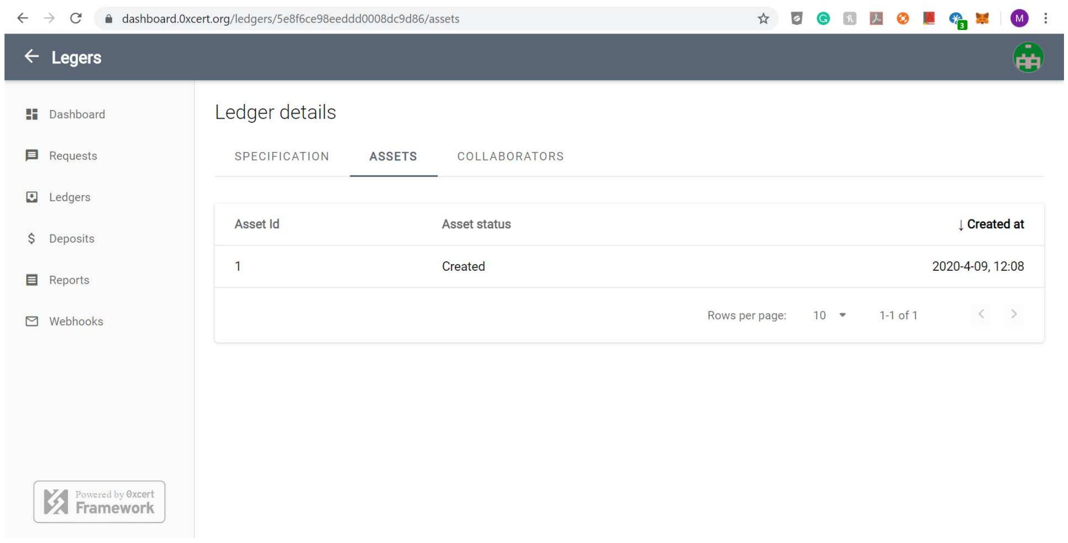
SPECIFICATION | ASSETS | COLLABORATORS

Name	Lakeside property
Symbol	LT
Id / Address	0x59a0A3988A8125aB124D61FCB65163Ba870e1A6B
Shcema Id	d6b353fc6a7aec8df01cf82277769d767bdd59f2d3a49fcd9422f788557ad69c
Uri prefix	http://127.0.0.1:8080/
Uri postfix	.json
Created at	2020-4-09 at 11:43
Ledger capabilities	• Toggle transfers • Destroy asset • Revoke asset • Update asset

Powered by 0xcert Framework

Figure 6: Ledger Specifications

An asset can be created by anyone who is authorized for abilities like asset creation inside the ledger. This authorization can be done by the Government authorities. A property owner can create an asset inside an object called "order". Multiple actions like creation of asset by the sender and transfer of an asset to a receiver is possible in a single order. This can serve use-cases like document generation and sending it to the right owner in a single blockchain call. Details of the contained asset can be seen on dashboard as in Figure 8.



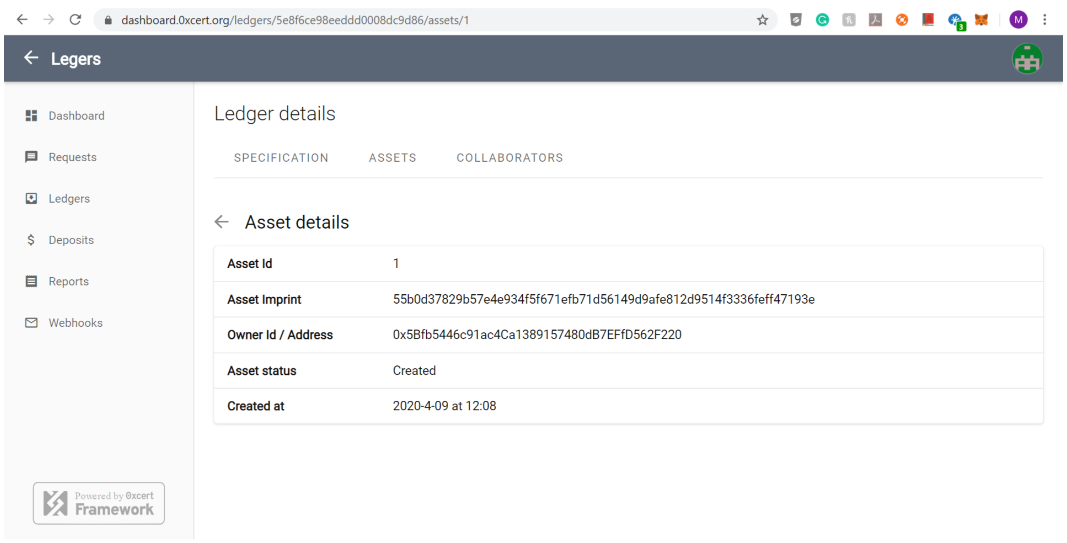
Asset Id	Asset status	↓ Created at
1	Created	2020-4-09, 12:08

Figure 7: Listed Assets

Third party authorities like lawyers can authorize the asset by providing their signatures. The real market value of the project lies in the fact that any type of transaction could be made depending upon the Government rules and use case requirements.

After asset creation, we can transfer an asset to someone with an address on Ethereum. The transfer-related transaction, or any other transaction information can be seen on the Rinkeby network as shown in the Figure 9.

The transfer of an asset can be reaffirmed by checking the new asset owner using the developed API. One important thing to note is the requirement of a relational


dashboard.0xcert.org/ledgers/5e8f6ce98eeddd0008dc9d86/assets/1

Legers

Dashboard
Requests
Ledgers
Deposits
Reports
Webhooks

Powered by 0xcert Framework

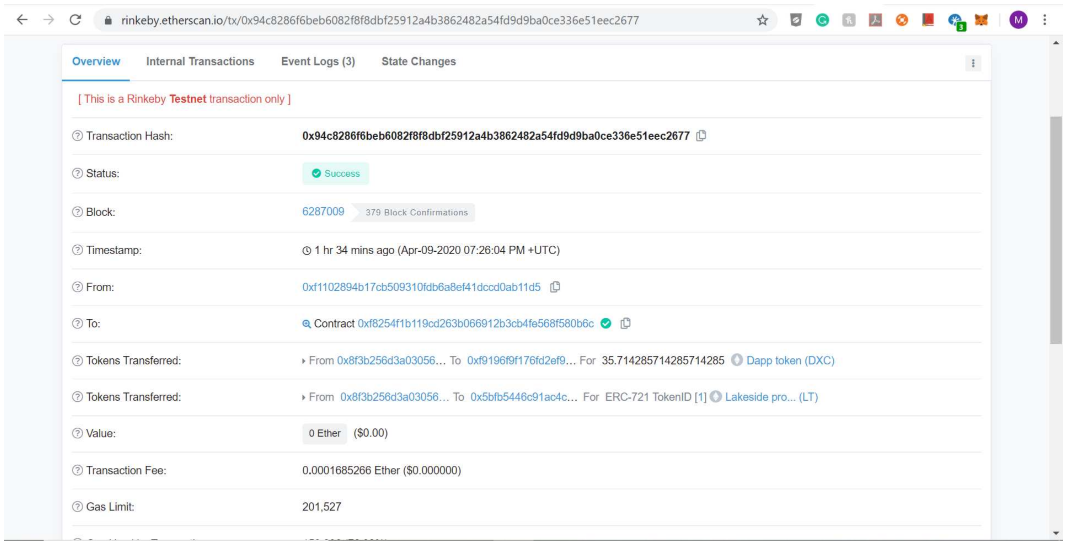
Ledger details

SPECIFICATION ASSETS COLLABORATORS

Asset details

Asset Id	1
Asset Imprint	55b0d37829b57e4e934f5f671efb71d56149d9afe812d9514f3336feff47193e
Owner Id / Address	0x5Bfb5446c91ac4Ca1389157480dB7EFfD562F220
Asset status	Created
Created at	2020-4-09 at 12:08

Figure 8: Asset Details


rinkeby.etherscan.io/tx/0x94c8286f6beb6082f8f8dbf25912a4b3862482a54fd9d9ba0ce336e51eec2677

Overview Internal Transactions Event Logs (3) State Changes

[This is a Rinkeby **Testnet** transaction only]

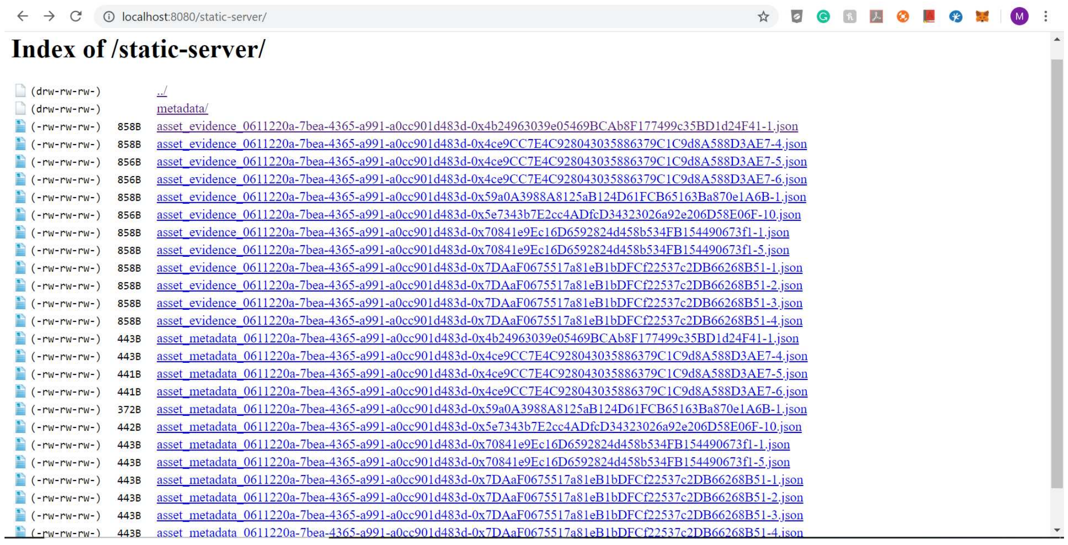
Transaction Hash:	0x94c8286f6beb6082f8f8dbf25912a4b3862482a54fd9d9ba0ce336e51eec2677
Status:	Success
Block:	6287009 379 Block Confirmations
Timestamp:	1 hr 34 mins ago (Apr-09-2020 07:26:04 PM +UTC)
From:	0xf1102894b17cb509310fdb6a8ef41dccd0ab11d5
To:	Contract 0xf8254f1b119cd263b066912b3cb4fe568f580b6c
Tokens Transferred:	From 0x8f3b256d3a03056… To 0xf9196f9f176fd2ef9… For 35.714285714285714285 Dapp token (DXC)
Tokens Transferred:	From 0x8f3b256d3a03056… To 0x5bfb5446c91ac4c… For ERC-721 TokenID [1] Lakeside pro... (LT)
Value:	0 Ether ($0.00)
Transaction Fee:	0.0001685266 Ether ($0.000000)
Gas Limit:	201,527

Figure 9: Asset Transfer details on Rinkeby network

database as well as a web server as shown in the figure 10 to host metadata and evidence files.

The asset metadata file will host publicly viewable information about assets as shown in the Figure 6.1 of the next chapter.

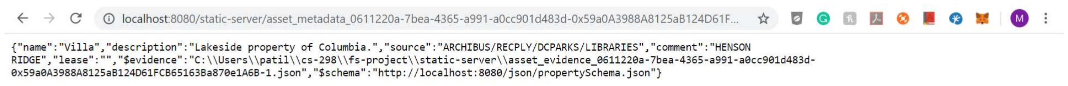
The asset evidence file as shown in Figure 6.2 of the next chapter, the asset-metadata file and the imprint from blockchain will help an individual verify the


```
localhost:8080/static-server/

Index of /static-server/

(drw-rw-rw-)          ../
(drw-rw-rw-)          metadata/
(-rw-rw-rw-)   858B   asset_evidence_0611220a-7bea-4365-a991-a0cc901d483d-0x4b24963039e05469BCAb8F177499c35BD1d24F41-1.json
(-rw-rw-rw-)   858B   asset_evidence_0611220a-7bea-4365-a991-a0cc901d483d-0x4ce9CC7E4C928043035886379C1C9d8A588D3AE7-4.json
(-rw-rw-rw-)   856B   asset_evidence_0611220a-7bea-4365-a991-a0cc901d483d-0x4ce9CC7E4C928043035886379C1C9d8A588D3AE7-5.json
(-rw-rw-rw-)   856B   asset_evidence_0611220a-7bea-4365-a991-a0cc901d483d-0x4ce9CC7E4C928043035886379C1C9d8A588D3AE7-6.json
(-rw-rw-rw-)   858B   asset_evidence_0611220a-7bea-4365-a991-a0cc901d483d-0x59a0A3988A8125aB124D61FCB65163Ba870e1A6B-1.json
(-rw-rw-rw-)   856B   asset_evidence_0611220a-7bea-4365-a991-a0cc901d483d-0x5e7343b7E2cc4ADfcD34323026a92e206D58E06F-10.json
(-rw-rw-rw-)   858B   asset_evidence_0611220a-7bea-4365-a991-a0cc901d483d-0x70841e9Ec16D6592824d458b534FB154490673f1-1.json
(-rw-rw-rw-)   858B   asset_evidence_0611220a-7bea-4365-a991-a0cc901d483d-0x70841e9Ec16D6592824d458b534FB154490673f1-5.json
(-rw-rw-rw-)   858B   asset_evidence_0611220a-7bea-4365-a991-a0cc901d483d-0x7DAaF0675517a81eB1bDFCf22537c2DB66268B51-1.json
(-rw-rw-rw-)   858B   asset_evidence_0611220a-7bea-4365-a991-a0cc901d483d-0x7DAaF0675517a81eB1bDFCf22537c2DB66268B51-2.json
(-rw-rw-rw-)   858B   asset_evidence_0611220a-7bea-4365-a991-a0cc901d483d-0x7DAaF0675517a81eB1bDFCf22537c2DB66268B51-3.json
(-rw-rw-rw-)   858B   asset_evidence_0611220a-7bea-4365-a991-a0cc901d483d-0x7DAaF0675517a81eB1bDFCf22537c2DB66268B51-4.json
(-rw-rw-rw-)   443B   asset_metadata_0611220a-7bea-4365-a991-a0cc901d483d-0x4b24963039e05469BCAb8F177499c35BD1d24F41-1.json
(-rw-rw-rw-)   443B   asset_metadata_0611220a-7bea-4365-a991-a0cc901d483d-0x4ce9CC7E4C928043035886379C1C9d8A588D3AE7-4.json
(-rw-rw-rw-)   441B   asset_metadata_0611220a-7bea-4365-a991-a0cc901d483d-0x4ce9CC7E4C928043035886379C1C9d8A588D3AE7-5.json
(-rw-rw-rw-)   441B   asset_metadata_0611220a-7bea-4365-a991-a0cc901d483d-0x4ce9CC7E4C928043035886379C1C9d8A588D3AE7-6.json
(-rw-rw-rw-)   372B   asset_metadata_0611220a-7bea-4365-a991-a0cc901d483d-0x59a0A3988A8125aB124D61FCB65163Ba870e1A6B-1.json
(-rw-rw-rw-)   442B   asset_metadata_0611220a-7bea-4365-a991-a0cc901d483d-0x5e7343b7E2cc4ADfcD34323026a92e206D58E06F-10.json
(-rw-rw-rw-)   443B   asset_metadata_0611220a-7bea-4365-a991-a0cc901d483d-0x70841e9Ec16D6592824d458b534FB154490673f1-1.json
(-rw-rw-rw-)   443B   asset_metadata_0611220a-7bea-4365-a991-a0cc901d483d-0x70841e9Ec16D6592824d458b534FB154490673f1-5.json
(-rw-rw-rw-)   443B   asset_metadata_0611220a-7bea-4365-a991-a0cc901d483d-0x7DAaF0675517a81eB1bDFCf22537c2DB66268B51-1.json
(-rw-rw-rw-)   443B   asset_metadata_0611220a-7bea-4365-a991-a0cc901d483d-0x7DAaF0675517a81eB1bDFCf22537c2DB66268B51-2.json
(-rw-rw-rw-)   443B   asset_metadata_0611220a-7bea-4365-a991-a0cc901d483d-0x7DAaF0675517a81eB1bDFCf22537c2DB66268B51-3.json
(-rw-rw-rw-)   443B   asset_metadata_0611220a-7bea-4365-a991-a0cc901d483d-0x7DAaF0675517a81eB1bDFCf22537c2DB66268B51-4.json
```

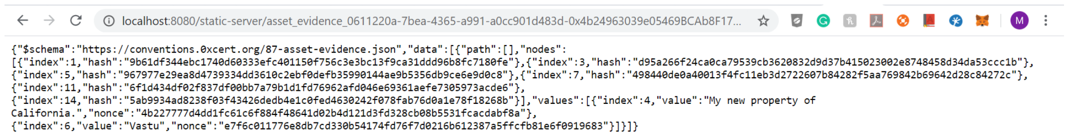
Figure 10: Static-server


```
localhost:8080/static-server/asset_metadata_0611220a-7bea-4365-a991-a0cc901d483d-0x59a0A3988A8125aB124D61F...

{"name":"Villa","description":"Lakeside property of Columbia.","source":"ARCHIBUS/RECPLY/DCPARKS/LIBRARIES","comment":"HENSON
RIDGE","lease":"","$evidence":"C:\\Users\\patil\\cs-298\\fs-project\\static-server\\asset_evidence_0611220a-7bea-4365-a991-a0cc901d483d-
0x59a0A3988A8125aB124D61FCB65163Ba870e1A6B-1.json","$schema":"http://localhost:8080/json/propertySchema.json"}
```

Figure 11: Hosted Asset Metadata file

authenticity of the records on the chain.


```
localhost:8080/static-server/asset_evidence_0611220a-7bea-4365-a991-a0cc901d483d-0x4b24963039e05469BCAb8F17...

{"$schema":"https://conventions.0xcert.org/87-asset-evidence.json","data":[{"path":[],"nodes":
[{"index":1,"hash":"9b61df344ebc1740d60333efc401150f756c3e3bc13f9ca31ddd96b8fc7180fe"},{"index":3,"hash":"d95a266f24ca0ca79539cb3620832d9d37b415023002e8748458d34da53ccc1b"},
{"index":5,"hash":"967977e29ea8d4739334dd3610c2ebf0defb35990144ae9b5356db9ce6e9d0c8"},{"index":7,"hash":"498440de0a40013f4fc11eb3d2722607b84282f5aa769842b69642d28c84272c"},
{"index":11,"hash":"6f1d434df02f837df00bb7a79b1d1fd76962afd046e69361aefe7305973acde6"},
{"index":14,"hash":"5ab9934ad8238f03f43426dedb4e1c0fed4630242f078fab76d0a1e78f18268b"}],"values":[{"index":4,"value":"My new property of
California.","nonce":"4b227777d4dd1fc61c6f884f48641d02b4d121d3fd328cb08b5531fcacdabf8a"},
{"index":6,"value":"Vastu","nonce":"e7f6c011776e8db7cd330b54174fd76f7d0216b612387a5ffcfb81e6f0919683"}]}]}
```

Figure 12: Hosted Asset Evidence file

These files are be generated during creation of the asset before their placement on blockchain. When any third party calls for verification, these files can be used to compare against the imprint stored on the blockchain.

4.3 Development of APIs

Following APIs have been developed for testing feasibility of the system. The implementation is focused on providing a core set of APIs for various asset operations. An asset is represented by a JSON object having any number of properties.

4.3.1 Initialize the client

An account is required on the Bitski developers portal with details like-

- Creation of application that generates a client Id.
- Authorized redirect URLs to connect with Bitski from backend.
- Name of the application.
- Backend credentials secret.

All these items will be used to create a client using Node.js backend code. For more details refer to Listing A.1.

4.3.2 Deploy the ledger

The asset ledger can be described with a name and a symbol. The URI prefix and URI postfix can be given to set the path to the static-server where asset metadata lives for that ledger. Metadata is description of the asset. URI prefix, with the asset Id and postfix will give the metadata location. Schema Ids, ledger owner address and privileges for handling assets inside are also mentioned. By default, the owner gets the abilities given to the ledger. For more details, refer to Listing A.2.

4.3.3 Get latest deployment related information

This API helps to know if the ledger is deployed successfully on the blockchain and this is confirmed if returned status equals to 7. For more details, refer to Listing A.3.

4.3.4 Create Asset

An asset Id is provided for indexing within the deployed ledger. This API also requires sender and receiver addresses, priority, client Id and a disclose parameter list. The steps involved in certification of assets are as follows, before the asset is added to the ledger.

- Create an imprint of the asset.
- Extract exposed metadata and exposed evidence from asset payload using disclose parameters. This is done using 0xcert certification API.
- Host exposed evidence file on the server.
- Update metadata with schema template URL for reference. A schema template defines how an asset structure should be.
- Hosting the metadata file on the server.
- Finally, create the order of asset creation.

For more details, refer to Listing A.4.

4.3.5 Get owner account Id

This API requires the asset ledger Id and account Id for getting current owner of the asset. For more details, refer to Listing A.6.

4.3.6 Get order information

An order can be of asset creation, asset transfer, asset update, etc. type of actions. The order's status can be checked using API by requesting its data. The order is said to perform successfully and asset is issued if we get status as 7. For more details, refer to Listing A.5.

4.3.7 Verify order

This API will make use of-

- Asset metadata file

- Asset evidence file
- Blockchain imprint

It requires client Id, ledger Id and asset Id to identify the asset on the blockchain after which it will verify the information. For more details, refer to Listing A.7.

4.3.8 Transfer Asset

The transfer of asset requires asset ledger Id, sender address, receiver address and asset Id within the ledger. If the sender is the owner of the asset, the asset can be transferred easily. But the receiver will require additional privileges if he has to trade the asset further. These privileges are granted using APIs to create value approval and create asset approval discussed in the next section.

Multiple signers can participate in the asset transfers. Thus, different stakeholders can provide their signatures for the transfer to pass through. For more details, refer to Listing A.8.

4.3.9 Authorize receiver account for transfers

- Create value Approval. Refer to Listing A.9.
- Get value Approval info. Refer to Listing A.10.
- Create Asset Approval. Refer to Listing A.11.
- Get asset approval information. Refer to Listing A.12.

The sender address has to first allow 0xcert API to transfer assets using his signature or name. This is done by approving value transfer for specific asset so that 0xcert can take payment for performing execution. Next we need to approving asset transfer for the sender address. This is to allow the ledger to transfer the asset for which 0xcert will take fees for themselves.

CHAPTER 5

Feasibility Analysis

Performance requirements are critical to any application. They help to determine how a system thrives under extreme load. A feasibility analysis is required for studying how a project handles the technical, legal, economic and security aspects. This helps the stakeholders know the benefits and drawbacks before investing in the project. The following sections describe few such important feasibility aspects.

5.1 Technology and Market feasibility

The use of blockchain technology is on the rise. Also, most of the supporting tools used and described in the previous chapter are widely used. This project can easily be extended due to the use of prominent technologies like MySQL or Node.js, that are open-source and have a big developer support. However, frameworks like 0xcert demand some learning since they are new in the market. The system might also face problems while overcoming the legal and administrative hurdles. It will require figuring out generalized development due to the laws and administration that differ across regions and countries.

Further, provisions have to be made to handle a large number of transactions right from the system inception. This is because the system needs to be robust enough to manage all existing users in the Government land registry along with the new ones. A huge user base thus makes this project market feasible. This system can ideally be shaped as a public project. It will prove highly beneficial to the common man due to decreased transfer and maintenance expenses.

5.2 Number of transactions and capacity of the system

A large number of property transfers take place every day. If the number of transactions rise, a blockchain system should be able to sustain and manage the excessive load. 0xcert keeps an hierarchy to segregate the assets on providers like

Bitski. Their API allows wrapping assets in a ledger, which are then wrapped in a client to associate assets and their users in a single application.

0xcert has forseen the above requirements of a high transaction and storage capacity. So the number of ledgers one can create in a DApp is basically infinite and really depends on how much credits one has bought. Credits are explained in detail in the next Section 5.3. Any number of ledgers can be created by an user using a single Ethereum address. The same address can be authorized to participate in any other transactions too. But there is a limit to the number of assets inside a ledger. The maximum number of assets that a ledger can hold is (2^{256}) - 1.

Therefore, the throughput of the system will depend more on the scalability and availability of the developed system architecture than the capacity of blockchain. In terms of performance, some delay is involved for each transaction to complete. This is normal and happens for all blockchain based systems, due to the time required for block acceptance on the chain while mining nodes.

5.3 Cost of transactions vs regular processing cost

Traditional transactions of buying or selling real estate include an average real estate commission that is around 5% to 6% of the selling price. Lot of time is spent waiting for the right buyer and then the transaction is performed. The fees for the lawyers, notaries, agents and registration, taxes, and other expenses amount to about 16% of the selling price of the property[41].

In contrast, blockchain based systems can help greatly to avoid the excess financial investment and facilitate high liquidity of non-fungible assets. However, the adoption of blockchain technologies is hold back by poor user experience. Users take time to understand trading using cryptocurrency due to its steep learning curve. As a solution, 0xcert decided to make use of a simplified purchasing process. They convert fiat currency paid with a credit card to 0xcert understandable tokens. The two different fungible tokens used by 0xcert are mentioned below -

- ZXC utility tokens - These are the utility tokens which can be converted into DApp tokens. They acts as a bridge between blockchain and real-world use cases and can be purchased via a credit card or through exchanges. Behind the scenes, fiat money gets exchanged for ZXC tokens and then swapped for DApp tokens(credits).
- DApp tokens/credits - These are spent by 0xcert on behalf of its users for performing actions. Using these in place of ZXC tokens hides the blockchain complexity and improves ease of use. They are modified ERC-20 tokens. The costs for different 0xcert asset operations and corresponding cost in Euros are mentioned in Table 1.

Action	Credits	EUR
Create asset	47.62	2
Transfer asset	35.71	1.5
Transfer value	35.71	1.5
Update asset	35.71	1.5
Set abilities	35.71	1.5
Deploy asset ledger	119.05	5
Deploy value ledger	119.05	5

Table 1: Actions price list[4]

The difference between the cost of transactions to create and transfer an asset using the developed system with the actual cost of transfer of an asset using traditional approach is considerably huge. This difference will remain substantial for other 0xcert like development frameworks, since they will only charge fees to perform underlying smart-contract related complex operations for the user, thereby cutting down excess third party payments whilst providing security and property management.

CHAPTER 6

Experimental Analysis of defense against theft

In this project, one can transfer various types of certificates, agreements, property contracts, and more. To defend against theft, a certification process generates proofs of asset metadata that can be used anytime by third-parties to verify asset information. This will save time and huge expenses required for proving the ownership of the property with traditional means. In this chapter, I review my experiments to show that my system defends against title theft.

In title theft, ownership transfer from title owner to the fraud person involves using a false identity. Loans are then procured by using the same property as collateral. Then the owner has to spend about two years to prove that he did not initiate a title transfer. This costs about $50,000 to hire experts. Even if it is proven that the house is not transferred and the loans are not taken, lot of losses incur in this process if the user does not have a title insurance[42].

Title theft has become a major problem in recent years. It is an under reported crime and is done by anyone very easily. Title fraud losses in the United States crossed more than $5 billion in 2015[42]. Since everything about us is found online, the documents required to steal the property can be downloaded. Once the address is found, the owner of the house is known. The mailing address is changed so the owner does not get any notices from the bank and can result in the foreclosure of the property.

To address this growing problem, there has to be a confirmed identity associated with the titles. This identity should be verifiable by all the agencies on a single platform. With such a system, the online documents would be released only to a verified identity. Such a provision is possible using document imprints maintained on a blockchain. A blockchain based system thus protects user information, which is the most important security advantage.

Following objects are maintained for an asset proof -

- Asset schema - describes the asset metadata structure.
- Asset metadata object - holds asset-related disclosed information. This object is a file hosted on static-server as discussed in Section 4.2. Sample metadata object is as shown in Listing 6.1.

Listing 6.1: Asset Metadata object file

```
{
        "name": "Vastu",
        "description": "This is a new property in Columbia on the river
            ↪  banks.",
        "source": "ARCHIBUS/RECPLY/DCPARKS/LIBRARIES",
        "comment": "CHEVY CHASE COMMUNITY CENTER / LIBRARY. MULTIPLE
            ↪ ADDRESSES",
        "lease": "THE LAB SCHOOL 3591.0980",
        "$evidence": "C:\\Users\\patil\\cs-298\\fs-project\\static-
            ↪ server\\asset_evidence_111.json",
        "$schema": "http://localhost:8080/json/propertySchema.json"
}
```

With the help of these objects, we generate -

- Asset imprint - This is a cryptographic fingerprint of the asset metadata generated using 0xcert AIH algorithm.
- Asset evidence JSON object - This describes disclosed asset metadata with the

blockchain's Merkle tree[43] hash values as shown in Section 6.2. Merkle hashes are maintained as a part of blockchain blocks to prevent tampering of values in the block. They are made by repeatedly hashing pairs of value nodes until only one hash value node remains.

- Asset schema Id string - generated using 0xcert ASH algorithm and uniquely identifies asset schema object.

Listing 6.2: Asset Evidence object file

```
{
        "$schema": "https://0xcert.org/conventions/87-asset-evidence.
            ↪ json",
        "data": [{
                "path": [],
                "nodes": [{
                        "index": 1,
                        "hash": "9b61d.."
                }, {
                        "index": 3,
                        "hash": "d95a2.."
                }, {
                        "index": 5,
                        "hash": "96797.."
                }, {
                        "index": 7,
                        "hash": "49844.."
                }, {
                        "index": 11,
                        "hash": "6f1.."
                }, {
                        "index": 14,
                        "hash": "5ab.."
                }],
                "values": [{
                        "index": 4,
                        "value": "This is a new property in Columbia on
                           ↪ the river banks.",
                        "nonce": "4bg.."
                }, {
                        "index": 6,
                        "value": "Vastu",
                        "nonce": "e7f.."
                }]
        }]
}
```

A step-by-step and mathematical certification process is followed in the system code. This will help in asset verification as well as prevention and detection of theft which is not possible in traditional systems.

6.1 Possible attacks and defense

In this system, we provide the client Id, ledger Id and the asset Id to uniquely identify an asset on the blockchain using 0xcert. This is the first line of defense, since knowing any asset details will first require access to the above key details.

The URL address of the asset metadata file can be obtained by the hacker easily since it is publicly available. Sensitive details should not be included in the metadata file. I performed a few experiments to see if the asset information truthiness gets verified upon modifying the files on the static-server.

Following attributes were modified from asset metadata file-

- Asset schema URL
- Asset evidence URL
- Name
- Description
- Lease Information

Following attributes were changed from asset evidence file-

- data.nodes - This includes Merkle tree[43] hashes.
- data.values - This will require modification similar to the asset metadata changes. Reconstructing the Merkle tree in this file after changing both the files is a challenge for the hacker.

If the location of the asset evidence and asset metadata files is changed by manipulating their locations stored in the MySQL database, this has to change in the asset metadata and evidence files too. If the file contents change by any means, generating the exact match of the hashes in the Merkle tree is a hard to solve problem. Hashes are one-way and deterministic that makes them reliable to use for securing blockchain. With higher computing power, even if the hashes are somehow reconstructed, the real strength lies in the use of imprint stored on the blockchain. The imprint was a result of hashing asset information at the time of creation. Due to this, any slight modification will result in hash mismatch during verification. We can construct one more barrier while developing the system by allowing access of the verification API only to authorized personnel.

15 different assets were used by me to test the verification process. All types of data tampering failed the verification test. This implies that any document modification or access to sensitive information is not possible for an attacker or users without proper authorization.

CHAPTER 7
Conclusion and Future Work

This project studied Ethereum as a blockchain technology along with smart contracts and Solidity language. Solidity is a feature rich language. Ethereum's Solidity smart contracts are heterogeneous in their typologies along with a reactive developer community. The discussion is followed by its use in creation of non-fungible tokens using ERC-721 standard, along with the internal data structures used for different token operations.

The project further discussed a system that is capable of performing safe property deals and maintaining a land registry on blockchain by making use of 0xcert framework. 0xcert leverages the power of non-fungible token standards to represent the assets. This is a significant use-case that will help revolutionize the real-estate industry by safeguarding against fraud. It will help preserve the true identity of the property owners, thanks to the blockchain principles. The system is still in the infancy state, but more work could be done to mature the system further as part of the future work.

7.1 Future Scope

The future of this project is open to a wide range of possibilities. The main challenge in building decentralized applications is a steep learning curve for Solidity smart contract development. Frameworks like 0xcert make this work easy, so the developer can only focus on the use case. The user interface and user management features can be added in the future. Many different types of transactions are possible when dealing with property contracts. Real world use-cases of land registration and dealership can be studied, implemented and be made to fit the concept of smart contracts through this project.

LIST OF REFERENCES

[1] G. Wood, ‘‘Ethereum yellow paper,’’ *Internet: https://github. com/ethereum/yellowpaper,[Oct. 30, 2018]*, 2014.

[2] ‘‘Erc-721 non-fungible token standard,’’ https://eips.ethereum.org/EIPS/eip-721, accessed: 2019-11-6.

[3] ‘‘district-government-land-owned-operated-and-or-managed,’’ https://opendata.dc.gov/datasets/district-government-land-owned-operated-and-or-managed, accessed: 2019-08-07.

[4] ‘‘0xcert api,’’ https://docs.0xcert.org/api/, accessed: 2020-03-14.

[5] ‘‘Erc-20 fungible token standard,’’ https://eips.ethereum.org/EIPS/eip-20, accessed: 2019-11-6.

[6] A. Pinna, S. Ibba, G. Baralla, R. Tonelli, and M. Marchesi, ‘‘A massive analysis of ethereum smart contracts empirical study and code metrics,’’ *IEEE Access*, vol. 7, pp. 78 194--78 213, 2019.

[7] ‘‘Cryptokitties key information,’’ https://www.cryptokitties.co/technical-details, accessed: 2020-01-06.

[8] ‘‘Left.gallery: Downloadable objects, blockchain, bitcoin,’’ https://rhizome.org/editorial/2018/jan/02/leftgallery-downloadable-objects-blockchain-bitcoin/, accessed: 2020-03-06.

[9] ‘‘Kittyhats,’’ https://kittyhats.co/#/, accessed: 2020-03-06.

[10] ‘‘Dada network,’’ https://dada.nyc/home, accessed: 2020-03-06.

[11] ‘‘Super rare,’’ https://superrare.co/, accessed: 2020-03-06.

[12] ‘‘Open sea,’’ https://opensea.io/, accessed: 2020-03-06.

[13] ‘‘Harbor,’’ https://harbor.com/, accessed: 2020-03-06.

[14] S. Chevet, ‘‘Blockchain technology and non-fungible tokens: Reshaping value chains in creative industries,’’ *Available at SSRN 3212662*, 2018.

[15] b. A. C. Bastiaan Don, ‘‘Real estate use cases for blockchain technology,’’ https://entethalliance.org/wp-content/uploads/2019/05/EEA-Real-Estate-SIG-Use-Cases-May-2019.pdf, accessed: 2019-08-06.

[16] S. Nakamoto, ‘‘Bitcoin: A peer-to-peer electronic cash system,’’ Manubot, Tech. Rep., 2019.

[17] V. Buterin *et al.*, ‘‘A next-generation smart contract and decentralized application platform,’’ *white paper*, vol. 3, no. 37, 2014.

[18] S. Enemark, ‘‘Building land information policies,’’ in *Proceedings of Special Forum on Building Land Information Policies in the Americas. Aguascalientes, Mexico*, vol. 26, no. 27.10. Citeseer, 2004, p. 2004.

[19] ‘‘Ethereum big upgrade,’’ https://www.coindesk.com/constantinople-ahead-what-you-need-to-know-about-ethereums-big-upgrade, accessed: 2020-04-19.

[20] V. Buterin, D. Reijsbergen, S. Leonardos, and G. Piliouras, ‘‘Incentives in ethereumâĂŹs hybrid casper protocol,’’ in *2019 IEEE International Conference on Blockchain and Cryptocurrency (ICBC).* IEEE, 2019, pp. 236--244.

[21] ‘‘Ethereum up over 13000% in a year,’’ https://www.cnbc.com/2018/01/10/ethereum-price-hits-record-high-above-1400-up-17000-percent-in-a-year.html, accessed: 2020-04-19.

[22] R. Tonelli, M. I. Lunesu, A. Pinna, D. Taibi, and M. Marchesi, ‘‘Implementing a microservices system with blockchain smart contracts,’’ in *2019 IEEE International Workshop on Blockchain Oriented Software Engineering (IWBOSE).* IEEE, 2019, pp. 22--31.

[23] ‘‘Propy,’’ https://propy.com/browse/, accessed: 2020-03-06.

[24] K. Salah, N. Nizamuddin, R. Jayaraman, and M. Omar, ‘‘Blockchain-based soybean traceability in agricultural supply chain,’’ *IEEE Access*, 2019.

[25] M. Mukhopadhyay, *Ethereum Smart Contract Development: Build blockchain-based decentralized applications using solidity.* Packt Publishing Ltd, 2018.

[26] M. Kim, B. Hilton, Z. Burks, and J. Reyes, ‘‘Integrating blockchain, smart contract-tokens, and iot to design a food traceability solution,’’ in *2018 IEEE 9th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON).* IEEE, 2018, pp. 335--340.

[27] M. Stefanović, S. Ristić, D. Stefanović, M. Bojkić, and D. Pržulj, ‘‘Possible applications of smart contracts in land administration,’’ in *2018 26th Telecommunications Forum (TELFOR).* IEEE, 2018, pp. 420--425.

[28] A. Mohite and A. Acharya, ‘‘Blockchain for government fund tracking using hyperledger,’’ in *2018 International Conference on Computational Techniques, Electronics and Mechanical Systems (CTEMS).* IEEE, 2018, pp. 231--234.

[29] S. Crafa, M. Di Pirro, and E. Zucca, ‘‘Is solidity solid enough?’’ in *International Conference on Financial Cryptography and Data Security.* Springer, 2019, pp. 138--153.

[30] ‘‘Multi-class fungible token,’’ http://web.stanford.edu/~achon/MCFT.pdf, accessed: 2020-03-07.

[31] ‘‘Creating your own erc721 token on moac blockchain,’’ https://moac-docs.readthedocs.io/en/latest/dapps/ERC721.html, accessed: 2019-11-04.

[32] K. Scarbrough, ‘‘Walking through the erc721 full implementation,’’ https://medium.com/blockchannel/walking-through-the-erc721-full-implementation-72ad72735f3c, accessed: 2019-10-20.

[33] ‘‘OpenZeppelin-contract,’’ https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/token/ERC721/ERC721.sol, accessed: 2011-11-04.

[34] A. Parker, ‘‘Jumping into solidity - the erc721 standard,’’ https://medium.com/coinmonks/jumping-into-solidity-the-erc721-standard-part-1-e25b67fc91f3, accessed: 2019-11-6.

[35] ‘‘About node.js,’’ https://nodejs.org/en/, accessed: 2020-03-15.

[36] ‘‘About npm cli,’’ https://docs.npmjs.com/cli/npm, accessed: 2020-04-03.

[37] ‘‘About express.js,’’ https://expressjs.com/, accessed: 2020-03-12.

[38] ‘‘About typescript,’’ https://www.typescriptlang.org/, accessed: 2020-03-14.

[39] ‘‘The beginners guide to using an ethereum test network,’’ https://medium.com/compound-finance/the-beginners-guide-to-using-an-ethereum-test-network-95bbbc85fc1d, accessed: 2020-03-14.

[40] ‘‘Bitski api,’’ https://www.bitski.com/, accessed: 2020-03-04.

[41] ‘‘Cost breakdown of selling a house,’’ https://www.opendoor.com/w/guides/how-much-does-it-cost-to-sell-a-house, accessed: 2020-03-07.

[42] ‘‘Home title fraud: How it’s done and how to protect yourself,’’ https://www.sandiegouniontribune.com/business/economy/sd-fi-title-lock-20180925-story.html, accessed: 2020-04-21.

[43] R. C. Merkle, ‘‘A digital signature based on a conventional encryption function,’’ in *Conference on the theory and application of cryptographic techniques.* Springer, 1987, pp. 369--378.

APPENDIX A

Additional Information

Listing A.1: Creating a Bitski client

```
{
        "path": "http://localhost:8000/ledger/initClient",
        "operations": [{
                "method": "POST",
                "summary": "Initialize a new Bitski client over Rinkeby
                    ↪  test network",
                "consumes": [
                        "application/json"
                ],
                "parameters": [{
                        "simpleClientId": 1,
                        "config": {
                                "sandbox": true,
                                "clientId": "0611..",
                                "network": "rinkeby",
                                "credentialsId": "9dc36..",
                                "credentialsSecret": "247qP8..",
                                "requiredConfirmations": 1,
                                "accountId": "0x8f.."
                        }
                }],
        "responseMessages": [{
            "status": "Success",
            "client_auth_signature": "0:0x26d652.."
        }]
    }]
}
```

Listing A.2: Deploying a new ledger

```
{
    "path": "http://localhost:8000/ledger/deployLedger",
    "operations": [{
        "method": "POST",
        "summary": "Deploy a new ledger on the initialized
           ↪ client",
        "consumes": [
            "application/json"
        ],
        "parameters": [{
            "simpleLedgerId": 1,
            "clientId": "061122..",
            "name": "Lakeside property",
            "symbol": "LT",
            "uriPrefix": "http://127.0.0.1:8080/",
            "uriPostfix": ".json",
            "capabilities": [
                    "toggle_transfers",
                    "destroy_asset",
                    "revoke_asset",
                    "update_asset"
            ],
            "ownerId": "0x8f..",
            "priority": "low"
        }],
        "responseMessages": [{
            "status": "Success",
            "info": "Call API /getDeploymentInfo for latest
                ↪ updates",
            "deploymentRef": "5e8f..",
            "exists": false
        }]
    }]
}
```

Listing A.3: Get the deployed ledger information

```
{
        "path": "http://localhost:8000/ledger/getDeploymentInfo",
        "operations": [{
                "method": "POST",
                "summary": "Get the deployed ledger info by providing
                   ↪ the database identifiers.",
                "consumes": [
                        "application/json"
                ],
                "parameters": [{
                        "simpleLedgerId": 1,
                        "clientId": "06112..",
                        "deploymentRef": "5e8f.."
                }],
                "responseMessages": [{
                        "status": "Success",
                        "data": {
                        "ref": "5e8f..",
                        "kind": 1,
                        "deploy": {
                        "kind": 1,
                        "makerId": "0x8f..",
                        "takerId": null,
                        "seed": 1586457831369,
                        "assetLedgerData": {
                           "name": "Lakeside property",
                                "symbol": "LT",
                                "uriPrefix": "http://127.0.0.1:8080/",
                                "uriPostfix": ".json",
                        "schemaId": "d6b35..",
                        "capabilities": [
                                        3,
                                        1,
                                        4,
                                        2
                                        ],
                                        "ownerId": "0x8f.."
                        },
                        "tokenTransferData": {
                        "ledgerId": "0x56..",
                        "receiverId": "0xf9..",
                        "value": "11904.."
                        },
                        "expiration": 1586630631372
                },
                "creatorId": "0x8f..",
                "claim": "0:0x811ca..",
                "createdAt": "2020-04-09T18:43:53.326Z",
                "status": 7,
                "priority": 1,
                "txHash": "0x35..",
                "ledgerId": "0x59..",
                "ledgerRef": "5e8f.."
},
                        "txnAccessUrl": "https://rinkeby.etherscan.io/tx/0x35
                            ↪ ..",
                        "exists": true
                }]
}
```

Listing A.4: Create a new asset

```
{
        "path": "http://localhost:8000/asset/createAsset",
        "operations": [{
          "method": "POST",
          "summary": "Prepare a new order for asset/property creation
              ↪ .",
          "consumes": [
                "application/json"
          ],
          "parameters": [{
    "simpleAssetId": 1,
    "assetLedgerId": "0x59..",
    "clientId": "0611..",
    "discloseParameterList": [
        [
            "name"
        ],
        [
            "description"
        ],
        [
            "source"
        ],
        [
            "comment"
        ],
        [
            "prototype"
        ],
        [
            "lease"
        ]
    ],
    "senderId": "0x8f..",
    "receiverId": "0x8f..",
    "priority": "low",
    "payload": {
        "description": "Lakeside property of Columbia.",
        "image": "https://troopersgame.com/dog.jpg",
        "name": "Villa",
        "object_id": 2,
        "address_id": 15584,
"ownership_type": "public",
"building_use": "public housing",
"use_code": 11,
"use_description": "Residential-Row-Single-Family",
"use_long_description": "(Class 1 or 2) : Single family
   ↪ dwelling with 2 walls build as common walls with another
   ↪  structure, 2 exposed walls; primarily used as a place
   ↪ of abode.",
"leased": "N",
"address": "3269 STANTON ROAD SE",
"agency": "DCHA",
"ownername": "DISTRICT OF COLUMBIA HOUSING AUTHORITY",
"ownername2": "",
"assessment": 371120,
"tax_rate": 0.0085,
"annual_tax": 0,
"square": 5890,
"suffix": "",
"res": "",
"par": "",
"lot": 169,
"ssl": "5890 0169",
"condo_lot": "N",
"lot_sf": 1525,
"book": 199,
"page": "1",
"source": "ARCHIBUS/RECPLY/DCPARKS/LIBRARIES",
"comment": "HENSON RIDGE",
"status": "CURRENT",
"part_lot": "N",
"prop_id": "DCHA_0038",
"newland": 114850,
"newimpr": 283350,
"newtotal": 398200,
"phaseland": 113430,
"phasebuild": 257690,
"xcoord": 401672,
"ycoord": 131051.23,
"ward": 3,
"anc": "ANC 8E",
"zoning": "R-3",
"site_name": "HENSON RIDGE: 3269 STANTON ROAD SE",
"old_prop": "DCHA_0001",
"proptype": "RESIDENTIAL-SINGLE FAMILY",
"commence": "2020-05-01T00:00:00.000Z",
"expiration": "",
"lease": "",
"shape_area": "",
"shape_len": "",
"condo_regime": "",
        "condo_book": "",
        "condo_page": ""
    }
}],
        "responseMessages": [{
    "status": "Success",
    "info": "Call API /getOrderInfo for latest updates",
    "assetRef": "5e8f729dcb5a8100073683c9",
    "exists": false,
    "exposedMetadataFilePath": "C:\\Users\\patil\\cs-298\\fs-project\\
       ↪ static-server\\asset_metadata_0611220a-7bea-4365-a991-
       ↪ a0cc901d483d-0x59a0A3988A8125aB124D61FCB65163Ba870e1A6B-1.
       ↪ json",
    "exposedEvidenceFilePath": "C:\\Users\\patil\\cs-298\\fs-project\\
       ↪ static-server\\asset_evidence_0611220a-7bea-4365-a991-
       ↪ a0cc901d483d-0x59a0A3988A8125aB124D61FCB65163Ba870e1A6B-1.
       ↪ json"
}]
}
```

Listing A.5: Get the order information

```
{
        "path": "http://localhost:8000/asset/getOrderInfo",
        "operations": [{
```

```
                    "method": "POST",
                    "summary": "Get the order/asset
                       ↪ information.",
                    "consumes": ["application/json"],
                    "parameters": [{
                            "assetRef": "5e8f.."
                    }],
                    "responseMessages": [{
                            "status": "Success",
                            "data": {
                                    "ref": "5e8f..",
                                    "kind": 0,
                                    "order": {
                    "signers": [{
                            "accountId": "0x8f..",
                            "claim": "0:0x18.."
                    }],
                    "wildcardClaim": null,
                    "actions": [{
                            "kind": 1,
                            "ledgerId": "0x59..",
                            "receiverId": "0x8f..",
                            "senderId": "0x8f..",
                            "assetId": "1",
                            "assetImprint": "55b0.."
```

```
                        },
                        {
                              "kind": 3,
                              "ledgerId": "0x56..",
                              "senderId": "0x8f..",
                              "receiverId": "0xf9..",
                              "value": "47619.."
                        }
                        ],
                        "seed": 1586..,
                        "expiration": 1586632088545
                  },
                  "automatedPerform": true,
                  "wildcardSigner": false,
                  "creatorId": "0x8f..",
               "createdAt": "2020-04-09T19:08:13.947Z",
                  "status": 7,
                  "priority": 1,
               "txHash": "0x077.."
            },
            "txnAccessUrl": "https://rinkeby.etherscan.io/tx/0x077
               ↪ .."
      }]
}
```

Listing A.6: Get the owner Id of property asset

```
{
        "path": "http://localhost:8000/asset/getOwnerId",
        "operations": [{
                "method": "POST",
                "summary": "Get the asset Owner Id",
                "consumes": [
                        "application/json"
                ],
                "parameters": [{
                        "simpleAssetId": 1,
                        "assetLedgerId": "0x59a.."
                }],
                "responseMessages": [{
                        "status": "Success",
                        "ownerId": "0x8f.."
                }]
        }]
}
```

Listing A.7: Verify the order

```
{
        "path": "http://localhost:8000/asset/verifyOrder",
        "operations": [{
```

```
                                "method": "POST",
                                "summary": "This API is for public
                                   ↪ verification.",
                                "consumes": [
                                        "application/json"
                                ],
                                "parameters": [{
                                        "simpleAssetId": 1,
                                        "assetLedgerId": "0x59a..",
                                        "clientId": "06112.."
                                }],
                                "responseMessages": [{
                                        "verified": "true"
                                }]
}
```

Listing A.8: Transfer the property asset

```
{
        "path": "http://localhost:8000/asset/transferAsset",
        "operations": [{
                "method": "POST",
                "summary": "Transfer the asset using signatures",
                "consumes": [
                        "application/json"
```

```
            ],
            "parameters": [{
                    "assetLedgerId": "0x59a..",
                    "senderId": "0x8f..",
                    "receiverId": "0x5bf..",
                    "simpleAssetId": "1"
            }],
            "responseMessages": [{
                    "transferOrderRef": "5e8f..",
                    "status": "Success"
            }]
    }]
}
```

Listing A.9: Create Value Approval

```
{
    "path": "http://localhost:8000/asset/createValueApproval",
    "operations": [{
            "method": "POST",
            "summary": "Create value approval for ledger",
            "consumes": [
                    "application/json"
            ],
            "parameters": [{
                "assetLedgerId": "0x59a..",
                "receiverId": "0x5bf..",
                "amount": 100
        }],
        "responseMessages": [{
                "status": "Success",
                "data": {
                        "ref": "5e8f..",
                        "kind": 2,
                        "approve": {
                                "kind": 8,
                                "ledgerId": "0x576F..",
                                "approver": "0x8f3..",
                                "spender": "0x59a..",
                                "value": "1000..",
                                "feeRecipient": "0xf91..",
                                "feeValue": "3571..",
                                "seed": 158..,
                                "expiration": 1586..
                        },
                        "creatorId": "0x8f3..",
                        "claim": "0:0x568..",
                        "createdAt": "2020-04-09T19:30:59.050Z",
                        "status": 1,
                        "priority": 1,
                    "txHash": null
                }
            }]
        }]
}
```

Listing A.10: Get Value Approval Info

```
{
        "path": "http://localhost:8000/asset/getValueApprovalInfo",
        "operations": [{
                "method": "POST",
                "summary": "Get value approval information",
                "consumes": [
                        "application/json"
                ],
                "parameters": [{
                        "assetLedgerId": "0x59..",
                        "accountId": "0x5b.."
                }],
                "responseMessages": [{
                        "status": "Success",
                        "data": {
                                "ref": "5e8..",
                                "kind": 2,
                                "approve": {
                                        "kind": 7,
                                        "ledgerId": "0x59..",
                                        "owner": "0x8f..",
                                        "operator": "0x67..",
                                        "isOperator": true,
                                        "tokenTransferData": {
                                                "ledgerId": "0x56..",
                                                "receiverId": "0xf9..",
                                                "value": "357.."
                                        },
                                        "seed": 1586..,
                                        "expiration": 1586..
                                },
                                "creatorId": "0x8f..",
                                "claim": "0:0xa6d..",
                                "createdAt": "2020-04-09T19:39:51.656Z",
                                "status": 1,
                                "priority": 1,
                                "txHash": null
                        }
                }]
        }]
}
```

Listing A.11: Create Asset Approval

```
{
        "path": "http://localhost:8000/asset/createAssetApproval",
        "operations": [{
                "method": "POST",
                "summary": "Create asset approval for ledger",
                "consumes": [
                        "application/json"
                ],
                "parameters": [
{
        "assetLedgerId":"0x59..",
        "accountId":"0x5b.."
}],
        "responseMessages": [{
    "status": "Success",
    "data": {
        "ref": "5e8f7..",
        "kind": 2,
        "approve": {
            "kind": 7,
            "ledgerId": "0x59..",
            "owner": "0x8f..",
            "operator": "0x67..",
            "isOperator": true,
            "tokenTransferData": {
                "ledgerId": "0x56..",
                "receiverId": "0xf9..",
                "value": "35714.."
            },
            "seed": 1586..,
            "expiration": 1586..
        },
        "creatorId": "0x8f..",
        "claim": "0:0xa6d..",
        "createdAt": "2020-04-09T19:39:51.656Z",
        "status": 1,
        "priority": 1,
        "txHash": null
    }
}]
        }]
}
```

Listing A.12: Get Asset Approval Info

```
{
        "path": "http://localhost:8000/asset/getAssetApprovalInfo",
        "operations": [{
                "method": "POST",
                "summary": "Get asset approval information",
                "consumes": [
                        "application/json"
                ],
                "parameters": [{
                        "approvalRef": "5e8f.."
                }],
                "responseMessages": [{
                        "status": "Success",
                        "data": {
                                "ref": "5e8f..",
                                "kind": 2,
                                "approve": {
                                        "kind": 7,
                                        "ledgerId": "0x59..",
                                        "owner": "0x8f..",
                                        "operator": "0x67..",
                                        "isOperator": true,
                                        "tokenTransferData": {
                                                "ledgerId": "0x56..",
                                                "receiverId": "0xf9..",
                                                "value": "3571.."
                                        },
                                        "seed": 1586..,
                                        "expiration": 1586..
                                },
                                "creatorId": "0x8f..",
                                "claim": "0:0xa6..",
                                "createdAt": "2020-04-09T19:39:51.656Z",
                                "status": 1,
                                "priority": 1,
                                "txHash": null
                        }
                }]
        }]
}
```

APPENDIX B

Glossary

Block:	Group of transactions stored on a blockchain.
Cryptoasset:	Non-fungible asset stored on a blockchain.
DApps:	A piece of software running on a blockchain.
Turing-complete:	A programming language property that can performs all the tasks performed by a regular computer.
Digital asset:	Any object that can exist in a binary format with rights to use.
ICO:	ICO(Initial Company Offering) is similar is creation and distribution of tokens in a blockchain context, which is similar to a company offering stocks in an Initial Public Offering.
Blockchain:	It is a decentralized data storage and transfer protocol that operates in the absence of a central authority.
Metadata:	This is specifications of data embedded in another piece of data like image or sound, which tells when and how the actual data was issued, where it resides, its structure, etc. Literally, this means data about data.
Miner/Mining:	An agent or a group of agents who provide computing power for mining blocks. In exchange, they get cryptocurrency in the form of fees.
Open-source:	Software maintained, distributed and developed free of charge or any form of licensing and a result of collaboration between several software developers.
Hash function:	This compression function is difficult to reverse engineer but easy to calculate the results. This means we cannot guess the inputs with the output.
Mining Pool:	These are co-ordinating miners who proportionately share their rewards depending on their contribution to the computing power for mining blocks together. This helps in maximising the overall profit.
RSA:	Baseline internet cryptographic algorithm.