Imperial College London

MEng Individual Project

Imperial College London

Department of Computing

NFT.finance
Leveraging Non-Fungible Tokens

Author:
Dragos I. Musan

Supervisor:
Prof. William J. Knottenbelt

Second Marker:
Dr. Arthur Gervais

June 15, 2020

Abstract

The impact of Blockchain solutions is more apparent everyday and it has enabled new types of economies to be developed which rely on different and improved principles in comparison to regular financial markets.

As new solutions are added, decentralized finance become more flexible and powerful, attracting more users.

In this project we have focused on the available applications, as well as the different types of token that exist, such as Fungible and Non-Fungible Tokens.

Initially, we had a broad scope of the project, namely to add Over-the-Counter derivatives to the Ethereum ecosystem and bring useful additions to the decentralized finance ecosystem.

In order to lay a strong foundation to our problem and to approach the development of the project, we have used techniques of Human Centered Design research. By doing so, we have improved the initial problem statement and guided the implementation of the project.

While researching the differences between decentralized finance and regular financial markets, we have identified a preliminary need of having financial instruments for Non-Fungible Tokens.

Therefore, we have set two main features as the objectives for the project:

- **Leasing:** which enables owners of Non-Fungible Tokens to lease out their assets to other users.
- **Lending:** which gives the ability to holders of Non-Fungible Tokens to get access to liquidity by locking their assets as collateral for loans.

Additionally, Over-the-Counter derivatives are an extension to this novel type of marketplace which would add more sophisticated financial instruments such as Forward Contracts.

To achieve a solution to our problem, we developed Ethereum Smart Contracts that contain the logic of the features mentioned above. Additionally, we have created a website to facilitate the interaction of the users with this new type of peer-to-peer network for leveraging of Non-Fungible assets.

Overall, the project resulted in a working solution, available to use on a testing network.

Acknowledgements

This project marks the end of my Computing degree and was an incredible opportunity to work on something I am passionate about.

This could have not been possible without all the encouragement and guidance from the people involved and I would like to thank the following:

- My project supervisor, Professor William Knottenbelt, for his support and for trusting me with this project.
- Dr. Arthur Gervais for his advice, as well as his course, Principles of Distributed Ledgers, which gave me a strong background in Blockchain technologies.
- My personal tutor, Dr. Mark Wheelhouse, for his guidance throughout my Imperial education.

Contents

1 Introduction **5**
- 1.1 Blockchain . . . 5
- 1.2 Motivation . . . 5
- 1.3 Problem Statement . . . 6
- 1.4 Contributions . . . 6

2 Background **7**
- 2.1 Blockchain . . . 7
  - 2.1.1 Ethereum . . . 7
  - 2.1.2 Nodes . . . 7
  - 2.1.3 Mining . . . 8
  - 2.1.4 Proof-Of-Work . . . 8
  - 2.1.5 Addresses, Keys and Wallets . . . 8
  - 2.1.6 Transactions . . . 8
  - 2.1.7 Forks . . . 9
  - 2.1.8 Proof-of-Stake . . . 9
  - 2.1.9 Smart Contracts . . . 10
  - 2.1.10 Ethereum Virtual Machine . . . 10
  - 2.1.11 Decentralized Applications (DApps) . . . 11
  - 2.1.12 Oracles . . . 11
  - 2.1.13 Decentralized Autonomous Organizations . . . 12
- 2.2 Ethereum Tokens . . . 12
  - 2.2.1 ERC-20 standard . . . 12
  - 2.2.2 Trading ERC-20 tokens . . . 12
  - 2.2.3 ERC-721 standard . . . 12
  - 2.2.4 Trading ERC-721 tokens . . . 14
  - 2.2.5 ERC-1155 tokens . . . 14
- 2.3 Decentralized Finance (DeFi) . . . 14
  - 2.3.1 MakerDAO . . . 15
  - 2.3.2 Uniswap and AMMs . . . 15
  - 2.3.3 Decentralized Lending Pools . . . 16
  - 2.3.4 Composability and "Money Lego" . . . 16
  - 2.3.5 Flash Loans . . . 17
  - 2.3.6 Decentralized Financial Crisis . . . 18
- 2.4 ISDA . . . 18
  - 2.4.1 Derivatives . . . 19
  - 2.4.2 ISDA Master Agreement . . . 19
  - 2.4.3 Schedule and Confirmation . . . 20

2.4.4 Smart Derivatives Contracts . . . 21
2.4.5 OTC derivatives trade life-cycles . . . 21
2.4.6 Central Counterparty Clearing House . . . 22
2.4.7 Financial products Markup Language . . . 22
2.4.8 ISDA CDM . . . 23
2.5 Derivatives in the blockchain space . . . 25
2.5.1 Hedgy . . . 26
2.5.2 Vega protocol . . . 26
2.5.3 Synthetix . . . 26

3 Design and Requirements **27**
3.1 Human Centered Design Research . . . 27
3.2 Planning . . . 28
3.3 Low-fidelity borrowing and derivatives prototype . . . 30
3.3.1 High-fidelity borrowing prototype . . . 32

4 Implementation **34**
4.1 Smart Contracts . . . 34
4.1.1 ERC-721 standard and OpenZeppelin . . . 35
4.1.2 Leasing smart contract . . . 36
4.1.3 Loans smart contract . . . 39
4.1.4 Deployment and testing . . . 43
4.1.5 Alternative Implementations . . . 44
4.2 DApp client . . . 46
4.2.1 Technologies used . . . 46
4.2.2 High-level structure of the React project . . . 47
4.2.3 Redux store . . . 47
4.2.4 OpenSea Services . . . 51
4.2.5 Web3 services . . . 52
4.2.6 Managing the state of the Application . . . 54
4.2.7 Components and Assets . . . 55
4.3 Final UI elements . . . 59

5 Evaluation **62**
5.1 Smart Contract Security . . . 62
5.2 Gas Prices . . . 63
5.3 Human Centered Design Process . . . 64
5.3.1 Final iteration . . . 64
5.4 Risks . . . 66
5.5 Comparison with existing tools . . . 68
5.5.1 Rocket LP DAO . . . 69
5.5.2 Lend721 . . . 69
5.5.3 Native leasing solutions . . . 69
5.6 Automated Market-Making for NFTs . . . 69

6 Conclusion **71**
6.1 Reflection . . . 71
6.2 Platform Use Cases . . . 71
6.3 Future Work . . . 72

A Appendix A **74**

B Appendix B **76**

Chapter 1

Introduction

1.1 Blockchain

In 2009 Satoshi Nakamoto published the Bitcoin white paper and introduced the idea of an online payments system which facilitates transfer of funds from one party to another without the need of having a centralised financial institution.[1]

To achieve this, it uses a peer-to-peer network which validates transactions based on an agreed consensus algorithm known as Proof-of-Work and proposes an alternative to traditional payment systems.

In the following years other types of blockchains have been developed in order to improve on or showcase a different implementation to Bitcoin. One project that we will particularly focus on is Ethereum which introduced the idea of "smart contracts".

In 2013 Vitalik Buterin published the Ethereum white paper which proposed a blockchain with a built-in Turing-complete programming language that can be used by anyone to program any asset stored on this peer-to-peer network. The code that gets executed is called a "smart contract" and led to the development of many decentralized applications. [2]

The DApp (decentralized application) space saw a boom in 2017 with many companies launched an ICO (initial coin offering) to raise capital and sell "shares" in the form of Ethereum tokens. This was conceptually similar to an IPO (initial public offering), albeit done in the early stages of the project, without having any users or even a working prototype.

By overpromising the possible applications of blockchain at that time, many ICOs failed to deliver on their promises or to attract users. Within a year nearly half of the projects that raised money using an ICO were closed and the price of cryptocurrencies dropped sharply. [3]

1.2 Motivation

One of the first types of useful projects to emerge from this crash were the DeFi (decentralized finance) projects which are leveraging the opportunity to replicate traditional banking ideas and solve its shortcomings using blockchain technology. Derivatives, lending platforms, decentralized exchanges and more, have all been part of the DeFi revolution. [4]

Being newly developed, there are many opportunities for improvement in the DeFi space by using existing solutions or adapting the concepts to new markets.

We set out to investigate the current shortcomings of DeFi and develop a novel implementation with applicability to Non-Fungible Tokens.

Since we are targeting such a wide applicability and vast scope, we had to follow a through method for design research in order to set the requirements for the problem. To achieve this we have used Human Centered Design Techniques with multiple iterations.

1.3 Problem Statement

Throughout the iterations of the Human Centered Design process we have established different goals and built different prototypes that will be discussed in this report.

The final goal was to add DeFi capabilities to Non-Fungible Tokens and bring novel use cases to unique tokens.

Even though Non-Fungible Tokens have market value, the owners of those assets do not have access to solutions for leveraging.

This would enhance their ownership and we will explain how it could potentially work.

1.4 Contributions

Our contribution was to come up with a novel and marketable product in the DeFi space addressed to owners of Non-Fungible Tokens.

Using Human Centered Design techniques we have built a peer-to-peer platform that proposes two new use cases:

- **Lending:** which releases a new stream of capital to owners of Non-Fungible Tokens by locking their asset as collateral for receiving a loan.
- **Leasing:** which enables owners to lease out their tokens to other users on the platform for a fee.

In this report we will motivate why we have chosen this approach, how it could be used and how we have implemented it.

On each stage of the implementation we have evaluated feedback from mentors and potential users that will be explained and laid out accordingly. We felt that this has provided a well-structured guide to our implementation and created a set of requirements that are more likely to have applicability on the market.

Chapter 2

Background

2.1 Blockchain

A blockchain is a decentralized network which keeps transaction records and acts as a source of trust. The data stored on the blockchain is immutable and updated by the peer-to-peer network.

The blockchain is made of "blocks", which hold transactions that have been broadcasted and verified as being valid.

Public blockchains are available for everyone to join and contribute to, in comparison to private blockchains which require permission from a central authority to become part of.

Public blockchains are pseudo-anonymous because all transactions are public, however it may be hard to link them to an identity.

2.1.1 Ethereum

Ethereum is a blockchain that builds on the concepts introduced by Bitcoin, therefore it has no central entity to handle the transactions and instead all events are recorded on a blockchain.

It is an open and decentralized network, secured using a consensus algorithm called Proof-Of-Work. Everything recorded on the blockchain is available to the public.

Ethereum tackles similar problems to Bitcoin, having Byzantine fault tolerance, defining how coins are mined and reaching a network-wide consensus.

Moreover, Ethereum enables the use of "smart contracts" which represent code executable on the blockchain in a decentralized manner by the Ethereum Virtual Machine.

This enables users to build Ethereum token systems which are Ethereum subcurrencies.

By combining smart contracts and tokens, Ethereum opened up the possibility to build a wide array of possible decentralized projects.

2.1.2 Nodes

The nodes are the participants to the blockchain. They are all connected to each other in the peer-to-peer network. They have access to the blockchain and can

verifiy incoming transactions.

Full nodes require a full copy of the blockchain to operate and need a lot of storage space. A user needs to run a full node to become a miner and contribute to the blockchain. On the other hand, a lightweight node stores only the hashes of the blocks and receives additional information from full nodes, so it occupies significantly less space.

2.1.3 Mining

Through mining new transactions are added to the blockchain.

Miners are incentivized to compete with each other because finding the right solution guarantees a reward in newly minted coins and transaction fees.

In comparison to Bitcoin, Ethereum does not have a fixed amount of possible Ether than can be mined.

Once a new block has been mined and verified by the network, miners start competing for the next one.

2.1.4 Proof-Of-Work

The consensus algorithm introduced by Bitcoin, Proof-of-work is also used in Ethereum. It is an algorithm that miners try to solve in order to find a good solution (a suitable hash) for the next block.

The difficulty of this algorithm can be adjusted as more miners join the network and increase the hash rate.

It is innovative because it enables the network to agree on a set of rules used to update the blockchain and allows free entry for anyone who is running a full node. Additionally it has Byzantive fault tolerance meaning that is can handle failures and bad actors that participate in the network.

2.1.5 Addresses, Keys and Wallets

A fundamental concept to blockchain is asymmetric cryptography.

A user can generate a random private key and use it to derive a public key. Using the public key, the address of the user can be generated. At this address the amount of funds is stored and the user can sign transactions from his address using his private key. Those transactions have to be verified by the network using the public key to check the origin.

Losing the private key means losing access to funds.

Users also have wallets to manage their funds as well as keys and addresses. In the case of Ethereum the wallet can also interact with smart contracts and implicitly decentralized apps.

2.1.6 Transactions

Transactions are messages that are signed by the address that has triggered them and can change the state of the network.

The algorithm used for Ethereum transactions is Elliptic Curve Digital Signature and has three main uses as explained below.

- **Proof of authorization:** The signature proves that the owner of the private key is the one who has authorized the transaction.
- **Non-repudiation:** The proof of authorization is undeniable.
- **Immutability of transaction data:** The data of the transaction is immutable.

The structure of a transaction on the Ethereum network is the following.

- **Nonce:** The number of transactions sent from a given address.
- **Gas Price:** The price that initiator of the transaction is willing to pay for its execution.
- **Gas Limit:** The maximum amount of Gas.
- **Recipient:** The destination address that the transaction is calling. This can be a different payable address or a smart contract.
- **Value:** The value in Ether associated with the transaction.
- **Data:** Binary data payload.
- **v, r, s:** Value of the signature of the transaction.

2.1.7 Forks

A "fork" of a blockchain can happen naturally when a parent block has multiple children blocks. It can be observed when blocks are mined by different miners at almost the same time. This is usually easily resolved when one child becomes part of the blockchain.

However, sometimes in order to change the specification of a blockchain, a fork with the new specifications has to be introduced. This can be done using a soft-fork or a hard-fork.

A soft-fork introduces specifications which are backwards-compatible with the network, so nodes that have not updated their software can still process transactions as long as they follow the new rules. On the other hand, hard-forks bring changes by loosening the rules of the blockchain, so the newly mined blocks may not be verifiable by old nodes.

If this creates a debate, the operating nodes may choose to follow a preferred implementation and two separate blockchains are created.

2.1.8 Proof-of-Stake

An alternative approach to Proof-of-Work is Proof-of-Stake, which calculates the "stake" of a node in the network by the amount of cryptocurrency that they hold, instead of their computing power.

Ethereum is planning a hard-fork to switch to Proof-of-Stake with the launch of Ethereum 2.0 because of its benefits in securing the network, while making it more efficient.

2.1.9 Smart Contracts

In 1994 Nick Szabo, defined smart contracts as [6] "a computerized transaction protocol that executes the terms of a contract".

The first blockchain to implement this idea was Ethereum [2] and it led to the development of many applications, especially in the finance sector.

In comparison to Bitcoin, which allows limited scripting, Ethereum is a blockchain on which code written in a Turing-complete programming language can be executed.

Once the code is written in a high-level language, it is compiled to low-level bytecode and can be deployed on the Ethereum network by sending it to the special **0x0** address.

Next the contract is assigned its own address which can be used to access the features of the contract and call its functions.

There are multiples high-level languages for writing Ethereum code, however our language of choice is Solidity.

2.1.10 Ethereum Virtual Machine

The Ethereum virtual Machine (EVM) is a virtual state machine on the blockchian which modifies its state by running smart contract code. [7]

It is a quasi-Touring-complete state machine, because the computation is bounded by the fee allocated for a transaction. This fee is called "gas" and it is necessary to avoid network-overuse and to make sure the blockchain is efficient. Anyone wanting to run code on Ethereum can purchase gas with Ether (the currency of Ethereum).

The architecture is stack-based and there are three main components of storage: [7]

- **Memory:** volatile, cleared for each message call. Reading is limited to 256 bits and writing to 8 or 256 bits.

  It can expand when accessing a different location, however this operation costs gas.

- **Storage:** persistent between calls and is more expensive to initialise or modify.

- **Stack:** all computations are performed on the stack, which has the bytecode of the smart contract loaded.

The EVM instruction set includes the following operations: [8]

- Arithmetic and bit-wise logic operations
- Execution context inquiries
- Memory, Storage and Stack access
- Control flow operations
- Logging, calling, and other operators

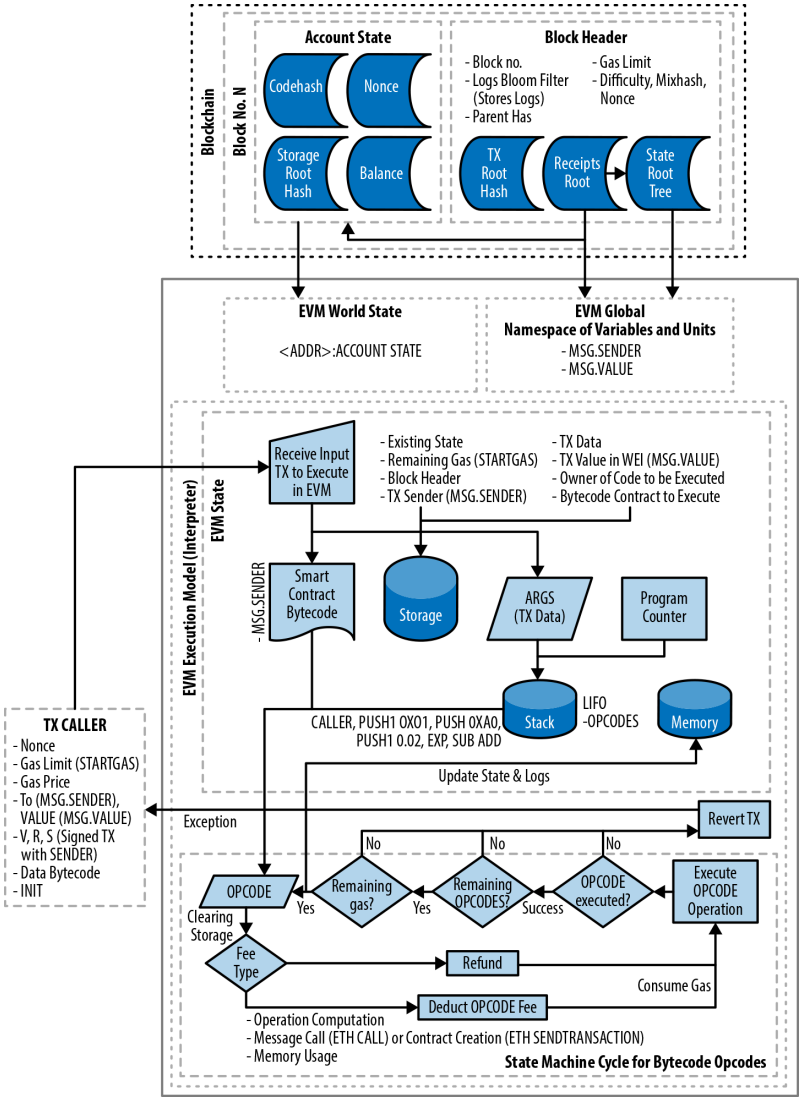


Figure 2.1: The Ethereum Virtual Machine (EVM) Architecture and Execution Context. [8]

2.1.11 Decentralized Applications (DApps)

Decentralized Applications or DApps are smart contract enabled platforms that run on distributed computing systems.

There are a wide range of DApps on Ethereum, which users can interact with.

In this report we will also discuss Decentralized Finance applications which are a subset of DApps that create new protocols for managing and trading blockchain assets.

2.1.12 Oracles

An oracle is external data, connected to the blockchain and impacting it. Oracles can be used with smart contracts to create DApps that act upon input coming from off-chain.

This approach is low-trust but it is up to the users to verify and find real-world data that they can rely on.

2.1.13 Decentralized Autonomous Organizations

Decentralized Autonomous Organizations or DAOs are organizations controlled by token holders, similarly to shareholders.

The tokens issued by the DAO enable their respective owners to vote on matters about the development of the organization and make decisions. Therefore the decision taking process is automated and a consensus is reached among the participants.

The tokens are transferable and the governance of the DAO is decentralized.

2.2 Ethereum Tokens

There are various types of tokens that can be built on Ethereum, usually used in DApps. Tokens can be understood as an asset that lives on the blockchain, which can be stored or managed using Ethereum addresses and smart contracts.

2.2.1 ERC-20 standard

ERC-20 tokens can be defined as a group of identical tokens, all with the same properties. They follow the ERC-20 standard which includes a common set of rules for creating and managing fungible tokens.

The use of ERC-20 tokens enables the creation of small economies that have liquid markets for different use cases.

2.2.2 Trading ERC-20 tokens

ERC-20 tokens can be traded on multiple types of platforms such as:

- Regular exchanges
- Decentralized exchanges
- Automated liquidity pool which can enable seamless token swaps using Automated Market Making algorithms to determine the price of buying ERC-20 tokens.

More about ERC-20 tokens will be discussed in the DeFi subsection.

2.2.3 ERC-721 standard

On the other hand, ERC-721 tokens are Non-Fungible Tokens (NFT). This implies that each token has a unique set of properties and values associated with it. [32]

Being unique makes ownership more desirable especially in the case of highly-sought after tokens.

The ERC-721 standard is an interface that each smart contract which creates ERC-721 tokens has to implement. [32]

There are multiple functions that enable interaction with NFTs such as:

- Finding the owner adderss of an ERC-721 token
- Approving the transfer of an ERC-721 token. Each address that we want to transfer to to be approved before initiating the transfer.

- Checking the approved addresses of an ERC-721 token.
- Transffering the ERC-721 token

ERC-721 tokens have found applicability in many domains of the Ethereum space, the most relevant being the following:

- **Gaming:** gaming items can be expressed as ERC-721 tokens. Each having a unique set of properties and potentially being a power-up, in-game user customization or any other utility.
- **VR real estate:** is real estate in Virtual Reality simulations defined as ERC-721 tokens.

  This could be a subsection of gaming, however it has a unique following composed of both gaming and non-gaming enthusiasts that own those assets to speculate on their market price.
- **Collectibles:** due to their uniqueness they are a perfect fit for collectible items. Examples can range from football cards to CryptoKitties, which are virtual pets, able to breed and create other unique tokens.
- **Utilities:** ERC-721 tokens can enable their owners to access resources. An example of this would be ENS names for managing multiple addresses into one readable string.

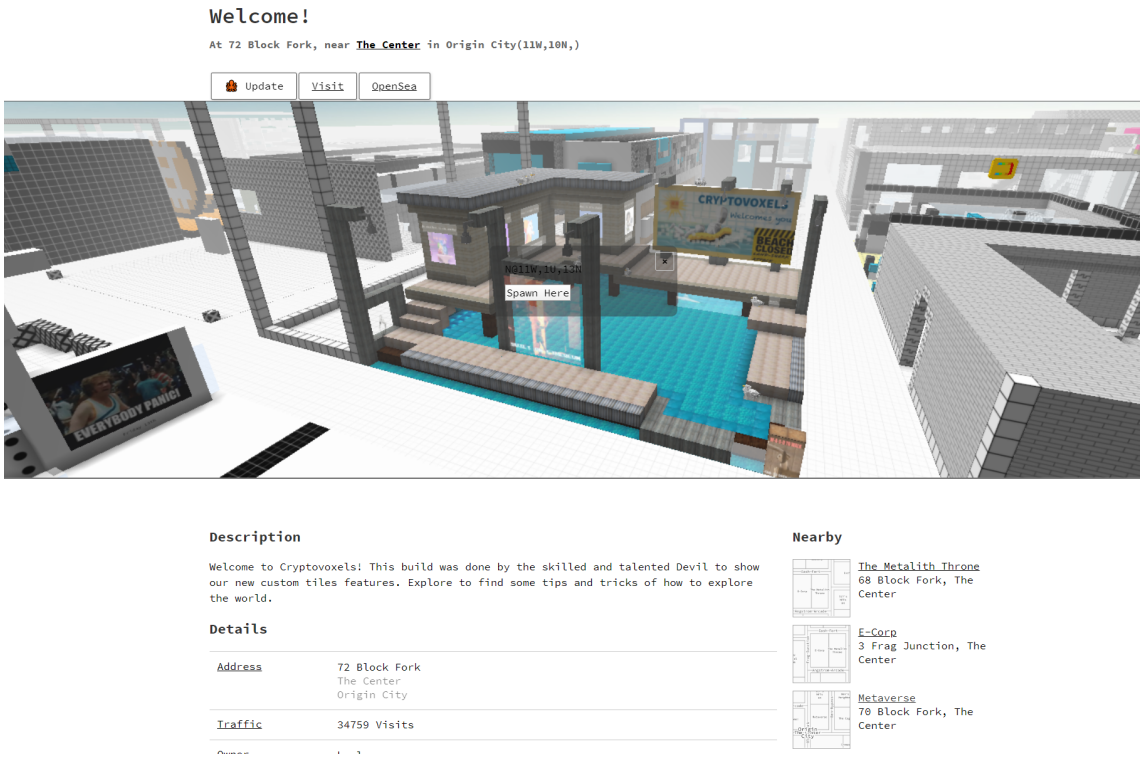


Figure 2.2: Cryptovoxels VR simulation Welcome property represented by an ERC-721 token.[9]

2.2.4 Trading ERC-721 tokens

Similarly to ERC-20 tokens, ERC-721 tokens can also exchange ownership and be traded for other tokens or Ether.

However, the way this is done is different. Since all ERC-721 tokens have unique properties, Automated Market Making algorithms are not feasible to be implemented.

The accepted way to trade them at the moment is to auction or sell them on peer-to-peer marketplaces specific to NFTs, the largest at the moment being OpenSea.

This creates new economies for digital collectibles with impressive statistics for a new type of asset: [29]

- 7 day trading volume of the top 3 kinds of NFTs is over 1,000 Ether.
- Decentraland, another VR simulation has a total volume of 44,000 Ether split amongst 3,662 owners. This gives an average ownership of LAND properties of around 12 Ether.
- The most expensive NFT sold in 2019 has been an F1 gaming item worth 415.9 Ether. [11]

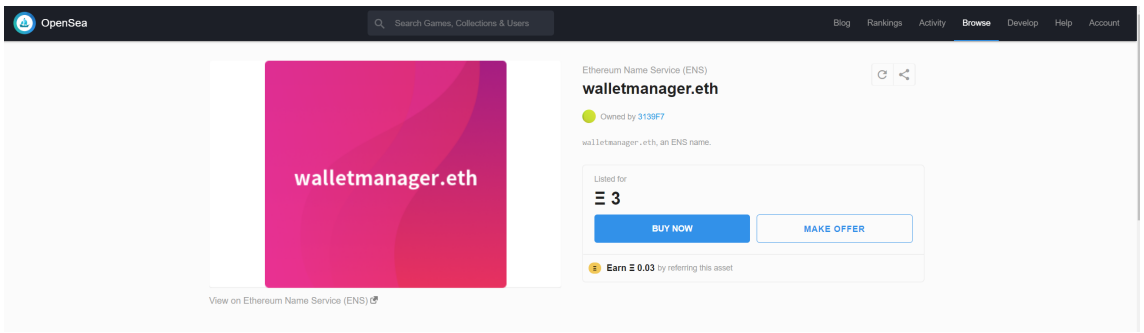


Figure 2.3: ENS name being sold on OpenSea marketplace.

2.2.5 ERC-1155 tokens

The ERC-1155 is a new token proposal standard aimed at creating both fungible and non-fugible tokens in the same contract.

This could be particularly useful for games that want to create more complex in-game economies.

2.3 Decentralized Finance (DeFi)

Decentralized Finance protocols are DApps that enable interoperable protocols for leveraging and trading exclusively ERC-20 tokens.

Currently there is close to 1bn USD of capital locked in DeFi, as of June 2020, with the largest protocol being Maker. [13]

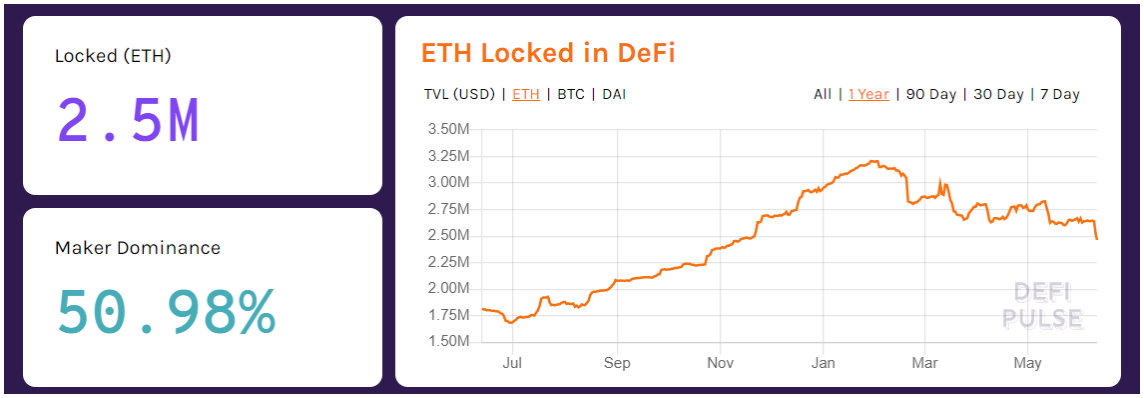


Figure 2.4: Total value locked in DeFi, taken from DeFi Pulse. [12]

2.3.1 MakerDAO

Maker is a DAO that offers multiple products such as:

- **Oasis:** non-custodial platform to exchange tokens or borrow and earn interest using DAI, a decentralized stablecoin, soft-pegged to 1 USD.
- **Migrate and Upgrade:** enables moving of DAI and Collateralized Debt Positions to newer versions.
- **Maker Ecosystem:** a marketplace to explore apps or services that user Maker protocols or DAI.
- **Governance Portal:** the governance of the DAO is done with MKR tokens. Participants can use MKR to vote for changes on the platform.

The target price of DAI is 1:1 USD and smart contracts try to keep the price stable. When DAI is created, Ether has to be deposited into a Collateralized Debt Positions (CDP). In exchange DAI will be created.

When returning DAI for the locked Ether, a stability fee has to be paid which is used to incentivize or make loans in DAI more expensive, which in turn can affect the price of DAI, making it lower or higher, respectively.

The stability fee is voted upon by the DAO community using MKR tokens.

However, the DAI price can fluctuate in case of unexpected "Black Swan" events such as the one that occurred on March 12-13, 2020. Due to the low price of Ether to USD, the total valued in USD locked in Maker dropped by almost 50% in a day and the CDPs for DAI have become under-collateralized.

2.3.2 Uniswap and AMMs

Uniswap is a smart contract based Automated Market Maker that dynamically calculates and adjusts the exchange rate for trading ERC-20 tokens.

In comparison to other decentralized exchanges (DEXs) it does not use order book mechanisms.

AMMs use liquidity providers who deposit assets in different pools and can create trading pairs using algorithms such as a Constant Product, which keep the product of a pair of assets constant at trading times.

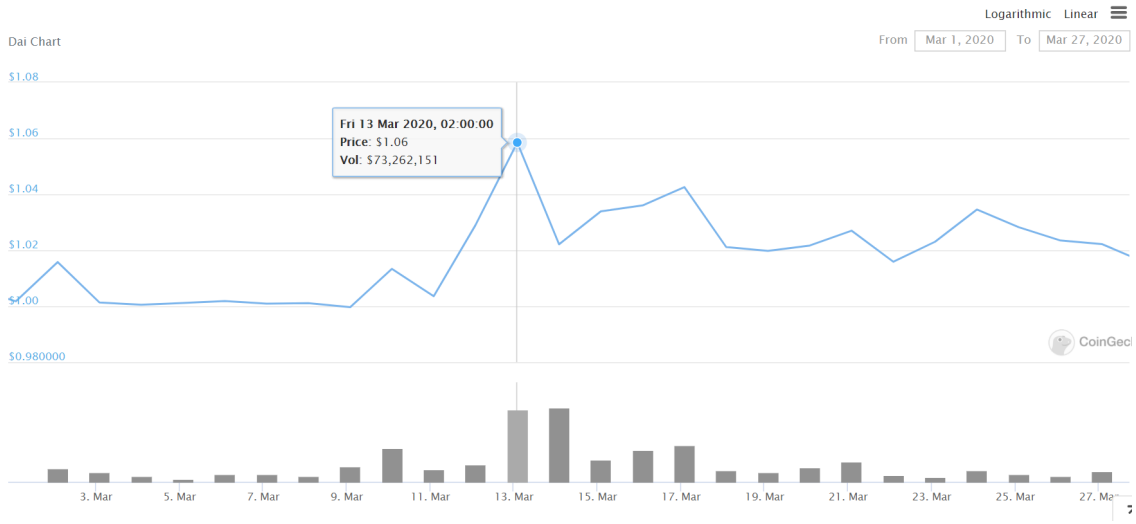


Figure 2.5: DAI price with emphasis on the Market Collapse of March 12-13, 2020. Screenshot taken from Coingecko [12]

If we have two trading pairs of tokens `X` and `Y`, with `x` and `y` being the number of tokens, the market price is determined such that the ratio of their product remains constant. [43]

2.3.3 Decentralized Lending Pools

Other popular DeFi solutions are Decentralized Lending Pools (DLPs).

DLPs enable lending that does not rely on peer-to-peer Order Matching. They work with liquidity providers for each token market. All the identical ERC-20 tokens are gathered in a pool, that borrowers can take advantage of and get liquidity from, providing they have deposited collateral.

Lenders instantly earn interest by providing liquidity and borrowers also earn interest in the collateral that they post.

Some of the features that make DLPs superior to P2P lending markets are the following: [45]

- Instant deposits
- Instant borrowing
- Different types of ERC-20 tokens as collateral
- Instant switching between fixed or variable interest rates for the loans

Preceding DLPs, there were DeFi platforms which enabled peer-to-peer lending.

However, since DLPs are a vastly superior solution with liquid secondary markets and lower risk, platform such as ETHLend (P2P loans) have evolved into DLP solutions, namely Aave. [44]

2.3.4 Composability and "Money Lego"

There are a multitude of DeFi solutions many able to interact with each other and pieced together as "Money Lego".

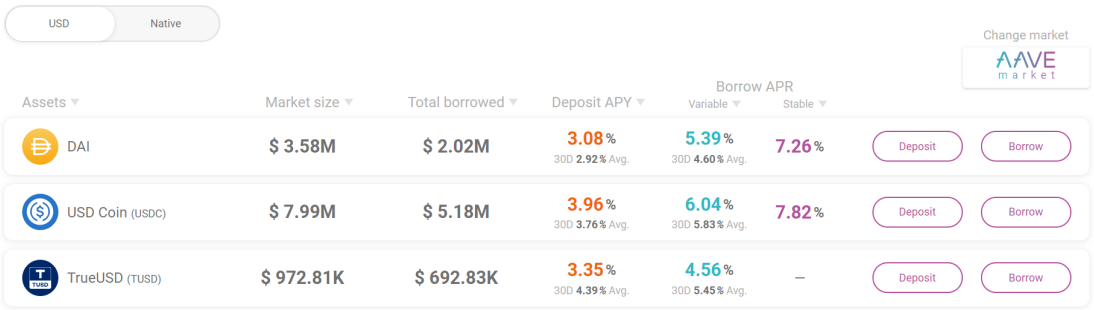


Figure 2.6: Screenshot of top assets in Aave. [12]

By combining different protocols, new financial tools can be created. For example you can place your Ether in Market DAO, mint DAI and then use DAI in other protocols to receive loans or keep them in a non-custodial wallet.

2.3.5 Flash Loans

Flash Loans are a new type of loan, exclusive to Ethereum DeFi, which enable instant borrowing, with no collateral, by anyone, as long as the principal amount is returned in the same block.

In case the principal is not returned the transaction can be reverted as if it never took place.

The novelties of flash loans are the following: [46]

- **No risk of default:** as explained above, if the transaction fails due to the user defaulting on his loan, the flash loan is not granted.
- **No required collateral:** Since there is no risk for the lender, there is no required collateral.
- **High Loan Size:** The loan size can be as high as the liquidity pool it is taken from.

There are vast use cases for Flash Loans and taking advantage of the "Money Lego" property of DeFi, can enable users to benefit from complex opportunities no matter what their address balance or trading history is.

Some use cases of Flash Loans are the following: [46]

- **Arbitrage:** Users can borrow liquidity and benefit from arbitrage opporunities between DEXs, witout being exposed to any risk. If their trade is successful, they pay back the principal amount of the loan and any required interest.
- **Wash Trading:** This is a type of trading aimed at manipulating trading volumes of assets and artificially increasing them using fake automated trades.

  Flash loans could be used for achieving this, without owning any of the assets traded, making use of the Lego Money property of DeFi and exchanging assets back and forth in one atomic transaction.
- **Collateral Swapping:** CDPs, as used in MakerDAO for minting DAI, require locking collateral, in this case Ether, until the DAI is returned.

However, the locked collateral in a CDP is exposed to risk of currency fluctuation.

Flash loans can be used to mitigate this risk without having the new asset by borrowing it and then closing the CDP. Immediately, a new CDP can be created with a different underlying asset to minimize losses from currency market fluctuations.

- **Flash Minting:** ERC-20 tokens can be minted and flash minting would enable instant minting of that ERC-20 token in a single transaction.

  The would be a momentary increase in the supply of the token and then the coins would be destroyed.

  It is a possible use case, with an application that is yet to be determined. [46]

2.3.6 Decentralized Financial Crisis

Throughout the different attacks and big arbitrage opportunities, drawbacks of DeFi protocols have been exposed. [47]

In case of sudden price drops of the assets locked in DeFi, overcollateralized protocols show deficiencies.

What is worse is that the failure of protocol can impact other dependable protocols and lead to additional losses.

As explained in [47], an attack on MakerDAO could have broader implications on other protocols such as Uniswap, Compound and more.

The composability of DeFi could have financial contagion in case of an attack that can lead to a DeFi crisis.

In case an asset becomes under-collateralized and it is used as collateral on another platform to issue a new asset, it would lead to the other asset being under-collateralized.

A simple example would be the scenario of DAI becoming under-collateralized. In this case, cDAI, which is Compound issued DAI would also become under-collateralized.

While DeFi is setting out to improve on regular financial instruments, it might fail to respond to high stress scenarios and in the face of malicious participants.

2.4 ISDA

The International Swaps and Derivatives Association (ISDA) is a trade organization that aims to improve the global derivatives market by making it safer and more efficient.

They were created in 1985, have more than 900 member institutions from 72 countries and their work focuses on reducing counter-party credit risk, increasing transparency and making the derivatives industry more stable. [14]

They have created the ISDA Master Agreement, which is the most used blueprint for creating over-the-counter (OTC) derivatives and executing these transactions.

This document has been updated over the years to keep up with current needs and technological advancements as well as cover any issues later exposed.

It provides the necessary infrastructure to simplify the negotiation process of the parties and prevent disputes.

2.4.1 Derivatives

Derivatives are financial tools that derive their value from an underlying asset.

They act as a contract between multiple parties and can be traded over-the-counter (OTC) or through an exchange.

OTC derivatives are tailor made for parties that do not want to go through an exchange or need a more flexible, exotic solution than vanilla derivatives. They carry higher counter-party risk and have limited transparency.

There are different types of derivatives such as

- **Options:** Used to guarantee the right but not the obligation to buy or sell an asset.
- **Swaps:** Used to exchange cash flows with a counter-party.
- **Future or Forward contracts:** Give the obligation to buy an asset in the future at a pre-specified price.

Common vanilla derivatives have been standardized and they have greater liquidity. They can be traded through an exchange and there is less risk involved when trading them in comparison to OTC derivatives.

As the market progresses and new types of exotic derivatives are used more frequently, they can also become standardized and placed on exchanges.

2.4.2 ISDA Master Agreement

The ISDA Master Agreement provides a documentation for the derivatives industry, to make the market safer and more efficient for the participants. The Master Agreement is a pre-defined contract that is not to be modified by any of the parties and applies to all the derivatives.

It has provisions to 5 main themes: [15]

- **Events:** Contains examples of situations that can affect one of the parties ability to perform its obligations.

  They can be two kinds of events, each with predefined contractual actions in case of occurrence: [15]

  - **Events of default:** Usually occur because one of the parties is unable to fulfill the agreement and is at fault for defaulting.

    In the ISDA agreement different types of default events are specified, but more can be added later on, if needed, as explained in the following subsection.
  - **Events of termination:** In case of termination events, none of the parties is at fault and all the involved participants are considered 'affected parties'.

    Such events can be Illegalities, Force Majeure events and more. [15]

- **Close out and Netting:** The steps followed by the parties involved, to close the transaction. This happens in case an Event occurs and the parties earn the right to exercise a close-out of the transactions.
- **Contract Formation and Legal:** In this section the legal compliance through the agreement for the parties and their representations are laid out.
- **Disputes:** Information about managing disagreements between the parties and how they could be potentially settled.
- **Payments and Deliveries:** This section enforces the economic obligations of the transaction, laid out in the Confirmation of the ISDA Master Agreement. However, it includes information about possible scenarios that can impact or modify the nature and delivery of the payments.

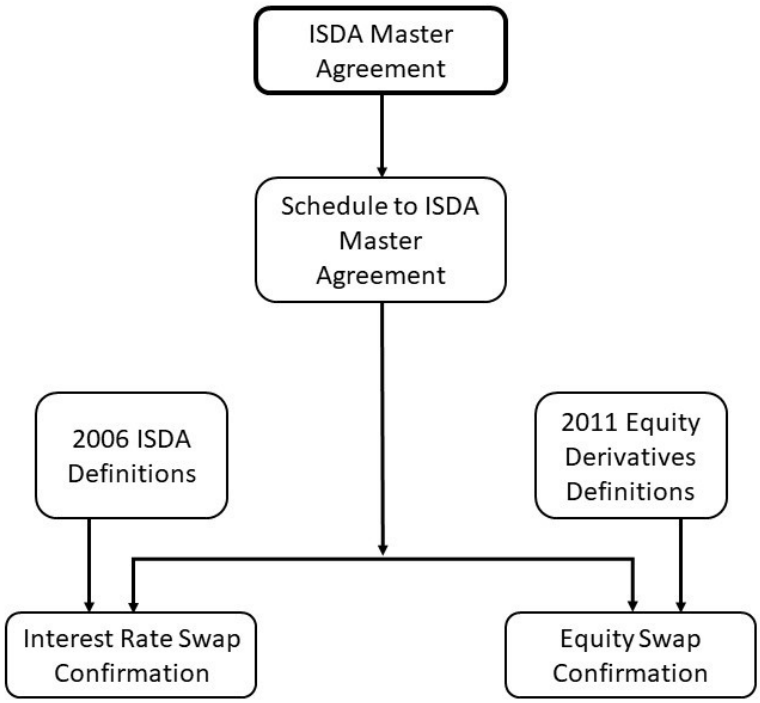


Figure 2.7: ISDA Document Components adapted from [5].

2.4.3 Schedule and Confirmation

In Figure 2.7 we have the following components

- ISDA Master Agreement which is standard.
- The Schedule which can be customized.

- There are multiple possible transactions, with two of them showcased in Figure 2.7 (Interest Rate Swap and Equity Swap).

  Each has a Confirmation document that uses different ISDA definitions.

Therefore, in order to introduce changes tailored to specific needs, supporting documents to the ISDA Master Agreement, such as the Schedule and the Confirmation can be used.

- **Schedule:** The parties involved can modify or add any details to the the Master Agreement and it provides a high amount of customization to the investors. Changes in the Schedule apply to all the transactions entered under this contract.

- **Confirmation**: Final changes can be made in the Confirmation of the transaction to modify any of the terms previously agreed upon in the Master Agreement or Schedule.

  This particular document is used to define custom economic terms or to take advantage of predefined ones applicable to a specific derivative and makes reference to other ISDA Definitions. [5]

2.4.4 Smart Derivatives Contracts

Following Nick Szabo's definition of smart contracts, it can be deduced that they can be applied to build derivatives, by automating some or all of the components of those financial instruments.

However, some of the specifications of the ISDA Master Agreement cannot be laid out in code and many issues that need to be solved can arise, such as the ones presented below.

- Processing human interventions
- Trustfulness of incoming data
- Transparency of the transactions and more

Smart contracts also represent a way to guarantee a certain behaviour, but they are currently not backed by legal enforcement and this could be crucial when reacting to human input and expecting an honest behaviour independent of any private motivation.

Therefore, in order to replace current derivatives, smart contracts have limitations due to the nature of transactions and should provide a combination of human input, legal enforcement of rights and tamper-proof code. [5]

The current proposal from ISDA states that "only those parts of a derivatives contract that are capable of being automated should be considered" and "effective automation should be based on legal validation". [16]

2.4.5 OTC derivatives trade life-cycles

There are multiple steps that form the life-cycle of a trade for OTC derivatives and it involves multiple stages as well as complex actions from the parties involved, as follows:

1. **Pre-trade:** The parameters for trading are set and credit checks are performed to verify the limits for a safe transaction.

2. **Trade:** The parties agree on the terms of the trade in the limits set in the Pre-trade stage.

3. **Post-trade:** At this stage the life-cycle of the trade is being managed through the following consecutive sub-activities:

   (a) **Trade capture:** The details that were agreed upon are formally set in writing or digitally.

   (b) **Economic Affirmation (Trade Verification):** Economic details are re-checked.

   (c) **Trade matching:** The parties match the terms of the trade to have the same final record of the transaction.

   (d) **Confirmation:** Final confirmation between the parties.

   (e) **Settlement:** The derivative is executed as expressed in the contract and the assets are settled.

2.4.6 Central Counterparty Clearing House

Central Counterparty Clearing Houses (CCP) have the role to mitigate the risk of default of the parties involved in financial contracts.

They manage the clearing and settlement parts of the transaction and have strict requirements for a accepting a trade.

As a result of the financial crisis of 2007, CCPs act as an intermediary in OTC derivatives transactions to prevent and minimize failures of market participants.

In 2009, G20 leaders have decided that standard derivatives should be traded on exchanges and cleared through CCPs. This enables more liquidity and mitigates risk. [18]

In OTC derivatives, the CCP exhibits the role of a counter-party for both sides involved in the trade and would decide the exact collateral needed for a safe transaction.

Through this supervision, a standard is imposed for creating and managing OTC derivatives that reallocates and lowers the risk by centralizing trades.

2.4.7 Financial products Markup Language

The FpML is an open source standard for representing OTC derivatives, first published by JP Morgan in 1999. It is based on XML and is the standard currently integrated by ISDA.

Participants in OTC trades use FpML for messaging and documenting workflows across different platforms and it is being expanded as new financial products are added. [19]

Even though the workflows and interactions of derivatives are documented through FpML, this standard does not addresses how they could be implemented in code. [20]

2.4.8 ISDA CDM

The ISDA Common Domain Model (CDM) is a standardized representation of the events and parts underlying a derivative in the post-execution trade lifecycle.

It enables the formalization through code of different types of financial instruments and provides a general structure for the whole market to build, trade and manage derivatives. [17]

The CDM model has components that can be found in the ISDA Master Agreement and its data structure also follows the Financial Products Markup Language.

In comparison to FpML, it also enables users to specify business logic in addition to data representation. It is more verbose and sets a new standard for representing OTC derivatives that is also compatible with blockchain technologies and can be translated to various programming languages.

The key advantages of ISDA CDM are the following: [21]

- Set a standard to recording and sharing trade life-cycle events.
- Consistency among all the participants to a trade and interoperability between firms.
- Accelerate automation and enable standards for processing derivatives that can be applied to both decentralized ledger technologies or centralized solutions.

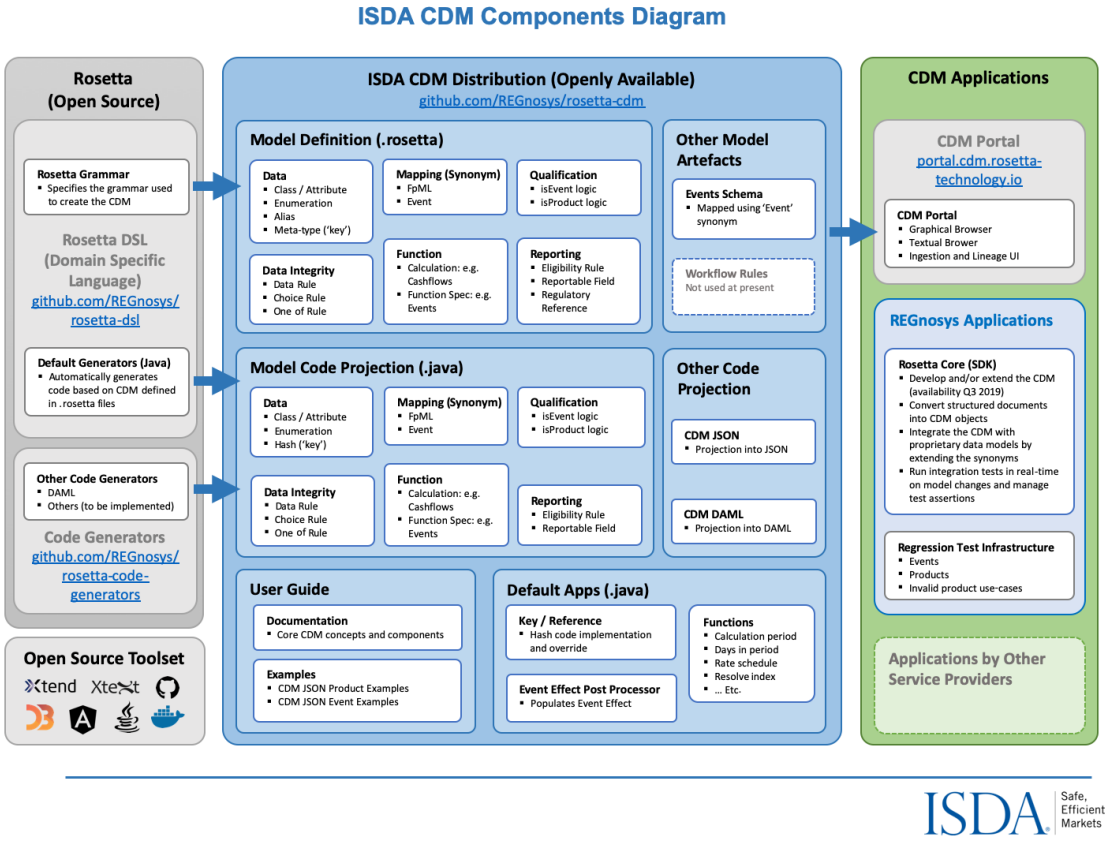


Figure 2.8: ISDA CDM Components Diagram from the ISDA Documentation. [21]

The ISDA CDM has different layers, having three main components:

- **Rosetta:** an Open Source domain specific language, having a syntax and code generator for translation to other programming languages.
- **ISDA CDM Distribution:** which includes multiple components such as the following
  - definitions of the comprising models (e.g. Data representation, Functions for different derivatives calculations, Reporting, Mapping, etc.)
  - Model Code Projection in Java or JSON.
  - Default Apps which are implementations of different functions that can be used in the life-cycle of derivatives as such, or modified to suit different differnt use cases.
- **CDM Applications:** solutions that use ISDA CDM.

  Currently, there is a web application that enables graphical browsing of different functions of the CDM, called the CDM Portal.

  Additionally, there have been implementations with different blockchain solutions of the ISDA CDM, such as Corda or Algorand as proof of concept. [22] [23]

  Those solutions are aimed at Financial Institutions and are using private blockchains.

The CDM has six components that are used for building derivatives: [21]

- **Product:** Contains the definitions of the financial instrument defined in the CDM.

  There are different product types (e.g. Tradable Products, Financial Products, Contractual Products, etc.), each with its own characteristics and custom economic terms.
- **Event:** Represents the different states that a transaction goes through and can be initiated by the parties involved.

  The events correspond to different parts of the ISDA derivatives life-cycle and define the following:
  - Trade execution and confirmation
  - Clearing
  - Allocation
  - Settlement
  - Exercise of options

  Events can be extended to and combined to build workflows which capture and modify the state of the trade.
- **Legal Agreement:** Provides a digital representation of legal contracts.
- **Process:** Using the terms of the trade previously defined, the Process implements the functions and behavior of the derivative.

- **Reference Data:** Representation of the Parties involved in the trade.
- **Mapping (Synonym):** Mapping to translate other industry standards, such as the FpML into the ISDA CDM.

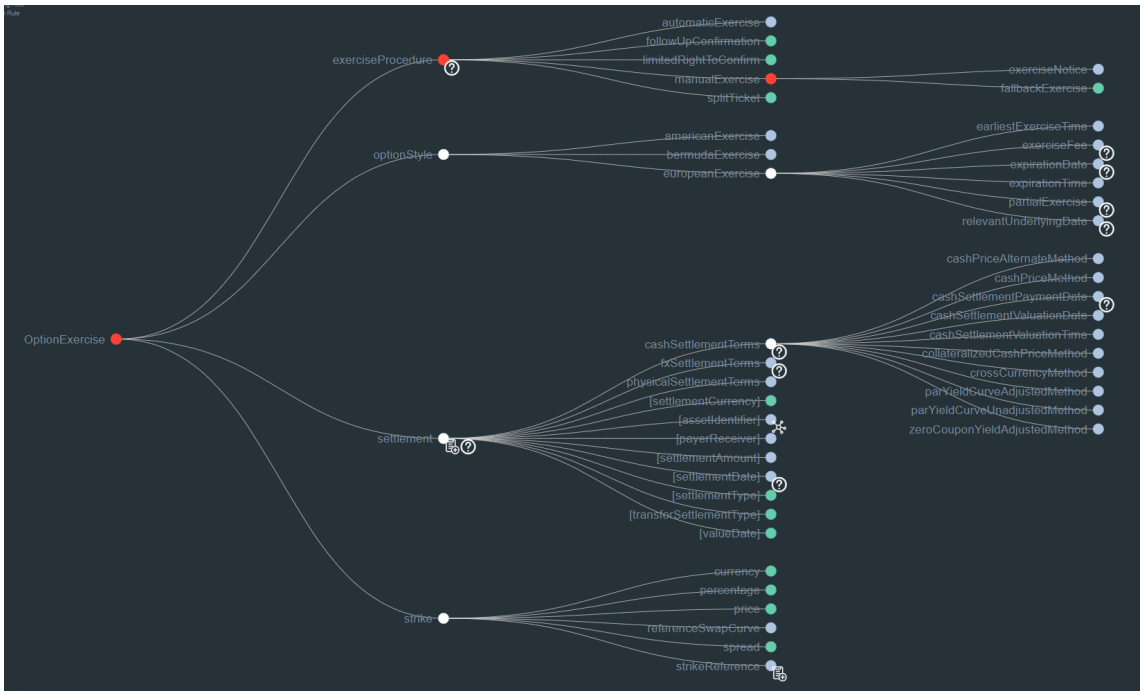


Figure 2.9: Screenshot of the components forming the definition for exercising an Option taken from the CDM Portal. [21]

In Figure 2.9 we can observe the components that form the definition of different functions and input data for an Option Exercise. An Option grants the right, but not the obligation to buy the underlying asset.

The Option style selected is an European Option, that can only be exercised on a pre-defined date, against a fee.

The Exercise procedure could be automatic or manual and the settlement can be in cash, where applicable, with the choice going through a clearing house for safer clearing of the trade.

Finally, the strike price is the fixed amount needed to buy the underlying asset at the date of maturity, in a specific currency or with a defined spread.

This is just an example of a simple financial instrument defined through ISDA CDM, but we can observe the high flexibility and thorough details that can be used as input for custom defined products. There are multiple possibilities and exotic derivatives can be created to suit the various needs of clients.

2.5 Derivatives in the blockchain space

There has been a wide range of crypto-derivatives exchanges that have launched lately, each focusing on a different problem or approach. They are a necessary stepping stone to reach a more mature cryptocurrency trading market.

2.5.1 Hedgy

Hedgy was a start-up which specialized in Bitcoin technology.

They have developed an OTC trading platform using Bitcoin script before smart contracts were introduced, following the structure of ISDA Master Agreements and supporting documents. [24]

They were one of the first companies to introduce the idea of derivatives using blockchain technology.

In 2018, they were acquired by Wyre, a compliance and payments solution, and integrated in their platform. [25]

Wyre currently offer an OTC trading desk addressed to high-profile, institutional investors. [**?**]

2.5.2 Vega protocol

Vega protocol are a startup which enables the creation of exchange traded derivatives on a decentralized network.

They are building their own blockchain and language for smart contracts dedicated to handling derivatives, which will enable any participant to create custom financial products. Once a new derivative is built, other participants can vote on creating a liquid market for it.

The underlying asset can vary from crypto-currencies to ERC-20 tokens or other types of assets from regular financial markets, as Vega provides connection to oracles for pricing updates. [26]

2.5.3 Synthetix

Synthetix focuses on the creation of synthetic crypto-assets (tokens that get their underlying value from oracles of fiat currencies, commodities, stocks and more) on Ethereum. They will enable the creation of different derivative instruments and all the tokens are ERC-20 tokens, created with the intent of being traded on their exchange. [27]

The prices of their synthetic tokens is determined by an average from a variety of oracle sources which feeds the data on-chain. It is not decentralized at the moment.

Their platform is part of the DeFi ecosystem and their smart contracts are open-source.

Chapter 3

Design and Requirements

In this chapter we will discuss our approach to identifying a gap in the market of NFTs and applying adequate DeFi principles as well as insights about the derivatives market gathered through the research presented in the Background chapter.

The requirements were adapted as we received more feedback and the problem statement was updated to bring novel features to owners of ERC-721 tokens.

The final problem we aimed to solve has been laid out after multiple iterations and an investigation of the DeFi market, where a gap has been spotted.

In order to structure and standardize our approach to receiving and reacting to the feedback of users or mentors, we have followed Human Centered Design Research Techniques.

Additionally, to get feedback for our solution, we have taken part in the HackMoney 30-day hackathon, organized by ETHGlobal. [59] This event was aimed at finding new solutions and improvements for the DeFi space and it enabled us to have access to mentors from the industry.

3.1 Human Centered Design Research

Human Centered Design is a well-defined approach to developing interactive applications, which constitutes a set of design activities.

This lays out a formal plan for the development of the platform while taking into account human factors and usability insights. [60]

Initially, we have planned the HCD process adapting the steps proposed by the ISO standards. [60]

There were many benefits to choosing this approach and it enabled a continuous assessment of our requirements and helped us redefine goals as we gained more information about the industry and users.

The steps, adapted from ISO, are laid out in Figure 3.1.

The first part of the HCD process, namely the planning, started with the background research and defining the motivation for the problem.

Following this, we have conducted three iterative rounds which re-assessed the definition of the problem, added to the requirements and improved the UX.

In this chapter we will focus on the first two rounds, the last one having a slightly different and more thorough approach and will be discussed in the Evaluation chapter.

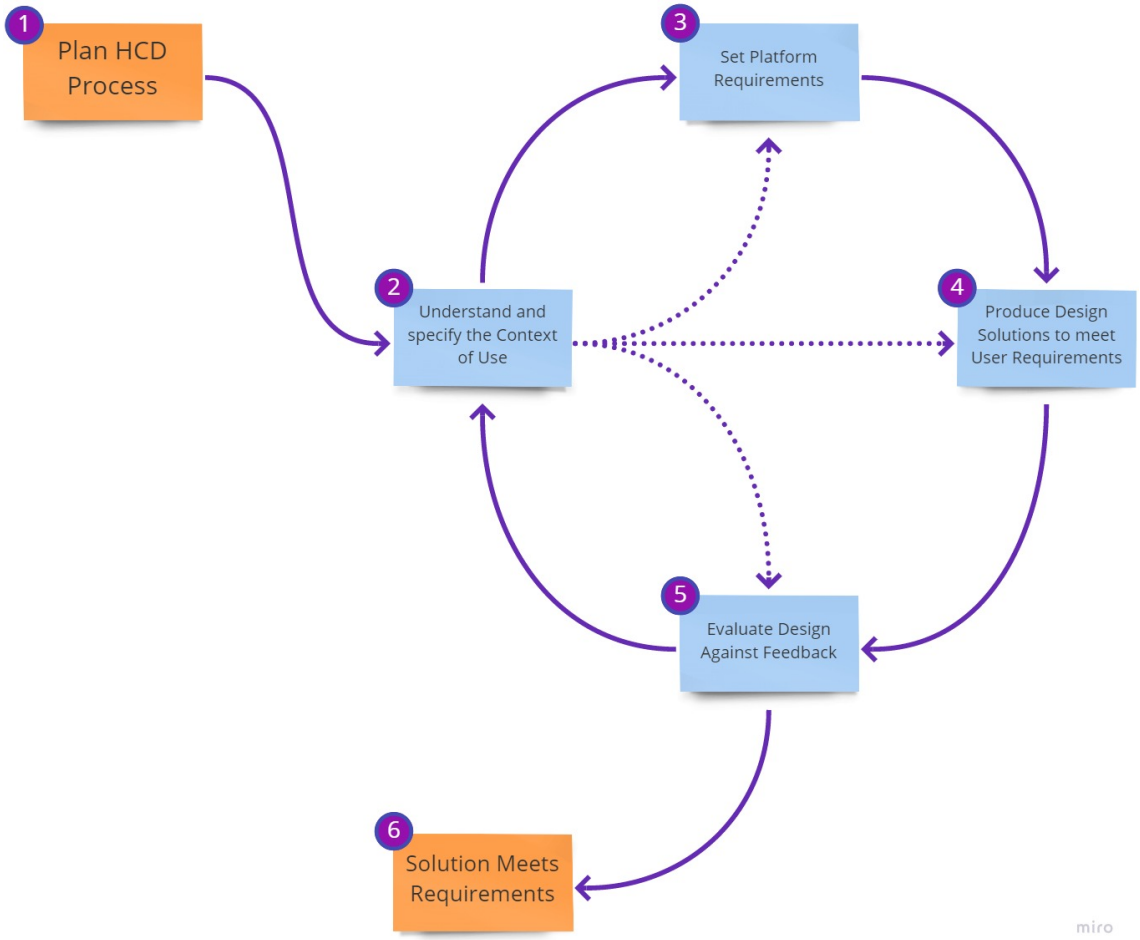


Figure 3.1: Human Centered Design Stept created with miro. [61]

3.2 Planning

Initially, we set out to build a peer-to-peer OTC derivatives platform following ISDA CDM standards for ERC-20 tokens.

In order to find product-market fit for OTC derivatives in the DeFi space, we have contacted Vega Protocol, [26] a platform where decentralized derivatives can be created and traded, as well as Hedgy, [24] an OTC derivatives platform built with Bitcoin Script, following ISDA standards. [25]

As explained in the Background sections, the ISDA CDM has a Product Model to represent the derivative or the nested underlying assets.

The available types of Products are the following: [21]

- **Contractual Product:** There are multiple types of Contractual Products which can be used in ISDA CDM.
  - **Interest rate derivatives:** such as Interest Rate Swaps, Swaptions, OTC Options on Bonds, etc.
  - **Credit derivatives:** such as Credit Default Swaps or Options on Credit Default Swaps.
  - **Equity derivatives:** Equity Swaps.
  - **Options:** Any OTC Option.

- **Loan**
- **Security**
- **Foreign Exchange (FX)**

It is important to re-iterate this because the possible derivatives can be split to address two kinds of markets in regular finance:

- **The Equity Market** which can be thought of as the corresponding market for ERC-20 tokens.

  In the Equity Market, derivatives are regularly traded through exchanges, since there is less risk for the parties involved because the market is more liquid and the participants are well collateralize.

  The same can be observed about derivatives of ERC-20 tokens or other blockchain solutions. As an example, Vega Protocol has the ability to enable an OTC desk (albeit limited in scope) for its users. However, it considers the approach of having peer-to-peer OTC trades inferior to a liquid market for derivatives.

  As discussed in the Background section there are other DeFi solutions that provide DEXs for trading derivatives.

  A peer-to-peer OTC solution could be addressing two main use cases.

  First of all, it would could be applicable to users who need access to more exotic financial instruments. However, this market would be limited and very niche. If a certain exotic derivative would become more mainstream, it would be placed on an exchange and it would have a liquid market.

  Second of all, it could be useful for users that want to trade extremely large amounts, more than is available on exchanges. However, a peer-to-peer solution would present too much risk for the participants and very complex regulatory measures would be needed. Moreover, most exchanges already have an OTC desk to address this scenario.

- **The Debt Market** is fundamentally different in DeFi in comparison to regular finance.

  Loans can be accessed instantaneously, interest rates can easily be switched from variable to fixed and liquidity pools offer interest to loan providers.

  Therefore, derivatives such as Interest Rate Derivatives or Credit Derivatives need a different approach to the one implemented in the ISDA CDM in order to be applied to DeFi.

However, one market where loans or derivatives has not been applied yet is the market for Non-Fungible Tokens.

Following this initial market research, we have focused on the characteristics of NFTs and their use cases and we set out to build new financial instruments for ERC-721 tokens.

3.3 Low-fidelity borrowing and derivatives prototype

This was the first iteration of HCD and the initial step was to understand the context of use and the user requirements.

We started with defining our target users. They represent a very specific demographic, having a certain background on ERC-721 tokens and are expected to have an understanding of blockchain solutions.

There are many different types of users that we address such as:

- Decentralized VR world users.
- Decentralized games users.
- Decentralized art holders.
- Other NFT holders who need access to liquidity.
- DeFi users who want to earn interest for their ERC-20 tokens.

To conceptualize all the expectations and define the ideal user we have created realistic User Personas, which represent UX tools used for creating descriptions of target users. This enabled us to compare their existing user journeys when interacting with DeFi products or NFTs and what changes a new solution could bring.

The Personas were created as a result of the background research and have guided the decisions about the features and technologies used for interacting with the platform.

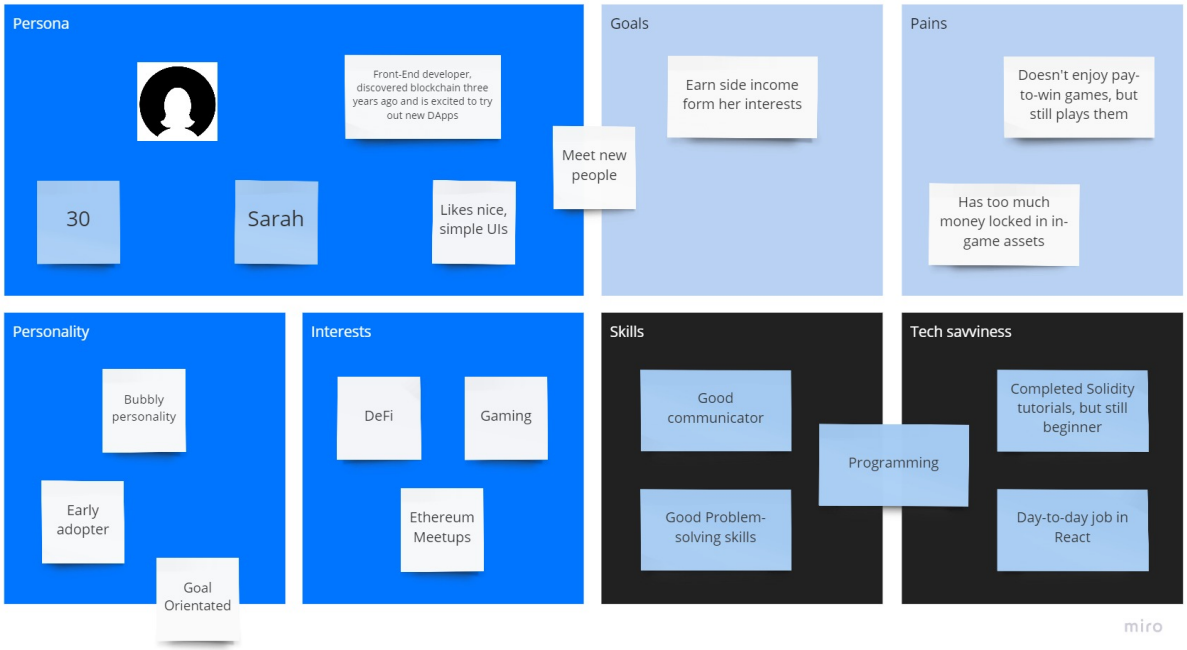


Figure 3.2: Example of User Persona.

In Figure 3.2 we have an example of a user persona we have used.

At this stage, since we have defined the market that we are addressing, we are able to introduce the components of the first iteration of HCD.

- **Context of use**

  In Figure 3.3 we have the different environments that user is accessing the platform from.

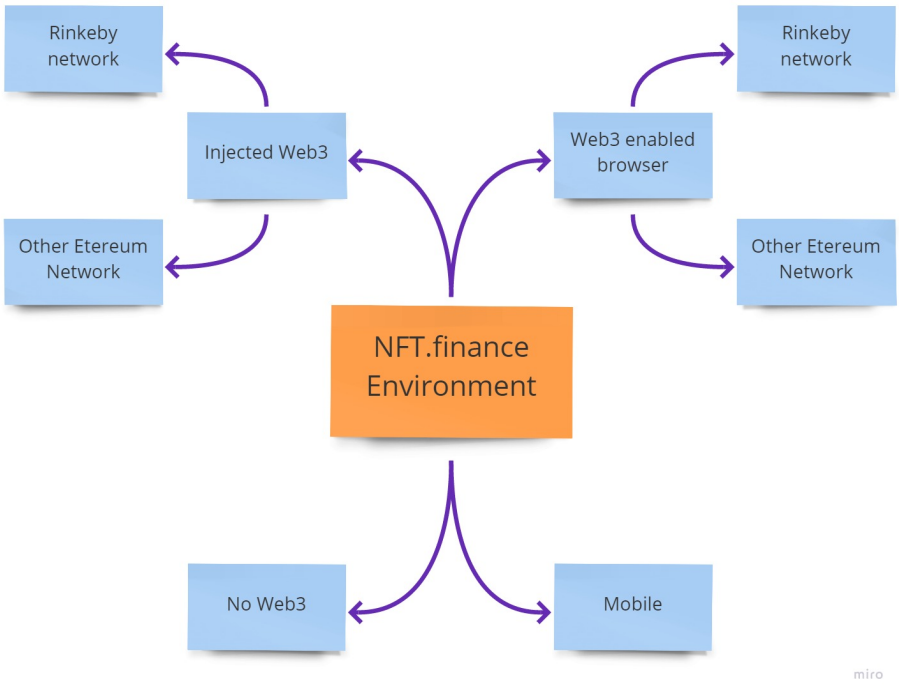


Figure 3.3: Context of use diagram.

  Should the user, access the platform with a Web3 enabling browser extension, such as MetaMask, which injects the web3.js API, his network will also be checked. In case he is on any other Ethereum test network than Rinkeby or on Mainnet, he will be prompted with a warning.

  A similar workflow will be expected if the user is accessing from a Web3 enabled browser.

  On the other hand, if the user does not have Web3 enabled, the website will not load and will show a warning.

  The UI will not be mobile friendly in the begging, however, this requirement might change.

- **Platform requirements** The platform requirements are all the possible tasks available for the users.

  In our case the initial requirements for the first iteration were to develop a basic prototype for the following features:

  – Locking NFTs as collateral to get access to loans.
  – Derivatives for NFTs, in order to minimize price fluctuations.

- **Design solution**

  The initial solution was low-fidelity prototype, which only showcased how the front-end for the loans would work.

To evaluate it, we decided to showcase it by presenting a walk-through to mentors from HackMoney. The entire discussion has been recorded and is available on YouTube. [62]

3.3.1 High-fidelity borrowing prototype

In second iteration, after having received feedback, we have changed the core platform requirements and the tasks users are expected to be able to perform using our solution.

- **Platform requirements**

  Most of the changes were made to the platform requirements, leaving the feature of OTC derivatives for NFTs as an experimental extension to the project and developing functionalities for leasing NFTs and getting loans with NFTs as collateral as shown in Figure 3.4.

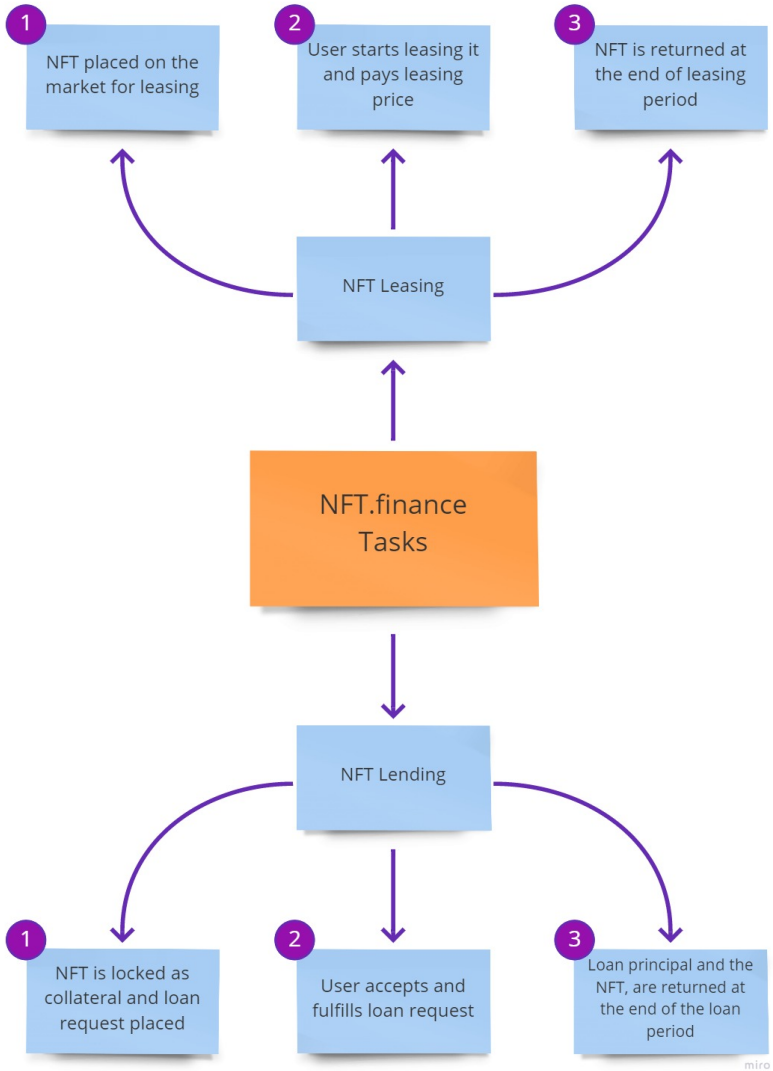


Figure 3.4: Platform requirements.

The requirements were also updated with all the inputs for creating a new lease offer or loan requests.

- **Design solutions**

  In this iteration, the feature of leasing NFTs was connected to the corresponding smart contract and a fully operational prototype was available for testing.

To receive feedback, it was again showcased to mentors from HackMoney and UX experts from Consensys [30] which gave us insights for the next steps.

The recording of the discussion and the feedback received is also available on YouTube. [63]

The main points addressed were:

- The platform can be used for shorting NFTs when borrowing the NFT at a certain price and speculating that its market value will increase in the future. By choosing not to return the NFT, if its price increases, and selling it at a later point, the user can earn a profit.

  This behaviour would not be encouraged on the platform, however there could be benefit in having OTC forward contracts which could lead to less volatility in the prices of NFTs.

- Being a novel use case, it has to gain the trust of the users and make the platform a safe peer-to-peer marketplace.

  Adding a "credit score" for users who post loan requests could be a good solution to incentivize building trust on the platform.

- More feedback should be provided to the users who initiate transactions. They should be informed of their status, without relying on their wallet to send notifications.

- A way to enable chat between addresses and receive messages about offers and requests would make the communication more transparent.

This is the final iteration discussed in the Design chapter. The third and last iteration presented in this report is part of the Evaluation chapter and involves DeFi and DApp users, once the implementation of all the Design Research conducted in this chapter is finished.

Chapter 4

Implementation

In this section we will discuss in detail all the components of the application and the decisions that were taken in the development process of the solution.

The core functionality of our DApp is to enable users to leverage NFT tokens on the Ethereum blockchain.

NFT.finance is a peer-to-peer platform that enables owners of NFTs to enrich their experience and creates a new market in decentralized finance, specific to NFTs.

As explained in the previous chapter, we have taken an iterative approach with a user-centered design in mind, always adapting the requirements to the feedback received.

The platform presents a familiar experience to blockchain and DeFi users, using popular tools and following industry standards.

The two main features of the platform are:

- **Leasing:** Gives users the ability to offer their NFTs for leasing on the market or subsequently lease available NFTs, themselves.
- **Lending:** Enables users to use their NFTs as collateral and to request loans in order to get access to liquidity without selling their asset.

This chapter will be split into 3 main sections and each will follow the development from different perspectives:

- **Smart Contracts:** This section will expand on the core smart contracts that were written to enable the described interaction between users and their NFTs.
- **DApp Interactions:** Facilitates the interactions with the smart contracts and gives users an abstraction to the on-chain data.
- **User Experience:** Because our application was built with the purpose of being released and marketable, we had to focus on researching user needs in order to reach their expectations and build trust.

4.1 Smart Contracts

As discussed in the background the EVM is capable of executing contract byte-code and is a quasi-Turing complete machine.

The smart contracts that are executed by the EVM for our application were written using Solidity, the official programming language used to modify the Ethereum state. [31]

For testing and user feedback purposes the smart contracts were deployed on an Ethereum testnet. This mimics a deployment on the main Ethereum network and allows us to measure the performance of the application and test its features.

We have used the ERC-721 standard for interacting with NFT tokens on the Ethereum blockchain and we expect any ERC-721 tokens to be compatible with our application.

4.1.1 ERC-721 standard and OpenZeppelin

We have covered what the ERC-721 standard means in the background section of the report.

In this subsection we build on those explanations and discuss how ERC-721 tokens are implemented, as well as our usage of this type of token.

ERC-721 is an open standard for smart contracts that is implemented by Non-Fungible Tokens (NFTs). Each ERC-721 token is a unique, scarce item and its interface specifies what functions should be implemented to create, interact and manage NFT tokens.

They have different characteristics and each has a unique `tokenId` that identifies it in the list of NFTs created by the ERC-721 smart contract which was used for minting. [32]

For the project we are using the OpenZeppelin contracts library. They have interfaces for ERC-721 smart contracts and we will focus on two of them: `IERC721` and `IERC721Receiver`. [34]

`IERC721` is an interface with function definitions for building ERC-721 compliant smart contracts and we will use it when interacting with the addresses of the smart contracts of the tokens that we will be receiving.

ERC-721 tokens have an owner address and a list of approved addresses that can initiate transfers. This is relevant in our case because we need our smart contracts to be able to transfer, receive and hold those tokens.

We will be calling the approve function pointing to the address of our smart contract to become approved by the ERC-721 token smart contract.

In case we need to transfer it we will call the safe transfer sending as parameters the `to` and `from` address, as well as the `id` of the NFT.

In our case the address that the token will be sent to is a smart contract. Therefore, the receiving smart contract has to implement the `IERC721Receiver` interface.

`IERC721Receiver` will be implemented by our smart contracts for security reasons and will indicate that we support handling of ERC-721 tokens, reassuring users that their assets will not be lost by being sent to an non-compliant smart contract.

The function that we need to implement is `onERC721Received` which has to return a selector to acknowledge the receipt of the token, as shown in Listing 4.1.

The selector is will acknowledge that our smart contract can handle receipt of ERC-721 tokens and is equivalent to the constant value `0x150b7a02` or the value of `bytes4(keccak256("onERC721Received(address,address,uint256,bytes)"))`.

```
1 function onERC721Received(address operator, address from,
```

```
                                uint256 tokenId, bytes memory data)
                                    public override returns (bytes4) {

    return this.onERC721Received.selector;
}
```

Listing 4.1: Implementation to become compliant receiver of ERC-721 tokens

4.1.2 Leasing smart contract

The first contract that we will go into detail about is the leasing smart contract. It will be used for the functionality of creating a new lease offer and storing it, as well as defining the mechanics of how the lease works.

All the lease offers are stored in a mapping from `LeaseID` to `LeaseOffer`.

The EVM has stack, memory and storage to hold variables for the smart contract and the storage is the most expensive. We have tried to keep the usage of storage to a minimum, however being the only location where the data is persistent we have chosen to keep the mapping of lease offers in this location.

Each lease offer is a `struct` which contains the following components:

- `uint leaseID`: used to identify and query for a specified lease offer in the mapping.
- `address payable lessor`: the address of the owner of the NFT who leases it to the `lessee`.
- `address payable lessee`
- `address smartContractAddressOfNFT`: needs to be stored in order to call the ERC-721 functions of the tokens through the `IERC721` interface using the `tokenIdNFT` to reference the exact NFT.
- `uint tokenIdNFT`
- `uint collateralAmount`: the amount guaranteed by the lessee in case they fail to return the NFT.
- `uint leasePrice`: the amount that has to be paid by the lessee to the lessor representing the rental payment.
- `uint leasePeriod`: the unix equivalent to the maximum duration that the asset can be leased for.
- `uint endLeaseTimeStamp`: the sum of the `leasePeriod` and the current Epoch time at the moment a lessee accepts the offer.
- `Status status`: an `enum` that represents the 4 possible states of a lease offer.

The States of the Offer are:

- **Pending:** The Lease Offer is on the market but no lessee has accepted it.

- **Active:** The Lease Offer has a lessee borrowing the NFT and it has not been ended.

- **Cancelled:** The Lease Offer has been cancelled by the lessor before being activated and is not available on the market, therefore the NFT has been transferred back.

- **Ended:** The Lease Offer has reached an agreed ending. The lessor has received his NFT or the `endLeaseTimeStamp` has passed and he has claimed the collateral amount.

The main functions of the leasing smart contract are `createLeaseOffer`, `acceptLeaseOffer`, `endLeaseOffer` and `cancelLeaseOffer`.

- `createLeaseOffer` is called by users who want to lease out their NFT. They must have approved the address of the leasing smart contract to begin the transfer of the NFT.

  This has to be done before calling `createLeaseOffer` and is a requirement that is checked by calling `getApproved` on the smart contract address of the NFT.

  The other requirements of this functions are that all the parameters for creating a new `LeaseOffer` are non-empty, except for the lessee and the end timestamp of the lease.

  The NFT is transfered from its owner (in this case the message sender who triggered the function call) to the address of the smart contract.

  Since the target address is a smart contract, as we have explained in the previous section, we will be using `safeTransferFrom`, following the implementation standards laid out by OpenZeppelin. [34]

  Finally, we have create a new `LeaseOffer` in storage and appended it at to end of the `allLeaseOffers` mapping, with the parameters received.

  The status is also set to `Status.PENDING` to highlight that the offer is available on the market and the NFT ready to be leased.

- `acceptLeaseOffer` is called by a potential lessee given the `leaseID` as a parameter. The requirements for the function caller are the following:

  - `leaseID` exists.
  - the status of the lease offer is `Status.PENDING`.
  - the address of the lessee is different from the address of the lessor.
  - the value of Wei sent in the function call is greater than or equal to the total sum required to lease the NFT, which is the collateral amount plus the lease price set by the lessor.

    The amounts specified are in Wei, which is the smallest denomination of ether.

If the requirements are all met, the message sender, i.e. the function caller, transfers the sum required to lease to the smart contract. The total is then split and the lease price is sent to the lessor and while the collateral is kept at the smart contract address.

The NFT is sent to the lessee, who can now use it. Unlike a regular lease contract where the lessor stays as owner throughout the lease period, this would not be possible with most ERC-721 tokens because only the owner of the NFT can control and use it. More will be discussed on this topic in the challenges subsection.

Finally, the status of the offer is set to Active to indicate that the NFT is being leased and other users do not have access to it.

The lessee is set to the message sender address and the end of lease timestamp updated with the current timestamp plus the specified maximum lease period.

- `endLeaseOffer` can be called by the lessee as well as the lessor based on certain circumstances. The requirements for calling the function are the following:
  - `leaseID` exists.
  - The status of the lease offer is `Status.ACTIVE`, therefore the lease has started and has not reached an agreed lease ending.
  - The message sender is the lessee and he can call this function to return the NFT before or after the end of lease timestamp has been reached, as long as the previous requirements are met.

    Alternatively, the message sender can also be the lessor but he can only call this function after the end of lease timestamp has passed.
  - If the message sender is the lessee it is also required that the lease smart contract is again approved to receive the NFT.

  If all the requirements are met, there are two corresponding scenarios:

  - **The caller is the lessee:** the NFT will be send form the lessee back to the lessor by calling `safeTransferFrom`.

    The collateral is transferred back to the lessee because he has followed the contract terms for the lease.
  - **The caller is the lessor:** as mentioned the caller can be the lessor only in the case that the lessee has not returned the NFT before the end of lease timestamp has been reached and has followed the contract terms for the lease.

    He is penalized and the lessor can ask to receive the collateral amount kept in the lease smart contract.

  Finally, the status is updated with `Status.ENDED` to indicate that the lease offer has reached an agreed predefined ending.

- `cancelLeasingOffer` the final core functionality is for the lessor to take the offer back from the market before a lessee has started a new lease with it.

  The requirements are the following:

- `leaseID` exists.
- The status of the lease offer is `Status.PENDING`, therefore the lease has not started.
- The message sender is the lessor.

Finally, a `safeTransformFrom` is initiated to transfer the NFT back to the lessor and the status is set to `Status.CANCELLED`.

This status has been chosen for lessors to have a history of their previous offers, in case they want to re-list the NFT on the platform.

Should any of the requirements not be met when calling the functions, relevant messages are returned to the users.

The contract also stores who the address that has created it is. This address is referred to as manager of the contract, who has the ability to pause the functions of the leasing smart contract. This decision has been made in the unlikely case that any security issues emerge.

The leasing contract creates a new type of NFT ownership that enables users to lease ERC-721 tokens on the short-term. Additionally it creates a new type of side income and makes the ownership experience even more valuable.

The ERC-721 use cases will be expanded further in the next subsection.

4.1.3 Loans smart contract

The second smart contract that we will cover enables users to get a loan in Ether, using their NFT as collateral. It is similar to the previous contract, albeit with different details to address the use case of lending instead of leasing.

It follows the same decentralized experience for users, creating a peer-to-peer market for loans, where the loan underwriter can be any person who wants to lend out their Ether. In exchange, they will receive an interest amount from the person requesting the loan.

In case of default, they would receive the NFT which is the collateral that covers the loss of the principal amount.

All the loan requests, just like the lease offers are stored in a mapping from `LoanID` to `LoanRequest`.

Following the example set by lease offers, loan requests are also a `struct` and they have the following components

- `uint loanID`
- `address payable lender`
- `address payable borrower`
- `address smartContractAddressOfNFT`
- `uint tokenIdNFT`
- `uint loanAmount`
- `uint interestAmount`

- `uint singlePeriodTime`
- `uint maximumInterestPeriods`
- `uint endLoanTimeStamp`
- `Status status`

The lender will the user who lends Ether and provides liquidity for the borrower.

The `loanID`, `smartContractAddressOfNFT` and `tokenIdNFT` are specified for a similar purpose, as described before. Both the lender and borrower are payable addresses, meaning they can receive and send Ether.

Each loan contract has an NFT as collateral. This is possible because NFTs have market value and can be traded, auctioned and individually priced based on their rarity, properties and usefulness.

The borrower receives a `loanAmount` in Ether from the lender and has to pay an agreed upon `interestAmount` for each `singlePeriodTime` that he borrows the Ether for.

The principal amount can be borrowed for a total number of consecutive periods equal to `maximumInterestPeriods`.

The `endLoanTimeStamp` represents the ending timestamp when the principal amount has to be returned. This can be extended with `singlePeriodTime` additional time.

Similarly to the leasing smart contract the Status can be the following:

- **Pending:** A Loan Request is on the market and a lender has to accept it and underwrite the loan.
- **Active:** A Loan Request is active and a borrower has receive Ether from a lender, having his ERC-721 token locked as collateral for the loan.
- **Cancelled:** The Loan Request has been cancelled by the borrower and removed from the market. The borrower has also received his NFT collateral back.
- **Ended:** The Loan Request has reached an agreed upon ending.

  The principal amount has been received by the lender.
- **Defaulted:** The borrower has defaulted and was unable to repay the principal amount.

  Therefore, the loan is marked as defaulted. This is useful to investigate the trading history of the participants in the lending market.

The main functions of the loans smart contract are `createLoanRequest`, `acceptLoanRequest`, `extendLoanRequest`, `endLoanRequest` and `cancelLoanRequest`.

- `createLoanRequest` is called to create a new loan request by the borrower.

  Before doing so, they have selected their NFT of choice and approved the loans smart contract to transfer the ERC-721 token from and to store it as collateral.

This is checked by calling `getApproved` on the smart contract address of the NFT.

Another requirement is for the parameters to be non-empty, except for `borrower`, `endLoanTimeStamp` and `status`.

Next, the ERC-721 token is transfered to the smart contract using `safeTransferFrom` as previously done.

A new `LoanRequest` is created in storage and added to the mapping of all loan requests.

The status is initialized with `Status.PENDING`.

- `acceptLoanRequest` is called by a lender when he is interested in underwriting a certain loan request. The requirements are the following:
  - `loanID` exists.
  - The status of the loan request is `Status.PENDING`.
  - The address of the lender is different from the address of the borrower.
  - The value of Wei sent to the function is equal to the total sum that needs to be lent to the borrower minus the first interest payment.

    This guarantees that the borrower has paid the first installment of the loan to the lender and has the rights to hold the principal amount for the first period of the loan.

  If all the conditions are met, the NFT is received by the loans smart contract and held as collateral. The Ether can be used, as described, by the borrower.

  The status of the loan is set to Active and indicates that the loan has started.

  The lender is set to the address of the message sender.

  The `endLoanTimeStamp` will be set to the current timestamp plus the specified time for a single period of the loan.

- `extendLoanRequest` is called by the borrower when he wishes to extend the ending period of the loan. The requirements of the function are the following:
  - `loanID` exists.
  - The status of the loan request is Status.ACTIVE.
  - The address of the message caller is the borrower.
  - The `maximumInterestPeriods` is greater than zero.
  - The value of Wei sent to the contract is equal to the `interestAmount`.

  The interest amount will be sent to the lender because the borrower has decided that he needs the principal amount for another period of the loan.

  The maximum interest periods is decreased by one and the end loan timestamp is set to the previous value plus the specified time for a single period of the loan.

- `endLoanRequest` is called by the borrower when he decides that he no longer needs the borrowed sum of Ether and wants to return it.

  Alternatively, it can also be called by the lender if the `endLoanTimeStamp` has passed and the collateral has not returned.

  The requirements are the following:

  - `loanID` exists.
  - The status of the loan request is `Status.ACTIVE`.
  - The message sender is the borrower and he intends to return the principal amount back before or after the end of loan timestamp, providing that the previous requirements are met.

    Alternatively, if the message sender is the lender, the end of loan timestamp has passed and the collateral has not been returned.

    He can claim the collateral (the ERC-721 token in this case).

  If all the requirements are met, the two cases of execution are the following:

  - **The caller is the borrower:** he returns the principal amount and will receive the NFT posted as collateral.

    The status of the request is set to `Status.ENDED` to indicate a finished loan where both parties have honoured their agreement.
  - **The caller is the lender:** he can claim the collateral and use it to cover the losses of not receiving the principal amount.

    This is an undesirable event and the loan will receive `Status.DEFAULTED` and remain stored in this manner.

    This will hold the borrower accountable on future loans and will influence his access to liquidity based on his history.

- `cancelLoanRequest` is called by the borrower when he decides to take his loan request off the market.

  The requirements are the following:

  - `loanID` exists.
  - the status of the loan request is `Status.PENDING`.
  - the message sender is the borrower.

  A `safeTransferFrom` is initiated to transfer the NFT back to the borrower and the status is set to `Status.CANCELLED`.

Similar to the previous contact a manager address is stored, being the address that has initialized the contract and has the ability to pause the functionality and minimize the impact of any unwanted behavior, in case of an attack.

This contract is a useful addition to the previous one and further enhances the utility of ERC-721 tokens by leveraging their market value.

4.1.4 Deployment and testing

The contracts were initially developed and compiled using the Remix IDE. [35]

We have found the IDE to be flexible and powerful enough for our needs and we used the latest version of solidity, which is 0.6.8, as advised in the docs, to compile the smart contracts. [36]

First the contracts were deployed to a local test chain running in Ganache, which enabled us to test how the transactions worked and debug any issues. [37]

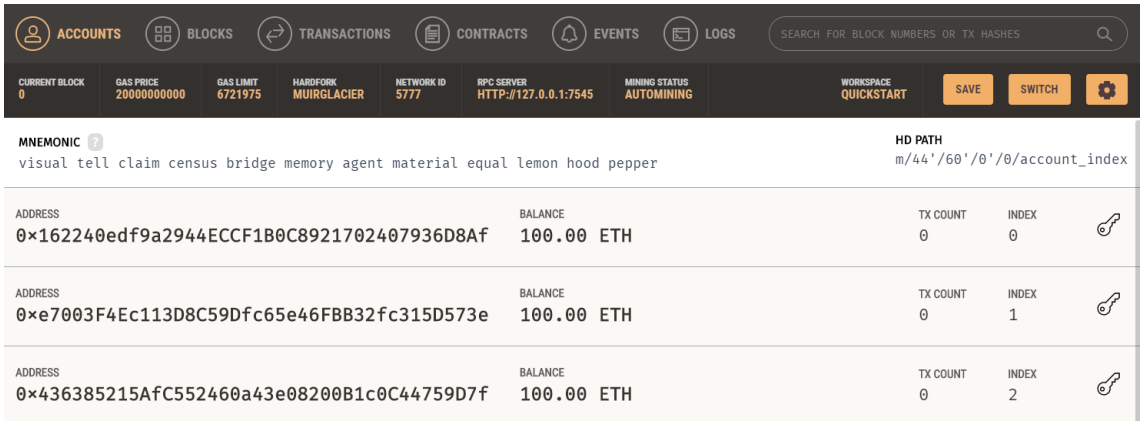


Figure 4.1: Some of the addresses of local test chain running in Ganache.

We connected the MetaMask wallet, which is a non-custodial wallet, [38] to one of the accounts in Ganache and using Remix we switched to the injected Web3 environment to deploy the smart contract to the local test chain.

However, we were not yet able to test the smart contracts since there were no ERC-721 token in local test network, so we decided to deploy our own "Basic ERC-721" token.

We used the OpenZeppelin `ERC721Full` smart contract to inherit all the functions of an ERC-721 token and deployed it, as it can be observed in Figure 4.2.

Doing so enabled us to have our own ERC-721 tokens and `mint` new tokens, `approve` addresses and use `safeTransferFrom` to interact with our smart contracts.

After testing the Loans smart contract and the Leasing smart contract we have decided to deploy them to a live testing network of Ethereum.

We had to choose one of the following test networks that are available: [39]

- Ropsten
- Kovan
- Rinkeby
- Görli

They all have different properties, such as different consensus algorithms and were launched by different teams.

We chose to deploy to Rinkeby because it is also the test network of choice for OpenSea. [29]

Figure 4.2: Some of the functions of the Basic ERC-721 token contract in Remix.

In the following section we will be using the OpenSea APIs and they are available only for ERC-721 tokens on the Rinkeby network. [40]

To deploy the smart contracts we needed Ether and we had to request it using the Ether faucet running on the Rinkeby network. [41]

We have again used Remix to deploy to Rinkeby and interact with the smart contracts, now having a wider choice of ERC-721 tokens have we could interact with, which were, this time, purchased from OpenSea.

The functions of smart contracts, deployed in Remix, are available in Appendix A.

4.1.5 Alternative Implementations

There are alternative implementations that we have considered and we will go over them in more detail comparing them with our final implementation.

We will discuss the advantages and disadvantages and we will motivate our choices.

- **Liquidity Pool Lending:** In the DeFi ecosystem liquidity pools are used for offering collateralized and over-collateralized loans. They use automated market making (AMM) algorithms to provide liquidity to markets and set

a deterministic spot price for the assets and interest rates for the loan, as explained in the background. [42]

The NFTs represented by ERC-721 tokens could have been wrapped in ERC-20 tokens, which could then be used for creating a liquidity pool. Users would be able choose their preferred NFT from the liquidity pool and lease it or lend their own NFTs to the pool and earn interest.

However, this would cancel the non-fungible property of the tokens and would not take into account the unique features of each NFT when pricing them using an AMM protocol.

- **Individual smart contracts for each loan:** An alternative implementation that was considered was to deploy a smart contract for each lease offer or loan request.

  This would have ensured that the funds or NFTs for each loan or lease, respectively, would have been held in separate contracts.

  It could have added an extra layer of security in case of potential attacks. Should there be any breaches in the `LoansNFT` or `LeaseNFT` contract, a hacker would be able to drain the contracts.

  However, there would be a higher cost associated with creating each lease offer or loan request and potential users would be discouraged to use the platform.

  The cost for deploying a smart contract would be around 0.04 Ether and our implementation is targeted at providing a simple experience for the user and lowering their on-boarding costs.

  For this reason, we have decided to have two main contracts which hold all the transactions and data, while focusing on making them secure.

- **Single contract for both loans and leases:** Another implementation would have been to create a single smart contract for both use cases of the platform.

  There would have been less overhead in the development of the contracts, however it would have increased the gas costs for the users.

- **No manager or pausable contract:** Having a manager that can pause the creation of new lease offers or loan requests is arguably a decision that goes against the core principles of decentralization.

  However, we consider it to be an important part of the functionality of the project aimed to protect the users in case any issues with the behaviour of the smart contract are spotted.

  Even so, the ability of the manager is limited. He can pause and un-pause only the following functions:

  - `createLeaseOffer`
  - `acceptLeaseOffer`
  - `createLoanRequest`
  - `acceptLoanRequest`

The manager has no access to the funds or the ERC-721 tokens held by the smart contracts. Moreover, if a loan or lease is already active he is unable to pause the function or end it. Similarly, if they are pending, the ability to cancel is not pausable.

- **DAO:** An alternative implementation would have been to create a "decentralized autonomous organization".

  This would have required the development of an ERC-20 token offered in exchange for Ether and used to vote on the platform.

  The holders could have taken part in the decisions regarding the future development of the platform and the smart contracts.

  This would be useful in the long term since it would create a crypto-economy and incentivize users to invest and contribute to the development of the platform.

  However, for the scope of the project it would not add any additional functionality or improve the user experience.

4.2 DApp client

In this section we will discuss the development of the front-end application which enables users to interact with the smart contracts and completes the experience of using this decentralized application (DApp).

We will go over the components and their connections and we will discuss the services and libraries used.

We will not be going over the UI/UX decisions which will be detailed in the User Experience section.

4.2.1 Technologies used

We have decided to build a website for the dApp and we expect our users to have a Web3 enabled browser or to use a browser extension that acts as a Web3 provider (such as MetaMask).

The technologies used for the front-end of the website and to manage the logic of the components are the following:

- **React:** a Javascript library used to update and render the UI. [48]

  It was used in the project to manage and create the Front-End.

- **Redux:** a Javascript library used to store and manage state. [49]

  It was used to hold the data from the smart contract and OpenSea APIs.

- **Web3.js:** a collection of libraries used to manage the connection to Ethereum. [50]

  It was used to connect to our smart contract and the user's wallet.

- **Axios:** a Javascript library that can be used to perform HTTP requests. [51]

  We used it for connecting to the OpenSea APIs.

- **Materialize:** a CSS framework based on Material Design. [52]
- **Rimble UI:** React components that provide a design standard for DApps. [53]

4.2.2 High-level structure of the React project

To understand the structure of the React project we will take a look at the main components, which are discussed below.

- **assets:** holds all the custom CSS properties and any other assets such as constants or images.
- **components:** stores all the layout components such as the pages of the website, the Navigation Bar or different UI cards.
- **services:** manages the Web3 connection and OpenSea API calls.
- **store:** the Redux store with 3 reducers (`accountReducer` , `leaseReducer` , `loansReducer`) and corresponding actions for each.

Next we will discuss how the components interact with each other starting with the Redux store.

4.2.3 Redux store

The whole state tree of the application is available inside the Redux store.

There are three main ideas of working with Redux: [54]

- **Single source of truth:** The global state of the application is held in a single store.
- **State is read-only:** The state is not directly updatable. We have to emit an action to show intent of replacing the state.
- **Changes are made with pure functions:** Reducers are used to take the previous state together with an action and create a new one.

There is a single Redux store and it is defined in `index.js` and passed as props to the `Provider` component to make it available to all the nested components which use the `connect()` function.

The store is defined as shown in Listing 4.2.

```
1 const store = createStore(rootReducer, applyMiddleware(thunk));
```

Listing 4.2: Creating Redux Store

We have chosen to apply a store enhancer called Thunk to our Redux store.

Thunk is a middleware that facilitates the dispatch of async actions to interact with the store. [55]

As an alternative implementation, we have tried making async calls in the components and dispatching the actions when the promises are resolved.

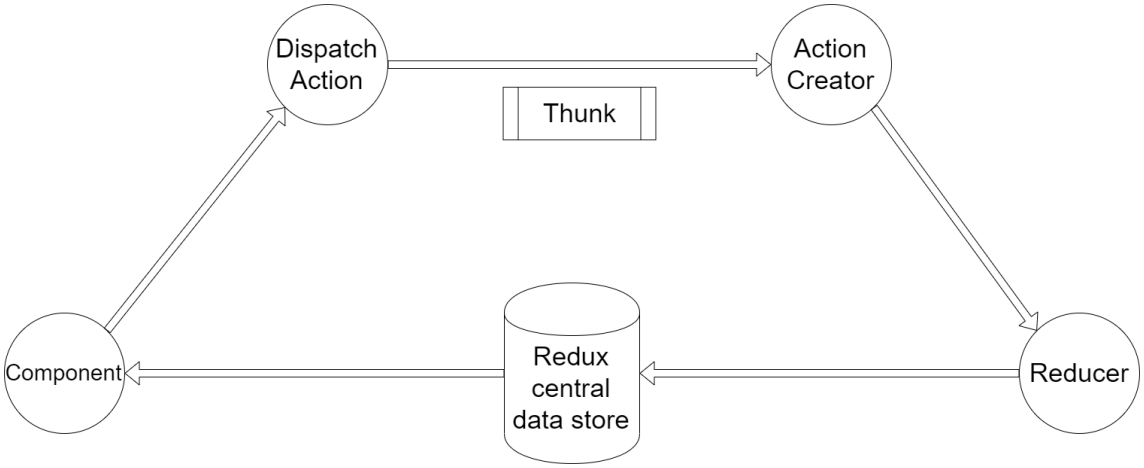


Figure 4.3: Redux overview.

However, by choosing to do so we don't have a true separation of the logic of the application. Moreover, it also introduces overhead and makes updating the project more difficult.

Using Redux, the state updates would be triggered as such:

1. **Redux central data store:** is defined as exemplified above. It is given as parameters the `rootReducer` and the thunk middleware.

2. **Components:** any component of our React project can connect to the store to access the data or dispatch actions.

   We will take the App component as an example to show how this works. We have mapped the state of the Redux store and the dispatch actions to the props of the this component.

```
const mapStateToProps = (state) => {
  return {
    userAddress: state.account.accountAddress.address,
    leaseOffers: state.leasing.leaseOffers
  }
}

const mapDispatchToProps = (dispatch) => {
  return {
    getAccountAddressAction: () =>
                    dispatch(getAccountAddressAction()),
    getAccountAssetAction: (address) =>
                    dispatch(getAccountAssetAction(address)),
    getLeaseOffersAction: (address) =>
                    dispatch(getLeaseOffersAction(address)),
    getLeaseAssetsAction: (leaseOffers) =>
                    dispatch(getLeaseAssetsAction(leaseOffers))
  }
}
```

```
20 export default connect(mapStateToProps, mapDispatchToProps)(App);
```

Listing 4.3: Connecting the App Component to Redux

We have chosen to connect to Redux in this component to manage its state as the application loads.

When the website is first accessed, we call `getAccountAddressAction`. This will trigger the corresponding dispatch action.

If the state of our store is modified, we have subscribed to any changes by mapping state to props.

In this case `this.props.userAddress` will always be updated with `account.accountAddress` from the state of the store, as shown in Listing 4.3.

This will enable easier management of any updates in the data as discussed in the following subsections.

3. **Dispatch Action and Action Creators:** there are three types of actions for our store

   - `accountActions:` which can add the user's address and the assets that he owns to the store

```
1  export const getAccountAddressAction = () => {
2    return (dispatch, getState) => {
3      getWeb3Account().then( account =>
4        dispatch({ type: 'ADD_ACCOUNT_ADDRESS',
5                   userAddress: account })
6      );
7    }
8  }
9
10 export const getAccountAssetAction = (address) => {
11   return (dispatch, getState) => {
12     getAssetsOpensea(address).then(response => {
13         dispatch({ type: 'ADD_ACCOUNT_ASSETS',
14                    accountAssets: response.data.assets })
15       }
16     );
17   }
18 }
```

Listing 4.4: Account Action example

Inside the dispatched actions there are async functions to retrieve the data that we want to update our store with. In this case `getWeb3Account` and `getAssetsOpensea`.

Action Creators are just functions that create actions and dispatch them, given an `action.type` and other action attributes.

- `leaseActions:` as an example `getLeaseOffersAction` creates and dispatches an action of type `ADD_LEASE_OFFERS`. Similarly, `getLeaseAssetsAction` that creates and dispatches an action of type `ADD_LEASE_ASSETS`.

  In addition we also process all the addresses received from web3 and update them to lowercase in order to facilitate string comparisons at a later stage.
- `loanActions:` which dispatch actions regarding the loans in a similar manner to `leaseActions`, but with the appropriate Web3 calls.

It is very important to note that the middleware we have applied, Thunk, is used at this stage in life-cycle of the Redux store.

It enables us to halt the dispatch and wait for the async request to finish before dispatching.

Normally, we would have only returned a dispatch call, however, by using Thunk we can call services such as Web3 or OpenSea before the dispatch, centralizing our service calls and making them easier to be reused in other components, enabling a separation of the logic.

4. **Reducers:** handle the changes to the application state in response to the actions.

   They check the type of action that has been dispatched and create a new state from a copy of the previous state with the updated attributes.

   Our `leaseReducer` handles the `ADD_LEASE_OFFERS` and `ADD_LEASE_ASSETS` actions as shown in Listing 4.5.

```
const initState = {
  leaseOffers: [],
  leaseAssets: []
}

const leaseReducer = (state = initState, action) => {
  if (action.type === 'ADD_LEASE_OFFERS') {
    let newLeaseOffers = action.leaseOffers;
    return {
      leaseOffers: newLeaseOffers,
      leaseAssets: state.leaseAssets
    }
  } else if (action.type === 'ADD_LEASE_ASSETS') {
    let refrshedLeaseAssets = action.leaseAssets;
    return {
      leaseOffers: state.leaseOffers,
      leaseAssets: refrshedLeaseAssets
    }
  }
  return state;
}
```

Listing 4.5: Reducer applied to leasing actions.

Similarly, we also have an `accountReducer` and a `loanReducer`.

However, all the states from our three reducers have to be combined into a single central state for the Redux store of the application, following the guidelines of using Redux.

To achieve this we have combined all the reducers into the `rootReducer` using the `combineReducers` function of Redux. We could have as many reducers as needed, but there has to be a single central store.

4.2.4 OpenSea Services

OpenSea provides access to their APIs for fetching ERC-721 assets. [40]

We have decided to use their API because it would cut overhead and switch the focus to the development of the core functionalities of the NFT.finance platform. Moreover, it has proven to be flexible and reliable enough for all our needs, which will be explained in this section.

It works on the Rinkeby test network and the main Ethereum network.

Since the application is not ready for release at the moment we have decided to use Rinkeby, where our smart contracts have been deployed. Moreover, there are multiple ERC-721 tokens available on Rinkeby for purchase through the OpenSea marketplace using Rinkeby Ether, which is free.

First of all, the APIs were used to retrieve and display the ERC-721 tokens of the user, as well as their properties.

```
export const getAssetsOpensea = async (account) => {
    return axios.get(
        'https://rinkeby-api.opensea.io/api/v1/assets/', {
            params: {
                order_direction: 'desc',
                owner: account,
            }
    });
}
```

Listing 4.6: Example of API call to OpenSea using Axios to retrieve user assets.

The API calls are made using Axios and will return a promise.

For the API to return the user's assets we have to specify the owner in the query parameters.

Finally, this function is exported to be used by the account actions, namely `getAccountAddressAction`.

Another use of the OpenSea APIs is to retrieve the assets for the NFTs of each individual lease offer or loan request.

This request is mapped on all the lease offers and loan requests, using the corresponding `smartContractAddressOfNFT` and `tokenIdNFT`.

Similarly to Listing 4.6, the function is exported and used by `leaseActions` and `loanActions`, respectively.

The API is free of charge and we were given an API key from the OpenSea team to use it.

We are also providing links to the OpenSea website to encourage our users to check out and use their marketplace.

4.2.5 Web3 services

To connect to the `leaseNFT` and `lendNFT` smart contracts we are using Web3.js. It is a library that facilitates the connection to the Ethereum ecosystem. [50]

```
if (window.ethereum) {
    window.web3 = new Web3(window.ethereum);
    window.ethereum.enable();
}
```

Listing 4.7: Basic example of Web3 connection.

First, we check if the browser is Web3 compatible and if `window.ethereum` exists.

`window.ethereum` is the provider object and supports the EIP-1193 standard [57] of Ethereum provider APIs. It is compatible with the native provider of Web3 that the browser of the user has.

Finally, `window.ethereum.enable()` asks for the permission of the user to connect to his wallet.

For all the smart contract calls we use the `web3.eth.Contract` object, specifying the JSON interface of each smart contract, derived from the Application Binary Interface (ABI) of the contract.

We also need to specify the address where the smart contract is deployed to and the address that makes the call.

In Listing 4.8 we have an example of how a contract instance for the `leaseNFT` smart contract can be created.

```
const leaseNFTContract =
    new web3.eth.Contract(contractInterface,
                          CONTRACT_ADDRESS,
                          { from: address });
```

Listing 4.8: Example of contract instance.

In Listing 4.9, we retrieve the list of lease offers and loan requests. In the smart contract they are both public mappings, so they can be easily accessed using their key-value pairs. First we need to call the `totalLeaseOffers` function to return the `totalLeaseOffers` number. The result will be used to retrieve each element of `allLeaseOffers`.

```
export const getAllLeaseOffers = async (address) => {
    const totalLeaseOffers =
```

```
        parseInt(await leaseNFTContract.methods
                                .totalLeaseOffers().call())

    return Promise.all(
        [...Array(totalLeaseOffers + 1).keys()].map(
            id => leaseNFTContract.methods.allLeaseOffers(id).call()
        )
    )
}
```

Listing 4.9: Example of smart contract data retrieval.

The result will be a promise that will be resolved before the action is dispatched by the action creator.

Similarly, we have the function `getAllLoanOffers` for the LoanNFT smart contract.

Before interacting with the NFT of the user we must approve our smart contracts. To do so, we have to connect to the smart contract of the token and also specify the JSON interface of the ABI which was taken for the ERC-721 implementation from OpenZeppelin.

```
export const approveNFT = (erc721Contract, userAddress, tokenIdNFT) => {
    const erc721crt = new web3.eth.Contract(erc721ContractInterface,
                                            erc721Contract,
                                            {from: userAddress});

    erc721crt.methods.approve(CONTRACT_ADDRESS, tokenIdNFT).send()
    .on('transactionHash', (hash) => {
        // transaction is sent
    }).on('receipt', (receipt) => {
        // transaction receipt is available
    }).on('confirmation', (number, confirmation) => {
        // called on each confirmation
    }).on('error', (error) => {
        // transaction has error
    });
}
```

Listing 4.10: Example of smart function call.

For each transaction, we can check their status based on the following events: [50]

- `transactionHash`: this notifies us that the transaction has been sent and we have a transaction hash available.
- `receipt`: this will be fired when the transaction receipt is available and the transaction will be mined.
- `confirmation`: signals for every confirmation up to confirmation number 24.

- `error`: fired in case an error is returned, if the transaction is rejected or if it fails.

The events are useful for informing the user about the status of their transaction. This should be in addition to MetaMask transactions and will be further discussed in the UX section.

Similarly, we are also able to call the functions of our smart contracts to create, cancel and end offers or requests. Moreover, after a number of confirmations, we trigger updates of the state of our Redux store since there could be changes to the ownership of the NFTs or the list of offers and requests.

4.2.6 Managing the state of the Application

Having explained how the Redux store works and how the calls to the services are made, we will go into detail about managing the state of the components in React and how we trigger Redux actions.

In this subsection we will focus on the logic of updating the `App.js`. It is the entry point of the website and we use the life-cycle methods to trigger and react to changes.

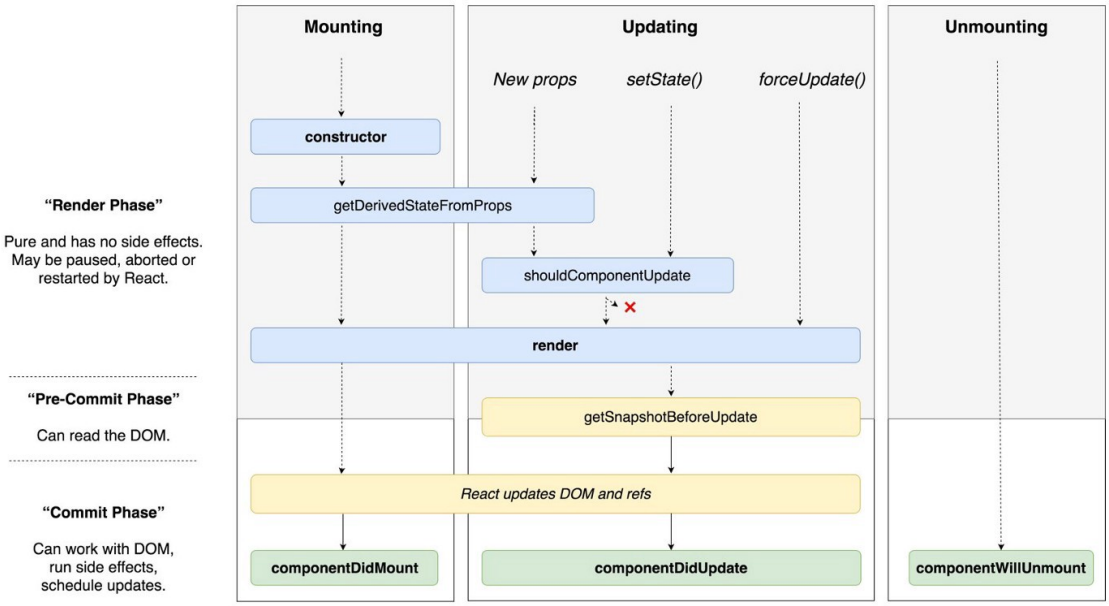


Figure 4.4: React life-cycle methods. [58]

The first life-cycle method used is `componentDidMount` which is called after the component is mounted and rendered. Our homepage is static and and we use this life-cycle method to call the `getAccountAddressAction`.

This will retrieve the address of the user as explained in the previous subsection, update the Redux store and subsequently update all the components that are connected to the Redux store, `App.js` being one of them.

The next life-cycle method we make use of is `componentDidUpdate` which is called after the an update of the state occurs.

It enables us to compare the previous props of `App.js` with the current props and check for any changes. It is critical to check for changes or re-rendering could occur in an infinite loop.

```
componentDidUpdate(prevProps, prevState, snapshot) {
    if (prevProps.userAddress !== this.props.userAddress) {
        // change in user address detected
    }

    if (prevProps.leaseOffers.length !==
            this.props.leaseOffers.length) {
        // change to the lease offers detected
    }

    if (prevProps.loanRequests.length !==
            this.props.loanRequests.length) {
        // change to the loan requests detected
    }
}
```

Listing 4.11: Logic of updating the state of the App component.

In the case of an update to the address of the user we call `getLeaseOffersAction`, `getLoanRequestsAction` and `getAccountAssetAction`. This ensures that any changes made to the user's address (e.g.: changing to a different account) is handled and the appropriate loans and leases are displayed, as well as the assets that the address holds. Moreover, it populates the Redux data store when the user first enters the website.

Subsequently, once the leases and loans are updated, we request their corresponding assets from OpenSea.

4.2.7 Components and Assets

The Assets and the Components enable the user interaction with all the services and smart contracts described up to this point, by rendering the UI.

Inside the Assets we store the following utilities:

- **images:** any images used in the rendering of the UI.
- **custom CSS:** for the CSS of the application we have used Materialize, a CSS framework, however we have also defined our own styles, for the cards and list of offers or requests.
- **constants:** in assets we also store the constants, such as the address of each smart contract and their pre-defined statuses.

The Components, connect to the Redux store, render the UI of the web-pages and react to user inputs. Their structure can be observed in Figure 4.6.

- **App:** this is the root component, first loaded when the website is accessed. The connection of the App component to the Redux store has been detailed in the previous subsection.

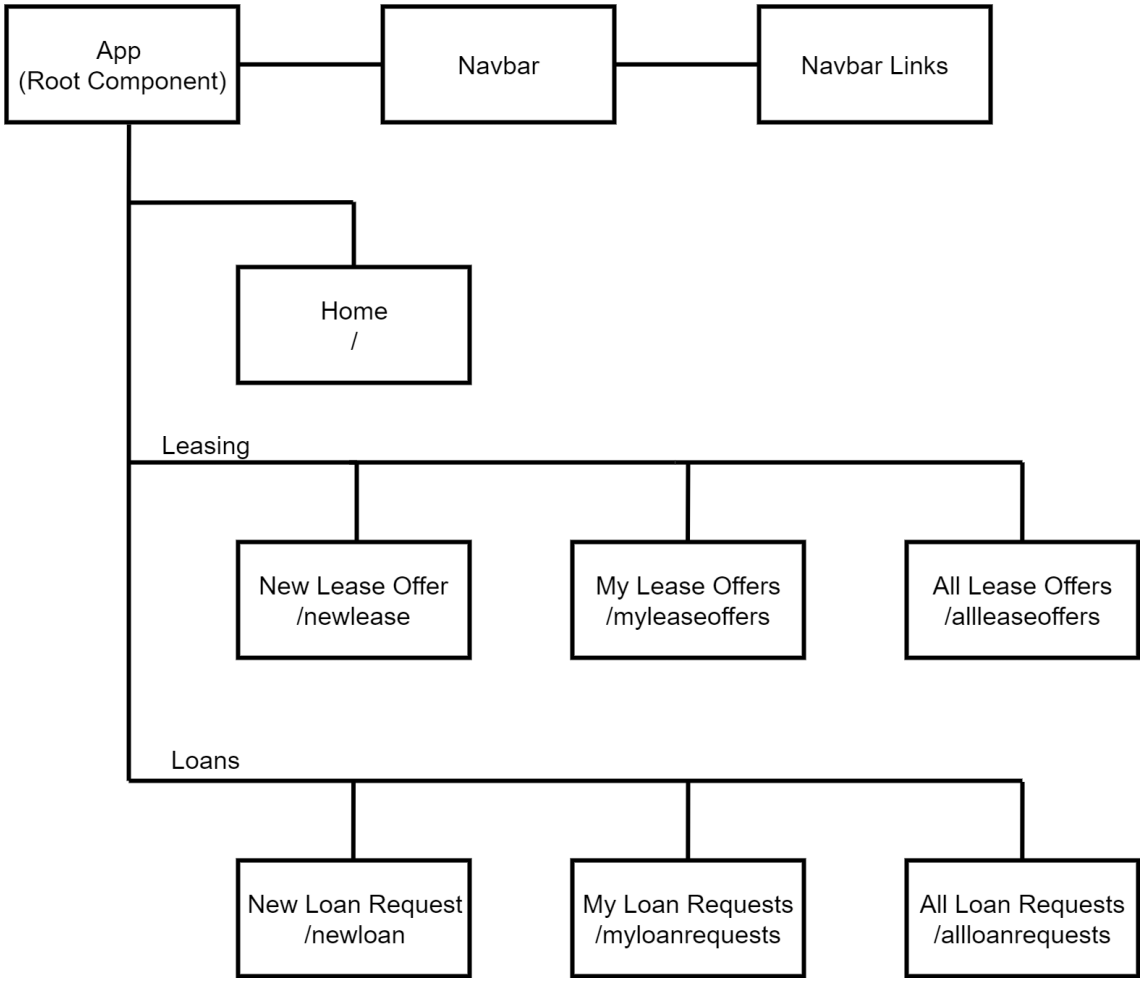


Figure 4.5: React components interactions.

In addition, it also loads the routes, which enable client-side routing, so the user can navigate the website without having to refresh the page, the components being dynamically loaded.

We have specified the path for each route in Figure 4.6, below the name of each component.

- **Navbar and Navbar links:** the Navbar will be nested in every route and displayed on all web-pages. The list of links is available inside Navbar Links, each containing a NavLink to the component that they point to.

- **Home:** the home will be component loaded when the path is empty and the user first accesses the website.

  Being the landing page of the website, we also display a `HomeCard` component to encourage users to read the Docs and guide them to begin using the NFT.finance platform.

- **Leasing:** the Leasing components will complete the logic and display the UI for making use of the Leasing feauture of NFT.finance, showing NFTs, connecting to smart contracts and interacting with them.

– **New Lease Offer:** on this page the user has all his NFTs displayed. He can choose an NFT to lease out and specify the details of his lease offer.
The `NewLeaseOfferPage` component is connected to the React router and it displays the `MyAssetsCards` component which in turn has other components nested. The structure is available in Figure 4.7.

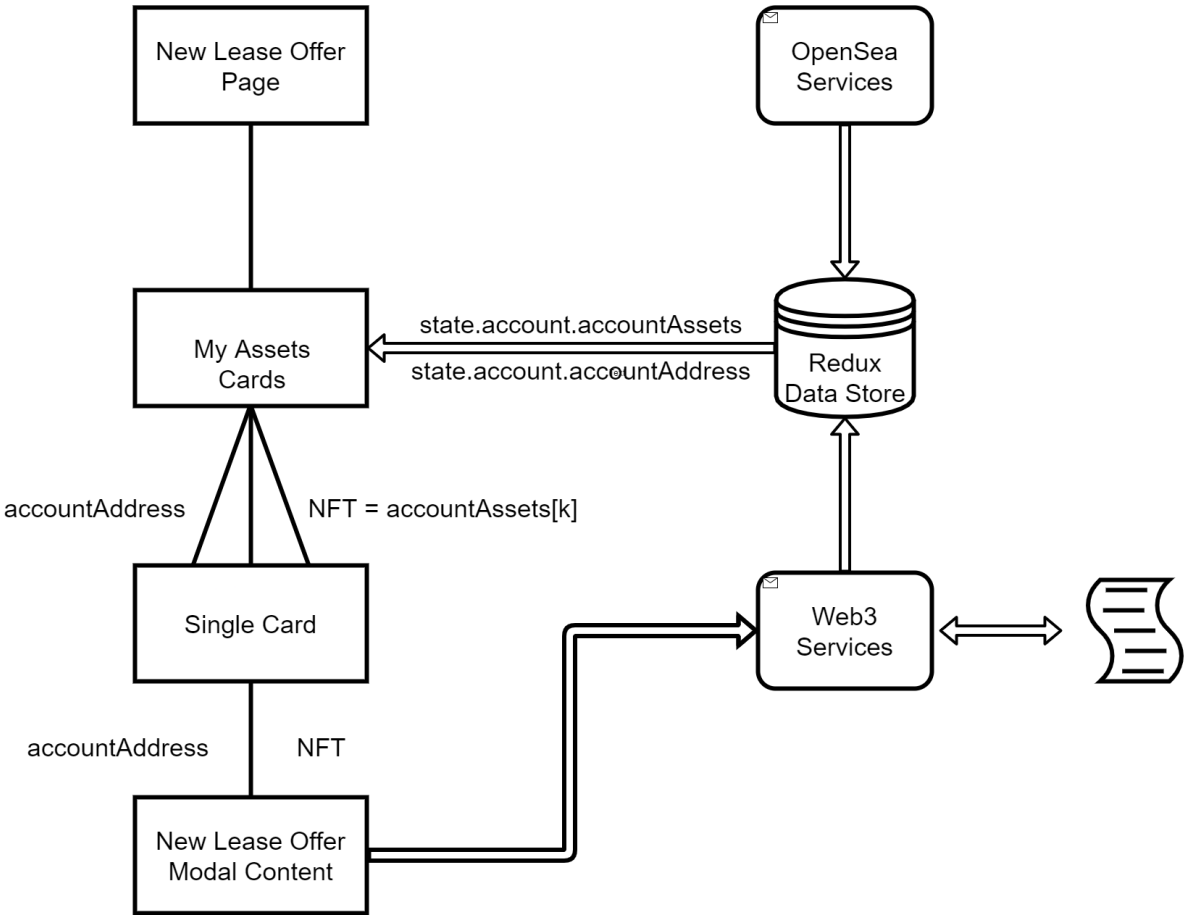


Figure 4.6: New Lease Offer Page components interations.

`MyAssetsCards` component connects to the Redux store and retrieves from the state the account address of the user (updated by the Web3 Services) and the account assets of the user (updated by the OpenSea services).

By mapping over the account assets we have create a list of `SingleCard` components, which receive the account address and the NFT assets as props, namely `accountAddress` and `accountAssets[k]` in Figure 4.7.

The `SingleCard` component is a Materialize card that displays properties of the NFT, such as the name, creator, smart contract address, number of previous sales and more.

The user can leverage their NFT to create a new lease offer. This dynamically creates a Materialize modal with the content loaded from `NewLeaseOfferModalContent`.

The props received by the `NewLeaseOfferModalContent` are again the corresponding NFT asset and account address. A form is available for

the user in which he can input the details of their new lease offer corresponding to the NFT. The fields correspond to some of the parameters of the `createLeaseOffer` function from the `leaseNFT` smart contract.

The user is also expected to approve the transfer of the NFT by calling the `approveNFT` function from Web3 services, through the UI.

Finally, the user can create the new offer. Again, he will call a function from Web3 services, namely `createLeaseOffer`. This will trigger a smart contract function call and on confirmation will update the Redux store, as described in the previous subsection.

– **My Lease Offers and All Lease Offers:** on those two pages users can check their lease offers or all the lease offers, respectively. Additionally, they can filter the lease offers based on their status and interact with them as described in the `leaseNFT` smart contract.

The `MyLeaseOffersPage` component and `AllLeaseOffersPage` component are connected to the React router and their structure can be observed in Figure 4.8.

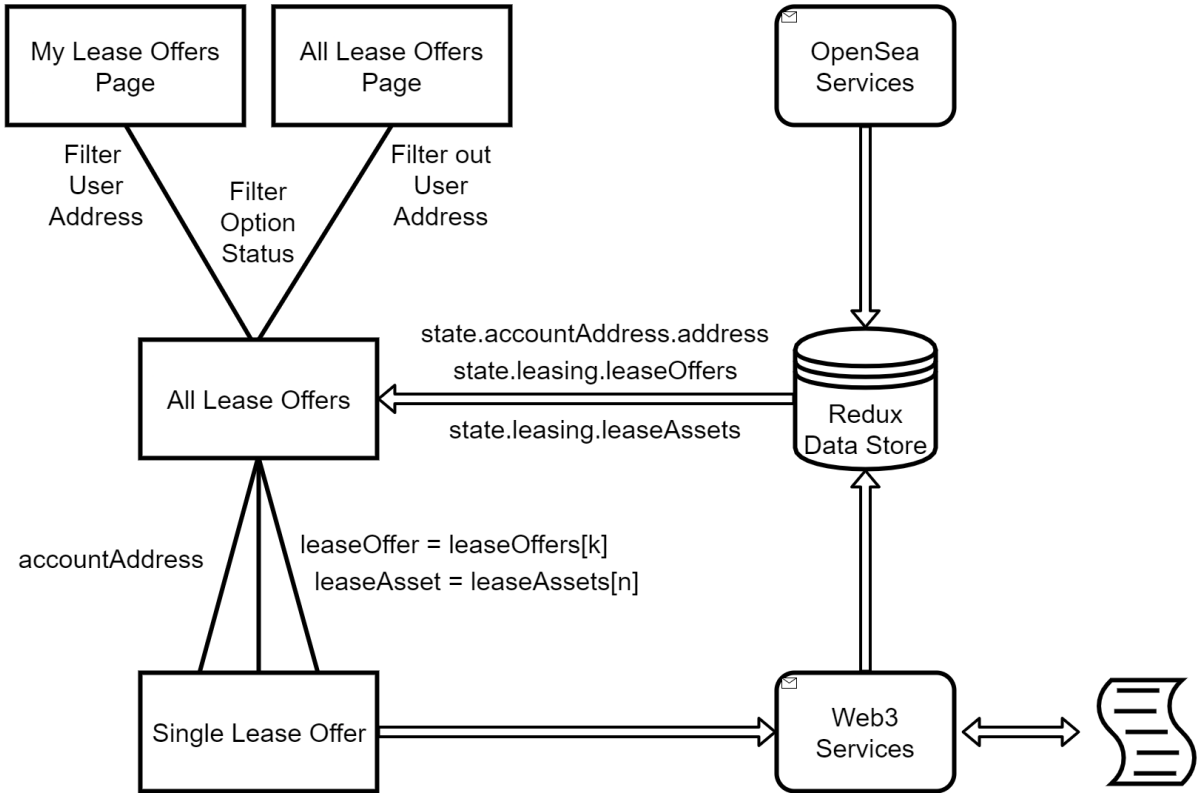


Figure 4.7: My Lease Offers Page and All Lease Offers Page components interactions.

My Lease Offers page and the All Lease Offers page both have options for filtering the offers based on their status. When an option is selected the state of the each page updates accordingly and sends the selected option as props to the `AllLeaseOffers` component.

Moreover, on the My Lease Offers Page we want to only display the offers of the user currently accessing whereas on the All Lease Offers Page we want to display all the offers except those of the user. In order to do this we send another filter accordingly.

The All Lease Offers component is connected to the Redux store. It maps the state from the store to the props in addition to the filters props.

The filters are applied to the lease offers received from Redux which were populated by Web3 Services.

For each lease offer we also access its corresponding NFT asset from the list of `leaseAssets` stored in Redux, populated by OpenSea Services.

Finally, we create a mapping of `SingleLeaseOffers` Components over each lease offer, sending as props the lease offer, the corresponding lease asset and the account address of the user.

Each Single Lease Offer component is a Collapsible Materialize Card which displays all the information of the offer.

Additionally, it enables the user to interact with the offer. It dynamically creates and displays the right Buttons based on the user and the offer. As an example one user does not have buttons to cancel or end another user's offer.

The interactions trigger functions from Web3 services, such as `leaseNFT`, `cancelLease` or `endLease`. They will trigger smart contract calls and update the Redux store on confirmation.

- **Loans:** the Loans components are similar to the Leasing component presented earlier, albeit with adapted functionality to meet the requirements of the `LoanNFT` smart contract.

  The interactions between the components can be deduced from the one presented earlier and the smart contracts. Therefore, there we will not go over the implementation of Loans to avoid repetition.

  However, the Loans are still a core part of the platform and they will be presented in the following sections.

The React implementation of the DApp is crucial for user interactions and provides an intuitive method of communicating with the smart contracts and implicitly the Ethereum network.

Additionally, in order to present a good experience for the user, we have focused on creating a simple and intuitive UI which will be presented in the next section.

4.3 Final UI elements

When the user first accesses the website, the Homepage loads and presents a card that enables access to the documentation and encourages interactions with NFTs.

The UI has simple elements in order to provide a familiar pattern for users.

The Navbar has sub-menus for the two main features: Leasing and Lending as it can be observed in Figure 4.10.

The New Lease Offer page and New Loan Request page both display the NFTs that a user owns, as shown in Figure 4.11. Clicking on any of the cards displays more information about the NFT, as well as a link to OpenSea. We have observed that having access to the history of the NFT is an important aspect of the platform because it is easier to determine what its market value could be.

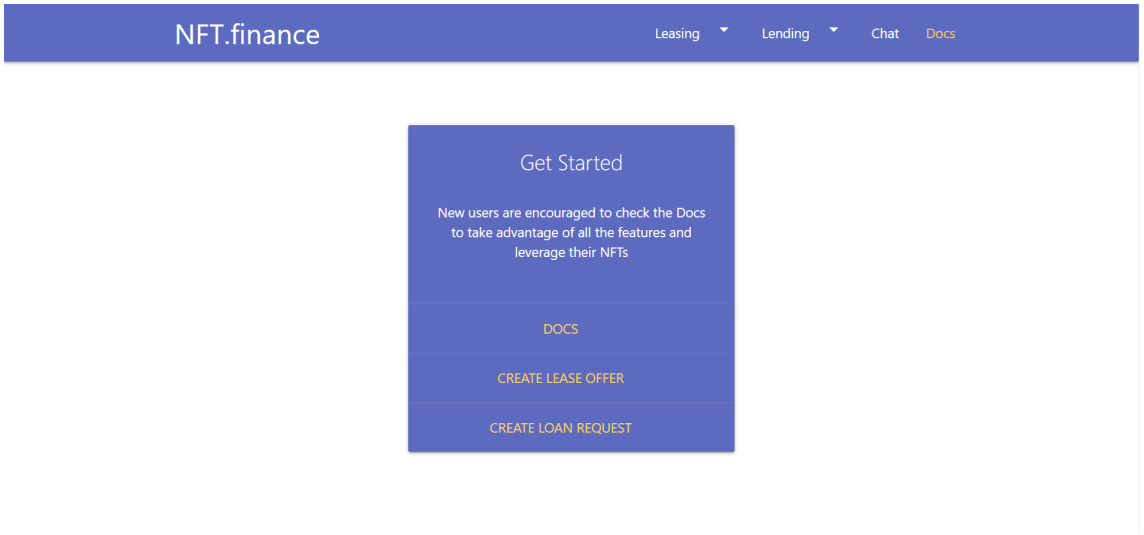
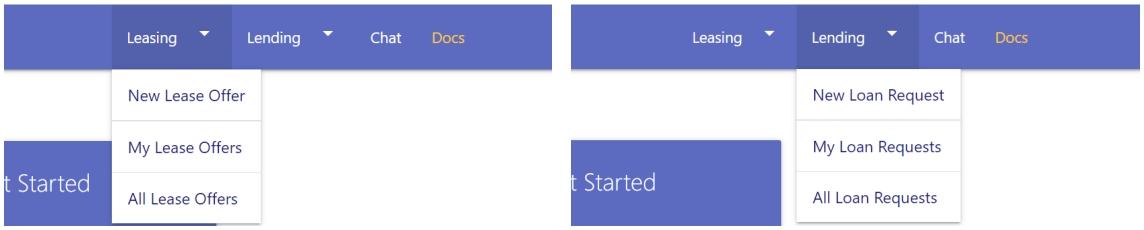


Figure 4.8: Home Page.



(a) Sub-menu for Leasing.

(b) Sub-menu for Loans.

Figure 4.9: Sub-menus.

If the user picks the NFT to create a new lease offer or loan request, he is presented with a Materialize Modal which has an input form with all the details necessary for the calling the corresponding function of the smart contract.

Users can check the address of their NFT for added transparency by clicking on it.

For feedback during transactions we have used Rimble-UI, to provide a toast message with the following information:

- **Transaction started:** users can follow the progress on Etherscan. They have a link with their transaction that they can access as in Figure 4.12 (a). [64]

- **Transaction completed:** for simple function calls, we wait for 7 confirmations to display the toast message as shown in Figure 4.12 (b). [65]

  We also update the Redux store after such transactions are completed to display the latest version of the offers and requests.

- **Transaction failed:** In case the transaction fails, the user is also notified and encouraged to check if the parameters and his actions follow the guidelines.

Under My Lease Offers and My Loan Requests, users are presented with a list of offers and requests that they have created.

The list can be filtered by the status.

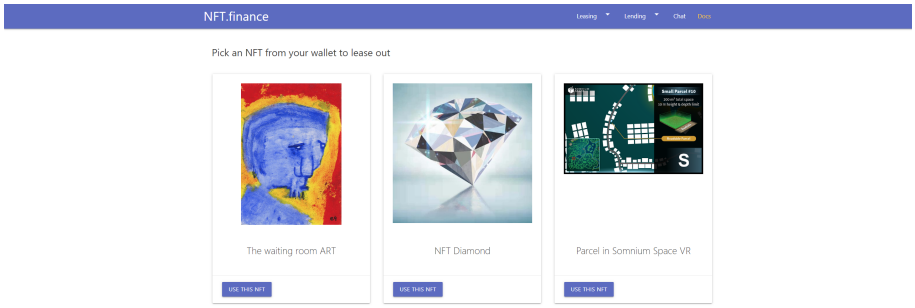
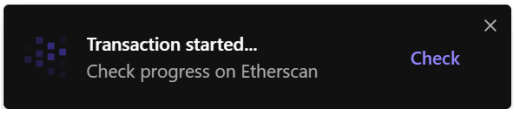
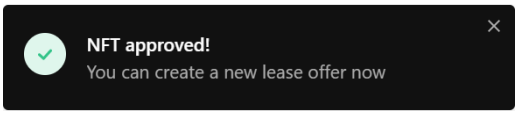


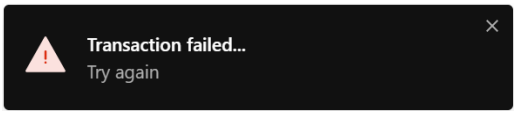
Figure 4.10: New Lease Offer Page.



(a) Transaction Started.



(b) Transaction approved.



(c) Transaction failed.

Figure 4.11: Transaction toast messages.

Each element of the list is a Collapsible Materialize element and can be clicked on to display more information and be interacted with.

All the information about the offer is shown when the Collapsible component is opened. Moreover, the buttons that enable user interaction are dynamically created based on the offer or request and the user that is currently connected.

In this section, we have only presented a few elements of the user experience. For an in-depth walk-through of the platform, there are more screenshots of the UI in Appendix B.

Chapter 5

Evaluation

The project set-out to be a useful addition to the DeFi ecosystem and in this section we will look at multiple metrics that measure its success in achieving this.

Being a product that we intend to release on the Main Ethereum network, we will also discuss more user feedback received during the final iteration of the Human Centered Design research process.

To begin this section, I will re-iterate the two main features of the platform:

- **Leasing:** peer-to-peer leasing of NFTs.
- **Lending:** peer-to-peer loans having NFTs locked as collateral.

They enable new interactions with NFTs, leveraging the ownership of those crypto-assets.

5.1 Smart Contract Security

We have taken the following security measures for our smart contracts:

- Use of OpenZeppelin standards such as `IERC721`, `Pausable` and `IERC721Receiver` as discussed in the implementation section.

  Using their audited smart contracts and standard practices enabled us to correctly handle ERC-721 tokens.

- Making the smart contracts `Pausable` will add a layer of security in case any breaches are to be found at a later stage. As discussed in the implementation chapter, the return of funds or the return of NFTs is not paused, only the addition of new offers.

- Usage of Checks-Effects-Interactions Pattern by performing checks and defining modifiers that are required before the function is executed.

- Interactions with external smart contracts are kept as last step of the functions in order to avoid state changes after the call, should a smart contract be malicious. This can also address re-entrancy attacks.

- Using `call()` instead of `transfer()` or `send()`.

- Unit tests were added to the Solidity functions to check their behaviour with regards to the following scenarios:
  - Edge cases
  - Expected updates to offers and requests
  - Correct handling of NFTs
  - Possible Overflows

5.2 Gas Prices

We have tried to keep gas prices low for users, to encourage usage of our platform.

To estimate the gas prices we have used `estimateGas()` from Web3.js and the gas costs estimated in Ether using Eth Gas Station are the following:

- `createLeaseOffer()` : 289.611 (approximately 0.0040 Ether or $0.95)
- `cancelLeaseOffer()` : 91.133 (approximately 0.0012 or $0.30)
- `acceptLeaseOffer()` : 197.216 (approximately 0.0027 or $0.65)
- `endLeaseOffer()` : 154.194 (approximately 0.0021 or $0.50)
- `createLoanRequest()` : 308.990 (approximately 0.0043 or $1.02)
- `cancelLoanRequest()` : 145.138 (approximately 0.0020 or $0.47)
- `acceptLoanRequest()` : 97.500 (approximately 0.0013 or $0.32)
- `extendLoanRequest()` : 48.891 (approximately 0.0006 or $0.16)
- `endLoanRequest()` : 142.013 (approximately 0.0019 or $0.46)

The cost in Ether was estimated using Eth Gas Station as of June 2020 with a Gas Price of 14 Gwei for an average speed transaction.

The highest cost is for `createLeaseOffer()` and `createLoanRequest()` since it involves writing to storage.

For a complete, successful lease, the lessee would have to pay $1.15 in gas fees, whereas the cost for the lessor would be $0.95.

For lending, the lender would have to pay $0.32 in gas fees, whereas the cost for the borrower would pay $0.78.

Overall, the cost for using the platform is very low and because of this we expect small transactions to take place, for low-value NFTs. We think this would be beneficial since it would attract more users.

As discussed in the implementation chapter, we could have deployed a new smart contract for each new offer, however this would have increased the gas costs by at least 0.04 Ether or $10 and it would have made the barrier of entry for using the platform higher.

5.3 Human Centered Design Process

Focusing on user feedback has brought major benefits to the development of the project.

It has influenced the motivation, the features and the user experience.

There have been 3 iterations of HCD, the first two having been presented in the Design chapter.

The final one is more extensive and involves qualitative research with potential users from varied backgrounds.

5.3.1 Final iteration

The first two iterations have both been conducted with mentors from industry who gave us insight into the Ethereum space and DeFi.

For the last iteration, we have received feedback from potential users, having different background and experience in using blockchain solutions.

They were seven blockchain users from Imperial Blockchain and Crypto-Technologies society, London Blockchain Labs [66] and StakeZero Ventures.

Since it was a qualitative study, the focus was not on having many participants, as it is the case in quantitative studies. Even so, the feedback was undoubtedly useful and insightful since we have been thorough with the questions asked. We were able to find common trends and analyze them.

The study was focused around three main usability metrics:

- **Overall reaction**
- **Perceived usefulness**
- **Perceived ease of use**

Additionally, the feedback was split into two parts:

1. Initially, the participants were each presented with a demo followed by a one-to-one interview about the platform. The interviews were between 15 and 30 minutes long and had open-ended questions. The discussion was adjusted to the knowledge and experience of each participant.

   The interviews have also been recorded for retrospective testing and better understanding of our participants and their opinions. This enabled us to compare answers and discusses the observed treds.

   The recordings will be kept for 30 days and can be retrieved upon confirmation of consent from each participant.

2. Following the interview, each participant completed a questionnaire about their experience and views about NFT.finance.

The interviews were tailored to each participant and analyzed more information and feedback. It also enabled them to ask any questions or request clarifications for using the platform.

The opinions received varied based on the participant and their understanding of blockchain, as well as their previous experience with DeFi products.

- **Overall reaction:** for this metric they were asked questions such as "What are your initial thoughts?" and this led to a conversation starter in which users not only expressed what they liked but also started asking questions about what they found confusing.

  Overall the feedback was positive, the platform being described as "something needed in the Crypto World" and that is looks "very good" and "useful".

  Some of the participants think it is also addressing a "very good market" with a lot of potential to grow.

  On the other hand, one of the participants had very basic financial knowledge and was not familiar with the difference between leasing and lending so he initially found the platform "confusing", even though the UI was straigthforward to use. This will be addressed in a future iteration of the platform.

  The participants have also asked questions about the gas fees and confirmed that lower gas fees would be adequate since the leasing feature would be addressed at users who want to save Ether by not owning an asset outright.

  For this metric, the initial reaction has been good and the use case deemed as a good addition to DeFi and NFTs.

- **Perceived usefulness:** the aim of this metric was to measure if the features of the platform can be useful for potential users. More importantly, finding what they liked about the demo, as well as areas of the platform that need improvement.

  Example of questions asked are: "What feature do you find most useful and why?" or "What was confusing?".

  The common opinion observed was that gaming users would find the leasing feature more useful, whereas DeFi users would be more interested in lending and speculating the prices of the NFTs.

  The participants were encouraged to also provide negative feedback and the answers enforced our previous findings. There might be users who are unfamiliar with financial concepts and our target users would be gaming users in the beginning, therefore it could be better to simplify the terms and provide more guidance to new users.

  Another good point made by one of the participants is that there could be two types of interfaces:

  - **Basic:** for new users who no not need access to complex functionality.
  - **Advanced:** for users who are more familiar with financial instruments and require more sophisticated features.

- **Perceived ease of use:** the focus for this metric was the user experience.

  This gave insight about the UI and example of questions asked are: "What was your overall impression about the UI?" or "What could be improved about the UI?".

  The answers were that it is "simple", but "easy to use" and "very clear". However, some participants have stressed that it might be too "basic" and "lack guidance" for some of the features.

The conclusions have been that there is still room for improvement with regards to the UI, while keeping it as simple to use as it currently is.

The open-ended questions encouraged critical thinking from the participants and gave us very useful insight a clear foundation for the future work.

The next part of the study was the questionnaire which where the participants have rated each feature and also gave an overall rating for the platform. This was not a quantitative study, but was aimed towards re-enforcing the findings from the interviews.

As expected, the responses showed consistency and the overall impression was very positive for the platform and its features with room for improvement on the UI.

In addition, using the questionnaire we have also profiled our participants based on:

- **Blockchain experience:** 5 of the participants had intermediate experience with the Ethereum ecosystem, while 2 were beginners.
- **Having used DeFi protocols before:** 4 had used DeFi protocols before.
- **Having owned NFTs:** all of the participants who were DeFi users have also owned NFTs or are currently owning. From our previous discussion, the NFTs that the participants own are ENS names or event badges.

In conclusion, the Human Centered Design Research process has been part of the whole project, not only the evaluation and guided not only the implementation but also set the problem that we have solved.

We have had continuous evaluation throughout the building process NFT.finance due to following a HCD approach and we will use this method going forward as we will explain the Conclusion chapter.

5.4 Risks

In this section we will discuss the different types of risks that we had to address and we will explain the actions that we took to mitigate or minimize it.

The types of risks encountered can be split into three main categories:

- **Motivation:** initially the goal of the project was very broad since we were addressing addressing the DeFi ecosystem in general.

  Setting out to improve existing DeFi solutions requires understanding of what is currently on the market, as discussed in the Background chapter.

  At this stage the risks involved with the motivation and the need to set a clear and well defined problem statement were even more relevant, since we wanted to build a tool that would bring novel and useful additions to DeFi, and OTC derivatives are not a good fit for this space since they have different core principles.

  As explained in the Design chapter, following our discussions with industry participants in the DeFi derivatives space (Vega protocol and Hedgy), who

have also used ISDA CDM, we have concluded that implementing OTC derivatives in DeFi would not be straightforward and would not bring any obvious benefits to the space.

Therefore, we were presented with the risk of developing a solution that would not complement exchange traded derivatives and would not improve the market.

However, by closely following DeFi loans and the NFT space, we have spotted a gap in the market that could enhance the ownership of NFTs and while using our background about derivatives.

NFTs could benefit from financial solutions such as leasing and lending but also forward contracts and other OTC insurance products, bringing ISDA regulations back into focus.

We think we have mitigated all the risks about the motivation of the project by focusing on a novel solution, which combined insight from different and apparently non-overlapping domains such as DeFi, NFTs and derivatives. A key process which guided and set a structure to our approach was Human Centered Design, as explained above.

- **Implementation:** having a problem statement that was improved throughout the project changed the focus from the use of ISDA CDM to using NFTs and then back to having derivatives as an extension.

  Moreover, having multiple iterations of the Human Centered Design process added requirements in a time frame limited by the length of the iterations. It was challenging to meet the requirements and present add implementation features at each iteration but this has also set clear objectives that we had to achieve.

  Additional implementation risks were related to the security of the smart contract.

  Since we are making calls to external contracts, that could act maliciously, there was a risk of re-entrancy attacks. We had to guarantee that any state or memory changes to the `LeaseNFT` and `LoansNFT` contracts would be resolved before making any external calls. Additional, security measures were discussed in the previous subsection.

  Other implementation risks involved handling the contract data in the React App and responding to changes. A lazy approach to solve this issue would have been to poll the smart contract for each component and periodically. However, this would affect the performance of the website.

  The solution chosen was to have a Redux store that holds the offers and requests of the smart contract as well as their corresponding assets.

  Should a transaction be executed, we can check its result using callback functions and update the store accordingly or inform the user about any error that might have occurred.

- **User risk:** Since NFT.finance is a peer-to-peer platform, a lot of trust amongst users is required.

However, this could have undesirable outcomes which will be explained below, but first we will introduce the following notions, adopted from [47].

- **Identity**
  - **Weak Identity:** the mapping of an agent and his online identity is not one-to-one and could be many-to-one or even change through time.
  - **Strong Identity:** the mapping of an agent and his online identity is one-to-one.
- **Agent Types:** adopting the BAR model of agents [68] with the extension of private valuations. [69]
  - **Honest:** will follow the protocol independent of their private motivations or external payments.
    In the case of leasing or lending, an honest agent would honour the details that he agreed upon when accepting an offer or placing a request.
  - **Rational:** maximize their utility, but generally follows the rules.
    Rational agents on NFT.finance might deviate from the expected behavior in the case of leasing, should the market value of the NFT that they have leased increase over the locked collateral. Similarly, in the case of lending a borrower could decide not to return the principal amount if the locked NFT depreciates.
  - **Adversarial:** maximize payoff in any way possible.
    Those would be malicious users, who will create false trades for their NFTs to inflate their market value to be able to present their asset as more desirable than it actually is.

Since our users will have weak identities, we expect a combination of all agent types on our platform.

This is a risk that our users are faced with and it is impossible to avoid completely on a peer-to-peer platform, but it can be minimized by having a transparent history of the trades made by each agent.

Moreover, we will track and incentivize honest actors and punish adversarial and even rational actors by keeping a "credit score".

5.5 Comparison with existing tools

In this subsection we will compare the NFT.finance platform with different tools that aim to solve related problems.

All the solutions presented are new to the NFT space, having just launched on Mainnet and even though they have similar features to our platform, the implementation is different or lacking in scope, having multiple constraints.

5.5.1 Rocket LP DAO

Rocket LP DAO is a DAO that enables loans with NFTs as collateral. They were the first to issue such a loan on the Mainnet, however it is different to NFT.finance because it is not a peer-to-peer marketplace. [71]

Instead, the DAO has shareholders that can vote to accept and fulfill a loan. However, since NFTs have such a large spectrum of potential use cases, to properly value them and take on a calculated risk the majority of the voters should have similar views and knowledge about the ERC-721 market.

It has also been observed that a peer-to-peer market would be a solution that has a broader applicability and we have concluded that it is superior to having a DAO for multiple reasons such as greater flexibility and easier user on-boarding and access.

5.5.2 Lend721

Another solution that aims to solve leasing of NFTs is Lend721. [72] Similarly to our implementation, it is a peer-to-peer market. The options for creating a lease are also similar, however, the design language of the website targets gaming users exclusively and lack features such as filtering of offers.

It is important for a new solution to earn the trust of potential users and the participants in our last user study have stated that NFT.finance brings a better user experience in comparison to Lend721.

5.5.3 Native leasing solutions

Platforms like Decentraland and Cryptovoxel have renting solutions built-in and users can create NFTs for rent and then sell them on platforms like OpenSea.

Currently, there are a limited number of NFT issuers that have this solution available for its users.

Moreover, this forces the users to take advantage of a different platform targeted at selling or auctioning NFTs. Therefore, it is not very intuitive to use since it was not designed for this use case.

To conclude this section, leasing and lending with NFTs is a technology in its infancy and as the adoption of ERC-721 tokens grows and their value increases, so will the need for a platform like NFT.finance.

The existing tools that are currently available to achieve the same use cases are have different drawbacks and are presented in a less-intuitive manner with limited functionality.

5.6 Automated Market-Making for NFTs

The current focus in DeFi is on Automated Market-Making algorithms which enable users to seamlessly get access to liquidity.

If a similar solution could be implemented for Non-Fungible Tokens it would be superior to a peer-to-peer market.

While researching possible implementations we have contacted the Aave Protocol [?] who offer the possibility of creating liquid money markets where users deposit their assets to earn interest or have the ability to borrow.

However, such a solution for NFTs is would not be possible to be implemented because of their core property of Non-Fungibility, which implies that each token has individual properties that determine their market value.

As an example, Cryptovoxels tokens all represent VR parcels, however they each have different characteristics that determine their market value such as:

- Dimensions of the parcel
- Position on the map
- Surroundings
- Daily traffic

Similarly, the average price of a Decentraland parcel is around 5 Ether. However, the prices have a high standard deviation since some parcels can be worth considerably more than others and there is a lot of market volatility. As an example, the most expensive Decentraland estate sold in 2019 was traded for $80k. [11]

A potential solution would be to wrap similarly priced and most traded ERC-721 tokens in ERC-20 tokens in order to be added to a liquidity pool. However, this would not be feasible since the trading prices of ERC-721 tokens could be manipulated and increased to just to fit a certain profile.

Therefore, the current solution that is suitable for NFTs would be a peer-to-peer market.

Chapter 6

Conclusion

6.1 Reflection

Non-Fungible Tokens are a market that is expected to grow by 50% in 2020, in comparison to the previous year. [73]

Through iterations of Human Centered Design research we have identified a use case that can improve the ownership of ERC-721 tokens and proposed a solution that brings new desirable opportunities to the market of NFTs.

As more valuable assets are stored in NFTs, the need for more complex financial instruments will be greater.

We want to address this market by enabling NFT owners get access to leverage for their tokens, as well as the ability to lease NFTs.

6.2 Platform Use Cases

There a wide range of possible use case and we target different NFT and DeFi users. We will mention some examples below.

- Passive NFT owners who want their assets to generate side income. The leasing feature can be applied to achieve this.

  The NFTs can be gaming items such as power-ups or tools. They can be useful to gain competitive advantage in pay-to-win games and would require a lower up-front costs for users, if leased.

  Additionally, it could be applicable to VR simulations that have have land parcels. It would enable users to lease properties and give specific rights to the lessee such as the ability to build on it.

  Similarly, art tied in the form of NFTs could be leased for the duration of events or to be displayed in a VR art gallery.

- NFT owners that need access to liquidity. The lending feature could be applied to this use case.

  It would reduce the overhead of having to sell the NFT. Additionally, it would be beneficial to owners that are reluctant to sell their assets because of potential long term gains.

The underwriters of such loans would also benefit from higher interest rates in comparison to other DeFi protocols.

- Prices of NFTs could be stabilized by new financial tools, such as forward contracts which would lock the future price making of the assets, making the market less volatile.

- If assets such as real-estate would be tokenized through the use of ERC-721 tokens, they could be used on the NFT.finance platform. However, this would imply additional legal requirements.

6.3 Future Work

Since we will continue working on this project as part of the Spark competition, we have decided on a list of improvements which build upon the current platform and add useful features that can enhance the user experience.

Even so, the core principles and ideas behind Lending or Leasing remain the same.

- After the final iteration of HCD it has been observed that a chat would be a useful addition and it would enable the participants to have a way to communicate or negotiate order details.

  Moreover, some users might requests more details about the NFTs or how the loans are intended to be used.

- An auction system for the loans could also be implemented. This has been suggested by one of the participants in the last round of interviews.

  Once a loan requirement is placed on the market, users could bid in an English style auction on how much liquidity they are willing to provide based on the NFT.

- Search box for filtering NFT items in addition to the provided filters.

  Having a way to find a particular NFT that a user requires, especially for Leasing could be a beneficial addition.

  Moreover filtering the NFTs based on the creator smart contract could also be important to users only looking for a certain type of ERC-721 token.

  However, since the platform is currently deployed on the Rinkeby test network, there are no smart contract addresses that are useful and we decided this would be an addition when moving to the main network.

- Another possible addition is to enable streaming interest for both loans and lending. This would not bring any new features, albeit make the interaction nicer for the users.

  This addition would make streaming payments as long as the lease or loan is running instead of paying interest periodically.

  A solution like Sablier could be used for this. [70]

- Another suggestion from our interviews was to have a simpler version, targeted at users that do not have a strong financial background and are not familiar with how leasing or lending works.

- A final improvement could be made to our handling of ERC-721 tokens, especially for improving the leasing feature.

  We will look into the possibility of creating a proxy token which would have the same functionality as the initial ERC-721 token, but not give owning rights to the lessee.

  This would remove the need to have a collateral locked for each lease offer and would guarantee that token is returned to the lessor at the end of the leasing period.

Appendix A

Appendix A

Figure A.1: Lease contract in Remix IDE.

Figure A.2: Loans contract in Remix IDE.

Appendix B

Appendix B

In this section we will include screenshots of the application and present a User Guide.

When the user accesses the website he is presented with a greeting card that helps him to get started to use NFT.finance, as shown in Figure B.1.

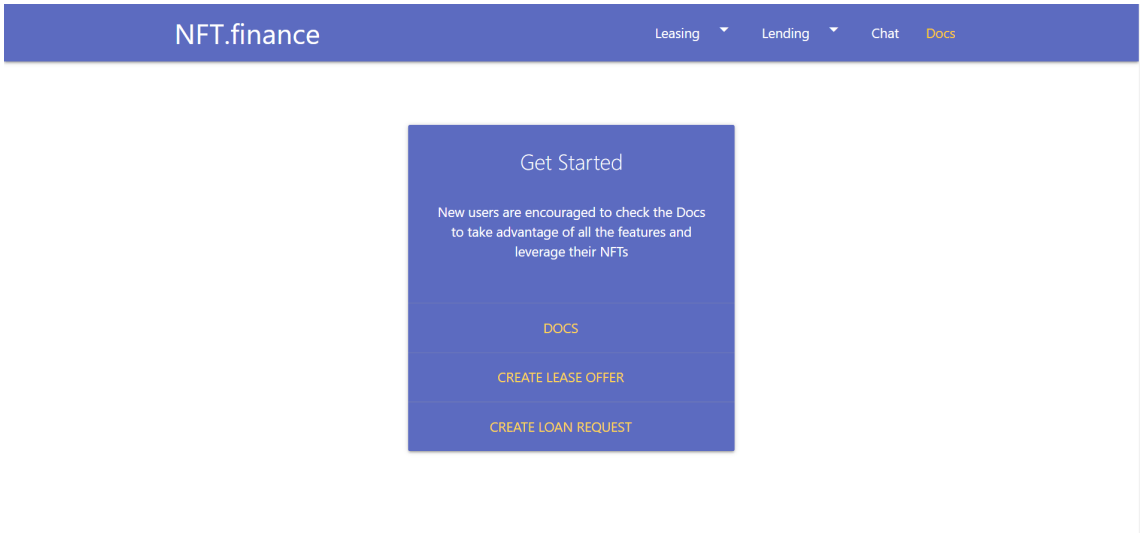


Figure B.1: Home Page.

He can navigate to a the New Lease Offer page, displayed in Figure B.2. to create a new lease offer. On this page all his NFTs are displayed. Any ERC-721 token that he has at his current address will be displayed. If he clicks on the picture of the NFT, he will receive more information about it and a link to OpenSea. We encourage all our users to check the trading history of their ERC-721 token.

If they click on the "Use This NFT" button they will be presented with a modal, as shown in Figure B.3, that has an input form. In this form, the leasing details will be specified, namely the collateral amount for the NFT, the leasing price and leasing period.

In order to create a new offer, the address of the leasing smart contract has to be approved to receive the NFT. Therefore, the user has to click on the "Approve" button first.

Once the transaction has been confirmed, a new lease offer can be created by pressing the "Create Offer" button.

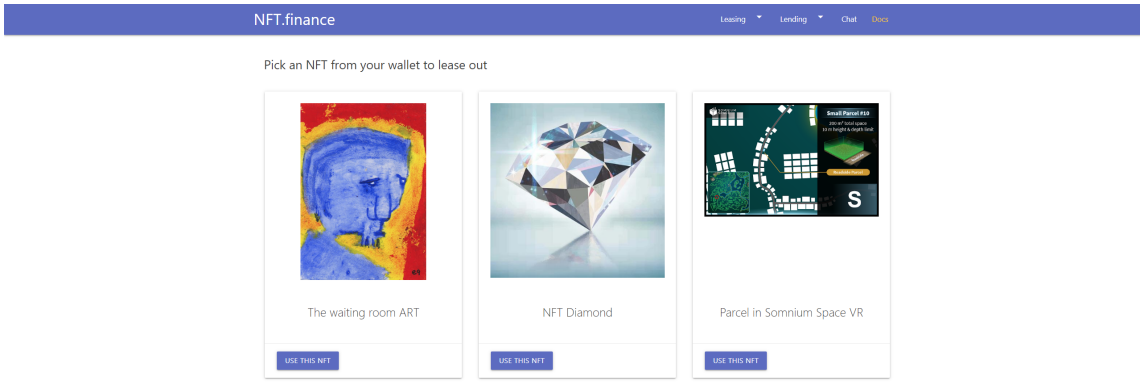


Figure B.2: New Lease Offer Page.

The offers of the user are available in My Lease Offers page. On this page he can filter the offers based on their status and also interact with them.

On the All Lease Offers page, any user can start leasing the available NFTs, as shown in Figure B.4. Again, on this page we have filters for the status of each offer. When clicking an offer, more information about it is displayed, such as the terms of the lease as well as information about the NFT and a link to OpenSea.

The Lending feature is intuitively similar to Leasing and the UI follows the same structure.

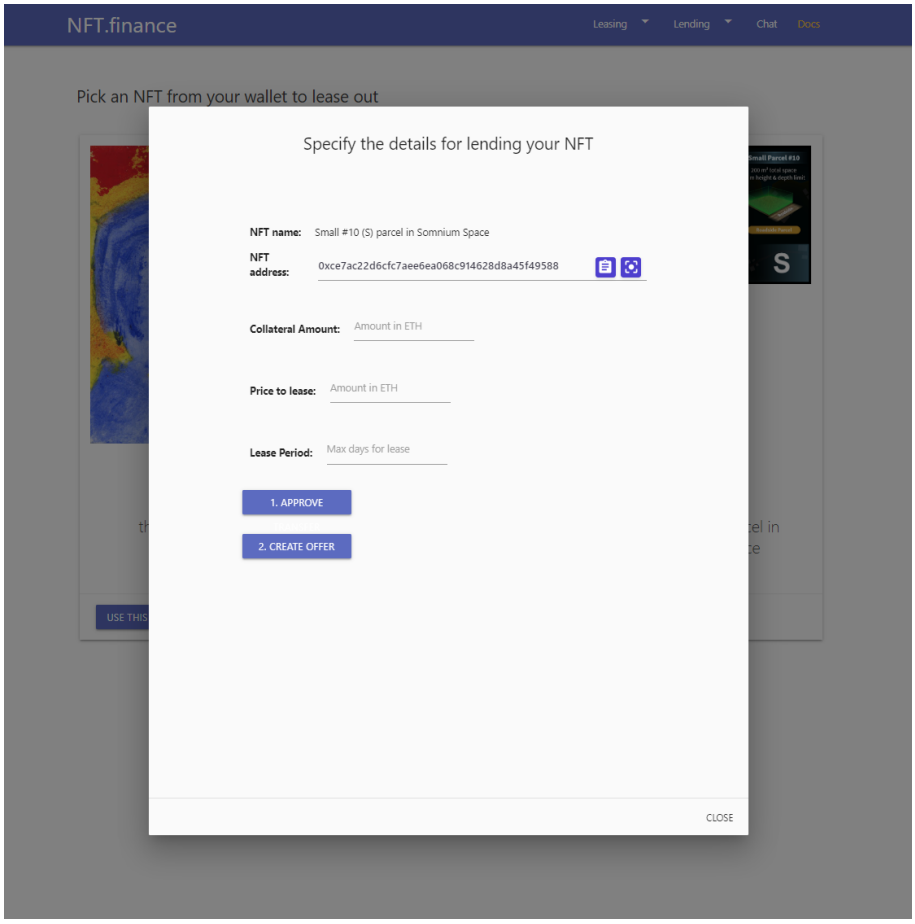


Figure B.3: New Lease Offer Modal.

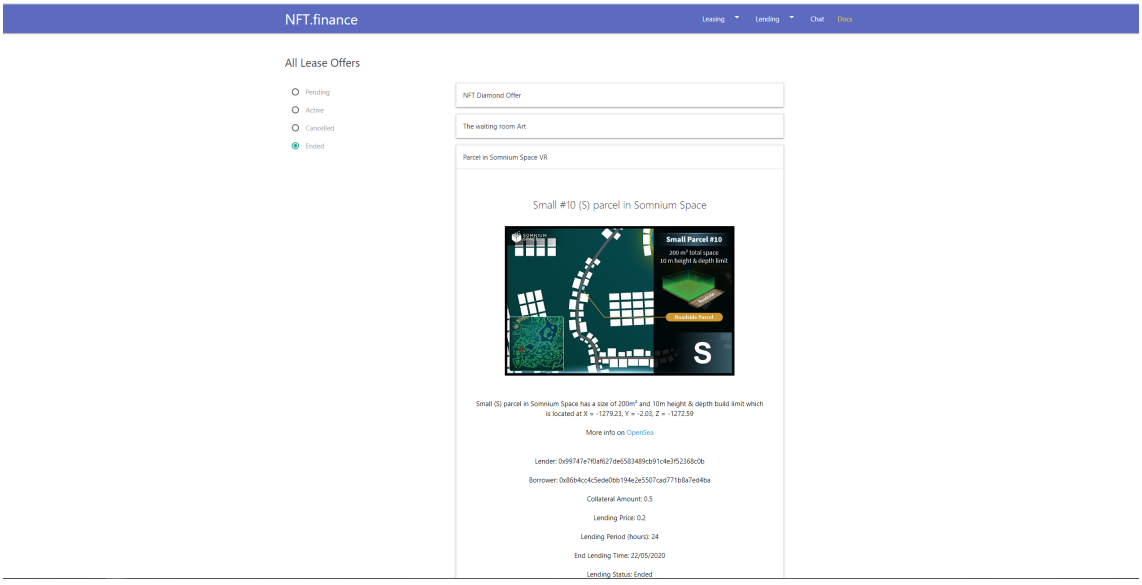


Figure B.4: All Lease Offers Page.

Bibliography

[1] Satoshi Nakamoto. The Bitcoin White Paper, Bitcoin: A Peer-to-Peer Electronic Cash System.

[2] Vitalik Buterin. Ethereum White Paper. Accessed 15 Jun https://github.com/ethereum/wiki/wiki/white-paper

[3] Aaron Hankin. Nearly half of all 2017 ICOs have failed. Accessed 15 Jun. https://www.marketwatch.com/story/nearly-half-of-all-2017-icos-have-failed-2018-02-26

[4] ConsenSys. The 100+ Projects Pioneering Decentralized Finance. Accessed 15 Jun. https://media.consensys.net/the-100-projects-pioneering-decentralized-finance-717478ddcdf2

[5] Christopher D. Clack and Ciaran McGonagle. Smart Derivatives Contracts: the ISDA Master Agreement and the automation of payments and deliveries. Accessed 15 Jun. https://arxiv.org/pdf/1904.01461.pdf

[6] Nick Szabo. Smart Contracts. Accessed 15 Jun. http://www.fon.hum.uva.nl/rob/Courses/InformationInSpeech/CDROM/Literature/LOTwinterschool2006/szabo.best.vwh.net/smart.contracts.html

[7] Gavin Wood. Ethereum Yellow Paper. Accessed 15 Jun. https://gavwood.com/paper.pdf

[8] Andreas M. Antonopoulos, Gavin Wood. Mastering Ethereum. Accessed 15 Jun. https://github.com/ethereumbook/ethereumbook/blob/develop/13evm.asciidoc#evm_architecture

[9] Cryptovoxels. Parcel 2. Accessed 15 Jun. https://www.cryptovoxels.com/parcels/2

[10] OpenSea. Rankings. Accessed 15 Jun. https://opensea.io/rankings

[11] John Robery. Cointelegraph. Five of the Most Expensive NFTs Sold in 2019. Accessed 15 Jun. https://cointelegraph.com/news/five-of-the-most-expensive-nfts-sold-in-2019

[12] DeFi Pulse. Accessed 15 Jun. https://defipulse.com/

[13] Maker DAO. Accessed 15 Jun. https://makerdao.com/en/

[14] ISDA. About ISDA. Accessed 15 Jun. https://www.isda.org/about-isda/

[15] ISDA. Legal Guidelines for Smart Derivatives Contracts ISDA Master Agreement. Accessed 15 Jun. https://www.isda.org/a/23iME/Legal-Guidelines-for-Smart-Derivatives-Contracts-ISDA-Master-Agreement.pdf

[16] ISDA. Legal Guidelines for Smart Derivatives Contracts Introduction. Accessed 15 Jun. https://www.isda.org/a/MhgME/Legal-Guidelines-for-Smart-Derivatives-Contracts-Introduction.pdf

[17] ISDA. ISDA CDM Factsheet. Accessed 15 Jun. https://www.isda.org/a/z8AEE/ISDA-CDM-Factsheet.pdf

[18] BIS. Central clearing: trends and current issues. Accessed 15 Jun. https://www.bis.org/publ/qtrpdf/r_qt1512g.pdf

[19] ISDA. FpML User Guide. Accessed 15 Jun. https://www.isda.org/book/fpml-5-user-guide-2012-edition-pdf/

[20] DAML Medium. New options for FpML and similar standards. Accessed 15 Jun. https://medium.com/daml-driven/new-options-for-fpml-and-similar-standards-878ad4abe0df

[21] Rosetta Technology. The CDM Model. Accessed 15 Jun. https://docs.rosetta-technology.io/cdm/documentation/source/documentation.html

[22] GitHub. Barclays Derivhack Corda. Accessed 15 Jun. https://github.com/corda/barclays-derivhack

[23] GitHub. Barclays Derivhack Algorand. Accessed 15 Jun. https://github.com/algorand/DerivhackExamples

[24] Joseph Young. NewsBTC. Hedgy Revolutionize OTC Financial Market. Accessed 15 Jun. https://www.newsbtc.com/2015/11/12/hedgy-to-revolutionize-the-otc-financial-market-with-blockchain-based-smart-con

[25] Bitcoin Exchange Guide. Hedgy Acquired by Wyre Crypto Payment Startup for Blockchain Smart Contract Team. Accessed 15 Jun. https://bitcoinexchangeguide.com/hedgy-acquired-by-wyre-crypto-payment-startup-for-blockchain-smart-contract-tea

[26] Vega Protocol. Vega Protocol whitepaper. Accessed 15 Jun. https://vega.xyz/papers/vega-protocol-whitepaper.pdf

[27] Synthetix. Synthetix litepaper. Accessed 15 Jun. https://www.synthetix.io/uploads/synthetix_litepaper.pdf

[28] Gnosis. Gnosis White Paper. Accessed 15 Jun. https://gnosis.io/pdf/gnosis-whitepaper.pdf

[29] OpenSea. About OpenSea. Accessed 15 Jun. https://opensea.io/about

[30] Consensys. Home. Accessed 15 Jun. https://consensys.net/

[31] Solidity. Read the Docs. Accessed 15 Jun. https://solidity.readthedocs.io/en/v0.6.8/

[32] ERC-721. Accessed 15 Jun. http://erc721.org/

[33] EIPS. ERC-165. Accessed 15 Jun. https://eips.ethereum.org/EIPS/eip-165

[34] OpenZeppelin. ERC 721. Accessed 15 Jun. https://docs.openzeppelin.com/contracts/3.x/api/token/erc721

[35] Remix. Ethereum IDE. Accessed 15 Jun. https://remix.ethereum.org

[36] Solidity. Docs. Accessed 15 Jun. https://solidity.readthedocs.io/en/v0.6.8/

[37] Truffle Suite. Ganache. Accessed 15 Jun. https://www.trufflesuite.com/ganache

[38] MetaMask. Accessed 15 Jun. https://metamask.io/

[39] EthHub. Docs. Accessed 15 Jun. https://docs.ethhub.io/using-ethereum/test-networks/

[40] OpenSea. Docs. Accessed 15 Jun. https://docs.opensea.io/reference

[41] Rinkeby. Faucet. Accessed 15 Jun. https://www.rinkeby.io/#faucet

[42] Chris Blec. Defiprime. Are Uniswap's Liquidity Pools Right for You? Accessed 15 Jun. https://defiprime.com/uniswap-liquidity-pools

[43] GitHub. Uniswap. Accessed 15 Jun. https://github.com/runtimeverification/verified-smart-contracts/blob/uniswap/uniswap/x-y-k.pdf

[44] ETHLend. Home. Accessed 15 Jun. https://ethlend.io/

[45] Emilio Frangella. Decentralized Lending Pool (DLP) Protocol Launched on Testnet — Swap Between Fixed and Variable Rates. Accessed 15 Jun. https://medium.com/aave/decentralized-lending-pool-dlp-protocol-launched-on-testnet-swap-between-fixed-

[46] Kaihua Qin, Liyi Zhou, Benjamin Livshits, and Arthur Gervais. Attacking the DeFi Ecosystem with Flash Loans for Fun and Profit. Accessed 15 Jun. https://arxiv.org/pdf/2003.03810.pdf

[47] Lewis Gudgeon, Daniel Perez, Dominik Harz, Arthur Gervais and Benjamin Livshits. The Decentralized Financial Crisis: Attacking DeFi. Accessed 15 Jun. https://arxiv.org/pdf/2002.08099

[48] React. Accessed 15 Jun. https://reactjs.org/

[49] Redux. Accessed 15 Jun. https://redux.js.org/

[50] web3.js. Accessed 15 Jun. https://web3js.readthedocs.io/en/v1.2.8/

[51] GitHub. Axios. Accessed 15 Jun. https://github.com/axios/axios

[52] Materialize. Accessed 15 Jun. https://materializecss.com/

[53] ConsenSys. Rimble. Accessed 15 Jun. https://rimble.consensys.design/

[54] Redux. Three Principles. Accessed 15 Jun. https://redux.js.org/introduction/three-principles/

[55] GitHub. Redux Thunk. Accessed 15 Jun. https://github.com/reduxjs/redux-thunk/

[56] Redux. Redux Actions. Accessed 15 Jun. https://redux.js.org/basics/actions/

[57] GitHub. EIP-1193. Accessed 15 Jun. https://github.com/ethereum/EIPs/blob/master/EIPS/eip-1193.md

[58] Saharsh Goyal. React-16 | New life-cycle methods inside out. Accessed 15 Jun. https://medium.com/@saharshgoyal/react-16-new-life-cycle-methods-inside-out-fdd269c43ccd

[59] HackMoney. Hackathon. Accessed 15 Jun. https://hackathon.money/

[60] ISO. ISO 9241-210:2019. Accessed 15 Jun. https://www.iso.org/standard/77520.html

[61] Miro. Accessed 15 Jun. https://www.miro.com/

[62] YouTube ETHGlobal. Hack Feedback Session. Accessed 15 Jun. https://youtu.be/bj8kkgLCHaw?t=1735

[63] YouTube ETHGlobal. Group 7 Judging. Accessed 15 Jun. https://youtu.be/Ej9ke_ug4mI?t=6555

[64] Etherscan. Home. Accessed 15 Jun. https://etherscan.io/

[65] Ethereum. Ghost. Accessed 15 Jun. https://ethereum.org/whitepaper/#modified-ghost-implementation

[66] London Blockchain Labs. Accessed 15 Jun. https://ethereum.org/whitepaper/#modified-ghost-implementation

[67] Stake Zero Ventures. Accessed 15 Jun. https://stakezero.com/

[68] Amitanand S Aiyer, Lorenzo Alvisi, Allen Clement, Mike Dahlin, Jean-Philippe Martin, and Carl Porth. BAR fault tolerance for cooperative services. ACM SIGOPS Operating Systems Review, 39(5):45, 2005.

[69] Dominik Harz, Lewis Gudgeon, Arthur Gervais, and William J Knottenbelt. Balance: Dynamic adjustment of cryptocurrency deposits. In Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, pages 1485-1502. ACM, 2019.

[70] Sablier.finance. Accessed 15 Jun. https://sablier.finance/

[71] Joshua Mapperson. First Loan Ever Issued With Ethereum Domain Name as Collateral. Accessed 15 Jun. https://cointelegraph.com/news/first-loan-ever-issued-with-ethereum-domain-name-as-collateral

[72] Lend721. Accessed 15 Jun. https://lend721.app/

[73] Zoup. Nonfungible.com. Accessed 15 Jun. https://nonfungible.com/blog/nft-yearly-report-2019