Borrowing digital currency against real world assets within decentralised finance

An exploration of collateralised lending within Decentralised finance

Christopher Staunton
B.A.(mod) Computer Science and Business

April 2022

A report submitted in partial fulfilment of the requirements for the degree of BA(Mod) in Computer Science and Business
Abstract

After the 2008 financial crisis, Bitcoin emerged as the foundation for a new era of peer-to-peer transactions. Its underlying technology, the blockchain, allows for the issuance and movement of digital assets without a central authority. This new technology spurred the innovation behind a new, unbiased and open financial system. This decentralised financial system has the ability to remove the restrictive power of central authorities and empower participants to gain control of their finances.

The goal of this project will be to explore decentralised finance by implementing a system to enable the use of real world assets on the Ethereum blockchain. This work follows a novel approach to representing real world assets as non-fungible tokens on the Ethereum blockchain. Furthermore, it explores the use of these tokenised real world assets as collateral which enables users to borrow a stable cryptocurrency. This will entail designing and creating a system to integrate with an existing lending protocol.
Acknowledgements

First and foremost, I would like to thank my supervisor, Dr Donal O’Mahony, for the opportunity to explore and research within this innovative space and for his patience, advice, and support throughout this project.

Secondly, I would like to thank the members of the Maker community for welcoming me and providing me with guidance.

Finally, I would like to thank my friends and family for always supporting me.
Contents

Abstract .................................................................................................................................................. 2
Acknowledgements ............................................................................................................................... 3
List of figures: ...................................................................................................................................... 7
Chapter 1 | Introduction: ..................................................................................................................... 8
  Motivation ........................................................................................................................................ 8
  Problem Statement .......................................................................................................................... 8
  Objectives ......................................................................................................................................... 9
  Structure .......................................................................................................................................... 10
  Glossary ........................................................................................................................................... 11
Chapter 2 | State of the Art .................................................................................................................. 12
  Blockchain: ..................................................................................................................................... 12
    Transactions .................................................................................................................................. 12
    Distributed Ledger ....................................................................................................................... 14
    Decentralised Consensus .............................................................................................................. 14
    Applications ................................................................................................................................ 15
  Ethereum ......................................................................................................................................... 16
    Accounts ..................................................................................................................................... 16
    Smart contracts ............................................................................................................................. 17
    Decentralised Applications: ......................................................................................................... 17
    Wallets .......................................................................................................................................... 18
    Tokens .......................................................................................................................................... 18
  ERC-721 - Non-fungible Tokens ...................................................................................................... 18
  Decentralised Finance ...................................................................................................................... 19
  Money: ............................................................................................................................................. 21
    Characteristics of money: ............................................................................................................. 21
    Functions of money: .................................................................................................................... 22
    Stablecoins: ................................................................................................................................. 23
  Decentralised Autonomous Organisations (DAO) ......................................................................... 26
  Maker Protocol ................................................................................................................................ 26
    Risk Parameters ............................................................................................................................ 27
    Components of the Maker Protocol: ............................................................................................. 28
    Internal Accounting module ......................................................................................................... 29
    Oracles: ......................................................................................................................................... 30
    Vaults: .......................................................................................................................................... 32
List of figures:

Figure 2.1: State transition caused by transactions (Rattanaveerachanon, Richards and Cursi, n.d.) __________ 13
Figure 2.2: Blockchain Architecture (Wang et al., 2018) ____________________________________________ 13
Figure 2.3: table of top 5 stablecoins by market cap (Coinmarketcap, 2022) ____________________________24
Figure 2.4: The Maker Protocol System Diagram (MakerDAO Technical Docs, n.d.) ______________________27
Figure 2.5: illustration of how shifting supply effect price (Maker Protocol 101, 2020) __________________ 29
Figure 2.6: Blockchains cannot connect to real-world data and events on their own (Oracles| Chainlink, 2021) 30
Figure 2.7: High-level overview of oracle Maker system (Ziu et al., 2021) ______________________________31
Figure 2.8: illustrating the locking and withdrawing transactions ______________________________________ 33
Figure 2.9: Illustration of the accrual of fees _________________________________________________________33
Figure 2.10: the repayment of Dai and freeing of collateral ___________________________________________34
Figure 2.11: Illustration of the discounted debt calculation, adapted from (Rates Module – Maker Docs, n.d.) 36
Figure 2.12: drop in collateral balance ___________________________________________________________ 37
Figure 2.13: illustrating collateral being confiscated and sent to the auction contract ___________________37
Figure 2.14: Displaying the timeline of Dutch auction ______________________________________________ 38
Figure 3.1: proof-of-concept architecture ___________________________________________________________ 42
Figure 3.2: high-level overview of NFT management ________________________________________________ 44
Figure 3.3: illustrating ‘locking’ of RWA NFT and withdrawal of Dai __________________________________ 45
Figure 3.4: illustrating the fee accrual _____________________________________________________________ 45
Figure 3.5: illustrating the repayment of debt and ‘freeing’ of RWA NFT. _______________________________46
Figure 3.6: updating real world asset value, adapted from (Maker Protocol 101, 2020) ___________________47
Figure 3.7: RWA NFT confiscated as a result of dropping below liquidation ratio ________________________48
Figure 3.8: high-level overview of auction operation. _______________________________________________ 48
Figure 4.1: System architecture implementation ______________________________________________________50
Figure 4.2: auth access modifier __________________________________________________________________ 51
Figure 4.3: operator access modifier __________________________________________________________________ 52
Figure 4.4: NFT adapter function to enter NFT as collateral in the system ________________________________ 53
Figure 4.5: Real World Asset NFT implementation _____________________________________________________ 54
Figure 4.6: NFT receiver interface function __________________________________________________________ 54
Figure 4.7: Vault function to lock RWA NFT __________________________________________________________ 55
Figure 4.8: Vault free collateral function _____________________________________________________________ 56
Figure 4.9: Vault draw Dai function __________________________________________________________________ 57
Figure 4.10: Vault repay function __________________________________________________________________ 57
Figure 4.11: Oracle function to modify value __________________________________________________________ 58
Figure 4.12: Vault health check function __________________________________________________________________ 59
Figure 4.13: Liquidation function ____________________________________________________________________ 60
Figure 4.14: Vault user interface ____________________________________________________________________ 61
Chapter 1 | Introduction:

Motivation

The current financial system is restrictive and centrally controlled. Governments and other central authorities can restrict access to financial services at will. The costs and inaccessibility of financial services can block employment and force someone to stay in poverty, in 2017 there were 1.7 billion “unbanked” people (Demirguc-Kunt et al., 2017). Intermediaries can charge premiums for services as they are an essential source of truth between parties in the current centralised financial system. Despite the need for trusted intermediaries, many times these institutions are closed books, forcing users to trust blindly. Participants of the current financial system must trust these large institutions to correctly manage their funds. The global financial crisis of 2008 provides a powerful demonstration of the problems associated with placing too much trust in centralised institutions (Reiff, 2021). Soon after the deceit and financial mismanagement by Lehman Brothers and other financial institutions, Bitcoin paved the way for a new decentralised financial system.

The motivation behind this project is to enable the adoption of decentralised financial services to a wider audience. Decentralised finance enables permissionless access to various financial services such as lending, borrowing, trading and investing. Enabling users to leverage their existing assets is at the forefront of decentralised finance development. In the coming years, we are likely to see great progress and adoption by larger institutions.

Problem Statement

In their paper “Bitcoin: A Peer-to-Peer Electronic Cash System”, Satoshi Nakamoto proposed a revolutionary solution to the issue of centralised trust in a financial system by using a decentralised, peer-to-peer method of tracking and verifying transactions (Nakamoto, 2008). This removed the need for a transaction to be supported by a centralised financial institution. This approach simultaneously solved the problems of decentralised consensus and double spending. This technological breakthrough spurred the innovation of a larger decentralised financial system. Built with blockchain technology, decentralised finance enables anyone to access financial services regardless of age, nationality, or gender. However, this system is in its infancy stage and is far from mainstream adoption.
Although there is huge growth in the decentralised finance area, the total assets in DeFi grew from $600 million in 2020 to $200 billion in 2022 (Canny, 2022), its size is still dwarfed by the traditional financial system. The growth of decentralised finance enables more people to access unbiased financial services. One approach to gaining mainstream adoption is the integration with the traditional financial world. By enabling people to utilise their existing assets from the traditional financial system in the DeFi system, adoption of the technology becomes a lot easier. This integration poses a problem as both systems have different underlying technologies and regulatory stances.

This project aims to investigate decentralised finance with the aim of designing a system suitable for the integration of real world assets on a blockchain-based financial system.

Objectives
The overall goals of this project are as follows:

- Gain an understanding of blockchain technologies and decentralised finance. Investigate the Maker Protocol and an approach to real world asset tokenisation.
- Develop a proof-of-concept system that enables the borrowing of cryptocurrency against real world assets.
- Implement smart contracts to create a prototype system that integrates with the Maker protocol to support the designed proof-of-concept.
Structure

Chapter 2 – **State of the Art** explores the decentralised financial system and the technologies that power it. It also provides an examination of the inner workings of the Maker Protocol, which forms the backbone of this project.

Chapter 3 – **System design** details the architecture of the proposed proof-of-concept system. The design goals and interactions within the system are highlighted.

Chapter 4 – **Implementation** provides the overview of the technical details implemented to convert the proof-of-concept into a functioning prototype.

Chapter 5 – **Evaluation** describes the difficulties encountered and the limitations of the design and implementation of the project. It provides an insight into the success of the developed project.

Chapter 6 – **Conclusion** provides a summary of the project and a brief look into other approaches the author could have taken. Additionally, it provides an overview of potential future work related to this project.
Glossary

On-chain: data stored on the blockchain

Off-chain: data stored off the blockchain

Fiat currency: Government-issued currency

Cryptocurrency: Digital currency enabled by blockchain technologies

Collateral Asset: Something of value used as security for a loan.

Over collateralisation: The value of underlying collateral exceeds the value of the loan

Crypto asset: Any digital asset that is enabled by blockchain technologies, including cryptocurrencies

Stablecoin: A cryptocurrency designed to have a stable value

Token: Interchangeable for crypto asset

Bitcoin: A blockchain network

Ethereum: A blockchain network

Ether (ETH): Native cryptocurrency of the Ethereum blockchain

Görli: a testing environment for Ethereum

Real World Asset (RWA): any tangible or non-tangible asset not yet represented on-chain.

Decentralised Finance: The group of financial products and services available through blockchain.
Chapter 2 | State of the Art

Blockchain:
A blockchain can be described as a public ledger of transactions that is shared among many different computers. Bitcoin and other cryptocurrencies are enabled by blockchain technology. Synonymous with its underlying technology, Bitcoin was first proposed by an anonymous person or group called Satoshi Nakamoto in their white paper called “Bitcoin: A Peer-to-Peer Electronic Payment System” (2008). This revolutionary approach removed the need for intermediaries in a financial transaction. The trust was removed from people and placed in math (Antonopoulos, 2014). It uses a cumulation of multiple technologies, such as cryptography, peer-to-peer networks, distributed ledgers, and consensus mechanisms to create a trustless and tamper-proof database. The nature of a blockchain is immutable, meaning once a transaction is valid and accepted it cannot be deleted.

Transactions
Transactions lie at the core of blockchain technology. They are messages that alter the state of the blockchain, see figure 2.1. For example, a simple transaction could include the instructions to send 10 Bitcoin from Alice to Bob. In order to ensure the integrity of a blockchain authorization and authentication of the transactions are done by asymmetric cryptography, where the private key is used to sign the transaction and the public key is used to verify it (Shrimali and Patel, 2021). This prevents forgery of transactions. As a result of a blockchain being immutable, the full history of transactions is recorded.
Transactions are batched together in “blocks” which are then “chained” together, where each block includes a cryptographic hash of the previous block (Shrimali and Patel, 2021). The cryptographic hash is found by inputting data into a one-way hash function. This produces a unique fixed-size string of bits. A hash function is one way because it is computationally infeasible to retrieve the input data if given only the output hash (Antonopoulos and Wood, 2018). This creates the linked list style data structure as seen in figure 2.2. The long chain of blocks makes it easy to spot tampering as a slight change in one block will alter every block that comes after it.

Figure 2.2: Blockchain Architecture (Wang et al., 2018)
The contents of a block in a blockchain network can be seen in figure 2.2, which includes the list of transactions, the hash of the previous block, a once-off number used to satisfy the previous block hash called the nonce, and the timestamp. The timestamp allows for the chronological ordering of transactions which prevents the double spend problem (Nakamoto, 2008). A double spend occurs if Alice tries to send 10 Bitcoin to Bob and 10 Bitcoin to Claire, but only has a balance of 10 Bitcoin. In practice, the valid transaction with the earliest timestamp should be taken as truth.

**Distributed Ledger**

A ledger is a record-keeping system usually associated with financial records (Kenton, 2022). A distributed ledger is a digital record of transactions that multiple computers hold the same copy of. With regards to blockchain, each node hosts a copy of the history of transactions (Shrimali and Patel, 2021). This feature prevents a single point of failure because every node maintains a copy of the synchronised “database”. To ensure the network agrees on the state of the distributed ledger, or blockchain, a consensus mechanism must be used.

**Decentralised Consensus.**

Consensus mechanisms allow untrusted parties to agree on a common state. In a centralised system, consensus can be reached via a trusted intermediary. For example, if Alice wanted to send Bob $10, the intermediary would check her balance and react accordingly. A core issue of a distributed and decentralised system is the lack of a central authority to provide a common agreement. For a blockchain network to reach consensus, 51% or more of the nodes must agree on the state of the blockchain. This is achieved through various consensus mechanisms that are implemented with cryptography.

**Proof-of-Work:**

The blocks within a Proof-of-work system are created by nodes, called miners, who compete against each other in a race to solve a mathematical puzzle. The puzzle involves producing the acceptable cryptographic link between the most recently accepted block and the new block (Nakamoto, 2008). Once a valid block is created, the miner will propagate the new
block across the network for other nodes to verify the correctness of the hash value (Zheng et al., 2017).

If successful they will add it to their copy of the blockchain and start working on the next block. At this stage, the state of the blockchain has been updated because a valid block has been added to the “chain”. For consensus to be reached the proof-of-work protocol should be combined with a chain selection rule. In the case of Ethereum and Bitcoin, the longest chain rule is used, which means that the longest blockchain will be accepted as valid by the network nodes (Nakamoto, 2008). This combination keeps the blockchain system secure as a malicious actor would need 51% of the networks computing power, which would likely be unprofitable (Richards and Collins, n.d.).

Applications

Bitcoin was proposed as an electronic cash system that allows “two willing parties to transact directly with each other without the need for a trusted third party” (Nakamoto, 2008). However, due to the attributes of decentralisation, immutability, transparency and cryptographic security, there are many other vastly innovative applications, one such example is Ethereum.
**Ethereum**

Ethereum is a public blockchain network first proposed by Vitalik Buterin (2014) in his Ethereum Whitepaper. In this paper, Buterin recognises the limitations of Bitcoin, mainly the lack of a Turing complete scripting language which hinders it from supporting loops. Ethereum improves upon Bitcoins innovation by creating a blockchain with built-in Turing completeness (Buterin, 2014). This allows anyone to write and create any conceivable computer program on Ethereum. This approach has enabled Ethereum to become a decentralised, peer-to-peer computing platform rather than just a payment system. In order to prevent excessive use of the network’s resources, Ethereum’s native currency Ether (ETH) is used as a pricing mechanism for computational power on the network (Nie et al., n.d.).

**Accounts**

An Ethereum account is an entity with an ETH balance that can send transactions, which may include sending or receiving ETH, and interacting with smart contracts (Nolan et al., 2022). Every Ethereum account has an Ethereum address, just like an email inbox has an email address. There are two types of accounts, externally owned accounts, and contract accounts.

**Externally owned accounts**

Externally owned accounts (EOA) are owned and controlled by anyone with the associated private keys. Each EOA is made up of a cryptographic pair of keys, the private key, and the public key. In order to send a transaction from an EOA the transaction must be signed with the private key (Nolan et al., 2022). This method of using public-key cryptography proves that the transaction was in fact signed by the private key holder, which prevents forgeries. Knowledge of the private key grants full custody of the funds of the account. The public address of an account is derived from the public key.

**Contract account:**

A contract account is a smart contract deployed to the network. It is controlled by the code written inside it. They act as “autonomous agents that live inside of the Ethereum execution environment”, that execute when they are “poked” via transactions (Buterin, 2014). Similarly, to EOAs they can hold, send and receive ETH. However, creating a contract has a cost because it uses network storage (Nolan et al., 2022).
**Smart contracts**
A contract is a legally binding agreement, which can be enforced in a court of law. They enable modern society to operate, they help regulate most of our professional and personal life. Coined by American Computer Scientist Nick Szabo, “a smart contract is a computerized transaction protocol that executes the terms of a contract” (Szabo, 1994).

A great example of a smart contract described by Szabo involves the operation of a vending machine. The logic programmed into the vending machine means that given the right inputs, a certain output is guaranteed. Szabo believed that in order for smart contracts to be effective, they need to be observable, verifiable, and enforceable (Szabo, 1997). Blockchain technology enables these traits.

**Smart contracts on Ethereum:**
A smart contract is a program that runs on the Ethereum blockchain. The code within a smart contract on the Ethereum blockchain defines rules and automatically enforces them when a user interacts with it. They are interacted with via transactions that execute certain functions described by the smart contract. They are usually written in a high-level programming language, such as Solidity, and then compiled into bytecode so the Ethereum Virtual Machine (EVM) can interpret them. The EVM is the state machine that receives instructions that update the Ethereum state (Hotchkiss, 2022). As a result of the Ethereum blockchain being public, writing and deploying smart contracts is permissionless. However, it costs a small amount of ETH to deploy and execute them. They can be thought of as open APIs as anyone can send transactions to them (Entriken et al., 2022).

**Decentralised Applications:**
Decentralised Applications or dapps are applications that have their backend code running on a decentralised network rather than a centralised server (Gallagher et al., 2022). Dapps use smart contracts for their app logic and store their data on the Ethereum blockchain. They usually have a frontend interface, which may be hosted centrally. Despite this, once a dapp is deployed it has no owner and cannot be modified, meaning it can be used by anyone regardless of a centralised frontend (Gallagher et al. 2022). The immutable nature of smart contracts makes updates and maintenance of dapps difficult even if security risks are found.
**Wallets**

A wallet is a software application that allows a user to manage their Ethereum account. They securely store the keys associated with the user’s account, which enables the user to sign and send transactions (Ethereum wallets | ethereum.org, 2022). A wallet application usually provides a user interface where a user can view their account balance. It is important to note that a wallet is not the same as an account.

**Tokens**

A core part of the Ethereum ecosystem is the use of tokens, which are blockchain native assets that can be bought, sold, or traded. These tokens, also known as crypto assets, can represent virtually anything such as digital currency, lottery tickets, financial assets, points in a game, etc. Ether, the native Ethereum cryptocurrency, is a token that can be used to pay for transactions and computing power on the network. Hundreds of thousands of tokens have been created on top of the Ethereum blockchain. In order to manage these various tokens, a user’s wallet must be able to interact with the underlying smart contract. This becomes unfeasible if there is no standardisation of token implementation. To improve the usability of the Ethereum ecosystem, contributors propose Ethereum Request for Comments (ERC) which outline token implementation standards.

**ERC-20 – Fungible Tokens**

The Ethereum Request for Comments 20 (ERC-20) standard deals with fungible tokens. Fungible means that one token is equivalent to any other token of the same type, like how one US dollar is the exact same as any other US dollar. ERC-20 is a technical standard for all smart contracts on the Ethereum blockchain that implement fungible tokens. It defines a common list of rules that fungible tokens on the Ethereum network should adhere to (Gontijo et al., 2021). The rules include how the tokens can be transferred, the total supply of said tokens, the balance of an account, and how transactions are approved. If a wallet can integrate with the ERC-20 standard it can manage any ERC-20 compliant token created.

**ERC-721 - Non-fungible Tokens**

Non-fungible tokens are a way to represent anything unique or one of a kind as an Ethereum-based asset (NFT | ethereum.org, 2022). The most common application of this technology is a token to represent a unique digital art piece, however, they could be used to represent physical assets such as deeds to a house. On the Ethereum blockchain, NFTs are
powered by smart contracts which track the ownership of their NFTs. As Ethereum is a public blockchain, ownership is public and can be easily verified by anyone (NFT | ethereum.org, 2022). The metadata of the asset may be stored on the IPFS and therefore considered immutable, meaning no one can modify the non-fungible token. This provides a solution to digital asset ownership because before NFTs, digital assets such as music, video files, etc could be easily copied and distributed. An NFT only has one owner and one source, both of which can be publicly verified.

ERC-721 is the standard smart contract implementation to represent ownership of non-fungible tokens (NFTs) on the Ethereum blockchain. It lays out the basic functionality required such as transferring NFTs, getting the owner of a token, getting the token balance of an account, and the total supply of tokens created from that ERC-721 smart contract (Moen et al., 2022). The ERC-721 smart contract is responsible for keeping track of the tokens it creates, much like how the ERC-20 smart contract tracks the fungible tokens it creates.

Decentralised Finance

Decentralised Finance (DeFi) is the collection of financial products and services built on decentralised networks, such as Ethereum. These services allow anyone, with an internet connection, to access a financial system and take control of their financial wellbeing. These financial services operate around the clock unlike the usual banking hours of traditional finance. As with other decentralised systems, there is no central authority meaning no one entity can block payments or deny access (DeFi | ethereum.org, 2022). Building on top of Blockchain implies that these products have mitigated human error as the processes are handled by public code. This means that users can inspect the internal workings of the products and scrutinize the operations.

In their Systemization of Knowledge on Decentralized Finance, Werner et al. (2021), examine the workings of DeFi, and what an ideal situation would look like for the DeFi ecosystem. They describe the ideal DeFi system as having four main attributes; User-owned funds: participants of the DeFi system should have full control over their funds, Permissionless: No one must gain permission to become a user, Transparent: The state of the system can be audited by anyone,
Composable: The system should be capable of being extended by others (Werner et al., 2021). The "money Lego" (Meegan and Koen, 2021) metaphor adequately describes this process of building on top of existing projects, which is relevant to the design of the proof-of-concept explored later in this paper.

**Comparison to traditional finance**

Decentralised Finance uses smart contracts and cryptocurrencies to provide financial services without the need for intermediaries. In the Traditional financial system, large financial institutions act as trusted intermediaries for transactions. As a result of large amounts of money flowing through these institutions, they hold a significant amount of power. In DeFi this power is removed from intermediaries and placed in smart contracts. A smart contract will hold funds and send/refund them based on certain conditions. Once the smart contract has been deployed, no one can alter or delete it and it will always run as programmed. There are no hidden transactions or risk of restricted access like in the traditional financial world. A major difference between DeFi and Traditional Finance is the fact that you hold your money in DeFi, whereas companies and intuitions hold your money in the traditional financial world (DeFi | ethereum.org, 2022). A user of DeFi controls where their money goes and how it is spent, but a traditional finance user must trust institutions to not mismanage their money.

**Applications of DeFi**

Arguably the first DeFi application was Bitcoin, a decentralised peer-to-peer cash payment system that allows users to own, control, and send value to anywhere in the world (DeFi | ethereum.org, 2022). However, the advent of smart contracts on the Ethereum blockchain allowed for a wider range of DeFi applications to emerge. Decentralised Finance on Ethereum allows users to borrow, lend, save, invest, and send money around the globe without the need for costly intermediaries. Some examples of applications include Maker Protocol for borrowing and access to a stable currency, Aave for lending and borrowing, uniswap for token swaps, and dYdX for trading and market prediction.
Money:
For any financial system to operate successfully, a stable form of money must be available. “Money serves to lower transaction costs by bringing about a form of social coordination” (Smit et al., 2015). As a tool of social coordination, money allows the public to make agreements and transact with one another to receive goods and services. In the case of the traditional financial world, we have fiat currencies which rely on the trust of government powers.

Despite the fact that Blockchain technology has provided a remarkable opportunity for adopters to overcome the tight grip centralised financial systems have on the public, there are some reasons why it has not gained widespread financial use. The original objective of Bitcoin was to create a decentralised payment system; however, it falls short of some hurdles on the way to mainstream money because of its fixed supply and speculative nature. In order to be considered a viable form of money, the following characteristics and functions must be provided.

Characteristics of money:

**Durable**

Money should be reusable and maintain its integrity and usefulness for the future. One example of non-durability would be using perishable goods as money. Cryptocurrencies achieve this as blockchain may be decentralised and distributed, meaning no one pint of failure. Additionally, in most blockchains, the full history of transactions is recorded.

**Portable**

Money should be convenient to use. It should have a worthwhile use value or be capable of being divided into convenient quantities.

**Fungible**

Units of money must have relatively uniform qualities so that they are interchangeable with one another.
Stable

The value of money should be relatively constant, whereby the cost of transactions does not fluctuate. A volatile form of money would make it very difficult for users to judge the cost of a transaction as the future value may differ significantly. Arguably, this is the largest deterrent for Bitcoin becoming mainstream money. One great vivid example of this difference in transaction costs is when Laszlo Hanyecz purchased two pizzas for 10,000 Bitcoins, which were worth 41 dollars at the time (Moneycontrol, 2021). However, due to the volatile nature of Bitcoin, the transaction cost Hanyecz $400 million at March 2022 prices, a hefty price to pay for two large pizzas. If Hanyecz had used the USD to purchase these pizzas, it would have only cost him roughly $54 due to inflation.

Functions of money:

Medium of exchange

The primary function of money is a medium of exchange. Money is a widely accepted and recognisable liquid asset used in the settlement of transactions, for example, the US Dollar. The value of money lies with its future purchasing power, or the ability to use it as payment for future transactions (Silver et al., 2021). Although cryptocurrencies facilitate safe and fast transactions and payments, the volatile value has proven to be a hurdle to widescale adoption. This means that it falls behind fiat currency as a medium of exchange.

Unit of account

An important feature of money is the ability to account for goods or services with units of that money. It must provide a common numerical unit of measurement of the market value of goods, services, and other transactions (Silver et al., 2021). Bitcoin facilitates this by being divisible into smaller amounts, with the smallest being a Satoshi or $10^{-8}$ Bitcoins. However, the unstable price of Bitcoin means that accounting for transactions becomes difficult as a profitable transaction may become unprofitable in a short time span.
Store of value

As the value of money is derived from its future usefulness, it should provide a means to store value. This requires the value of the money to be relatively stable. Inflation decreases the purchasing power of fiat currencies, which damages their ability to store value over time. On the other hand, Bitcoin has proved to be a good store of value so far, as it has outperformed inflation.

Stablecoins:

Many cryptocurrencies were created with the goal of improving the current global financial system and better serving the public good. However, the majority fall short of an ideal medium of exchange because of high levels of speculation and volatility. The instability of the value of cryptocurrencies, such as Bitcoin, make them hard to use for everyday transactions. If Alice bought groceries for 1 Bitcoin, worth $50 at the time, and the next day the value rose to $100, she would regret this transaction as she could have saved 0.5 Bitcoin or $50 if she had waited a day to purchase. This problem prevents many cryptocurrencies from becoming mainstream money because the future value cannot be relied upon to be stable, making it hard to judge and account for transactions.

Stablecoins solve this problem by providing a global, open, secure, fast, and stable form of digital money (Stablecoins | ethereum.org, 2022) Built on top of blockchain technology, they derive their value from other stable assets, which allows them to minimize price volatility keep their value relatively consistent. This makes them an ideal form of money as they host all the characteristics and functions of money. The top five stablecoins can be viewed in figure 2.3. There are three main approaches to stablecoin implementations, algorithmic, centralised, and decentralised stablecoins.
Algorithmic stablecoin

Algorithmic stablecoins aim to reduce its price volatility by dynamically changing its supply. They achieve this by creating more supply when the price rises above the target price and destroys some supply when the price drops below the target price. The balances of existing holders are rebased according to the proportion of the total supply they own. The supply will increase if the value rises above the target price and it will shrink if the value drops below (Zhou et al., 2021). The viability of this type of stablecoin is questionable as recently USDN lost its dollar peg (Ghosh, 2022).

Centralised Stablecoin (USDC)

One approach to tackling the problem of a stable cryptocurrency is by backing the stablecoin with fiat currency reserves. To do this, the issuing company must hold significant funds in a bank. It works by taking the users money and depositing it in a bank and then issuing the user with the appropriate amount of cryptocurrency. This 1:1 backing means that in theory holders of this stablecoin can trade their tokens in exchange for fiat currency (Stablecoins | ethereum.org, 2022). Tether or USDT is an example of a centralised stablecoin. “Tether’s claim that a USDT was worth a dollar, is backed by the fact that the
company maintained a bank account in Hong Kong which held a dollar for every USDT in circulation” (O'Mahony, 2021). Despite the huge success and adoption of USDT as a stablecoin, many still question the use of a centralised entity as an intermediary for issuing currency, which erodes the decentralisation of the system. Using a legal entity to facilitate the operation of this stablecoins may require compliance with legal regulations. This may result in lack of access for certain jurisdictions, or loss of user’s funds (O'Mahony, 2021). The low price volatility of a centralised stablecoin depends on the trust placed in the centralised system overseeing the operation.

**Decentralised stablecoin**

This approach to stablecoins uses crypto assets, instead of fiat currency, as collateral backing for the stablecoin. In order to maintain a low price volatility, a decentralised stablecoin usually requires over-collateralisation. This means backing the stablecoin with crypto assets worth more than the stablecoin value. This helps to counteract the volatile nature of the underlying crypto assets because the crypto assets can be sold to cover debt before their value drops too low (Stablecoins | ethereum.org, 2022). The benefits of this type of stablecoin are transparency, decentralisation, and no external custodians. By keeping this stablecoin system on-chain, we avoid the risks associated with centralised entities. One example of a decentralised stablecoin is Dai, an ERC-20 token that is over-collateralised by a wide range of crypto assets. The Dai stablecoin is maintained by the Maker Protocol which will be discussed later.

**IPFS (Interplanetary File Storage)**

The IPFS is a distributed peer-to-peer protocol for storing and accessing data. The backbone of IPFS is Content addressing which involves identifying a piece of content, for example a file, by the cryptographic hash of the content (IPFS Docs, n.d.). This allows for each piece of content to be stored by its content by having a unique address. To access files, the searcher’s computer will ask the network (other nodes) to share the file, rather than asking a central server for the file. The files on IPFS are immutable as any change in the original file would lead to a new cryptographic hash.
Decentralised Autonomous Organisations (DAO)
A Decentralised Autonomous organisation (DAO) is a blockchain native flat organisation that is managed and owned by its members. There is no CEO or central authority, the decisions are made via a voting mechanism and the outcomes are implemented automatically without an intermediary (DAOs | ethereum.org, n.d.). The transparency of a DAOs operation means that members who do not know or trust each other can collaborate on a common goal. Often, membership is based on owning governance tokens. These tokens allow a user to vote on proposals and the voting weight is usually proportional to the amount of governance tokens held. The backbone of a DAO is a smart contract that defines the rules. Once deployed, the rules can only be changed via a successful vote. A proposal could allocate the collective funds or modify its code (Buterin, 2014).

Maker Protocol
The Maker protocol is a decentralised financial application on the Ethereum blockchain that allows anyone to access a stable cryptocurrency, the Dai stablecoin, via over-collateralised borrowing (MakerDAO, 2020). In order to borrow, users must lock crypto assets as collateral into the system to generate debt. Failure to repay the debt will result in the system selling the locked collateral to recover the generated funds. This is similar to the process of using a house as collateral for a mortgage. The value of the collateral must be greater than the level of debt. Users may also use the system to earn savings.

The core design considerations within the Maker protocol are token agnostic, verifiable and modular components (Maker Protocol 101, 2020). The system is indifferent to any token type as the system balances are stored internally within a smart contract called the vat. There are many adapters which deal with the various tokens and update the internal balances regardless of token implementation. The core smart contract, the vat, is designed to be verifiable where each state, such as token balance, is defined and can be proved. This is achieved by having no external dependencies or calls to other smart contracts. The operations within the internal accounting system are kept simple to reduce risk. The Maker protocol is designed to be modular so components and parameters can be upgraded or updated. As the entire system is very complex (see figure 2.4), only the relevant components to this project will be explained.
Figure 2.4: The Maker Protocol System Diagram (MakerDAO Technical Docs, n.d.)

Risk Parameters
The Maker protocol has a few system variables that are adjusted to control various types of risk. There are

- Stability fees
  - The annual percentage rate of interest to be paid for borrowing Dai. Each collateral type has a different stability fee depending on how risky the crypto asset is, for example the collateral asset ETH A has a stability fee of 2.25% (Makerburn.com, 2022). The fees are constantly accumulated per second.

- Debt ceiling
  - The maximum amount of debt that can be generated and exist from a single collateral type.

- Liquidation ratio
  - The minimum acceptable collateral-to-debt ratio required before a vault is deemed too risky.

- Liquidation penalty
A fee added to a vault's total outstanding bad debt of Dai when a liquidation occurs. This helps prevent self-liquidation becoming profitable.

- **Dai Savings rate**
  - Dai that a user does not immediately need can be deposited into the system to earn savings. The Dai savings rate is the return earned from locking Dai in the Dai savings rate module. Similar to the savings rate of a traditional bank account.

**Components of the Maker Protocol:**
This section will discuss the various components that make up the Maker Protocol.

**Dai:**
The Dai contract is the user-facing ERC-20 token contract maintaining the accounting for external Dai balances. The Dai token functions similarly with any other ERC-20 token. This contract can mint, burn, and transfer Dai tokens. The Dai stablecoin is the main product of the Maker Protocol (Maker Protocol 101, 2020). There are two representations of the Dai token, internal Dai and external Dai. As a result of the Maker system being token agnostic, an internal balance is used to account for the Dai tokens within the system. The external Dai, or ERC-20 Dai, is generated by the system but used externally in other dapps. Both of these represent the same currency and are backed by collateral, the main difference appears in where they can be used.

**Economics of Dai:**
Dai is a decentralised, unbiased stablecoin soft-pegged to the United States dollar (MakerDAO, 2020). This means that it aims to keep a consistent value close to the target price of $1. The dollar peg is maintained via shifting the supply of Dai (see figure 2.5). This is achieved via incentive mechanisms. The first of which is the change of risk parameter such as stability fees or Dai savings rate. When the value of Dai drops below $1, MakerDAO can increase the Dai savings rate or decrease the stability fees to encourage users to create more Dai. The opposite is true when the value rises above $1. Additionally, arbitrage seekers can trade an off-target Dai by swapping it for other stablecoins in a module called the peg stability module, that was designed for this purpose.
Collateral Assets
Collateral assets are crypto assets, that have been approved by the governors of the system, to be used as security for loans issued by the Maker Protocol (Maker Protocol 101, 2020). The risk parameters, such as, stability fee, debt ceiling, etc. are individually set for each separate collateral type. Some collateral assets, such as ETH, are broken down into different pools which provide different risk parameters for the same underlying asset. Some examples of utilised collateral assets include Ether, BAT, and Link.

Internal Accounting module:
The Vat is the core accounting module of the Maker Protocol, and it is known as the ‘single source of truth for the system’ (Maker Protocol 101, 2020). It acts as a database as it stores the fundamental primitives, such as the risk parameters, and the internal Dai, collateral, and debt balances. It provides operations to update these values. To ensure security and upgradability of the Maker Protocol the Vat is designed to be functional without external dependencies. Whereas the other modules are designed to be upgradeable and replaceable (Vat – Maker Docs, n.d.).

Peg Stability module:
The peg stability module aims to maintain the dollar peg. It achieves this by facilitating the trade of other stablecoins for the equivalent amount of Dai. For example, users may swap 100 USDC for 100 Dai without taking on any debt or opening a vault. The users of the peg stability module do not retain ownership of the asset they enter into the swap with, unlike a
vault. The peg stability module creates an arbitrage opportunity whereby an undervalued Dai can be traded for a $1 stablecoin or vice versa when the price of Dai rises above a dollar (Peg Stability Module, n.d.).

**Oracles:**

Blockchains and smart contracts face a fundamental limitation – they cannot inherently interact with data and systems existing outside their native environment (Oracles | Chainlink, 2021). Oracles solve this problem by acting as a bridge between the real world and the blockchain. In the context of DeFi, they are third parties that report the price of assets from real-world, “off-chain” sources to “on-chain” smart contracts (Liu et al., 2021).

![Real World Data and Events](image1.png)

![Blockchains](image2.png)

*Figure 2.6: Blockchains cannot connect to real-world data and events on their own (Oracles | Chainlink, 2021)*

Oracles are entities that connect blockchains to the off-chain environment, thereby enabling smart contracts to execute based upon data from the real world. In order for the blockchain network to come to consensus, each node needs every transaction to end up with the same result. This poses a problem when using external APIs that produce varying results, so an oracle is built to post the data onto the blockchain. This way, nodes will use the same immutable data and thus agree on the state transition caused by a transaction (Ashimine et al., 2022). Oracles are made up of a smart contract and some off-chain components that can fetch data and relay it to the smart contract. An example use case, Alice makes a bet with Bob that Team A will defeat team B, so they place the respective ETH into a smart contract.
After the match is complete and a winner is declared, this result is streamed to the smart contract from a trusted oracle. Now the funds will be sent to the winner of the bet.

Oracles are a vital component of the Maker Protocol as the value of a collateral asset is derived from its free market price in USD. An oracle is a trusted entity that provides a collateral assets free market price. An individual oracle module is deployed for each collateral type, where it feeds USD price data from trusted sources to the internal accounting module (Maker Protocol 101, 2020).

Each individual oracle broadcasts its corresponding collateral asset price onto the blockchain. These values are fed to the collateral asset’s oracle module where the list of prices is gathered. The median value of this list is used to update the collateral’s internal price (Median – Maker Docs, n.d.). By using the median price, the system gains a layer of protection as outliers are ignored in the case that an oracle is compromised. Figure 2.7 gives a high level overview of this architecture. There is a delay of one hour to allow the ecosystem to analyse and react accordingly, such as preventing liquidations.

![Figure 2.7: High-level overview of Maker oracle system (Ziu et al., 2021)](image-url)
Vaults:
Maker Vaults are smart contracts which are the core mechanism that allows a user to leverage their collateral assets to generate Dai. Many third parties have created integrations with the Maker Protocol to enable users to easily create and manage vaults, one such example is Oasis. The creation of a vault is straightforward; however, users should have a basic awareness of the Maker protocol to operate them effectively. The generation of Dai locks the collateral in the Maker Vault smart contract, which cannot be released until the Dai debt plus any incurred stability fees are paid back. This process is analogous to a mortgage, where a bank will provide a loan by “locking” ownership rights of a house with them. The debt must be repaid to “free” the bank’s ownership of the house (Maker Protocol 101, 2020).

Maker Vaults are non-custodial, meaning the user interacts directly with their vaults and they have complete and independent control over their vault, generated Dai, and deposited collateral as long as the value of the collateral doesn’t fall below the minimum required level. As a result of the stability fees constantly accumulating, the total debt also increases indefinitely. This means that over time the collateral to debt ratio will decrease, assuming no change in collateral value. There is no obligation to regularly pay back the debt, unlike a traditional loan. However, the locked collateral is always more valuable than the total debt, and a liquidation is likely to cost more to the user than paying back the debt.

Interacting with a Maker Vault:
Lock Collateral:
The first step is creating a vault via an interface such as Oasis. After this the user is free to deposit collateral assets into the vault. The vault is now collateralised and ready to generate Dai. Collateral may be freed from the vault at any stage, for no cost other than gas costs, if no debt has been generated.

Generate Dai:
Once collateralised a user may generate Dai from their vault, which is equivalent to taking a loan. Initiating this transaction will create ERC-20 Dai tokens and transfer them to the owner’s address, while increasing their debt balance within the system. The collateral asset,
ETH, is locked into the vault and may not be freed until the Dai debt plus stability fees has been paid back.

**Fee accrual:**
The Dai debt is constantly accruing every second according to the collateral’s stability fee. For this example, the user has left their vault idle for 1 year and has accrued 100 Dai in stability fees. This is added to the original debt to find the total debt balance. In figure 2.9, the vaults collateralisation ratio, or collateral to debt ratio, has decreased to 167% as a result of the increase in debt. A change in value of the collateral would impact this ratio too.

**Repay Dai:**
The repayment of debt plus stability fees can only be paid in Dai, so a user must retrieve the extra Dai from another source. Completing this transaction will destroy the users external
Dai and simultaneously decreases their outstanding internal debt balance by the same amount.

**Free Collateral**

Once the vault has no outstanding debt, the user can transfer their collateral asset from their vault to their personal address.

In this example scenario, the user comes back after a year and decides to retrieve their collateral. In order to do so, the user must pay back the outstanding Dai they have generated plus the stability fee incurred. Both can only be paid in Dai, so the user must purchase Dai in the secondary market. In figure 2.10 below, Alice repays debt of 500 Dai plus 100 Dai interest, equalling 600 Dai in total, and retrieves her 10 ETH collateral.

![Figure 2.10: the repayment of Dai and freeing of collateral](image)

**Stability Fees / rate module**:

The Maker Protocol is faced with a problem where each Maker vault owes a different amount of stability fees. The total stability fee owed depends on how much Dai was borrowed, how long the Dai has been outstanding, and the stability rate.

To keep track of fees owed, the Maker Protocol would have to calculate and store these figures for each vault, which is inefficient and costly for large numbers of vaults (Rates Module – Maker Docs, n.d.). To avoid this problem the Maker Protocol only stores two values, the normalised debt (art), which is unique for each vault and represents the value of
debt if discounted by the rate to the start time. The other value stored by the system is the compounded per second rate of interest (rate). The rate is calculated by converting the annual stability fee to a compounded per second rate, which is achieved using this formula:

\[
\text{stability rate} = \left( \frac{\text{seconds in a year}}{\sqrt{(1 + \text{stability fee})}}, \text{where seconds in a year} \right)
\]

\[
= 31536000
\]

To find the variable rate for each collateral type at any time, the maintainers of the system must calculate the following formula:

\[
\text{rate} = \text{old rate} \times (1 + \text{stability rate})^{(\text{time since last fee update})}
\]

The normalised debt (art) is calculated by this formula:

\[
\text{art} = \text{old art} \pm \frac{\text{debt drawn/wiped}}{\text{rate}}
\]

By using this convention, anyone can efficiently calculate the total debt of a vault by using the formula below instead of storing each vault’s debt balance.

\[
\text{Total Debt (Dai)} = \text{vault normalised debt} \times \text{collateral type rate}
\]

(Rates Module – Maker Docs, n.d.)

To illustrate with an example (see figure 2.11), if a user draws 20 Dai at time 0, the normalised debt is set as 20. With a stability fee of 50%, the total debt will have grown to 30 Dai in one year. The fees will accumulate indefinitely at a constant rate unless the amount of debt or the stability fee changes. If the user draws an additional 10 Dai at time 1 year, the art is recalculated using the above formula, which gives 20 + 6.67 = 26.67. The total debt can be calculated at year 2 by using the formula above to give 26.67 \times (1.5)^2 = 60 Dai.
Liquidation:

Liquidation is the selling of collateral assets to cover the debt generated by a user’s vault. To ensure the over-collateralisation of the Maker Protocol, liquidations occur when the value of a vault’s collateral drops too low, or the debt grows too large. The protocol classifies these risky vaults by comparing the liquidation ratio to the current collateral-to-debt ratio of a vault. When a vault becomes too risky it is automatically liquidated, where the vault collateral is confiscated and sent to the auction module (Liquidation – Maker Docs, 2022). Here the collateral is auctioned off in a Dutch style auction, where the price starts high and gradually decreases until the bad debt plus liquidation penalty is covered, or all of the collateral has been sold. In figure 2.12 below, the ETH collateral drops in value, from $100 to $80. Thus, the collateral-to-debt ratio drops below the liquidation ratio. 133% < 150%
In figure 2.13 above, Liquidation has occurred, and the 10 ETH collateral has been confiscated and sent to be auctioned. The bad debt to be covered is the outstanding debt plus the liquidation penalty. The current auction process in the Maker Protocol works by a Dutch auction system, where the price starts high and gradually decreases until someone bids. This bid includes a maximum amount of collateral the bidder is willing to purchase. The total bid value is found by multiplying price per collateral by maximum amount of collateral to buy. If this value exceeds the total amount of debt to be covered, the amount of collateral is reduced so that only the exact amount of debt is retrieved. This means that only the necessary amount of collateral is sold off and the rest is returned to the original owner. In the case that a bid’s total value is less than the total debt to be covered and there is collateral remaining, the auction will continue.

**Bidding:**

For the example, the initial price is set at $110 per collateral because the oracle price before liquidation was $100 and the liquidation penalty is 10%. This price gradually falls until it reaches a level that a bidder, Bob, is comfortable with. Bob bids at $85 per collateral with a maximum amount of collateral to purchase at 10 ETH. As the total value of his bid is greater than the debt to be covered 660 Dai, he will not receive the full 10 ETH. To find the amount of collateral Bob will receive we divide the debt to cover by the price per collateral that Bob bid at, this gives us 660/85 = 7.76 ETH. The remaining 2.24 ETH is sent back to the original owner.
vault. The Dai is sent from Bob to the Maker protocol, where the 600 Dai bad debt is written off and the remaining 60 Dai is sent to the surplus buffer as a profit.

![Figure 2.14: Displaying the timeline of Dutch auction](image)

**Keepers:**

Keepers are external actors, usually automated bots, that are incentivised to contribute to decentralised systems. With regards to the Maker Protocol, these keepers are incentivised by arbitrage opportunities to automate certain operations such as Dai peg stability, or liquidations and auctions. Arbitrage opportunities are risk-free profitable trades. “Keepers are market participants that help Dai maintain its Target Price ($1)” (MakerDAO, 2020). Arbitrage opportunities occur when the price of Dai drops below the target price: using the peg stability module, they can buy Dai and swap it for a stablecoin trading at $1, and vice versa when the price rises above the target price. They also monitor the Maker protocol and will trigger liquidation events when the Liquidation Ratio is breached. On top of this, auction keepers will participate in auctions via a pre-set bidding profile to make profit.
Core units
The Maker Foundation was an organisation that created the Maker Protocol and maintained it through its infancy. They directly employed contributors to bootstrap the system. In July 2021, the Maker foundation was dissolved to allow the Maker protocol and MakerDAO to become fully decentralised (Christensen, 2021). The MakerDAO replaced the foundation and core units replaced the internal foundation teams, however many foundation members continuing to work with the protocol. Core units are the decentralised workforce that enable the efficient running of the Maker Protocol. These teams act independently, but with the common goal of improving the Maker protocol. The various core units include protocol engineering, collateral engineering services, and governance communication to name a few. Funds are allocated to the core units to pay for salaries and expenses, by the MakerDAO.

Surplus buffer
The surplus buffer is the collection of all Dai profits of the Maker Protocol. The total amount of Stability fees accrued from all vaults are recorded in the buffer. The main purpose of the buffer is to protect the system by covering the bad debts (Surplus Buffer, n.d.). On top of this, the retained profits are used to compensate the contributors of the system and to pay for maintenance fees, such as oracle costs.

MakerDAO Governance
MakerDAO is the community, consisting of MKR token holders, that manage and govern the protocol. “Through a system of scientific governance involving Executive Voting and Governance Polling, MKR holders manage the Maker Protocol and the financial risks of Dai to ensure its stability, transparency, and efficiency” (MakerDAO, 2020). The MKR token is a governance token used to vote on changes to the Maker protocol, however anyone can submit proposals for a vote by the MakerDAO. Voting weight is proportional to the amount of MKR a voter holds. The proposals usually appear on the forum where the community discuss its impact and ultimately decide if the proposed changes are acceptable. The process follows two stages: firstly, a governance poll is created to gauge community interest, the result of this is non-binding. However, if the governance poll receives positive feedback, then the proposal is converted into executable code called a spell, and an executive vote is held to determine if the proposed changes will modify the live Maker Protocol. Possible
changes could include modifying the risk parameters, adding new collateral types, approving core unit budgets, adding new technical implementations, etc.

An additional function of the MKR token is a safeguard for the Maker Protocol. If collateral auctions do not raise enough Dai to cover a vaults debt, the deficit is converted into Protocol debt. This is accounted for as a loss and must be covered by retained profits from the surplus buffer or through the sale of MKR tokens. If there is not enough Dai in the surplus buffer, a debt auction occurs. During a debt auction, MKR tokens are minted and sold to bidders for Dai (Flopper – Maker Docs). This is the last resort because increasing the circulating MKR will dilute the holdings of existing MakerDAO members. Therefore, MakerDAO members are incentivised to minimise risk and ensure the security of the protocol.
Chapter 3 | Design

Current state:
Currently the technical implementation of Real World Assets on MakerDAO is very limited and the process is vague. These RWA Collaterals are represented by ERC-20 tokens that do not have strong links to the underlying Real World Assets. Although the agreement documents are available for browsing on the forum or IPFS, there is not a clear indication of what the Real World Assets consist of. In terms of technical implementation, there is no oracle system or automatic liquidation/auction system. The value of the collateral asset is determined manually by a trusted party, which makes the system less decentralised. A lot of work is being done to improve and streamline this process within MakerDAO. New proposals are created frequently and core units such as Collateral Engineering Services, Real World Finance, and others continuously work on the process.

Approach
To investigate the viability of using real world assets as collateral within decentralised finance lending protocols, the author developed a proof of concept utilising Ethereum, Non-fungible tokens, the Dai stablecoin, and the Maker Protocol. Creating a system that integrates with the Maker system required the author to gain a deep and broad understanding of the protocol. The Maker Protocol whitepaper, technical documentation, presentations, talks, forum posts, and direct communication with community members and contributors were used to acquire knowledge and an understanding of the complex system.

Design Goals
• Create an innovative way of integrating Real World Assets with the Maker Protocol
• Represent and track Real world assets on the Ethereum blockchain using Non-fungible Tokens (NFT)
• Enable the minting of Dai via locking an NFT as a collateral asset
• Updating value of Collateral Asset within the system
• Initiate an auction of Real World Asset NFT if a loan becomes undercollateralised.
• Implement a functioning auction where the winner is transferred the Real world Asset NFT
System Architecture
The Author implemented and tested the methods and processes using the proof-of-concept architecture detailed below in figure 3.1. However, this architecture has limitations which will be considered.

Proof-of-Concept architecture:
The proof-of-concept system was implemented as an integration to the Maker Protocol, where the Maker system consists of a personal instance of the Maker protocol. Approximately eighty smart contracts had to be configured and deployed to the Görli testnet to replicate the Maker protocol.

Figure 3.1: proof-of-concept architecture
For purposes of clarity the author will illustrate an example use case of the designed system. This example follows Alice, who owns a $25 million piece of real estate and wishes to expand her real estate portfolio. She approaches MakerDAO to borrow against her real estate. MakerDAO assigns a team to value the asset and the valuation is set at $25 million. Her proposal is accepted, and the terms are set as follows: stability fee of 2%, debt ceiling of $20 million, liquidation ratio at 150%. This explanation will follow the process of Alice interacting with her vault, encountering a liquidation event, and participating in an auction.

**NFT Management**

In order to interact and trade real world assets with Decentralised Finance, they must first be brought onto the blockchain. The tokenisation of real world assets is straightforward, assuming ownership of a Non-fungible Token that represents the asset is seen as proof of ownership of the asset in the eyes of the law. As MakerDAO is not a legal entity, it cannot hold legal ownership of assets. The proposed solution to this is creating a legal entity that will hold ownership of these assets and follow orders directly from MakerDAO. For the illustration of the design, the legal entity is referred to is Tokenise LLC.

A ERC-721 Compatible smart contract is created, deployed, and managed on the Ethereum Blockchain by the Tokenise LLC. This contract tracks the various Real World Asset NFTs (RWA NFT) that have been approved for use within the Maker Protocol. This contract allows Tokenise LLC to prove ownership and authenticity of RWA NFTs. When the asset originator, Alice approaches MakerDAO, she allows Tokenise LLC to tokenise her asset. This involves, uploading the relevant data to the Interplanetary file system (IPFS), in Alice’s case, this involves the coordinates of her real estate, the area, the number of lots, etc. Before minting the RWA NFT, this important information must be filed and uploaded to the IPFS, where it cannot be modified. The link received from hashing the contents of this file will become the uniform resource identifier (URI) of the NFT, this ensures that the linked data will not change. When the NFT is minted a tokenId will be generated and ownership will be granted to Alice’s account. Figure 3.2 outlines this structure.
Vault Management

A vault is a tool that lets users deposit collateral and generate Dai. It acts as an interface to the Maker Protocol where all the underlying complexities are abstracted away. Each Maker vault is a smart contract deployed on the Ethereum blockchain that allows the owner to interact directly with the Maker Protocol. For this project the vault allows the use of NFTs, on the condition that the NFT comes from a whitelisted address, has the correct tokenId, and the sender of a transaction is the owner of the vault. These checks prevent malicious actors taking advantage of the Maker Protocol. If these security checks are satisfied, the vault smart contract can enable a user to manage and modify their protocol debt position. The changes to a debt position must adhere to the risk parameters set by Maker governance.

Figure 3.2: high-level overview of NFT management
Lock and Draw

In our example, Alice deposits her RWA NFT into the vault, and proceeds to lock it by generating 15 million Dai. The ownership of the NFT is transferred to the Maker system and in return 15 million Dai is sent to Alice’s address. She may use these funds as she wishes. However, conforming to the function of the Maker protocol, this NFT as the collateral asset cannot be returned to Alice unless she pays back the amount of Dai she has borrowed, plus the stability fees incurred.

Figure 3.3: illustrating ‘locking’ of RWA NFT and withdrawal of Dai

Fee accrual

The fee accrual works the exact same as the existing Maker system. Stability fees are accrued per second and will update when the rates module is called. In our example, Alice takes her 15 million Dai and uses it to fund her new project, a year later her vault has accrued 300,000 Dai in stability fees (15M @ 2%APR). This has decreased her collateral-to-debt ratio as the value of her collateral has not changed.

Figure 3.4: illustrating the fee accrual
**Repay and free NFT**

The final components of vault management are the ability to repay Dai debt and to retrieve the locked real world asset NFT. For our illustration, Alice decides that she would like to retrieve her real world asset NFT and use it within another system. In order to do this, Alice must pay back her outstanding debt, plus accrued stability fees. In this example, Alice will pay back 15.3M Dai to clear her debt. After this, she may free her NFT and use it as she pleases.

![Diagram illustrating the repayment of debt and ‘freeing’ of RWA NFT.](image)

**Figure 3.5: Illustrating the repayment of debt and ‘freeing’ of RWA NFT.**

**Oracle management**

Just like the current Maker system, where the price feeds for crypto collateral assets are sourced off-chain, the price data for real world assets must be sourced off-chain too. The design of this system is very similar to the current Maker system, only the sources change. The sources for each asset class would be different, for example the oracles for real estate would be vastly different from stock market price feeds. An example oracle for a real world asset such as real estate could use local house prices from housing marketplaces such as Zillow to roughly estimate the value of the asset. In practice this process would have to be more closely monitored than the crypto price feeds as it would be hard to get exact value.
Liquidation

Liquidation events occur when the value of the collateralisation ratio of the user’s vault drops below the liquidation ratio. This may occur if the value of debt rises too much, or the value of the underlying collateral drops too low. Keepers of the Maker protocol, constantly monitor the system and automatically liquidate unsafe vaults. In the current system a one hour delay is given to the collateral price updates, which allows users to remedy their situation. In this proof of concept design, a longer delay may be appropriate as real world assets generally hold more utility.

When a liquidation event occurs, the Real world asset NFT gets confiscated from the vault and added to the list of auctions. The outstanding debt remains in the user’s control, but they lose their collateral asset, which in most cases should exceed the value of the outstanding debt. Within the Maker system the outstanding debt is added to the and debt queue until it can be covered by auctions or proceeds from the surplus buffer.
Liquidation Auction

Once a liquidation event occurs, the collateral asset, our real world asset NFT, is automatically transferred to the auction contract. Here anyone can view the asset NFT and bid on it. These auctions follow an English style auction structure where anyone can bid increasing amounts of Dai until the auction expires and the highest bidder receives the NFT. This auction is less complex than the current Maker system as only a whole collateral asset can be auctioned, rather than fractions of the collateral asset. This method relies on an efficient market, where the winning bid will be relatively close to the market value, i.e., no major information asymmetry.
Chapter 4 | Implementation

Overview
This chapter will discuss the implementation of the proof-of-concept, from chapter 3, into a working prototype. Creating a functioning prototype involved uploading data to the IPFS, implementing smart contracts that integrated with the Maker Protocol, and creating a user interface.

Tools:

Solidity:
Solidity is an object-orientated, high level language designed for implementing smart contracts on the Ethereum blockchain. It is heavily influenced by other object-orientated languages such as C++, Python and JavaScript. The programs written in Solidity are compiled into bytecode to be read and run by the Ethereum Virtual Machine (EVM).

Dapptools:
Dapptools is a suite of command line tools and smart contract libraries used for Ethereum smart contract development. The Maker Protocol was developed using this suite, so this project was also developed using the Dapptools suite. The tools from this suite that the author used for this project included dapp and seth. Dapp is a command line tool used to build, test, and deploy smart contracts. Seth is a command line Ethereum client tool that allows automated deployment and testing.

Services:

Infura:
Infura provides fast and reliable access to the Ethereum blockchain through APIs and developer tools. The suite of tools allows for easier blockchain development as it removes the need to run your own node. The API service was used to access and interact with the Ethereum network for this project.

Etherscan:
Etherscan is free to use “block explorer” for the Ethereum blockchain. It acts as a search engine that enables users to discover, view, and verify transactions that have taken place on
the public blockchain. In the context of this project, Etherscan was used to view deployed smart contracts, monitor transactions, and to debug failed transactions.

System Architecture implementation
The implemented system architecture closely resembles the architecture described in figure 3.1. However, this diagram, figure 4.1, highlights the interaction between different elements of the implemented prototype. The frontend user interface was hosted on the authors local computer, which communicating with the Ethereum blockchain via Infura. As this was a prototype, the use of a third party service such as Infura removed the need for running a node. The smart contracts interacted with each other as outlined in the design chapter. The IPFS was used as the storage system for the Real world assets NFTs metadata. To implement this, an IPFS node was run on the local computer which allowed the uploading of documents. In a practical application the running of the node would be carried out by the NFT issuing entity.

![System Architecture Implementation Diagram]

*Figure 4.1: System architecture implementation*
Smart contract implementation
This section outlines the various smart contracts that were implemented to create the functioning prototype system. It will explore the logic and design rationale behind the smart contracts. The functionality of each smart contract closely resembles the corresponding proof-of-concept design diagrams.

Access control modifiers
Smart contracts on the Ethereum blockchain are public and as a result anyone can call a function if they know the address of the smart contract. This behaviour is not desirable in the proof-of-concept because of the security risks associated with modifying the values within the Maker Protocol. For this implementation two access modifiers were used:

Auth – this acted as a layer of protection for the Maker Protocol. It limited administrative operations of the Maker system to trusted addresses, such as the oracle contract.

```solidity
// --- auth ---
mapping (address => uint256) public wards;
function rely(address usr) external auth {
    wards[usr] = 1;
    emit Rely(usr);
}
function deny(address usr) external auth {
    wards[usr] = 0;
    emit Deny(usr);
}
modifier auth {
    require(wards[msg.sender] == 1, "RwaUr/nt-authorize");
}
```

**Figure 4.2: auth access modifier**

Operator – this provided a layer of protection for the user of the vault. It limited operations of the vault to the owner of the vault. Operations include locking collateral, drawing Dai, etc.
Join Adapters

The Maker Protocol was designed to be token agnostic, so any new implementation of a token could be used as collateral in the future. Under the hood, the external tokens that are used as collateral are locked into these adapters. When a token is locked into an adapter, it will send a transaction to the internal accounting system of the protocol to update the users internal collateral balance accordingly, and vice versa when taking collateral out of the system. To integrate NFTs as collateral in the Maker System, a new join adapter that could handle ERC-721 tokens had to be created. The token agnostic design of the Maker Protocol made it far easier to integrate a new token type compared to modifying the internal workings of the system.
Function join(address usr, bytes32 ilk, uint256 tokenId) external auth {
    require(live == 1, "GemJoin/not-live");
    //require collateral initialised
    require(ilkTokenIds[ilk] == tokenId, "GemJoin/incorrect-tokenId");
    // increase vault internal collateral balance by ONE
    vat.slip(ilk, usr, int256(ONE));
    // transfer NFT to this contract
    gem.safeTransferFrom(msg.sender, address(this), tokenId);
    emit Join(usr, tokenId);
}

Function exit(address usr, bytes32 ilk, uint256 tokenId) external auth {
    require(live == 1, "GemJoin/not-live");
    // require collateral initialised
    require(ilkTokenIds[ilk] == tokenId, "GemJoin/incorrect-tokenId");
    // reduce vaults internal collateral balance by ONE
    vat.slip(ilk, msg.sender, -int256(ONE));
    // send nft back to vault
    gem.safeTransferFrom(address(this), usr, tokenId);
    emit Exit(usr, tokenId);
}

Figure 4.4: NFT adapter function to enter NFT as collateral in the system

**Non-fungible token Contract**

The implementation of the Real world asset (RWA) NFT contract utilises the ERC-721 interface. This allowed the creation of RWA NFTs and their required functionality. These RWA NFTs can be transferred from owner to anyone else. Before creating the NFT, the asset’s ownership documentation must be uploaded to the IPFS. When creating a RWA NFT the IPFS link to the ownership documents must be used as the uniform resource identifier (URI) of the NFT. This creates an immutable and persistent link to the underlying asset. As well as this, a unique tokenId is associated with each NFT on creation, which allows each user to easily identify their asset. A simple incrementing tokenId was used to avoid duplication of tokenIds.
Figure 4.5: Real World Asset NFT implementation

In order for smart contract accounts to hold ERC-721 NFTs, they must implement the `ERC721TokenReceiver` interface. This includes the function in figure 4.6, which will return the appropriate value when an NFT is sent to the smart contract. If this function is not present, the transfer will fail. Within the context of this prototype, the `NFTAdapter` contract and the vault contract had to implement this function.

Figure 4.6: NFT receiver interface function
Vault management.
The vault smart contract is the main port of interaction between the user and the Maker protocol. The provides functions to lock & free a RWA NFT, generate debt by drawing Dai, and repaying Dai debt. In order to receive and hold ERC-721 NFTs this smart contract has to implement the ERC721TokenReceiver interface. As this smart contract can modify the internal vat balances, appropriate access modifiers are of huge importance. The auth modifier is used to ensure only the authorised parties can set up the vault, and the operator modifier is used to ensure only the owner of the vault can interact with it.

There are four main functions within the vault smart contract:

**Lock collateral**
This function facilitates the insertion of the users RWA NFT to the Maker system. To account for the internal collateral balance, the NFT is locked in the NFTAdapter and the internal collateral balance of the vault is increased by one. The internal collateral balance represents the amount of collateral a vault has inside the token agnostic Maker system. After this stage the user is able to generate Dai debt.

```solidity
// --- cdp operation ---
// n.b. that the operator must bring the gem
function lock() external operator {
    //send nft to thisUr
    ERC721(gemJoin.gem()).safeTransferFrom(msg.sender, address(this), tokenId);
    //approve NftAdapter
    ERC721(gemJoin.gem()).approve(address(gemJoin), tokenId);
    //send nft to nftAdapter
gemJoin.join(address(this), ilk, tokenId);
    //increase vault collateral balance by 1
    vat.frob(ilk, address(this), address(this), address(this), int(WAD), 0);
    emit lock(msg.sender, WAD);
}
```

*Figure 4.7: Vault function to lock RWA NFT*

**Free collateral**
The unlocking of collateral from the vault was implemented as a reversal of the locking function. The users internal collateral balance is decremented by one, dependent on the
vault having no internal debt balance. A call to the \textit{NFTAdapter} will remove the internal NFT collateral balance and transfer ownership of the NFT to the users address.

This is when the user wants to retain full ownership of their NFT. They can unlock their NFT or asset from the vault and Maker protocol overall, by repaying all of the vaults debt plus accrued stability fees. Their “external” ERC-20 Dai is transferred to the \textit{DaiJoin} smart contract and destroyed. This transfers the Dai into internal Dai which is used to reduce the internal debt balance. When the debt of a vault is zero the NFT can be withdrawn. The \textit{NFTAdapter} will send the NFT to the users address. Now the user cannot increase debt or draw more Dai unless then lock their collateral again.

```
// n.b. that the operator takes the gem
// and might not be the same operator who brought the gem
function free() external operator {
  //decrease collateral balance by 1
  vat.frob(ilk, address(this), address(this), address(this), -int(WAD), 0);
  //send nft to operator
  gemJoin.exit(msg.sender, ilk, tokenId);
  emit Free(msg.sender, WAD);
}
```

\textit{Figure 4.8: Vault free collateral function}

\textbf{Generate Dai}

This action may occur when the NFT is locked in the vault, i.e., the vault is collateralised. A user may draw up to the predetermined credit ceiling, ‘line’. Additionally, the collateral value to total debt must not drop below the liquidation ratio. When the user draws Dai the vault will call \textit{Jug.drip} to update the accumulated fees and then the normalised internal debt balance of the vault will be incremented. The \textit{DaiJoin} contract is called to mint ERC-20 Dai and transfer them to the users address. Now this external Dai can be used however the user sees fit, including interactions outside of the Maker system.
Repay Dai

This function will be called when the user wants to deleverage their vault. Firstly, it calls drip on the Jug contract to update the accumulated fees and uses the discount rate to calculate the actual debt on the vault. Then it attempts to transfer the specified amount of ERC-20 Dai from the user to the system. The vaults internal debt balance is reduced by the same amount.
Oracle Management

One limitation of the implementation was the oracle system. The author discovered that development of an oracle system for pricing real world assets was too time consuming and was out of scope for this project. However, the author did implement a manual version of the oracle functionality whereby the author could update the value of the collateral asset. This consisted of directly inputting the updated value rather than finding the median value of a price feed. In an ideal implementation, the sources of collateral values could be data feeds from trusted real estate markets. The following functions were implemented for the oracle module.

Update collateral value

The initial value of the asset would have to be judged by an independent source, instructed by MakerDAO, however it could be verified by multiple sources to increase trust. This initial value would have to be manually entered when the new collateral asset type is added the system via a successful governance proposal, a spell. The process of updating the collateral value in the prototype involved passing a new value into the smart contract from a whitelisted address. This action would send a transaction to the relevant smart contract, the spot, to update the collateral value in the vat.

```
// --- valuation adjustment ---
function bump(bytes32 ilk, uint256 val) external auth {
    DSVValue pip = DSVValue(ilks[ilk].pip);
    require(address(pip) != address(0), "RwaOracle/unknown-ilk");
    //update collateral value
    pip.poke(bytes32(val));
    // pull updated price into vat
    spotter.poke(ilk);
    emit Bump(ilk, val);
}
```

Figure 4.11: Oracle function to modify value

Monitor vault health

This function checks whether a vault is healthy or not. A vault is considered healthy if its collateral to debt ratio is above the liquidation ratio. This function uses the adjusted collateral value to determine the health of the vault. The adjusted value is calculated by dividing the collateral value by the liquidation ratio, so for a value of $25 million, and liquidation ratio of 125%, the adjusted value would be $20 million. In other words, the
adjusted value is the threshold of Dai debt available before a vault will be liquidated. This allows for a simple number comparison which will return a Boolean representing if the vault is healthy or not.

```solidity
// --- liquidation check ---
// to be called by off-chain parties (e.g., a trustee) to check the condition of the vault
function good(bytes32 ilk) external view returns (bool) {
    require(ilks[ilk].pip != address(0), "RwaOracle/unknown-ilk");
    // get vault debt and adjusted value
    (uint256 Art, uint256 rate, uint256 spot,) = vat.ilks(ilk);
    // calculate outstanding debt
    uint256 outstandingDebt = (mul(Art, rate));
    if(mul(spot, WAD) > outstandingDebt){
        return true;
    } else{
        return False;
    }
}
```

*Figure 4.12: Vault health check function*

**Liquidation**

This event occurs when a vault has become unsafe or unhealthy. In an ideal implementation, events like this would be caught and triggered by keepers of the Maker Protocol. In the prototype system this event was triggered manually as bot creation and maintenance was unfeasible. However, the author was able to replicate the actions required to liquidate a vault. The first step in this process is check if the vault is unhealthy, by calling the check health function. This is required to confiscate the vault. The debt balance is sent to the vow as bad debt, and the collateral is confiscated from the vault and sent to the auction contract. The vaults debt and collateral balances are set to zero. Then the auction is started, with the appropriate parameters, such as expiry time, debt to cover, id, etc.
The auction contract houses all the functionality related to the liquidation auctions. When a vault has been liquidated, the NFT gets transferred to the auction contract and a new auction starts. The list of auctions is accessible to anyone. These liquidation auctions work on an English style, where the bidding starts low and increases until expiry. When the auction has expired, the highest bidder is the winner and will receive the RWA NFT.

Each auction has these parameters:

- **Bid expiry:** time when last bid will expire
- **Auction Expiry:** time when auction will expire
Tab: goal debt coverage

Guy: current highest bidder

Beg: minimum percentage increase on previous bid

tokenId: identifier for RWA NFT

**User interface Implementation:**

A simple user friendly interface was created to display and enable the functionality of the system while abstracting away all the complexities of the underlying smart contracts. The UI included a vault page, NFT page, Oracle page and Auction page. The entire app was created using a web development framework, Next.js, and the integrations with the smart contracts were possible due to the web3.js framework. This UI is in no means a final product but just a tool used to better explain the system during the presentation.

![Figure 4.14: Vault user interface](image-url)
Chapter 5|Evaluation:

Difficulties encountered

Throughout this project many difficulties were encountered. Firstly, the author had no previous experience with blockchain development. The first step in this learning curve was to research the Ethereum and Bitcoin whitepapers. In order to gain a more in depth understanding of the Ethereum blockchain the author read the Ethereum book (Antonopoulos and Wood, 2018), and gained hands on experience following tutorials. The programming language used to development the prototype, Solidity, is based on Javascript which the author had some experience in, so this did not pose a huge challenge. The usage of Command line interface development tools such as seth required research and learning.

The second difficulty encountered was understanding the Maker protocol. As previously describe in chapter 2, the Maker protocol is a very complex system with many intertwined components. To add to this difficulty, the smart contracts are written in “Dainese”, where the naming convention is very short and simple but usually give no indication of the function of the code. For example, the function to start an auction is called “kick”, instead of “startAuction”. The technical documentation provided clarity of functionality and use cases. In order to create a functioning integration prototype the author had to gain a deep understanding of the system. Some barriers to the creation of this prototype included out of date documentation. However, only small pieces of information were missing or modified. Contributors and members of MakerDAO kindly helped to fill in these gaps.

The Görli testnet was used to develop the prototype as a result of gas fees being too expensive on the Ethereum production environment. Although the Maker protocol is deployed on this testnet, it is a replica used for testing of the actual system. The test version is permissioned and admin privileges are held by the protocol engineering core unit. As an external developer it was not feasible to integrate with their test environment in case of damages to their system. To enable full control and the ability to experiment with the system, the author deployed their own instance of the Maker Protocol on the Görli testnet.
Evaluation of prototype:

In its current state, the implemented prototype enables a user to utilise their real world assets as collateral on the Maker protocol. It provides a novel approach of using non-fungible tokens as the representation of real world assets on the Ethereum blockchain.

Looking back at the entirety of this project, the author is very satisfied with the outcome. This journey provided an opportunity to gain hands on experience working with blockchain development. The knowledge gained from exploring the Ethereum blockchain and the Maker Protocol are invaluable to the author. A deep understanding of the complex Maker Protocol was gained and displayed with the integration of the proof-of-concept system. The creation of a user friendly interface enabled the abstraction of the complexities associated with the underlying smart contracts.

Evaluation of design goals:

Procedure to represent real world asset on-chain:

High success –

The prototype enabled the representation of real world assets as non-fungible tokens on the Ethereum blockchain. The related data was intrinsically linked via an immutable storage system. While the technical implementation of this goal was a success, this implementation holds large assumptions about the regulatory operation.

Vault management:

High success

The prototype produced included a procedure for interacting with a Maker vault that held a real world asset, represented by an NFT, as collateral. The system replicated the management of a vault regardless of

Oracle system:

Medium success
The interaction between the internal accounting system and an external pricing feed was implemented successfully. However, one limitation of this goal was the lack of a reliable price source. For this prototype, the collateral pricing was entered manually.

**Liquidations and Auctions:**

High success

The implementation of an English style auction was a success. The initiation of auctions worked well as the appropriate liquidation checks were put in place. One limitation of this functionality was the lack of an automated keeper system. However, the functioning Maker keepers could easily integrate with this system in a practical application.

**Limitations**

The feasibility and practicality of the proposed solution is met with several limitations. Some of these limitations will be outlined in this section.

**Legal issues**

Due to the broad scope of the area this project is based on, some limitations were encountered and acknowledged. The area of tokenising real world assets is still in its infancy stage, where it is faced with many challenges, especially regulatory challenges (Wang and Nixon, 2021). The regulatory issues arise when a user tries to claim ownership of a physical asset. Within the blockchain, this process is very easy and verifiable, however taking delivery of the physical asset may be difficult if the original party fights the transfer of ownership. For example, if a house was used as collateral for a loan that was liquidated it may be difficult to remove the original owners. This process is a grey area and fraught with ethical issues. These problems are likely why MakerDAO is planning for onboarding large institutions as “customers”, where a costly legal battle is less likely to occur.
**Oracle system.**
Arguably the largest limitation of this prototype is the lack of a functioning oracle system. In a practical application of this design, the oracle system would require many hours of research, investigating, and verifying sources of price feeds. However, it plays a vital role as the safety of the Maker protocol and the stability of the Dai stablecoin cannot be ensured without trusted and accurate sources of price data. On top of this, unlike crypto currency markets, there are no accurate pricing mechanisms for many real world assets. Due to time constraints, a mocked oracle system was not created.

**Keeper network**
The efficient running of the Maker protocol relies on the constant monitoring by keepers. This prototype did not include a keeper network, which lead to the manual monitoring of the system by the author. However, the existing keeper network would effectively monitor this integrated prototype because the internal values mirror that of the existing Maker Protocol.
Chapter 6|Conclusion

Conclusion

The work produced in this project reflects an insight into the world of decentralised finance. Firstly, knowledge of the underlying technologies, Ethereum and blockchain were researched. The architecture and economics of a decentralised financial system were investigated through the Maker Protocol. A protocol that enables access to a stable cryptocurrency via over-collateralised lending. Throughout the authors research it was evident that stablecoins play the vital role of ensuring stability in a volatile crypto world. In the authors opinion, the Maker Protocol provides a valuable service as it enables users to utilise the value of their crypto assets without selling them. Leveraging assets to gain liquid capital can be a powerful tool in the hands of an experienced investor.

The proof-of-concept designed took a unique approach to allowing the use of real world assets as collateral on the Maker Protocol. Non-fungible tokens enabled the representation of such real world assets on the Ethereum blockchain. The open source nature of Decentralised finance applications, such as the Maker Protocol, allowed the author to explore and experiment with integrations of the prototype. Building on top of an already established protocol enabled the creation of a functioning prototype in a short period of time. The area of real world asset tokenisation is in its infancy stage; however, this project displayed a novel approach to the problem. The evaluation of this project demonstrates the assumptions and limitation of this implemented prototype.

Future work

The areas surrounding this project are so vast and interesting. Constant innovation is progressing each day as the Decentralised Finance, and the wider blockchain community, keep building. If given more time and a chance to start the project over again the author would have taken a few different approaches. Firstly, other asset representations would have been explored, such as yield bearing assets and/or more specifically the tokenisation of stocks. The approach of using non-fungible tokens provides an abstract representation of
many assets however it lacked precise details of more complex asset types. Another approach that the author could have taken is the deeper exploration of other real world asset protocols. Although the investigation of the Maker Protocol provided deep knowledge of the system, other protocols may provide innovative solutions to the real world asset tokenisation problem.

Additionally, within the design of the proof-of-concept, the author would have liked to have implemented a keeper bot to automatically monitor the operations of the prototype system. Although not necessary, this implementation could have been used to explore arbitrage trading and auction bidding strategies. Another feature within the design that could have been implemented given more time, was the oracle system. Potentially a functioning oracle system could be created using existing APIs. An interesting approach would be reading stock prices from existing exchanges or querying housing markets for the average local house price.

Within the context of the tokenisation of real world asset, future work should focus on regulation. The technical implementation of this process is straightforward as explored in this project; however, the regulatory side has lacked behind. The main problem occurs when a tokenise asset is traded, who ensures the transfer of such asset? There are many ethical issues regarding the instant liquidation of tokenised assets that should be addressed. Within the Maker Protocol, future work should focus on ensuring the transparency of reporting and operations.

In the authors opinion, there are two future pathways for the DeFi ecosystem, with valid points on either side. The first path would be the regulatory path. In order to integrate with the Traditional financial system, DeFi protocols would have to adhere to some regulations, potentially removing the anonymity associated with DeFi. This integration would provide quick growth and open a huge market to trillions of dollars. The other pathway the DeFi ecosystem could take is the separate path from the traditional system. By avoiding the regulations of large central authorities DeFi could provide a better alternative to the current system. Building the new financial system from the ground up could better serve the public good. However, the growth and adoption of this system would take a lot longer than integrating with the traditional system.
Appendix

Link to github repository: https://github.com/chris-staunton/fypProject
Bibliography


