Imperial College London

BEng Individual Project

Imperial College London

Department of Computing

Creating a Token Economy for Film Rights (FilmChain and the Centre for Cryptocurrency Research and Engineering)

Author:
Theerathat Pornprinya

Supervisor:
Knottenbelt, William

Second Marker:
Thomas Heinis

June 25, 2020

Abstract

There is no question that the current financial industry gained much attention from blockchain and its application. FilmChain, the first startup company to automate blockchain platform for collecting and allocating incomes from films. The platform distributes the revenues and splits amongst various types of stakeholders using the revenue waterfall concept. Each stakeholder would own a virtual wallet for visualising and withdrawing revenues that is received.

In this project, an online marketplace that is developed on blockchain specifically for FilmChain to be able to trade a stakeholder's position in the revenue waterfall of a movie. The marketplace allows FilmChain's clients to be able to buy and sell stakeholders tokens that is owned from the films in exchange with money. This FilmChain secondary marketplace not just only provides buy and sell, but clients are also able to make offers at a certain price for a specific stakeholder in case the listed price from the seller is not suitable. List of offers can be viewed and accepted as well as withdrawal if it is desired. This platform also provides full support to user experience including How it works? page and articles to learn more on the features of each of the application. Additionally, clients are also welcome to message via the Contact Us page if a problem or a question arises. Smart contract infrastructure for tokens and the marketplace are implemented as a back end for the project. The features for the marketplace that allows clients to interact with the platform is implemented using React Component front end. The created platform will allow FilmChain to extend its product line to support client trading of stakeholders.

Acknowledgements

I would like to thank my supervisor, William Knottenbelt, and my second marker, Thomas Heinis for providing me an opportunity to work on this project and his guidance throughout the course of the project. Special thanks go to Irina Albita, co-founder of FilmChain and Big Couch, Olivia Stannah, full stack developer and Evangelos Barakos, lead software engineer to also give me the opportunity for this project and provide ideas for the secondary market which enlightens my passion for the film industry. Throughout our meetings every week, which inspired me to develop a clear vision of the system and also to grow as an engineer. Also, help me develop my creativity and critical thinking skills for the challenges encountered. And finally, my parents for their support throughout my study. Without their support, this project would not have been possible.

Business Confidentiality

In this report, there exists some confidential information that is private for FilmChain. Distribution of this report is rigorously not allowed without permission from FilmChain.

Contents

Chapter 1

Introduction

1.1 Motivation

FilmChain is the first automatic blockchain platform that collects, allocates and analyses revenues for films, TV, and digital on the blockchain. Upon obtaining consent from stakeholders, the platform creates and automatically executes the recoupment schedules (a contract specifying how incoming revenue should be split amongst various tiers of rights holders, structured in what is termed a "revenue waterfall") using smart contracts and private databases. The platform enables governance for all stakeholders, and visualises the process for all the parties: producers, investors, directors, actors, crew etc. When new incoming revenues are split, each stakeholder has an individual wallet where they can see and withdraw their revenues instantly, creating automatic statements. The robust architecture caters for complex studio budgets as well as micro-budgets videos.

1.2 Objectives

The primary objective of the project is to create a token economy for film rights on the Ethereum blockchain. FilmChain is creating two types of tokens. One class is revenue token (representing incoming revenue received from e.g. cinema receipts) while the second class is a non-fungible token (NFT) (representing ownership of rights).

FilmChain provides clients with the dashboard to manage and monitor the economic flows through the use of the blockchain ledger. This results in real-time revenue explorer to track money flows in a swift. As a result, the stakeholders could withdraw money when they receives one into their bank account, however, there are still possibilities that the stakeholders may want to exchange their rights with the money straight away without waiting for the recoupment waterfall or if they already withdrew their shares from the revenue waterfall and their rights still has equity and still own the percentage of the film, one might want to sell this as a value to someone instead of waiting for the films to get profit. As a result, FilmChain and our supervisor have seen an opportunity in this situation to capitalise a software with a secondary market for stakeholders of the films to be able to trade their ownership rights with a value similar to selling stocks in the stock market. The new owner will now own the share that has been sold and all the incomes of the films after all of the stakeholders in the revenue waterfall has been recouped will be their profits according to how much the share is.

The project consists of several subobjectives:

1. Research in which tokens are appropriate for representing shares in the revenue streams of films, (e.g. ERC-20 vs ERC-721)

2. Determine how these tokens get used to represent the recoupment schedule/revenue waterfall

3. Develop a system where stakeholders can visualise their ownership tokens and their value
4. Develop a secondary market for ownership tokens to be traded

This project can be divided into two sub projects with the first focusing on the blockchain and smart contract technology and the second on the decentralised application. I also research into the Hollywood Stock Exchange which is an online stock platform that uses information to conduct research in the film industry.

1.2.1 Blockchain and Smart Contract

The first step for solving this problem is to understand what are blockchain tokens and the different kinds of tokens. Next is to analyse which token is more suitable to the project. Then understand what is a smart contract and what functionality it has to deploy on a blockchain. The task is to implement the smart contract for the secondary market using Solidity smart contract language which would then be deployed on the blockchain. The smart contract should provide basic functionalities for transactions, transferring ownership rights, withdraw money, place bids, etc. It should also have a unit testing to ensure it is secure against attacks.

1.2.2 Decentralised Application

The second sub-project is to develop a decentralised application (dApp) which is essentially a web-based distributed app that enables human interaction with the contract such as visualising the different ownership rights that are listed by stakeholders on the platform and it's list prices. The user should be able to explore the info that relates to a specific ownership rights and able to place bids or buy it now. When the deal is made, the system should automatically transfer the money from the buyer to the seller and the ownership rights should be transferred to the buyer. This dApp front end must connect to the Solidity smart contract backend using Trufle, Gnache and Metamask. The whole end-to-end process then must be tested with edge cases on the local blockchain before deploying to the real blockchain.

Chapter 2

Background

In this chapter, it will be discussed of FilmChain and its technical design of the front end and back end in order to achieve a platform the is compatible with FilmChain from this project. This chapter also includes Blockchain and smart contracts which are the main backbone for building this decentralised marketplace application and the technology of Ethereum. Also, describing the concepts of different types of tokens in order to be able to justify the ownership rights for each stakeholder. Next will give an insight of active projects that has a similar idea call Hollywood Stock Exchange and conclude with the suitable programming tools for this project.

2.1 FilmChain

FilmChain is the platform building on the blockchain purpose of collecting and distributing revenues and profits of a film to stakeholders. Revenues are split in the "Revenue Waterfall" financial structure. Every stakeholder must first sign a CAMA (collection and management agreement) that describes the distribution process and structure of the funds coming from the revenues. FilmChain's platform has a web based application that for the distribution of revenues and each client has their virtual wallet for recouping the revenues. In the backend, Ethereum smart contract is used as a pipeline and the logic for revenue allocation implemented.

2.1.1 Revenue Waterfall

This is a hierarchical structure for representing income flow distribution priority tranches amongst stakeholders of different types. When the money flows into a tranche(priority level), each stakeholder in a tranche will get distributed equally until all the required amount has been recouped in this tranche level. Then the rest of the money will flow into next tranches until every stakeholder gets recouped. Equity stakeholders are stakeholders that will only be recouped once the film breaks even and other stakeholders get recouped in the revenue waterfall. Equity stakeholders will receive the funds as a predefined percentage of the film.

2.1.2 FilmChain's Technology

FilmChain platform design gravitates around 3 main concepts which are agility, scalability and transparency. FilmChain platform is built using a wide range of technologies for its front and back end. It is important to understand each tool in order to build a compatible project successfully.

FilmChain database compose of 3 main technologies as follows:

- GraphQl - an API query language,
- Prisma - Database ORM, and

- Express - a backend framework.

For the front end, 3 main technologies are used as follows:

- React - use as the main front end libraries for building web application
- Apollo - use as a front end client for Graphql
- Styled Components - using CSS for styling, but makes it easier and quicker design implementation

FilmChain platform runs on blockchain with 3 major technologies as follows:

- Ethereum - a blockchain technology for deploying smart contracts
- Solidity - Ethereum programming language for writing smart contracts
- OpenZeppelin in Solidity - a library to build secure smart contracts

FilmChain is currently running on a private local blockchain, but will be deployed on the public blockchain in the very near future.

2.2 Blockchain

As a blockchain project, this section would provide an insight of the blockchain technology and its integration with Ethereum and smart contracts.

Over the past few years, the absorption of blockchain technology into the financial industries has been a very significant factor. More and more applications in the service are continuously developing towards blockchain applications which make the blockchain technology rapidly growing at the same time.

This fast development of blockchain technology and blockchain application started to transform the financial industries. One of the major examples is Bitcoin, but not only that, blockchain applications range from systems that process money exchanging to an equity trading marketplace.

2.2.1 Basics of Blockchain

A blockchain Michael J.W. Rennock, 2018 is a centralised application and it is based on the Distributed Ledger Technology(DLT) where all transactions could be recorded publicly and privately. Blockchain technology encourages transactions without the requirement of a middleman therefore your data is private from the server. After the transaction is approved, this new block is added to the chain of transaction and every transaction cannot be reverted. As a result, this provides a transparent environment and the authentication of every transaction.

Distributed Ledger Technology is relying on the decentralisation and the distribution of the network. All of the transactions are secured and recorded by the network of computers instead of having one central server.

Blockchain innovation draws a lot of attention from a wide range of industries, however, the primary industry that uses most of the technology is the financial industry. Not only because of the Bitcoin cryptocurrency which is very well-known, there are few other reasons than that. Blockchain reduces substantial processes and the massive expenses in the business. Also, it records the truthful owner where even in the financial industry finds it very hard to find the truthful owner of an asset.

For the validation of every transaction, one of the two consensus that is used is called Proof of Work. Members of the network called miners subsequently validate the transactions and collect the transaction fees before contending to add its block to the blockchain network. The second consensus that is used for validation is called Proof of Stake. In this consensus, members of the

network called validators each own a percentage of the block by referencing the digital money it owns. In the validation process, the chance of successfully validating is determined by this percentage.

Blockchain networks are divided into 2 types: Permissioned and Permissionless blockchain.

- The first type is the Permissioned blockchain. It can be referred to as a private blockchain where only the member of the network can create transactions.
- The second type is the Permissionless blockchain. This can be referred to as a public or open source blockchain because it is open to all public users to make transactions. One of the major examples is Bitcoin where anyone can transact using bitcoin.

In the blockchain world, virtual currency is used instead of normal currency. Virtual currency is similar to the normal currency apart from it having its representation digitally where it still provides the functionality of trading. However, virtual currency is still not a flat currency, therefore it is not perceived as a lawful delicate by the government yet. In the creation of digital currency, both digital tokens and digital coins are issued by the DAO and may own the right to be traded in the secondary market or refunded. Any possibilities of trading are accepted ranging from the exchanges to real money or to other tokens as well.

Crytocurrency is considered as a virtual currency. Cryptography is used for securing instead of a middleman, which leads to rises of cryptocurrencies including Bitcoin, Ethereum, Ripple.

2.2.2 Key Characteristics

In summary, blockchain is defined by 4 main characteristics which are decentralised, persistent, auditable and anonymous.

- Traditionally, when a transaction is to be made, it needs to go through a centralised server for example a bank. The major disadvantages for this is that in every transaction, there will exist an overhead. However, in blockchain application, it removes the need for this centralised server and allows 2 members to transact without the need to pay for overhead.
- Every transaction that is made via blockchain is recorded in blocks where it is not possible for individuals to alter them. This provides the application to be persistent.
- Every transaction on the blockchain are transparent. Since all transactions as well as the owner of the transaction are recorded on a block, it makes every transaction traceable.
- In the blockchain environment, no real identity address is recorded since there does not exist a centralised server. This means that the data privacy of every user is protected. Each user would have an address associated with them before making a transaction or one can generate an address themselves.

2.2.3 Blockchain and Smart Contracts

The concept of combining protocols, and user interface provides an agreement for a contract. This concept was presented by Szabo and became very well-known with its integration with blockchain which provides applications that can be used to control the ownership right of an asset. This cutting edge approach might be replacing other real world application that deals with asset or ownership rights either it is tangible or intangible, such as banks in the future.

Another major application of smart contract is used with Ethereum. The idea is originally proposed by Buterin but extends the concept of Bitcoin to support a decentralised server application. The idea is to remove the necessity of centralised server which is used for transactions in the past to provide more transparency and security as well as reducing the overhead cost using the concept of smart contract via a blockchain.

Since blockchain records all transactions, this means that everything can be tracable and referred to. This is essentially a database that provides a potential to store information for both tangible and intangible assets.

Blockchain is an innovative concept that is going to transform the business world in various aspects. Blockchain could be used to keep a record for every transaction history or any trade history that occurs in its application. The concept can also be extended at the governmental level. Blockchain could be utilised to use the observing and controlling of specific undertakings, for example, the election, the registry to the piece of land or any other applications.

Blockchain could be referred as a database that is managed by several computers since it is a peer to peer network consisting of several computers and allows individuals to create a transaction. Once the transaction is made, it will go through a verification process and will then be recorded in a block once it has been verified. This makes blockchain a trust-able application since all information are publicly available and cannot be hidden once the transaction is completed.

Blockchain removes the need for a centralised server for every transaction. Instead, it acts as a trustful third party which is more transparent and allows the two entities to contract with each other safely by entrusting this neutral decentralised server. This also prevents any conflicts that might be happening from two different unknown entities.

Not only that, blockchain is trying to achieve all functionalities that banks nowadays could do, for example, transferring money between two individuals by providing a trustful environment. We could take a simple example from the real world to refer to. Nowadays, if one person wants to send money to another, the process needs to be done by a bank. In this case, the bank is acting as a centralised server that contains all the processing information, including the amount of money transfer, transaction time, etc. The process that banks would do is simply programmed the system to change the balance of both sender and receiver, but the processing time might take a few minutes, hours, or days depending on the type of transfer made. It also costs some overhead in the process as well, such as commission fee. However, everything depends on a bank where the two users have no access or control over them. So the question arises what if this transaction got altered. Since only the bank has access to all the information, what would happen.

This is where blockchain comes into place. As it can be seen from the example above, users have no access or control over the information at all because everything is kept with the bank. Blockchain removes the need for the centralised server, which in this case is a bank, and provide a decentralised system instead. Users have control over the process instead of the bank being the intermediary and all the transactions are recorded on the network, which leads to a more transparent and higher security transactions. Blockchain could be used in other banking aspects as well not only transferring money. For this reason, blockchain becomes a more relevant accompany in the financial industries and currently growing in this field.

Many enterprises are currently changing to use blockchain instead of the normal banking transactions. Blockchain would reduce the redundant time consumption that would not be necessary if done by the network. For example, if blockchain is to use for paying credit cards, it would reduce the settlement processes from days to immediately. This has already shown why businesses are changing to use blockchain.

2.2.4 Applications of Blockchain and Future Trends

Blockchain innovation also play a numerous parts in our everyday life, not just only in the financial industry. Ranging from the food industry, health care, automobile industry in commercial business, has started to integrate using blockchain applications to tackle the problems in their fields thus providing persistent and decentralisation.

2.2.5 Application in Commercial Business

The first major application is cryptocurrencies. Cryptography is used to to ensure all security aspects of every transaction occurring in the system when trading especially Bitcoin, etc. When using blockchain, enterprises are making transaction without the need for an intermediary medium

like a bank, therefore, these applications need to ensure all securities for trading and repayment. Blockchain applications could include the secondary market for trading equities amongst the members of the network. Additionally, the blockchain application could also be applied to insurance fields where properties or assets are registered in the network, and while a transaction or a trade is happening, everything is recorded into the network and cannot be altered.

2.2.6 Non-Financial Applications

One of the application is in the music industry. Blockchain allows the determination of the owner or right ownership easier than the general record. Blockchain stores the ownership rights as well as documents that are used to store the timestamp and information regarding to the right on the blockchain in a decentralised manner.

Another major application of blockchain is used in the food industry. Food safety is becoming a number one concern for national health and the need for it to be traceable. Therefore, blockchain is decided to be used since it provides the benefits of everything being traceable. Starting from the process of using sensors and RFID to collect data and utilise it using the BlockchainDB.

2.2.7 Future Trends

AS it can be seen that blockchain has become more relevant in many industries nowadays since it removes the need of a centralised trusted server. There are many advantages using blockchain application ranging from securities to reducing the cost. Therefore, the growth of blockchain is becoming even more rapidly nowadays and in the future.

2.3 Smart Contracts

Nick Szabo proposed an idea of smart contract, a digital software contract that acts as an agreement between two entities, which is based on the real world contract. For example, in a vending machine environment, the buyer can be the ownership of the item inside the vending machine by simply paying into the vending machine. The vending machine provides a contract with a user for that no one would be able to own the goods inside of it if no money is given to this contract. Therefore, after a transaction into this contract has occurred, the ownership right of the item inside will be transferred to the one who pays for it.

Not only transferring ownership rights of an item inside the vending machine, he extended this concept to be used in the digital world where a more complicated transaction could be solved using this idea. Ranging from trading or negotiating for an equity shares to the contract of allocating the owner of an asset.

The integration of blockchain and smart contract bring many advantages. Apart from reducing the cost of overhead for removing the centralised server, blockchain also makes the transaction faster because it is automated so working 24/7 comparing to the original way via a bank, which would take an amount of time for transferring money abroad. This has already shown that using blockchain would be much more efficient in the business aspects as well as reducing the cost overall.

However, the concept of blockchain is still very new. Also, the implementation for it is continuously developing at a rapid rate, but still remains very complex and there are still some bugs regarding to it. But in several years, time would give it a chance to integrate and advance it to the government level.

Also, another major loop hole of blockchain that needs to be fixed is anonymity. Anonymity is considered as an advantage as the description above, but also leads to major problem when it comes to cryptocurrencies. Criminals could use this gap of anonymity to hide their true identity and carry out drug trafficking or other illegal actions.

Therefore, smart contracts Liehuang Zhu, 2019 come in handy to prevent such scenario. Since

2 distrust entities are supported in a transaction by blockchain, smart contract that is executed on the blockchain would record and verify the signer of the contract for both entities and allowing both entities to review the contract before transacting it into a public blockchain.

Smart contract is used broadly in many blockchain aspect since it offers the functionality of every step in the chain. For example, in a trading scenario, smart contract would record every transaction made, including offering or bids or any rejections. Also, providing real-time feedback on the current status for the entire transaction system end to end.

A simple example of smart contract would be a contract providing deposit, trade and withdraw functionalities. These 3 functionalities can be referred to the process similar to a banking transaction. When a deposit function is called, an amount of money would be deposited into the smart contract. A trade functionality can happen with a condition, for example, a can be traded with 2bs and if a trade function is called with an amount of 1 a, 2bs will be given in return. Lastly, for the withdraw function, once the withdraw function is called, the contract would return all of the money the caller if he is the owner of the money.

All of the transactions can be called continuously in a chain. One example might be calling deposit then make a trade then withdraw the return from that trade. All of these transactions are recorded and needs to be verified before the final return money could be withdrawn to that user. This could be further enhanced in a more complex situation which leads to the reason why a smart contract plays an important role in blockchain.

2.3.1 Ethereum Smart Contract

Ethereum is launched in July 2015 and is the primary framework that uses the concept of Turing complete and extends the limitations of Bitcoin to highlight the possibilities to create autonomous software and agents. Smart contracts are built with ethereum, where there are 2 possible nodes to build on. The first one is Ethereum Virtual Machine or EVM is used for executing smart contracts. The second one is the miner's node where the transactions in the contract are pushed onto the blockchain after executed.

The usage of smart contract has been already well spread with applications ranging from equity trading to a Kickstarter. Ethereum allows individuals to create an application that is both secure and anonymous as well as removing the need for a third party, this application is called a decentralised application (dApp). This open up opportunities to create an application with different functionalities with all records being kept if one might think that the current one is not as good as expected. So many models could be tested until the best one is found.

The extension of smart contract from Bitcoin changed the design of the old proof of work concept. In smart contract, the algorithm chooses to not favour those in the mining pools. As a result, ethereum transaction is faster and removes the boundary of using cryptocurrency. Ethereum has its own execution currency called gas, which can be thought of an amount of money to pay for each transaction to occur. The more complex the function is, the more gas need to be paid especially in loops. However, there are currently many developments which provide no gas cost which can be seen in a private blockchain network.

2.3.2 Security

The smart contract is a relatively new concept where the developer is developing it at a rapid rate. However, new products are added and new bugs might occur which makes the whole concept to still be very vulnerable. One study case is a reentrancy bug which allows hackers to hijack and interact with the ERC20 token contract and calling transfer multiple times. Every time this function is called, several ethers(unit for ethereum) are stolen. This concept of attack was very similar to the DAO attack in the past. As it can be seen that even when time past, some bugs that might occur in the past could be happening again due to the lack of time and the nature of continuously developing project.

When developing a smart contract, the priority is to provide users with all security aspects and test all of the vulnerabilities that might be occurring before deploying and having to go through

the huge lose in the attacks.

2.4 Blockchain Tokens

A resource that can be digitally transferred between two individuals/accounts in the blockchain ecosystem is called token. The vast majority of these tokens are issued on the Ethereum blockchain. Most of the time each token would have a specific property and classification regarding to it depending on the properties when the token is created. The token can belong to a blockchain like Bitcoin or can be hosted on an existing blockchain via smart contract or could even be deployed on the private blockchain. There are a large number of interests on deploying it on an existing blockchain because it is cheaper and could be controlled easier than creating a new one.

Blockchain tokens could be used in many different aspects ranging from currency to security. Blockchain removes the need of a middle man so open to opportunity to create a border-less transfer between two entities and make it possible for a transparent trade to occur.

Blockchain tokens Chen, 2018 consist of two major types. The first type is currency or a cryptocurrency because blockchain is built on a virtual currency of cryptocurrency. Blockchain uses a peer to peer network that allows users to be able to transfer tokens for example a Bitcoin. Another example is the Ethereum blockchain, which allows client to send or receive ether (the currency in Ethereum blockchain) as well as purchasing in with fiat currency. In ethereum, the computational cost of executing a function would be determined by the gas value which the functions are called by smart contract and its decentralised platform network.

Tokens are built inside smart contract and are standardised using the ERC-20 token standard, which requires the basic functionalities for example transfer or send money between accounts on the blockchain network. There are many different forms of tokens with their specific properties and functionalities. The most common tokens are fungible token and non-fungible token.

Fungible tokens can refer to as a share, dollar, Bitcoin, etc. Whereas non-fungible token can be referred to as an art piece, person's avatar, game avatar, etc. To define which type of a token by determining its use value character of an asset rather than its technical characteristics.

2.5 Fungible Tokens, ERC-20

ERC20s Zoup, 2018 can be used in many different ways, since the values remained fixed against a state currency. According to a dictionary, the definition of a fungible asset is: (especially of goods) being of such nature or kind as to be freely exchangeable or replaceable, in whole or in part for another of like nature or kind. Also, the definition of fungibility according to Investopedia is being able to interchange or transfer the same type of individual goods or assets. Being fungibility means that both assets or goods have equal values. The creation of ERC-20 tokens plays a large part in the Ethereum world since it allows digital assets to be created, transfer, etc.

Fungible token can be compared to a token used to play the rollercoaster in a fair. The token for playing rollercoaster only has the value (being able to get on the ride) inside that fair to allow individuals to go on a ride or a place inside the fair. Similar to a fair token, a company can sell goods and services and only allows to be purchased using its token managing in blockchain.

Examples of projects that issue their own tokens on top of the Ethereum blockchain to provide specific services from each platform are as follows:

- Augur is essentially a prediction market on the outcome of an event where user will be paid back in Augur tokens if the bet is correct.
- Filecoin is a blockchain-based storage network, which in 2017, they raised up to $257 million of Filecoin token sale
- Golem is a platform where users can purchase idle computing power of remote computers with Golem token to use in their own personal PC.

- Harbor, a platform for trading securities such as real estates, fine art, equity, etc. using tokens
- Basic Attention Toke is the use of token to power a decentralised advertisement exchange on the blockchain
- GIFTO is a digital platform for gifting market using their own GIFTO token.

All of the examples above use Ethereum blockchain as their backbone. For example, there are already many existing prediction markets, but what makes Augur unique is its decentralised system where bets will be paid out automatically based on smart contracts and the system run on Ethereum rather having a centralised authority. This results in a more secure and diverse environment for the bets.

The ERC-20 token is a standardised token that are agreed by the community to make transferring between 2 types of token possible. Imagine having 2 completely different token, without a standard, how could one be interchanged with the other. The ERC-20 token provides basic functionalities and simple guidelines for a token to follow.

There are also 3 types of theoretical token to be considered:

- Utility tokens: it is a token that allow access to the publisher services
- Security tokens: similar functionalities to non-fungible token, but added extra security to verify each transaction before being transacted. Can be in the form of equities or shares, etc.
- Commodity tokens: use as a virtual currency

2.6 Non-Fungible Tokens, ERC-721

A non-fungible Chevet, 2018 asset must be unique, irreplaceable and non-interchangeable.

Crypto asset becomes popular amongst other blockchain usage from the boom of Rare Pepes. Matt Furie created a Meme that went viral about a green frog call Pepe which is originally in a comic series. The first image comprised in internet clients making Pepe character images in different scenarios and end up providing the community with with a Rare Pepe. Since it is a rare artwork, a blockchain system was created to provide users with this scarcity, allowing them to buy, sell or trade Rare Pepes which can be seen on this website (http://rarepepedirectory.com). The platform is a decentralised blockchain platform with each Rare Pepes linking with tokens that user can buy.

The process of how this works on the website is as follows:

- Rare Pepe is created on the website (http://rarepepedirectory.com).
- The system decides if the created Rare Pepe is considered to be rare enough to be featured on the website
- If the system verifies to be rare, then the created Rare Pepe image will be listed for sales on the website
- Since it is rare, only a limited amount of this Rare Pepe image is listed and normal users can buy this token that is linked to the Rare Pepe image
- The token linked to Rare Pepe defines the proof of ownership. It is not possible to fake a token or create a new one because the system needs to verify all tokens in the network. Therefore, owning a token means that that user have right of that token.

The token is launched only in blockchain and provides users with a secure environment for the ownership rights for each Rare Pepe image. This concept of ownership rights could be extended to use for other purposes or applications, for example a trading marketplace.

2.6.1 Ownership

The concept of managing rights for digital assets was created before. A concept calls Digital Rights Management (DRM) is used to manage and control the rights ownership and restriction for digital art. It is essential to prohibit individuals from copying online digital content from other sources, especially in the digital art environment and the creative industry. Many methods have been used for restrictions of access. For example, in an online game, users are forced to install the game platform for download and need to register with a username and purchased the copy to be able to download and access its content. Another example would be to download a local copy of a music to a mobile device where users must create an account and link to a payment method and it will specify the price of a song for each download to give the access rights to the user to the digital piece. However, by doing this, the owner of the music platform or game would have too much power of controlling users. One of the main incidents that went viral was Amazon removed thousands of e-book without letting users notice. It could be summarised to 3 major concerns against DRM which are

- publisher could monitor the activities of users in its platform which raised against the concert of data privacy
- publisher gains too much power as can be seen from the Amazon example above where the publisher has the rights to do anything which the users might not be notified before
- publisher should limit how much content is exposed to the general users that do not own the right for the product comparison to the user that purchased the product

Today, advance possession of digital ownership has been enhanced to be very similar to physical right ownership. Using DRMs would provide ownership principle and making owning and transferring digital ownership more secure where a specialised developer need to be in controlled. This is where blockchain would do a perfect job of providing a standard for trading and securing the transaction.

2.6.2 ERC-721 standard, Cryptokitties

ERC-721 is a new standard for token creation which is developed on top of the ERC-20 standard. It is similar to the ERC-20 standard that provides a framework for users to be able to create/transfer/destroy of digital assets on a blockchain network. However, the ERC-721 token is used for non-fungible token and could be tracked on the network.

A significant example of platforms that based on the ERC-721 token is Cryptokitties. Crytokitties is launched on November 2017 by Axiom Zen which is essentially a blockchain based collection game where users can buy, breed or trade the digital cats. Each cat would have a unique specific trait depending on its parent where this makes each cat interesting in a way. The random traits are depending on the creation smart contract and new traits are bred on the platform every 15 minutes. The concept of owning a digital cat is similar to that of the Rare Pepes where a buyer will buy a specific token that determines to be this specific digital cat. Owning this token would mean the ownership right for this cat. This concept of trading is also very similar to Bitcoin but in this case, each token cat has a very specific trait which is considered by their DNA for determining its appearance (skin color, size, etc.) which is embedded in the contract code. If one cat combines with another one and produce a child, that child cat would then inherit the DNA from its parent (the two combined digital cats) and has characteristics of its parent. This characteristic would then be stored on the blockchain network in the form of token. This token determines the proof of ownership if a user owns this token on the blockchain network.

As mentioned before, only the token that determines the specific traits and the ownership and signature of the owner would be stored on the blockchain network whereas the images for each digital cat would be kept elsewhere. The success of Cryptokitties has gained much attention for non-fungible token and open up the possibilities for other collectible inspired applications or game of possession that could be deployed on the blackchain instead. However, since it was the first online system, the development team needs to put much effort into making this platform reliable

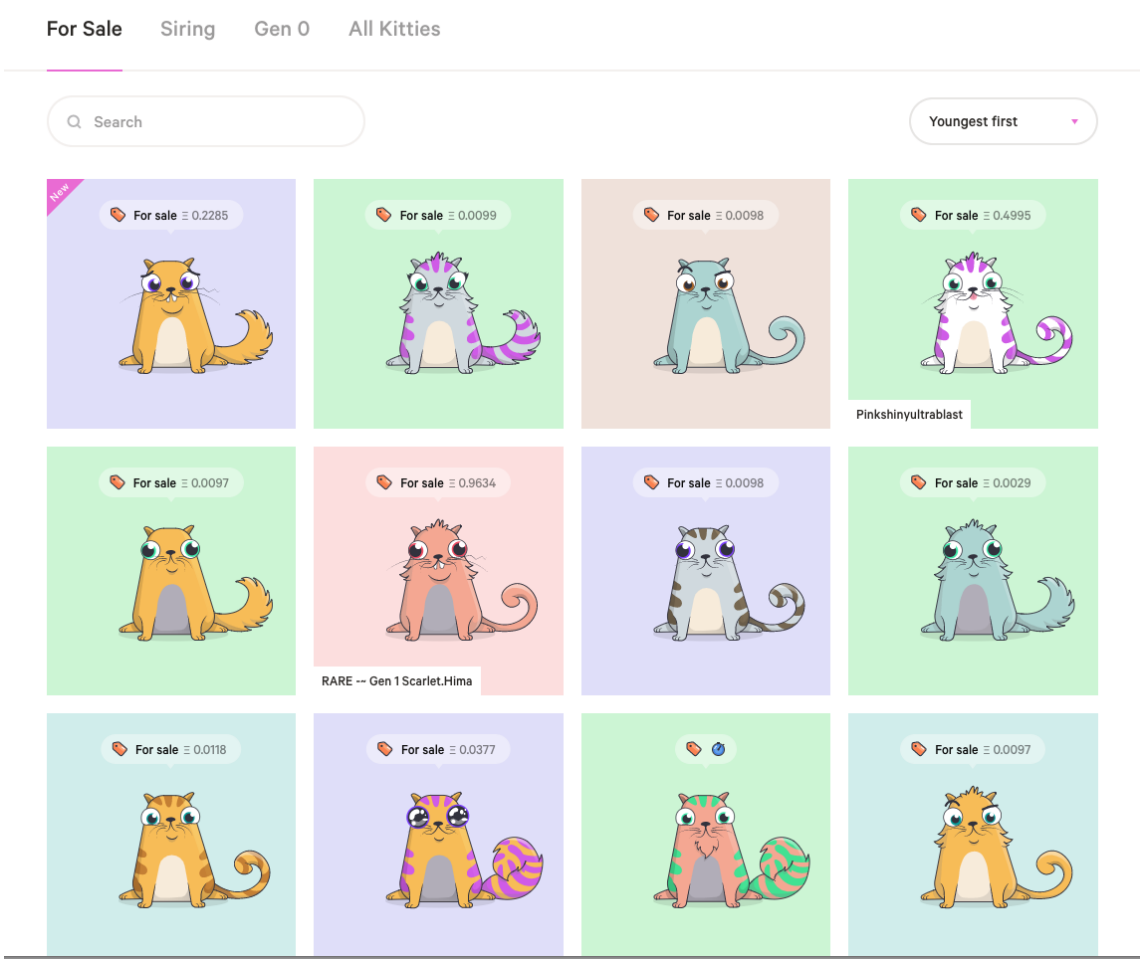

Figure 2.1: Examples of Cryptokitties

and trustful for people to spend money online for digital assets. This has already proved that the integration of the blockchain platform makes it possible for item to consider as an asset.

2.6.3 Non-Fungible Tokens Limits

Even though that Cryptokitties is a boom but there are still not many regular users on the platform.

One of the major concerns of what it is lacking is that for every art piece, an owner would want to actually own the art piece. In this case of Rare Pepes, they are rare, however, it still lack the sense of really owning the art piece or a real ownership. The real question arises what does it mean to own this piece of art? Is it just being able to download the art piece digitally whilst comparing to the real painting where one could actually hang up on a wall and enjoy looking at the work.

The second concern is about legal issues. Similar to the real world, each country has their own law. As well as the digital art piece that needs to take this into account where a law need to play a part in how the token is being able to transfer between one another and what would happen in the case of a fraud. Or what is the limit that publisher can monitor activities of a user in its platform and ensure user data privacy from leaking.

Lastly, blockchain and cryptocurrency are still a continuously developing environment, where bugs could appear anytime and there are still several ongoing issues of owning a crypto asset.

2.6.4 Security Tokens, ERC-1411

Security Tokens coinhouse, n.d. are based on the ERC-20 tokens. However, it focus on a securer aspects and the law. Similarly to the ERC-20 tokens, these tokens can be used to represent the financial data, such as debts, equity or shares, etc. Additionally, it could be traded or other functionality that the normal financial system could do such as investing. However, when it comes to trading or transfer of these tokens, it requires a third person to consent to the law of managing digital asset before authorising the transaction. These tokens overcome the legal issue that is mentioned above from owning a digital artwork where the piece is managed and created by a smart contract.

2.6.5 What is Solidity?

Solidity Fernandez, n.d. is an object oriented computer language inspired from the popular computer languages such as Python, C++ and Javascript which is created solely for creation of smart contract ranging from the electoral system to a secondary market for trading equities.

In solidity, there are a few type values. The first type is a boolean which has the value of true and false. The second type is integer. There are 2 types of integer, unsigned int(uint) and signed int(int). In solidity, it is able to specify the size of the integer ranging from 8 to 256 bits. For example, uint8, uint16, ..., uint256. If the size of the int or the uint is not specified, it will automatically convert to the size of 256 bits. So uint = uint256 and int = int256.

The second type in solidity is bytes. Bytes has a size of 1 to 32 bytes which is bytes1, ..., bytes8, ..., bytes32. However, it is allowed to not specify the size of the bytes which will initiate to an array of dynamic size. The third type is a string. A string is considered as array of dynamic size similar to bytes, but the difference is that in the creation of a string, the system will consume more gas from creating the dynamic size. The final type is an address. The address is bytes of size 20 and to start with 0x.

2.7 Hollywood Stock Exchange

The Hollywood Stock Exchange Ting, 2005 is an online marketplace similar to the Virtual Stock Market and is used as a research tool and study case. In Hollywood Stock Exchange, people invest their money on their favourite movies, movie stars or their favourite music artists. The concept is

Figure 2.2: Movie Stocks

to produce a game for users to win or lose money out of their predictions as well as the company gaining the data about what to predict how successful the outcome of the movie would be when it's on air. Since it is based on financial markets, the user would like to buy low in price, but sell as high as possible, therefore, when the user is a confident investor, one will make a move on the platform. Recently, Hollywood Stock Exchange launched a prediction market for the health care industry as well.

At that time, new stocks called IPO are continuously issued, which adds the brand spanking new security to the exchange and stocks stocks are constantly getting cashed out with payoffs. It is free to sign up for an account of Hollywood Stock Exchange. After signing up, users will receive an HSX account with $2 million HSX inside of its account. As a guideline, users can rely on what the box office has reviewed on each specific movie. The platform will only allow trading to happen initially on the release day and setting the stock price equal to the expectation of the outcome of the first four weeks after release day. A price might already be shown on the website before the release day to represent what expectation the movie is in real time but there is no guarantee of success of it. Movie stocks can also be affected by the Hollywood events or news happening at that moment. One example is a news about Arnold Schwarzenegger being elected as a government which makes the stock of the Terminator 4 to immediately drop 10 percent after the news. In a transaction or a trade, the platform will deduct 1% commission fee from the account.

2.7.1 Movie Stock

Hollywood Stock Exchange is essentially a prediction market where users make predictions on how successful a movie one think will be and invest with an amount of money. This will only be determined by the amount of money that a movie made during its first 4 weeks after release day. On the d-day (4 weeks after), the platform will calculate the total money made during the time and for every $1,000,000 is equal to $1 HSX. Users are open to invest since the development stage until the release day of the movie in the theatre.

Users have many options to invest on ranging from their favourite actor, singer or director. Algorithm calculating how much money a star of a film would make at the box office is determined by the market structure of demands and supplies and how. The box office would then get this number and list it in the Starborn card, an example is illustrated in the Figure above. In some special occasions, stocks could be released for example, in an Oscar event. In some cases, there could be a highly anticipated price depending on what a movie is expected and how the market reacts to the release of a movie.

2.7.2 The Payoff Method

Four weeks after the release day HSX would remove the current movie stock from the market. The total amount of money made is determined by the revenue made and the platform will pay out $1 HSX for every $1,000,000.

The formula is as follows:

$$D_m, T_1 = \frac{Z_m, T_1}{1,000,000} \tag{2.1}$$

where

Zm, T1 is the amount of money made for a specific movie after 4 weeks from the release day M is the index of movie reference to the platform

The Hollywood Stock Exchange only allows marketplaces to happen between weekdays(Monday-Friday) on the first week of the release day. Movie normally releases on Friday where all the prices and prediction would then be paused if there is going to be a commemoration event during the weekends to see how the market reacts. After the event, the new price of a movie stock is calculated based on the feedback of the event calculation with the prediction of the outcome of the movie. For users that make predictions before Friday and after the event, major increase would then be shown publicly in ranking systems.

2.7.3 Daily Summary

The stock prices for movies on the Hollywood Stock Exchange platform are listed online and update daily using an XSLT script. Its job is to gain information for a specific stock and calculate the stock index value and records it in the stock index history.

Chapter 3

Overview

FilmChain Marketplace is a secondary market that is built on the Ethereum blockchain network. The purpose for this marketplace is to be able for its user to trade their shares of the revenue streams of film and exchange for money or vice versa. The secure transaction happens via MetaMask and every transaction are recorded on the blockchain.

Users can access the platform online via web application specifically called decentralised application where they can start a trade by browsing through available stakeholders that are listed for sales on the market on the homepage or to list their own stakeholders for sales onto the market in exchange of money. In a trade, platform allows clients to choose between 2 options, to buy it now or to make an offer. Every transaction has a 2% processing fee for using the FilmChain marketplace platform. This gives client more choices to choose when buying a specific stakeholder. Buy it now essentially means that the client agrees to buy the stakeholder at the listed price. By making an offer, the client can specify a comfortable price where if the seller has agreed and accept the offer, that price would be the sales price for this stakeholder. A list of stakeholders are displayed on the View Offers page where the highest offer is displayed at the top and has an accepted button associated with it. Once the sales are finalised, either from the offer or buy now, if the client has made offer and it is not accepted, the client can withdraw the money from the withdrawal page. A list of stakeholders that a client owns are displayed on the profile page where it is required to login first.

This FilmChain Marketplace project is implemented using Solidity smart contract language backend and a React Javascript front end using web3 as the pipeline for connecting the front and backend.

Chapter 4

Project Management

This chapter will discuss of an overview of this project has been managed including technologies that enhance the quality of communication and project development since project management is one of the most important aspect to successfully reach the target in this project.

4.1 Technologies

2 main communication platform is used for tracking the progress of the project and to communicate with both supervisors and FilmChain members regularly.

4.1.1 Slack

Slack is a great collaboration hub, especially for teamwork. Slack provides workspace and communication channels for members to send messages, upload files, comments, etc. It also provides features for private messages as well as group messages to all members in the workspace. A schedule call is set up as a weekly basis to keep track and update the progress status of the project to FilmChain members. Slack call is used as a main presentation and communication methods for a meeting and it is capable up sharing the screen in the case of showing the front end update or bug issues. Feedback and comments are received every week to improve the project and to set up a goal for the upcoming week of what needs to be done. When facing a problem, a message can be dropped down in the channel and FilmChain member would try to guide what might be the case of it for me to figure it out by myself. This communication channel is one of the most important tools in succeeding this project.

4.1.2 Trello

Trello is used as a management tool to visualise the current status of the project. It is divided into 3 columns which are To Do, In Progress and Deployment. To Do column contains the list of requirements in order to fulfill the goal of the project. It is a very important task to understand the project in the beginning in order to divide the requirements into tasks. Any new bugs or some more functionalities that might arise in the feedback from communicating with FilmChain team members will be added as a new card in the To Do column to reflect the new progress that need to be done. In Progress column is used for displaying tasks in progress and will be moved to the Deployment column once it is finished. In the end, all the cards should ideally be moved to Deployment section and this would display all the bugs that occur on this project and all the tasks done. These cards help to keep track of the project and ensure that all the bugs are debugged before deployment.

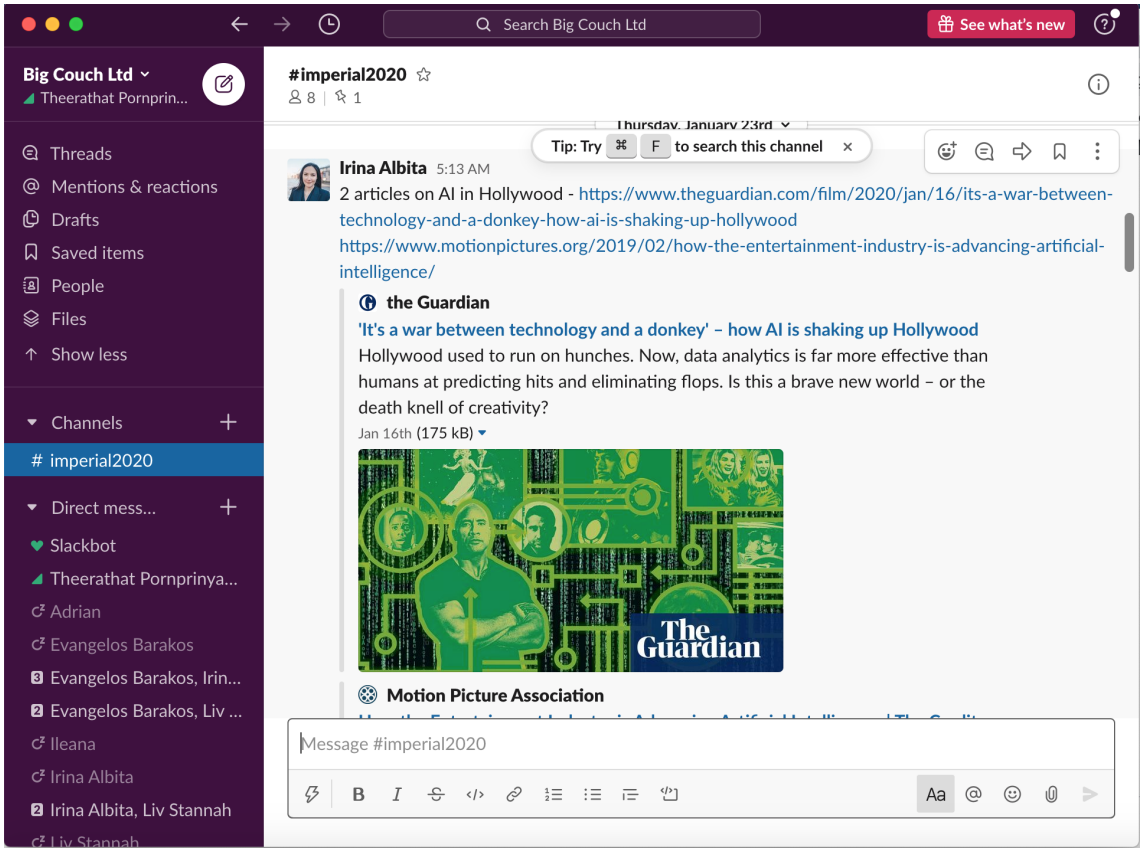

Figure 4.1: Slack Channel

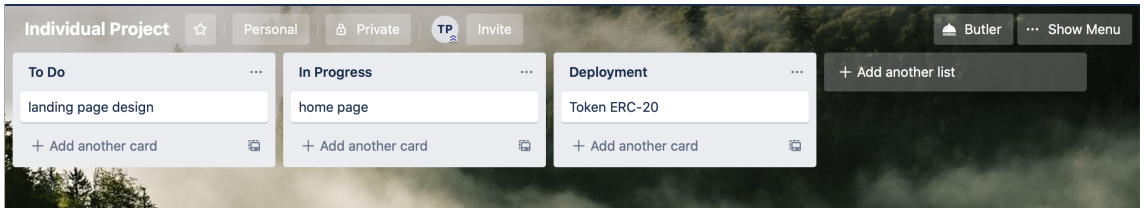

Figure 4.2: Trello Management Structure

4.1.3 Git

Git is a version control that is used to update and track changes in source code for the development. Since this project is considered to be very large, using git will provide support for backlogs and branching systems for different feature's implementation. Git is also used as a communication channel with FilmChain members to see the update progress because a shared git project is used. Also, FilmChain members can also give comments and feedback as well as having an easier time helping when a bug is struggling with because the code is shared. Code updates are pushed inside the branches with an understandable commit message in order to keep track of each commit. After a feature is implemented, it will be merged to the master branch and any conflicts would be checked before merging to ensure that the overall code would not break.

4.1.4 Whimsical

Whimsical is used as a design platform for creating flowcharts and mock up wire-frames. It is substantial to give the reader a clear understanding of how it looks visually and what is the flow in the platform, Whimsical makes the design of web-pages easier and faster.

4.1.5 Programming Code Styles

Along with testing the features of the implementation, how the code appears to visually be a major concern. Comments are added to function to specify the purpose of the implemented function as well as data structure and output. Emit messages are noticed when deploying, destroying or cancelling. Meaningful variable name and good indentations are used to keep the code implementation clean. This provides clarity when a client might want to see the source code of the project that might be deployed in the future.

Chapter 5

Design

In the first step of the project, after understanding the goal of this project and read through some documentations, I came up with an idea how to layout and create end-to-end decentralised application.

5.1 Back End

In this section, technologies and tools that will be used for implementation of the back end will be discussed ranging from smart contract, Truffle, Ganache, MetaMask, Infura and Truffle HD Wallet Provider. At the end of the section, data structure and code design will be talked about.

5.1.1 Smart Contract

The main goal for the first half of the project is to create a solidity smart contract back-end in order to cope with all the transaction activities and functionalities of a marketplace to be able to trade the stakeholders on the platform.

A question might arise why I chose to use smart contract as the back-end of the application. Smart contract and blockchain removes the need for a third party central server which makes the platform more transparent and prevents conflicts and security issues as well.

As can be seen from several blockchain projects, for example, a kickstarter project, kickstarter is a third party platform that allows supporters to donate an amount of money to a desired project which will get a reward in return. However, this requires both the supporter and the developer to trust this third party kickstarter software as well as paying the commission fee.

Instead, if a smart contract is used then if a target amount of money has been raised for a project, the funds can be automatically transfer directly to the developer and if the project is successful, then the reward will be automatically transferred to individual supporters.

This already answered why smart contract is the best choice for my project in creating a secondary market for trading stakeholders where user can list a stakeholder on the platform and if it is sold to another user, the stakeholder will automatically transferred to the new owner and the funds will be automatically transferred to the seller. Also, smart contract is immutable meaning that when the contract is deployed, it cannot be altered.

Truffle Sahu, n.d.

Truffle Suite provides a framework for smart contract development for the Ethereum Virtual Machine. Truffle suite consists of Truffle, Ganache and Drizzle where in this project, I use only two tools from the above which are Truffle and Ganache. Truffle can be thought of a all in one development framework where it supports from implementation, libraries, deployments to testings.

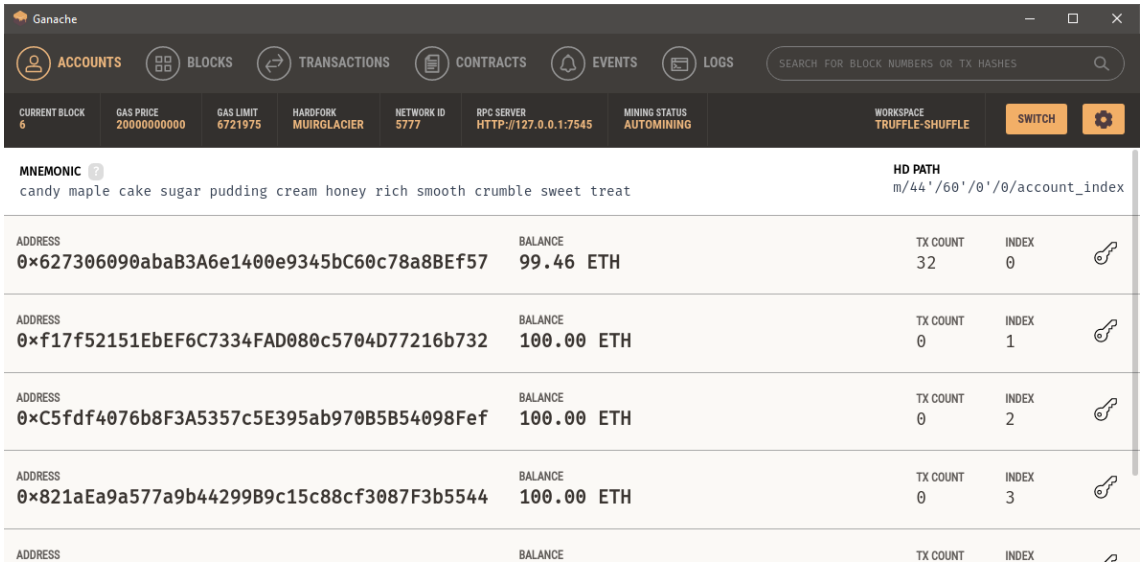

Figure 5.1: Ganache Examples

It handles contract artefacts and support automated tests. Truffle also manages the network artefacts for the decentralised application. In this project, the test is written in javascript to test out the deployment and functionalities of the contract. In this project, Mocha automated contract testing framework is used.

Ganache is a one-click blockchain that provides all the visual mnemonic and the list of accounts in the local blockchain network. Ganache makes it easier to visualise what's happening when a contract is deployed as well as a live-test smart contract.

Drizzle provides libraries that are useful for developing the front end application that connects with a smart contract backend.

The structure of this project is similar to a truffle project where the 3 main folders are contracts, tests and migrations. Specifically for this project, I create a new folder called "ethereum" to handle all the contracts related work including build folder to store the json files when the contracts are compiled. Ethereum folder also stores the compile and deploy script for the smart contracts which is essentially the migration path.

Test folder of the project stores all the related testing where truffle has a built-in testing framework that is suitable for writing tests in Javascript. In this project, javascript and Mocha testing framework is used to test the functionalities of the contracts. The mocha testing framework makes it possible to write synchronous and asynchronous tests on the network and run on Node.js or the browser. Mocha allows a Test Driven Development approach which consists of a following steps:

1. Writing a test to test a function
2. Run the test
3. Implement the function
4. Fix the errors until it gets to correct functionality
5. Repeat the steps for new function

However, in this project BDD style-interface is used. A pattern for BDD testing style for the mocha testing framework is shown in the figure below:

MetaMask

MetaMask Choi, n.d. is a web browser extension that allows user to interact with a decentralised application. MetaMask is based on a concept of Distributed Web. Distributed Web is built on the

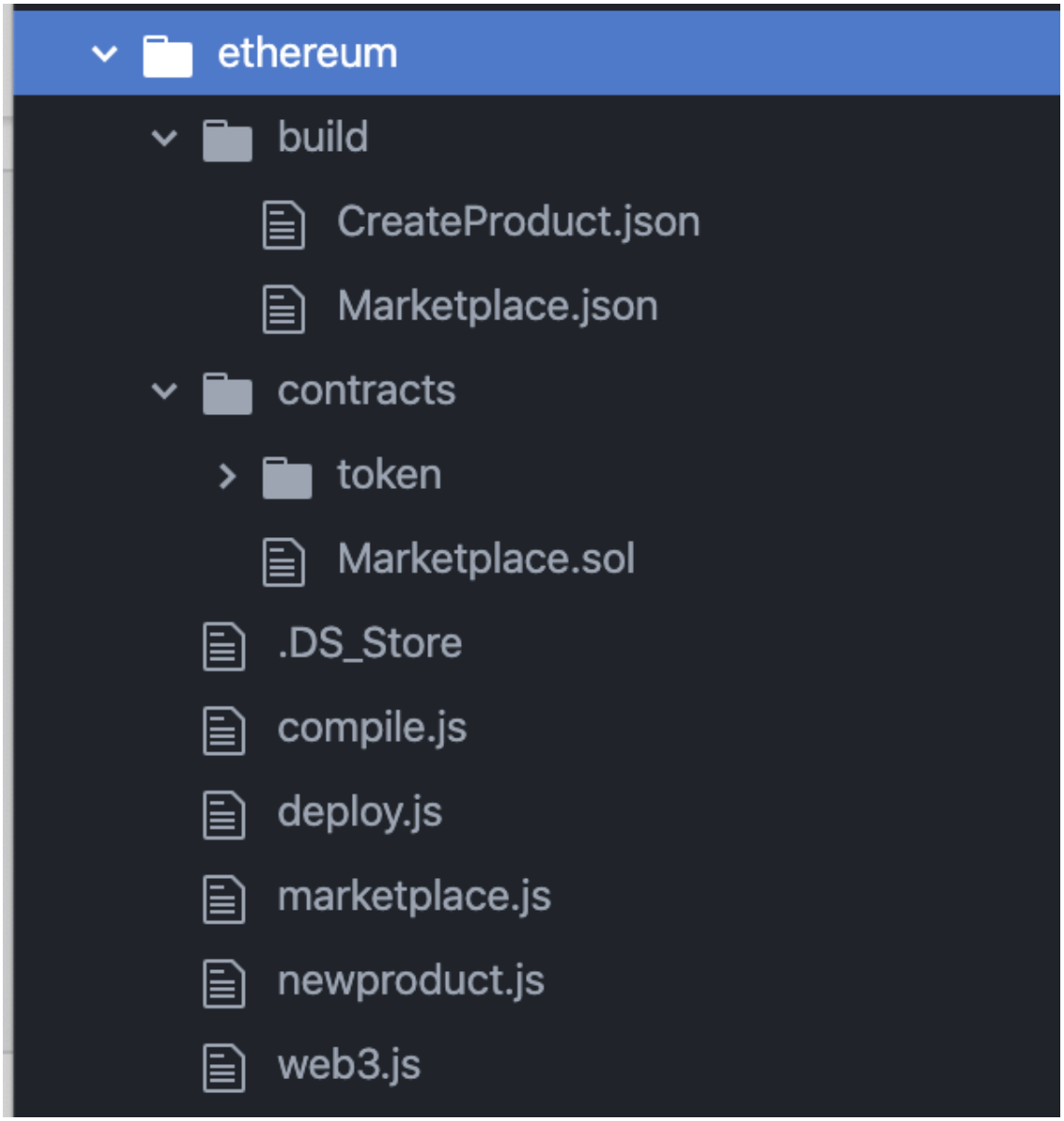

Figure 5.2: Ethereum Folder

concept of Centralized Web by having one large central server. However, having one large data center is not very convenient and not very safe. There is a very high risk of data leakage as well because all the information is stored in one place. A concept of individuals have a sheer bit of information and distributed in a network and can retrieve information by connecting the people with part of the information and download their bits to get the full information. This is called Distributed Web.

MetaMask runs the decentralised app and connect to the Ethereum node using INFURA. MetaMask contains an account that holds ethereum wallet in ether and allows user to connect to the different blockchain network. Users can use MetaMask to send or receive ethers while interacting with the decentralised app. MetaMask can be used to interact with the Ethereum blockchain. In the Ethereum blockchain network, the cryptocurrency for the network is ether and tokens where tokens are cryptocurrency that are built using smart contracts. MetaMask can store both ether and tokens in its wallet of an account.

There are several reasons why MetaMask is chosen to use in this project. Firstly, MetaMask is an open-source online software that is free of charge that provides users backup if the account is lost. Secondly, it is very simple to use and new user can get familiar with it in a short period, which

```
beforeEach(async () => {
    // some testing logic before each test is run
});

// This initiate the test of Marketplace
describe('Marketplace', () => {

  it('function 1', () => {
    // logic of test for synchronous function 1
  });

  it('function 2', async () => {
    // logic of test for asynchronous function 2
    await .....

  });

  // More tests ....
}
```

Figure 5.3: BDD style-interface

make it suitable for a marketplace application wallet.

However, there are also downsides of MetaMask BitDegree, n.d. to consider as well. Since MetaMask runs in the web browser, your data will be private from MetaMask itself, but the browser that runs MetaMask can still monitor the activities. This might be an issue for some users because letting web browser collect data from your transaction might feel not very comfortable from data privacy. The second issue is that MetaMask is an online platform that runs on a web browser. Since MetaMask is only a web extension, it doesn't guarantee to secure all the information submitted to the platform. All transactions are happening online so users are exposed to more risks from hackers.

There are a few other options of Ethereum wallets. The two main wallets that I first decided to compare are MetaMask and MyEtherWallet. MyEtherWallet is more widely used in the blockchain community. MyEtherWallet allows user to create their own wallet and view their cryptocurrencies that are stored in the wallet simply by using MEW, an Ethereum private key. MyEtherWallet gives full control of the private key including storage and can connect directly to the blockchain node. However, Metamask needs a trust from central server such as infura in order to communicate with the blockchain. MyEtherWallet also provides functionality to write smart contracts. Eventhough MyEtherWallet seems to provide more functionalities than Metamask to the user, both wallets offer very good wallet package already. So in this project, Metamask is used because it already provides all the functionalities that are needed in trading and it is easier for users to get started and use it.

The last issue to consider would be the security of MetaMask. From researching, it has shown that MetaMask never exposed to major data leaked or hacks. Since MetaMask provides with HD backup and it is a continuously developing open-source software, the community helps each other to keep MetaMask safe. However, MetaMask is still an online platform, therefore, there are still some risks regarding to hackers comparing to having a hardware wallet storage.

The major risk that MetaMask face is the phishing attacks where personal information is stolen. This is very common for online wallet since it is a popup menu. This attack can happen when multiple tabs are open and a transaction is made in one tab. Hackers can intercept by sending a pop up menu saying that user's transaction is failed and make him reenter the password. Since a transaction fails case is very common in blockchain so the user might not be aware. The password of the MetaMask account now got stolen to the hacker and hacker can use this password to access the MetaMask wallet and withdraw all the ether.

When creating a MetaMask account, it will require users to create a password with at least 8 characters and MetaMask will provide users with unique 12 words. It is required to store these 12 words somewhere safe and accessible because it is the only way to restore the MetaMask account. When interacting with decentralsied app, MetaMask allow user to choose which network to transact in. In the main network, it would cost real money in the transaction of ether. However, in this project, the Rinkeby Test Network is used instead.

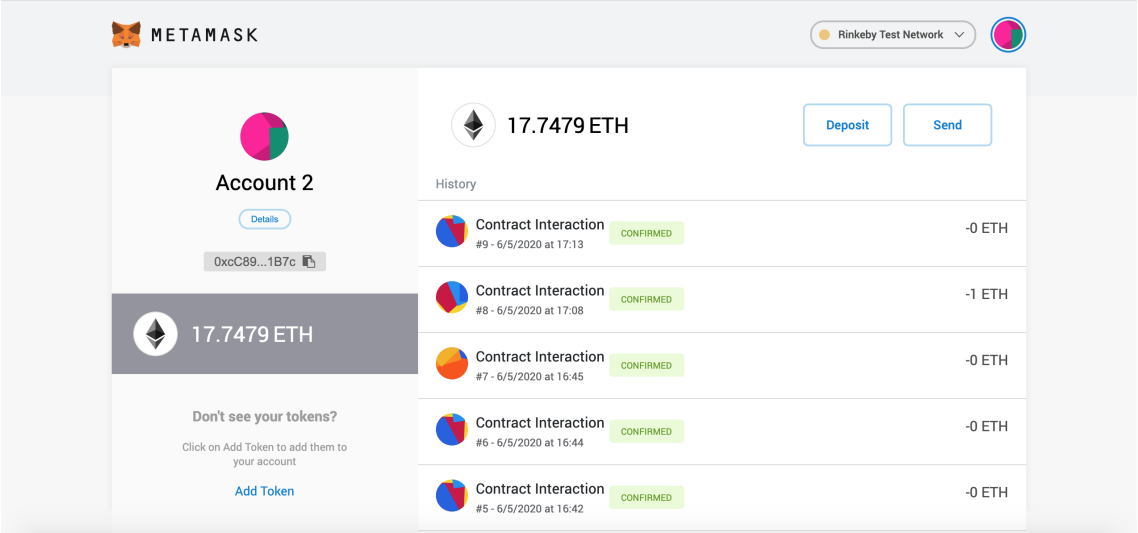

Figure 5.4: MetaMask Account and Transaction Details

Getting ether for testing in the Rinkeby Network requires several steps. User needs to visit https://faucet.rinkedby.io and user is required to make a public post from a twitter or a Facebook account containing the Ethereum address inside the post. Then copy the URL of the public post that contains the address and paste it in the form of getting ethers inside the website.

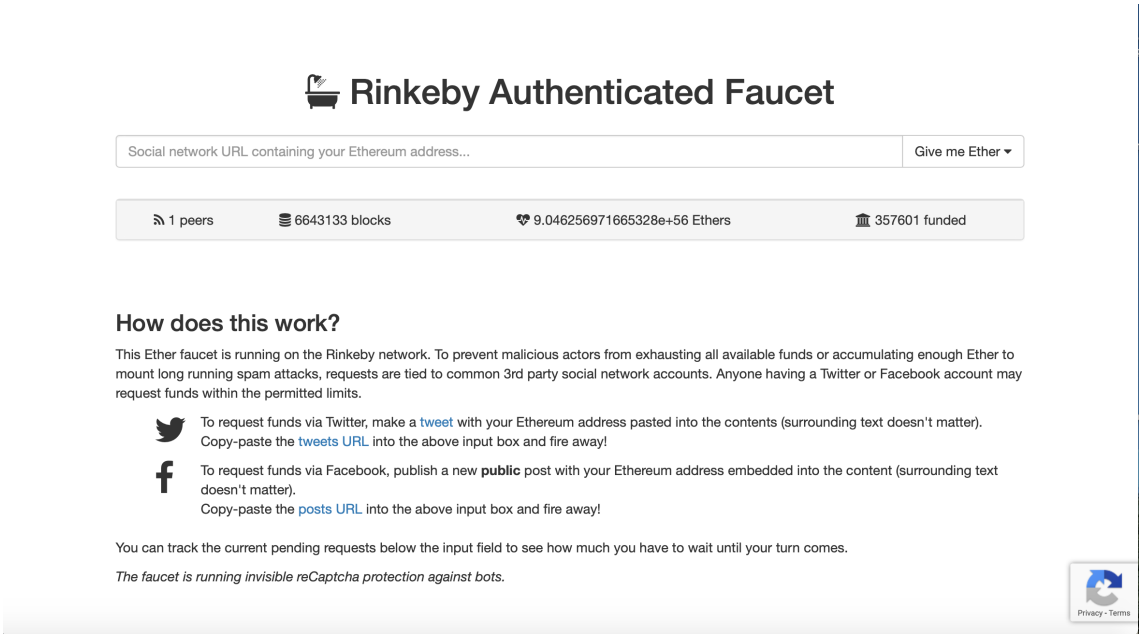

Figure 5.5: Rinkedby Faucet Website

Infura and Truffle HD Wallet Provider

Infura Copeland, n.d. is a tool to help build decentralised application on the blockchain easier and quicker by interacting with the blockchain and runs the complicated infrastructure node them-

selves. Normal developers might find it hard to connect to the blockchain network and storing data is very expensive and takes up lots of storage memory. Infura gives a faster access to the blockchain network and uses a hash for data storage, therefore, only the hash needs to be stored on the network and the real data could be stored elsewhere.

However, since infura Curran, n.d. acts as a centralised server, it can suffer from attacks from a third party. This single point of failures brings many concerns especially the privacy that might not be secure. It is very unlikely for infura to fail but if it does happen in any case, all the decentralised applications running on top of it would crash because those apps cannot connect to the blockchain. Since infura is an online platform, it also suffers from data privacy problems in the case of the transaction data hash and IP address are traceable.

In conclusion, infura is still a very powerful tool that helps connect the decentralised application to the ethereum blockchain node which results in a better security assurance and scalability in the long term.

Trufle HD Wallet Provider is a tool helping with signing transactions using provate key. It is essentially a tool to manage the accounts of the provider. Also Truffle HD Wallet Provider makes it simple to configure connection to ethereum network with infura.

5.1.2 Data Structure & Back End Design

In this project, it is required for the platform to be compatible and scalable to other FilmChain products. Since the FilmChain user data structure is too complicated for this project, a simpler data structure is designed.

Data Structure

```
"data": {
   "user": {
     "id": "ckaf6x1e1jm88094147eedyg2",
     "email": "investor@test.com",
     "profile": {
       "firstName": "Investor",
       "lastName": "Test"
     }
    "stakeholders": [
      {
        "id": "ckaf914rao6vb09926excugp1",
        "movie": {
          "id": "ckaf907vto6pv0992w074u15g",
          "title: "Stranger Things",
          "description": "Strange things happen",
        }
        "fixedAmountToRecoup": 100000,
        "percentage": 0.33,
        "type": "CappedGrossPercentage",
        "group": {
          "id": "ckaf907vto6pv0992w074u15g",
          "name: "US"
          "tranche": {
            "id": "ckaf907vto6pv0992w074u15g",
            "displayName": "Position 2",
            "position": 2,
          }
        }
      },
      {
        "id": "ckaf914rao6vb09926excugp2",
        "movie": {
          "id": "ckaf907vto6pv092h7074usi8",
          "title: "Lord of the Rings",
          "description": "A journey to a lava-filled hole",
        }
        "fixedAmountToRecoup": 0,
        "percentage": 0.5,
```

```
        "type": "NetPercentage",
        "group": {
          "id": "ckaf907vto6pv0992w074u15g",
          "name: "US"
          "tranche": {
            "id": "ckaf907vto6pv0992w074u15g",
            "displayName": "Position 1",
            "position": 1,
          }
        }
      },
      {
        "id": "ckaf914rao6vb09926excugp3",
        "movie": {
          "id": "ckaf907vto6pv092h7074usi8",
          "title: "Harry Potter",
          "description": "A boy becomes a wizard",
        }
        "fixedAmountToRecoup": 5000,
        "percentage": 0,
        "type": "FixedAmount",
        "group": {
          "id": "ckaf907vto6pv0992w074u15g",
          "name: "US"
          "tranche": {
            "id": "ckaf907vto6pv0992w074u15g",
            "displayName": "Position 5",
            "position": 5,
          }
        }
      },
    ]
  }
}
```

Above is an example of the user data structure that is used in this project. This user contains user data, including user ID, e-mail address, first name and last name. This user also owns 3 stakeholders token address as can be seen inside the stakeholder struct which includes a specific address ID for each stakeholder: "ckaf914rao6vb09926excugp1", "ckaf914rao6vb09926excugp2", "ckaf914rao6vb09926excugp3".

For each stakeholder, information attributes include stakeholder token address, movie id, movie title, movie description, fixed amount to recoup, percentage, stakeholder type, group id, group name and group tranche. The tranche struct has it's tranche id, tranche display name and tranche position.

Contract Design

The initial design for the contract to provide core functionalities including list stakeholder for sales, buy stakeholder now or make an offer.

5.2 Front End

In this section, it will discuss about the technologies used for creating the front end as well as the logic behind each page and its mock up.

5.2.1 Technologies

For the front end of this project. I use mainly React Components framework along with Semantic UI React and links pages with Next js.

CreateProduct Contract

Variables		
deployedProducts	address[]	list of addresses of stakeholder on the marketplace
deployedProductsCount	uint	number of stakeholder on the marketplace

Functions	
newProduct	create new stakeholder token address
getDeployedProducts	get list of addresses of all stakeholder on the marketplace

Marketplace Contract

Variables		
manager	address	address of the person who owns this stakeholder
listedPrice	uint	price that this stakeholder is listed for
stakeholderType	StakeholderType	type of the stakeholder
productCondition	ProductCondition	state of the product (whether it is on sales)
fixedAmountToRecoup	uint	amount to recoup from owning this stakeholder
percentage	uint	percentage of the share

Functions	
Marketplace	Constructor function that set the information attributes
listForSales	list this stakeholder address for sales at a certain price
buyEquity	buy now this stakeholder at the listed price
makeOffer	make offer to this stakeholder at a certain price
acceptOffer	accept one of the offer that is made to this stakeholder
offerOwnership	transfer the ownership of this takeholder to the beneficiary

Figure 5.6: Initial Contract Design

React

For the front end of this marketplace project, it is mostly implemented by React components and jsx. React JS is a javascript library for building web application. React provides reusable libraries such as jsx and virtual DOM, which reduce the chance of getting errors and development time. JSX makes it easier to create and modify DOM, Document Object Model which is the tree of how a webpage is arranged, using HTML pattern. JSX supports dynamic web development for handling events of clicking buttons.

Next JS

Next js Vercel, n.d. is a React framework that makes linking and routing inside the directory more easily. It is very suitable for this project because many pages are linked to each other.

Semantic UI React

Semantic UI React is a react UI component framework that provides prebuilt components that allows user to use.

5.2.2 Decentralised Application & Mock Up Design

In this marketplace, the first main functionality that a user could do is browse through the list of available stakeholders that are on sale on the platform. On the homepage, the platform list available stakeholders that are for sales and allow user to scroll through all of them. If a user wants to see more information about a specific stakeholder, user can simply click the "View Project" button which will directly link to a show page that list out all the information for that specific stakeholder including description of the stakeholder, movie related to this stakeholder, stakeholder type, percentage of the share, information of the percentage, amount of money you can recoup in the future as well as the condition of this stakeholder whether it is still available for sales on the platform. The price of this stakeholder is shown in bold text on the right column, there exists 2 cards which allow user to choose to buy this stakeholder now at the listed price or to make an offer for this stakeholder at a certain price that he is comfortable at. Users can also visit to see the list of offers that are already made to this specific stakeholder by pressing the " View Offers" button where this will automatically direct to the offer page that list out the offering price, account address of the bidder, and the description of this offer. Below offers card on this page, there exist a card that renders information of the seller of this stakeholder.

As mentioned above, in order to buy/own a stakeholder, new users have 2 available options which are buying it now and make an offer. Buy now option is suitable for users that might think that the listed price is right for him. When buying it now, it is guaranteed that that user will immediately own the listed stakeholder after the payment process went successfully. The ownership of this stakeholder will be directly transferred to this user when all the processes are completed. From the stakeholder show page, after "Buy Now" button is pressed, this will first direct user to the term and condition page first where this will describe the process of how it works as well as there is an anchor tag or term & conditions where the user can read the whole term & conditions of this platform. The user would have to accept the term and condition first by clicking the green "I understand" button before it will direct to the checkout page. This process is very important for the user to understand this platform terms & condition first before any transaction might occur. If the user decides to click "Cancel" then it will direct the user back the previous product show page.

The project checkout page will be the last step for the user to confirm that this is actually the stakeholder that is willing to be purchased, this page will give a summary of all the information such as Stakeholder types, stakeholder description, amount of money that will be available to recoup in the future. This is the final step for user to agree that the listed information is correct and this stakeholder is actually the one that he is willing to buy. On the right column, the price of the stakeholder will be shown. Additionally, the system will automatically calculate the subtotal

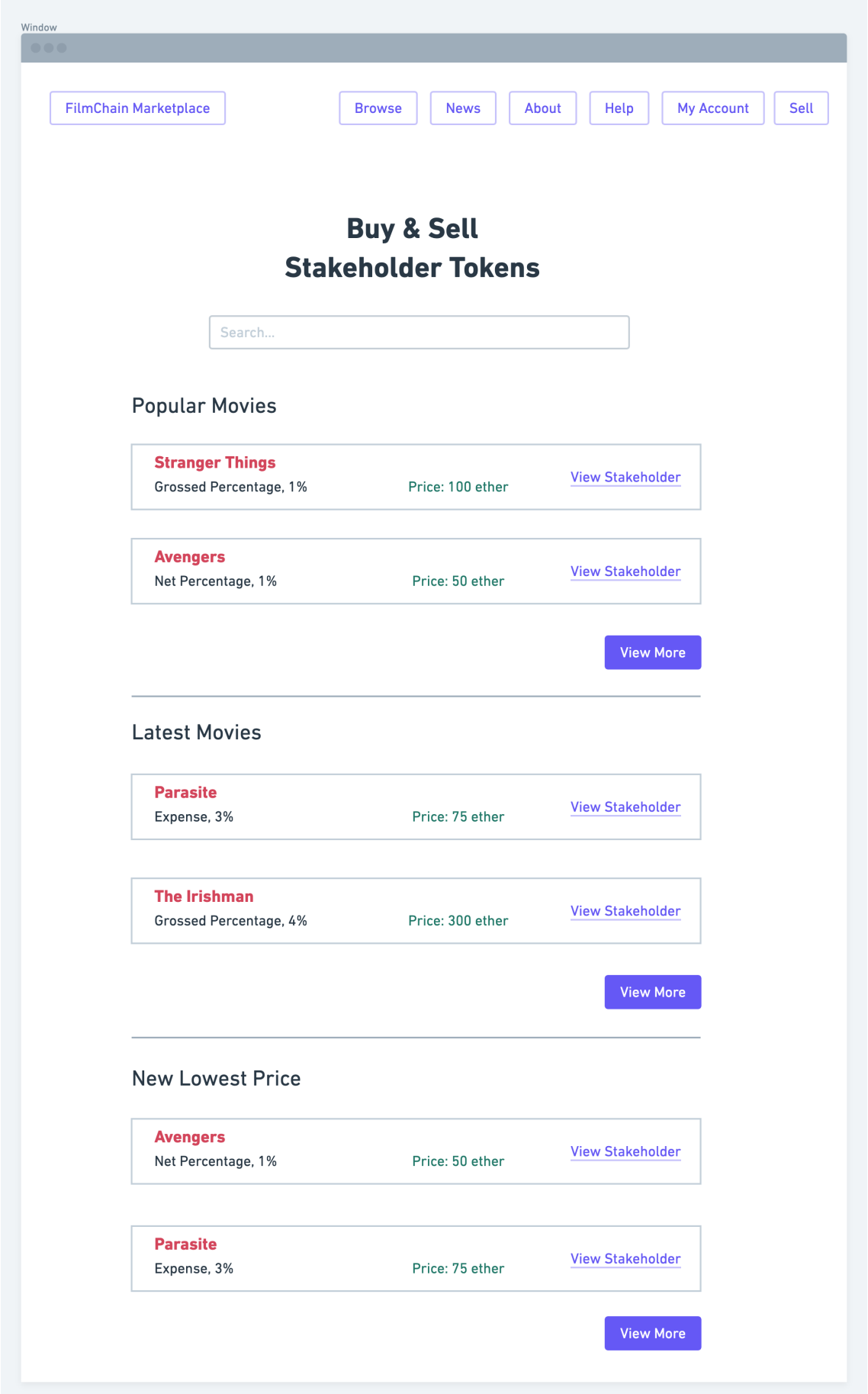

Figure 5.7: Homepage Design

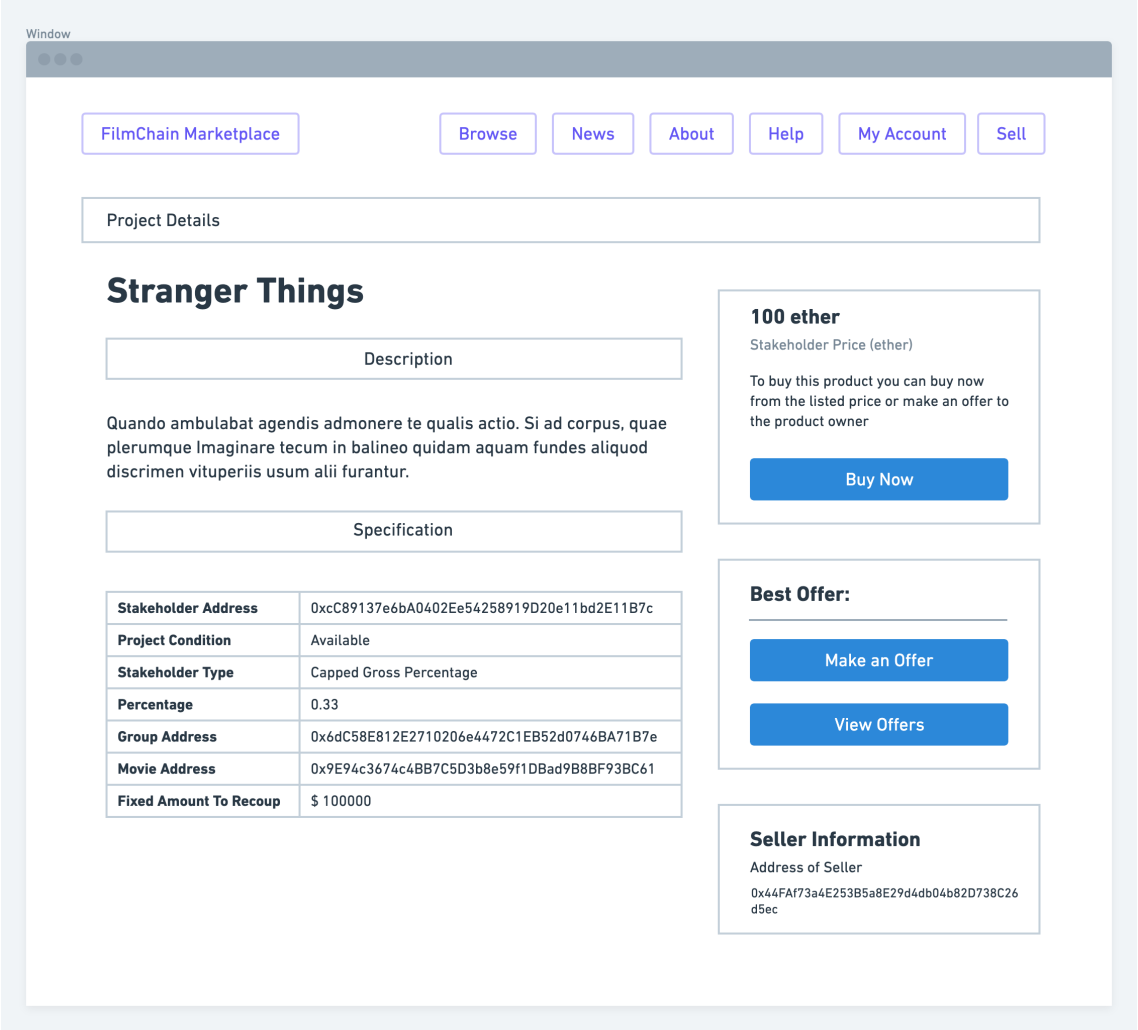

Figure 5.8: Project Show Page Design

price that the user would be paying when the sales is final. This includes the processing fee by using our FilmChain marketplace which the processing fee is equal to 2% of the stakeholder price. This processing fee will be added to the stakeholder listed price and the subtotal is displayed. Below the subtotal price, there exists a “Confirm and Pay” button. If the user agreed with the information and the price, this button should be pressed in order to process the payment and complete the transfer of ownership. When this “Confirm and Pay” button is pressed, the MetaMask account will pop up and displaying approve and reject buttons for this transaction as well as the total cost including the gas price will be shown. The gas price is essentially the cost that the user needs to pay for the system to process a function. If the user would like to continue with the transaction, simply press accept button, then MetaMask will process this transaction onto the blockchain. Just relax and wait while the transaction is processed since it could take a while due to communicating with blockchain. The loading button will appear instead of normal button to show the status of the transaction that it is still in progress. Once MetaMask successfully transact, the loading button will disappear and the system will automatically direct the user to his profile page. Once a user successfully purchased, the ownership of the stakeholder will be immediately transferred to this account. The new stakeholders will be shown on the list of stakeholder for the account and will be removed from the list of stakeholders of the previous account owner. All the ownership rights will also be transferred to this account, meaning that if this user is willing to sell the current stakeholder, it will also be possible or this user can wait until the recoupment is available for recouped and recoup that money instead.

The second option that is possible in the platform is by making an offer. Making an offer will allow user to purchase the listed stakeholder at a more comfortable price at the price he is willing

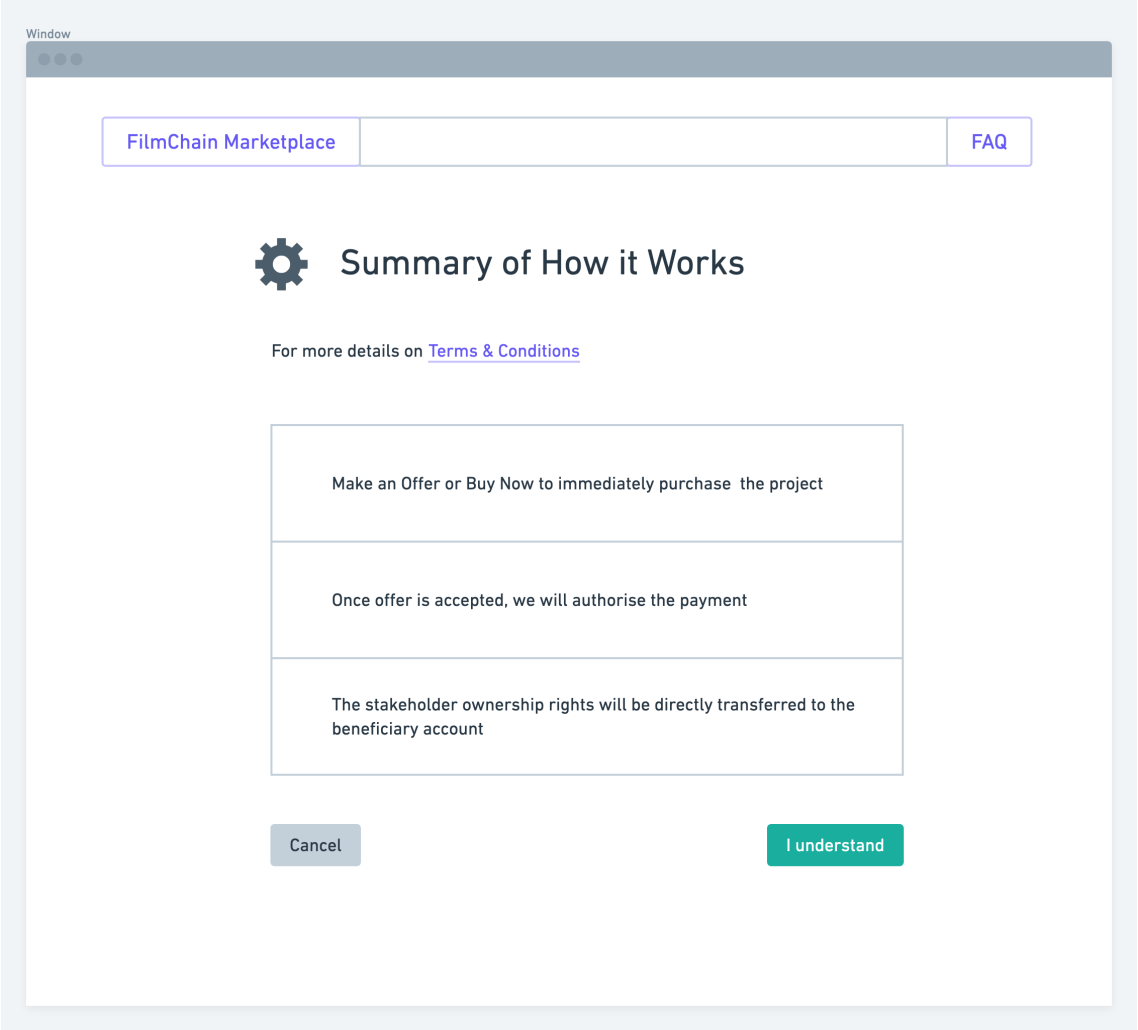

Figure 5.9: Terms & Conditions Page Design

to pay. All the offers made To the current stakeholder will be shown on the View Offers page. This View Offers page is directed by clicking the button "View Offers" in the project show page. This View Offers page will display all the offers that are made to this stakeholder including the offer price, the address of the bidder of that offer and the offer message. However, only the highest offer will be shown at the top of the table and this is the only offer that the seller can accept. If the seller accepts this offer. It means that the seller has agreed to sell this stakeholder at this price instead. When the seller accepts this offer by pressing the "accept" button, this stakeholder ownership will be automatically transferred this highest bidder similar to the process of buying it now. Once the transaction is done, the highest bidder will see this stakeholder available on his personal profile page.

A question might arises why did I choose to display the accept button only for the highest offer. In a practical world, after I have looked into several marketplaces, 99% of the time, the seller would wait for the highest offer and go for them instead of accepting a lower offer if they see there is another higher offer. However, when determining that this is actually the most highest offer that this user will be offered depends on individual techniques for different users. Most of the time, the seller would wait until the offers have settled, for example, by letting people make new offers until no one make a higher offer in 3-4 days. Another reason behind is that having multiple accept button is costly and waste of resources as well. Having to have an event listener for each of the buttons for a specific offer is also not very practical. Therefore, I decided to display only the highest offer with an accept button next to it. Users can view the list of all offers below the table. Another question that I have been researching on is why shouldn't the page list offer bids in ranking order from highest to lowest. I actually had tried to sort the order of the offers from highest to lowest

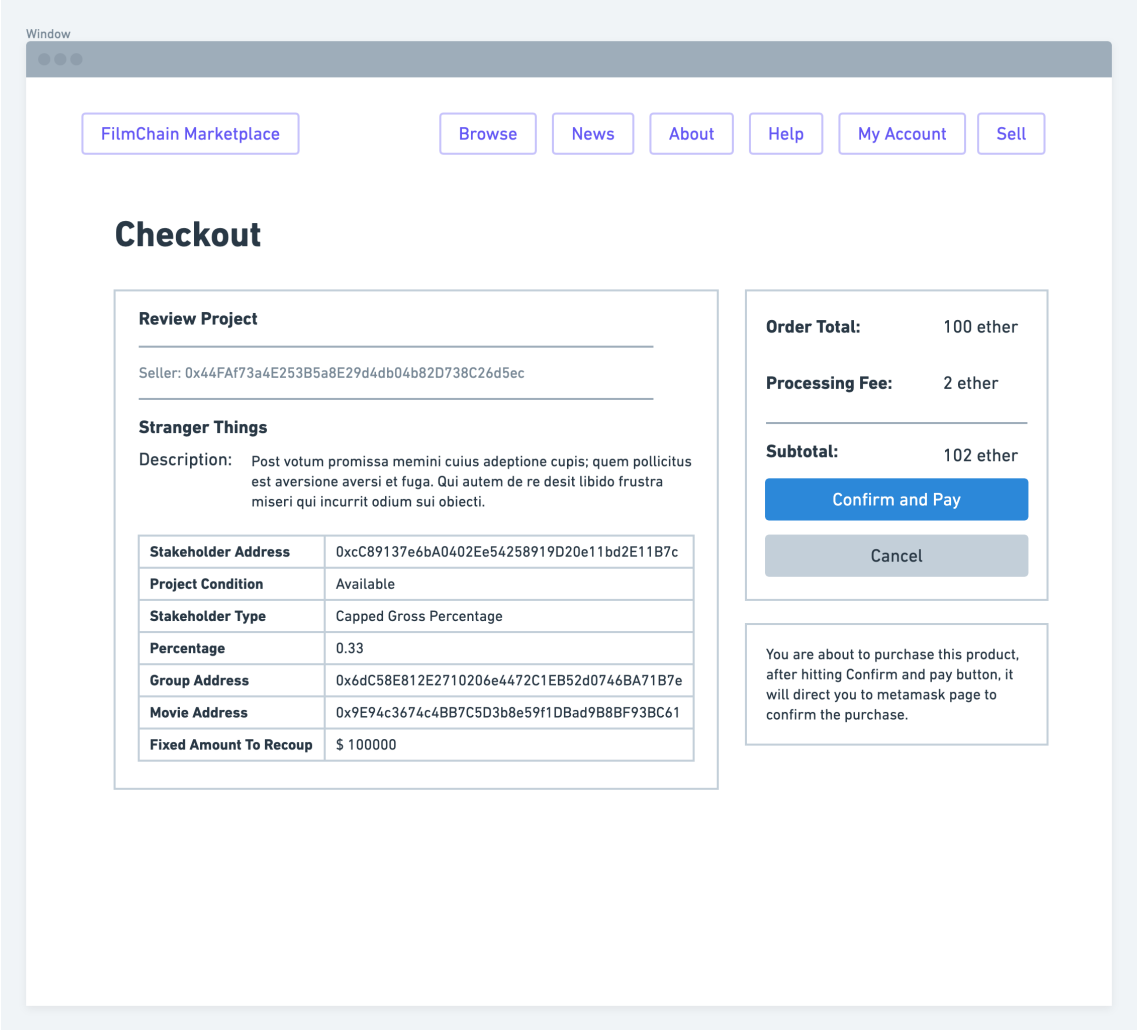

Figure 5.10: Checkout Page Design

using many sorting algorithms. At first, I tried using Quicksort algorithm. The algorithm works with a small subset of the array. The array is being passed as a storage reference which make it possible to sort the array. However, this approach is too costly because it costs too much gas to execute just one execution of sort. As I mentioned earlier that in solidity, every execution will require a gas for it to do the work. In this case, when sorting an array, it requires to contract storage to be modified, therefore, this would cost too much gas for just one execution. Overall, for the system to use the Quicksort algorithm to render just a table of sorted offers is not worth it when compared to the amount of gas consumed. This also applies to other sorting algorithms.

There are 2 places that are available for users to create a new offer for a specific stakeholder. It will be available through clicking the button “Make an Offer” where this button will be display one in the project show page and another one on the View Offers page. After clicking the “Make an Offer” button, the web page will direct users to the Make Offer page. In the Make Offer page, the user is required to insert the price that he is comfortable to offer in and think that the seller might accept it and also the offer message. When all the fields are inserted, the user can click “Create” button in order to progress to the next page. The system will automatically direct to the terms & condition page that will display the steps of hot the system works as well as the option to see the full term and conditions of the FilmChain marketplace platform. In this step, the user also has two options, whether to accept this term and conditions or to cancel the offer. By clicking “I understand” button will be the confirmation for the user to accept the term & conditions otherwise the offer will be cancelled. Once the term & conditions are accepted, it will direct the user to the checkout page that is similar to the buy now checkout page mentioned earlier. This page will display the information for this stakeholder including stakeholder type, project condition, the

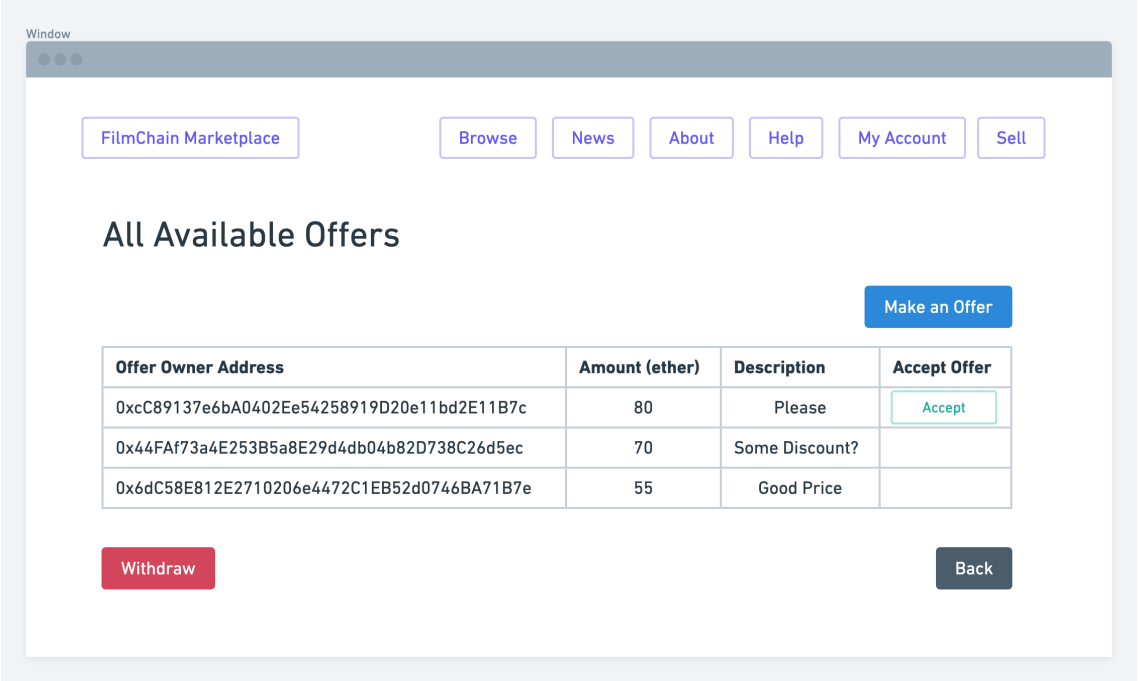

Figure 5.11: View Offers Page Design

amount that can be recouped, movie of this stakeholder, etc. However, on the right side instead of showing the listed price, it shows the offer price that this user is going to make. It will also show the 2% processing fee and calculate the subtotal that the user will have to pay in total in order to make this new offer. This price will be the sum of the offer price and the processing fee of 2%. Underneath the subtotal price, there exist a "Confirm and Pay" button that once the user click this, the system will signal to the back end that this user is willing to create a new offer. The user's Meta<ask will pop up with a transaction hash and showing the total ether that need to be paid, which is the sum of the offer plus processing fee plus a few gas price for executing the function. The user can choose to accept or reject in this stage where if the user rejects, the transaction will be failed and result in an error pop up on the web page. If the user chooses to accept, then MetaMask will continue to process this transaction and the loading button will appear on the page instead of the normal button the show that the transaction is in progress. Once the transaction between MetaMask and blockchain is successful, the spinning button for loading will disappear and replace with a normal button. Then the page will be automatically directed to the View Offers page of this stakeholder and your offer will be shown on this page. If your offer is considered to be the highest, it will be at the top of the table and an accept button will be displayed next to your offer. Only the seller will be allowed to press this button to accept the offer. If other users try to accept, an error will occur instead. However, if your offer is not the highest, then it will be shown down below on the table instead. It is believed that a user should only make an offer if they could afford a higher offer than the previous highest offer, otherwise the seller would not accept the lower offer anyway. Also, if a new offer and the previous offer have the same offer amount, the system will consider that the previous highest offer is still higher than the new offer recently made and still appear as the highest offer. This is to prevent user from offering the same price forever until the seller accept at this price.

In this marketplace, new users required to sign up to our platform first before all listings or sales can occur. Users can create an account by visiting the sign-up page where it is available from the Menu header and also available on the login page. New sign up will required one to submit their personal details, including first name, last name, email address, password, and a MetaMask account address. It is required for a user to own a MetaMask account beforehand before creating an account on our platform.

Creating a MetaMask account is nice and simple. Users can simply add a MetaMask extension in Google Chrome or visiting the MetaMask account and create a new login password. After creating MetaMask account, MetaMask will give out a 12 word password that you must note-down to be

able to login elsewhere or use the same account. After logging in to your Metaask account, you can copy your MetaMask account address to clipboard and simply insert it in FilmChain Marketplace platform in order to create a new account with us. This MetaMask account is very important because every transaction a user make in our platform such as, buy now, make an offer, etc, would require a user to approve the transaction in the MetaMask account as well as a confirmation. Additionally, one MetaMask account can only link to one FilmChain account, therefore, it is very important to be able to remember the password for the MetaMask account.

User can log in to our platform by browsing to the login page. Users must login to our platform before any transaction can happen. A login will require the user to submit the metamask account address and a password of our platform. Note that the password that must be provided for logging in to our password is the password that user submit when creating an account on our platform not the password for their metamask account. If login is successful, it will automatically link users to his profile page, otherwise an error would be shown. The profile page will show unique profile information for each specific user including first name, last name, email address, and the most important information which is the stakeholder that this user owns. A list of stakeholders will be shown on the cards with some preview information and there will be a view-info button inside each card that will direct the user to a unique stakeholder page.

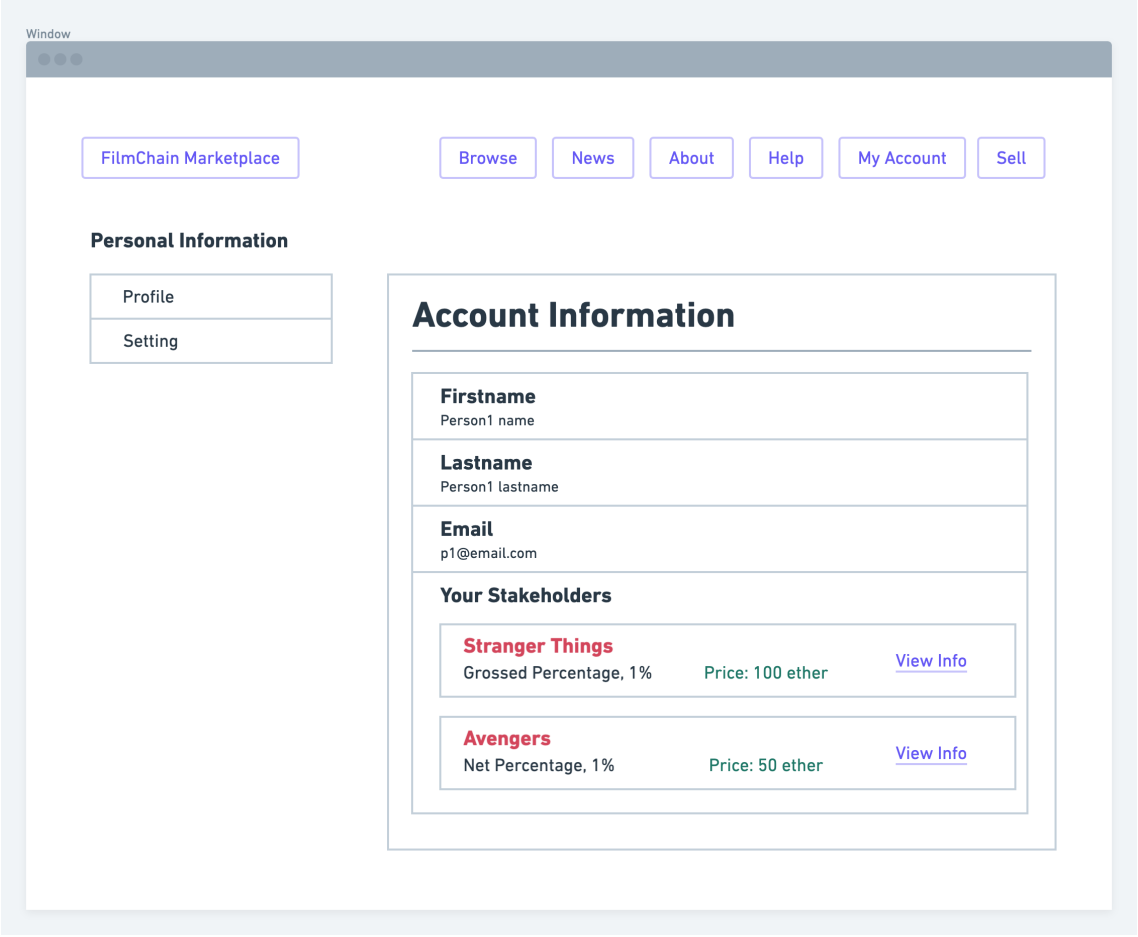

Figure 5.12: User Show Page Design

The view-info page for each stakeholder is very similar to the stakeholder show page that are listed for sales where it renders all the available information such as the description, movie of this stake holder, stakeholder type, amount of money that are available for recouped, percentage of the share in the movie and some information about the share. However, what is different is that on the right column of this page, there exists a form for listing this stakeholder for sales if one might be interested to sell this stakeholder on the secondary market. The form will require the user to insert a price that he is willing to sell for. Underneath, there will be a "List For Sales" button which essentially a button that once clicked, the form will be created and this stakeholder will be listed

for sales on the platform. After clicking "List For Sales" button, if the inserted price is of a correct type (uint) then the form will be submitted and the system will communicate with the back-end to create this listing onto the platform. MetaMask account will pop up for the user to confirm the transaction. If the user rejects, then an error message will pop up instead otherwise, MetaMask will send this transaction to the blockchain network and the button on the page will change to a spinning button to clarify that the system is currently loading. Once the transaction is successful, the spinning button will revert back to normal button and the page will automatically direct to the project show page. Once the stakeholder is on sales, the product condition will be changed to Available, which means that it is available for trades.

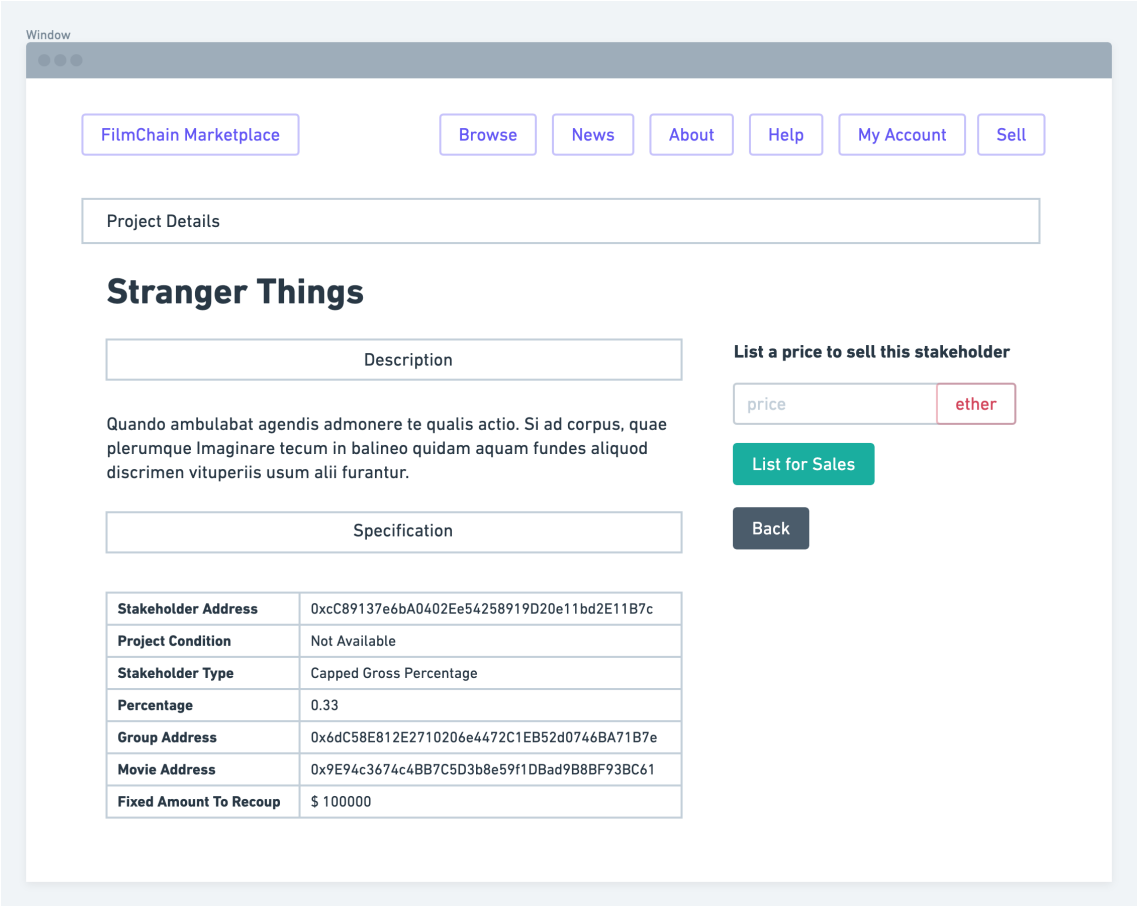

Figure 5.13: View Offers Page Design

This decentralised app might not be very familiar with new users, therefore, I have created a How it works? page as a summary of step by step of what this platform is and how to get started using this platform. Starting the page with some background information about the company FilmChain and the motto for this marketplace. Down below, Short summary of "How to Buy?" and "How to Sell" are displayed in simple steps where users can browse into the "Learn More" button to view more information. From each of the learn more option, it will direct users to the articles page according to its.

To Learn More about Buying page, description of what is an offer and how to buy is rendered below. As mentioned above that in this marketplace platform, there are 2 options that users can buy a stakeholder which are by buying directly or making an offer. Any offer price can be made to a specific stakeholder, however, only the highest offer will be displayed at the top and has an accepted button associated with it. This accepts button is for the seller to decide whether or not to accept this offer at a certain price. This seller can decide to ignore this offer until he manages to get another higher offer that might be interested in. If the seller decides to accept the offer, then essentially that highest bidder successfully offer at the right price at getting transact. The ownership of the current stakeholder will be transferred to this user and get removed from the previous owner.

The second option for buying is to select buy it now, which means buying this stakeholder immediately at the listed price. This guarantee the success of the user who decides to buy it now for this stakeholder that once the transaction is successful, the stakeholder will be transferred to him immediately. This option is suitable for the user that is comfortable at the listed price and do not want to risk to compete with other people in order to win the possession of this stakeholder. After a deal has been made, either by seller accepting the highest offer or the stakeholder is purchased directly, all the funds that are made from the offers will be released.

Moving on to the Learn More about Selling page, this page describes how to list a project/stakeholder for sales in this marketplace. For every project that can be listed for sales, it first need to have an owner which is essentially the user who owns this stakeholder of a movie. A user can list a stakeholder for sales by logging in to the profile page and visit a specific stakeholder page. On that page, there will be a form on the right to specify a price if that user wishes to try to sell this stakeholder. Once a certain price has inserted and the user decides to list this stakeholder for sales, after all the communication of MetaMask and blockchain is successful, that stakeholder will be available for sales for another user to buy or make an offer on the marketplace. The user can decide to withdraw the stakeholder from sales at any point in time apart from during the sale is successful.

There are also several other useful articles that are listed in the suggestion tab on the right corner. The user must understand the buying and selling processing fee, which will be included in the subtotal money to pay. For this marketplace, both users and sellers will need to pay a 2% processing fee both for every transaction they make including buying and making offers. The subtotal price is the calculated price of the sum that the user needs to pay by making this new offer or buy the stakeholder.

This web application also provides with a Reviews page where users are allowed to write a review of this marketplace as well as examined other users opinion about this platform. The first column will be top reviews that might be written from a celebrity. Down below will displayed all the review comments from general users. This Review section also plays an important part in improving the marketplace. Our team will read all the reviews and try to improve the platform to make it even better!.

Also, if there are any other problems or users might have a specific question, users can visit the FAQ page which will have lists of several common frequently asked articles or to contact us directly via Medium/Twitter/Email. If the user chooses to contact us by email, user must specify his name and email address as well as the message or question and this message will be sent to us and our team will try to reply every email as soon as possible.

In summary, the overall design of the steps how the marketplace works are as follow:

1. A stakeholder of a client is listed for sales
2. All available stakeholders that list for sales are displayed on the homepage of the web application
3. More information regarding to a stakeholder is rendered inside each stakeholder page
4. Clients can choose to Buy Now at a listed Price or Make an Offer at a certain price.
5. Terms & Conditions page must be accepted before any checkout for the client to understand the process and agree with the rules
6. At checkout, the subtotal price including FilmChain processing fee of 2% will be displayed along with Confirm and Pay button to agree with the price
7. If Buy Now is selected, the ownership stakeholder token will be transferred to the beneficiary instantly after the transaction. However, if an offer is made instead, the client needs to wait until the seller has agreed and accept the offer then the ownership stakeholder token will be transferred.
8. If the offer is not accepted and the sales have already been made, the client can withdraw the offer money from the withdraw page

5.3 ICO FilmChain Token Exchange

A platform that allows the client to exchange FilmChain tokens must also be created. This platform should allow user to insert an amount of token inside an input field. After a button is pressed, it should exchange the ether to the amount of Filmchain tokens that the client wants to exchange.

The mock up design for this part is as follows:

Instruction is given on how to purchase FilmChain tokens and how much each Filmchain is worth for the client to understand the exchange value. Your wallet will also be shown on the current amount of FilmChain tokens that you own as well as your account address at the bottom of the page. In the input field with an associated Buy Tokens button is used for specifying the amount of tokens that the client wants to exchange. Below the input field, a record of the number of tokens that have been sold on the platform is also recorded.

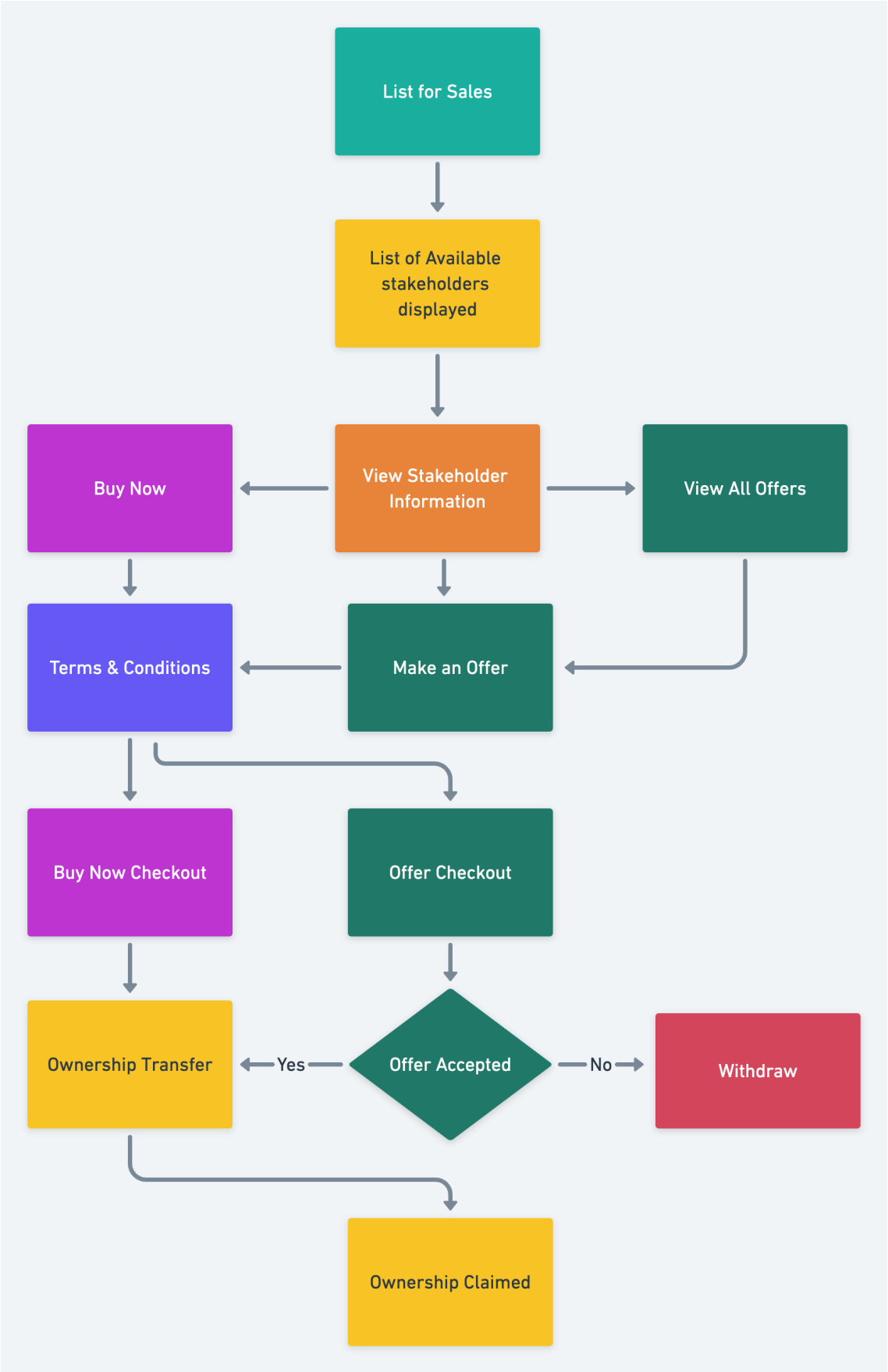

Figure 5.14: Marketplace Process

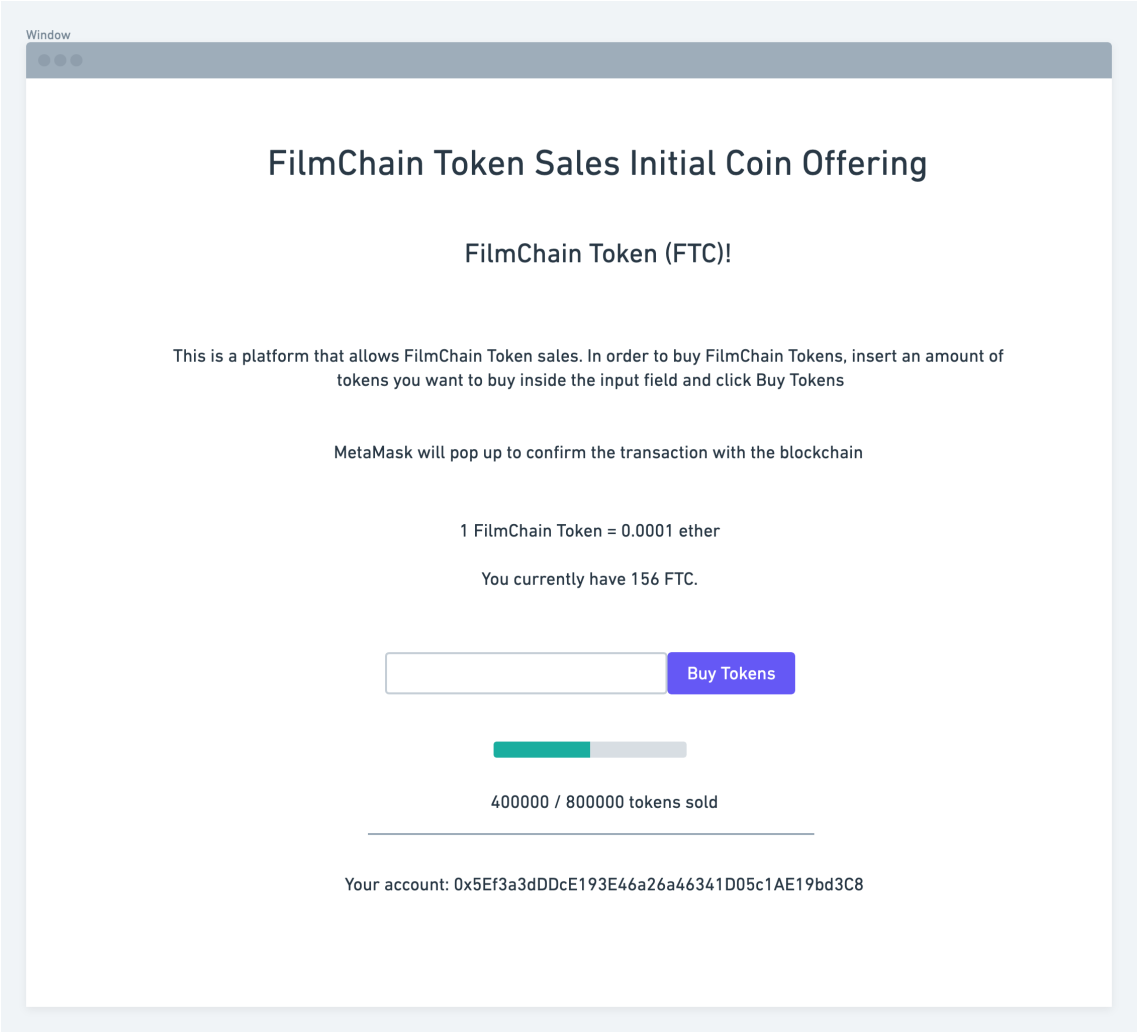

Figure 5.15: FilmChain Token ICO Web Mock up

Chapter 6

Implementation

The core functionalities of this project is to implement an end to end decentralised application marketplace to be able to trade for beneficiaries in the film industry for stakeholder position. In summary, user A might be a production team in a movie and owns 3% of the share in this movie. After the first recoupment, he decided to sell his share in a specific movie which might worth $x in his opinion so he listed this share for sales onto the marketplace. The platform allows other users to make a purchase directly by buying it now or can make an offer at a certain price. The offer will pop up to user A for him to decide whether to accept this offer or not or to wait for a higher offer. Once a deal has been made, the ownership of the listed stakeholders for sales will be transferred to the buyer and the buyer would become the new owner of this stakeholder and has all the ownership rights of this possession.

This project is divided into 2 main parts. The first part includes the implementation of smart contract infrastructure and the web application for the FilmChain Tokens Sales ICO environment and the second part of the project is creating a decentralised application or essentially a web-based marketplace to support users purchase the stakes or to list stakes for sales.

6.1 Back End for FilmChain Token Sales

In the back end of this part, I divided the implementation into 2 main smart contracts. The first contract dealing with the creation of FilmChain token as well as provide features that are standardised to the ERC-20 token protocols such as transfer, approve, etc. The second smart contract is dealing with the token sales ICO environment especially for buying the FilmChain tokens from the platform and to exit the sales.

6.1.1 FilmChain Token Smart Contract

In this contract, the core functionlaity is to create FilmChain Token. The name of the token is specified by a string, "FilmChain Token" with a symbol "FCT". The constructor of this smart contract accepts a number for initial supply to specify the amount of tokens in total. Also, the smart contract must implement the transfer, approve and transferFrom functions in order to become an ERC-20 token.

```
1  mapping(address => mapping(address => uint256)) public allowed;
2
3  constructor(uint256 _initialSupply) public {
4      balances[msg.sender] = _initialSupply;
5      totalSupply = _initialSupply;
6      // allocate the initial supply
7
8    }
9
10 //Transfer amount of token to balance
11   //Trigger transfer event
```

```
12   function transfer(address _addressTo, uint256 _value) public returns(bool success
       ) {
13     require(balances[msg.sender] >= _value);
14     balances[msg.sender] -= _value;
15     balances[_addressTo] += _value;
16
17     emit Transfer(msg.sender, _addressTo, _value);
18     return true;
19   }
20
21   function approve(address _approveAddress, uint256 _value) public returns(bool
       success) {
22
23     allowed[msg.sender][_approveAddress] = _value;
24     emit Approval(msg.sender, _approveAddress, _value);
25     return true;
26   }
27
28   function transferFrom(address _addressFrom, address _addressTo, uint256 _value)
       public returns(bool success) {
29     require(balances[_addressFrom] >= _value);
30     require(allowed[_addressFrom][msg.sender] >= _value);
31     balances[_addressFrom] -= _value;
32     balances[_addressTo] += _value;
33
34     allowed[_addressFrom][msg.sender] -= _value;
35
36     emit Transfer(_addressFrom, _addressTo, _value);
37     return true;
38   }
```

When an event of Transfer or Approval happens, it must emit to the blockchain in order to keep a record of the transaction happening. Therefore, Transfer and Approval events are implemented and triggered when an event of transaction happens.

```
1 event Transfer(address indexed _accountFrom, address indexed _accountTo, uint256
      _value);
2   event Approval(address indexed _tokenOwner, address indexed _tokenSpender,
      uint256 _value);
```

6.2 Front End for FilmChain Token Sales

The balanceOf function is implemented to keep track of the current balance of this account by returning the value inside the balances mapping for the account specified by the key value.

```
1 mapping(address => uint256) public balances;
2
3 function balanceOf(address tokenOwner) public view returns(uint256) {
4     return balances[tokenOwner];
5 }
```

6.2.1 FilmChain Token Sales Smart Contract

In this smart contract, the core functionality is to provide the environment for buying tokens. The main function for this smart contract is the buyTokens function. First it checks whether the inserted amount is enough in order to purchase the token and check if this client has enough balance. Then the contract will try to transfer that amount of money to the token address. If it is successful, it will keep track of the number of tokens sold and emit the TokensSell event to the blockchain network.

```
1 uint256 public numSoldTokens;
2 function buyTokens(uint256 _value) public payable {
3     require(msg.value == times(_value, tokenPrice));
4
5     require(tokenAddress.balanceOf(address(this)) >= _value);
6
```

```
    require(tokenAddress.transfer(msg.sender, _value));

    // number of tokens sold
    numSoldTokens += _value;
    emit TokensSell(msg.sender, _value);
  }
```

TokensSell event is emitted when tokens are sold. It records the address of the buyer and the amount of tokens bought.

```
event TokensSell(address indexed beneficiary, uint256 _value);
```

The constructor for this smart contract specifies the FilmChain token address and the token price (how much one Filmchain token is worth). It then sets the attributes and record the manager of the contract.

```
address payable manager;
  FilmChainToken public tokenAddress;
  uint256 public tokenPrice;

  constructor(FilmChainToken _tokenAddress, uint256 _tokenPrice) public {
      // admin
      manager = msg.sender;
      tokenAddress = _tokenAddress;
      tokenPrice = _tokenPrice;
  }
```

The exitSales() function is called when the manager wants to terminate the contract where once it is called, it will transfer all the balances in this platform to the token address and self destruct this smart contract.

```
function exitSales() public restricted {
    require(tokenAddress.transfer(manager, tokenAddress.balanceOf(address(this))));
    selfdestruct(address(manager));
  }
```

6.2.2 FilmChain Token Sales Web Application

The web page is implemented using javascript bootstrap. The core functions of the web page is implemented in index.html file and the js/app.js file.

```
initContracts: function() {
    $.getJSON("FilmChainTokenSales.json", function(filmChainTokenSales) {
      App.contracts.FilmChainTokenSales = TruffleContract(filmChainTokenSales);

      App.contracts.FilmChainTokenSales.setProvider(App.web3Provider);

      App.contracts.FilmChainTokenSales.deployed().then(function(
    filmChainTokenSales) {
        console.log("FilmChain Token Sale Address:", filmChainTokenSales.address);
      });
    }).done(function() {
      $.getJSON("FilmChainToken.json", function(filmChainToken) {
        App.contracts.FilmChainToken = TruffleContract(filmChainToken);

        App.contracts.FilmChainToken.setProvider(App.web3Provider);

        App.contracts.FilmChainToken.deployed().then(function(filmChainToken) {
          console.log("FilmChain Token Address:", filmChainToken.address);
        });
        App.listenForEvents();
        return App.render();
      });
    })
  },
```

The initContract function creates an instance of a contract and deploys the contract to the Ganache network.

buyTokens function

The buyTokens function will be called when the Buy Tokens button is submitted. It will try to call the buyTokens function from the smart contract back end with the amount of tokens inserted in the input field.

```
buyTokens: function() {
    $('#content').hide();
    $('#loader').show();

    var numberTokens = $('#numberOfTokens').val();
    App.contracts.FilmChainTokenSales.deployed().then(function(instance) {
      return instance.buyTokens(numberTokens, {
        from: App.account,
        value: numberTokens * App.tokenPrice,
        gas: 500000
      });
    }).then(function(result) {
      console.log("Tokens bought");
      $('form').trigger('reset');

    })
  }
```

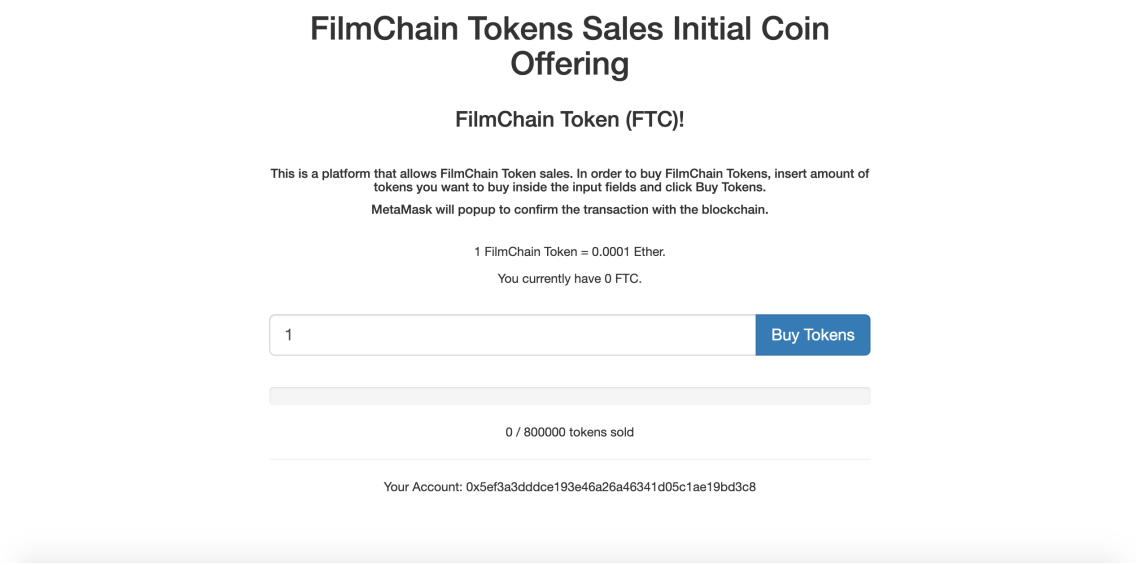

Figure 6.1: FilmChain Token Sales ICO Actual Web Page

initWeb3

The initWeb3 function instantiates the connection with the provider and the Ganache network.

```
initWeb3: function() {
    if (typeof web3 !== 'undefined') {
      // If a web3 instance is already provided by Meta Mask.

      App.web3Provider = web3.currentProvider;
      web3 = new Web3(web3.currentProvider);
    } else {
      // Specify default instance if no web3 instance provided
      App.web3Provider = new Web3.providers.HttpProvider('http://localhost:7545');
      web3 = new Web3(App.web3Provider);
    }

    return App.initContracts();

  },
```

render function

The render function will try to retrieve all the information to render on the web page such as the price for each Filmchain token, amount of tokens sold, amount of tokens that are still available for sales and the current balance of this account that display the number of FilmChain token that he owns.

6.3 Back End for Decentralised Application

In this section, a closer look at the implementation of the smart contract infrastructure, compile and deploy script and connecting the deployment with infura to connect to the blockchain node.

6.3.1 Smart Contract Implementation

In this project, 2 main smart contracts are implemented. The smart contract simulates marketplace operation by adding new stakeholders for sales, the stakeholders can be bought by buying now from the seller or making an offer, functions for checking whether the stakeholder is available for sales. Additionally, the stakeholder can be removed from the marketplace if it has not been sold to anyone yet. The implementation will be divided into 2 phases. The first phase is implementing a complete end-to-end marketplace without using a token. The second phase is to add ERC20 token as a payment method in the marketplace.

Phase 1: Marketplace without Token

In this phase, it will discuss about the 2 main smart contracts that are used to create the marketplace environment.

CreateProduct Contract

The CreateProduct smart contract is considered as a factory contract. Below are the technical implementation details of this factory contract that is used to create a new stakeholder address to the platform. Additionally, it contains other aspects of a factory for the marketplace includes storing all stakeholders on the platform and user information. The technical details are described below.

User Struct In this FilmChain marketplace platform, it is required for users to have an account with us first before any trading can happen. Therefore, the first step is that a client needs to sign up for an account. The user data structure has to contain all the relevant information for a specific user. User array is created to store user accounts that are created on this platform.

```
struct User {
        uint userId;
        address id;
        string email;
        string firstname;
        string lastname;
        address[] stakeholders;
        string password;
    }

User[] public users;
```

Create User In order to create an account, a client must specify the attributes such as first name, last name, email address and password. With the creation of an account, the MetaMask account address will also be automatically linked to the user account. Therefore, a MetaMask account can

only link to one user account on this platform. The create user function will store the relevant user information and set into a new User struct and push to the users array to keep track of all the new created accounts. The account address and password are also stored separately inside a mapping because it is more convenient to justify when a client logs in. This process does not require to search into the struct.

```
mapping (address => string) loginPassword;

function createUser(string _email, string _firstname, string _lastname, string
    _password) public {

        User memory user;
        user.userId = userCount;
        user.id = msg.sender;
        user.email = _email;
        user.firstname = _firstname;
        user.lastname = _lastname;
        user.password = _password;

        users.push(user);
        addressId[msg.sender] = userCount;
        loginPassword[msg.sender] = _password;

        userCount += 1;
    }
```

Get Users Information To retrieve information for an individual account, the address of the account can be inserted The information are stored in the User struct in the users array so the account address will specify the correct user account that account information is desired and the system will pull data from the User struct.

```
function getUserIdFromAddress(address _addressId) public view returns (uint) {
    uint _id = addressId[_addressId];
    return _id;
}

function getUser(address _addressId) public view returns (address, string, string,
    string, address[]) {

    // need to add check whether a user exist
    uint _id = getUserIdFromAddress(_addressId);
    User storage user = users[_id];
    return (
        user.id, user.email, user.firstname, user.lastname, user.stakeholders
    );
}
```

Log-in & Log-out After creation of an account, clients still need to login to the platform before any trading logic can occur. The system only allows an account owner to login the system. This means that in order to login, User must currently sign in to the MetaMask account and the account address must match with the one inserted into the log in page. In order to check whether the password matches with each other, an alternative way of comparison string is used. Since solidity language does not support string matching, a keccak256 hash of the password is used instead. The password hash is then being bytes compared to the one stored inside the User struct of an address. Once log in to the platform, the loginAddress and boolean login status will be updated. When the client wishes to log out, these attributes will be set to initial state instead.

```
bool public login;
address public loginAddress;

function login(address id, string _password) public {
        uint _id = getUserIdFromAddress(id);
        User storage user = users[_id];
        require(msg.sender == id && user.id == msg.sender, "Only owner of this
    address can login");
```

```
8           require(keccak256(abi.encodePacked(_password)) == keccak256(abi.
    encodePacked(user.password)), "Password does not match");
9           login = true;
10          loginAddress = id;
11
12          updateStakeholders(id);
13  }
14
15  function logout() public returns (bool) {
16          login = false;
17          loginAddress = address(0);
18      }
```

Create Stakeholder Since this platform has not yet been integrated into the actual FilmChain database yet, therefore, a way to create end-to-end testing must be used instead. This function essentially creates a new smart contract address for a stakeholder which includes its description, amount to recoup, amount already recoup, the percentage of this stakeholder, the movie that is associated with this stakeholder and it's stakeholder type. The new instance of the stakeholder is created by inserting these attributes and call the Matketplace contract constructor. After the stakeholder is created, it is added to the arrays of stakeholder contract addresses to keep track of all the stakeholders in the marketplace. After creation, an emit event message is sent to the blockchain and the stakeholder is added to the user profile. However, before the creation of this stakeholder is possible, it is required for client to login to the platform first.

After the stakeholder is created, it is pushed to the deployedProducts array which store stakeholder addresses, in order to keep track of all the stakeholders on the platform.

```
1  address[] deployedProducts;
2
3  function newProduct(string description, uint amountToRecoup, uint
       amountAlreadyRecouped, uint percentage, uint movieId, uint stakeholderType)
       public {
4           // check login is true first
5           require(login == true);
6
7           address product = new Marketplace(description, amountToRecoup,
       amountAlreadyRecouped, percentage, movieId, stakeholderType, msg.sender);
8           deployedProducts.push(product);
9           emit ProductCreated(msg.sender, product);
10
11          uint userId = getUserIdFromAddress(msg.sender);
12          User storage user = users[userId];
13          user.stakeholders.push(product);
14  }
```

Get All Stakeholders on the Marketplace deployedProduct is an array of addresses of stakeholder in the marketplace. By calling getDeployedProducts, it will return the list of stakeholders to be rendered on the homepage.

```
1    function getDeployedProducts() public view returns (address[]) {
2           return deployedProducts;
3  }
```

Emit Event of Stakeholder Creation After the creation of the stakeholder in the newProduct function, the contract must emit an event to the blockchain for this creation by calling emit ProductCreated. This event emits the owner of the stakeholder address as well as the address of the stakeholder to notify the node on the Ethereum blockchain that a creation has occurred.

```
1  event ProductCreated(address indexed owner, address indexed productId);
```

Transfer Stakeholder to Beneficiary Inside the marketplace logic, if a stakeholder is bought or the offer is accepted by the seller, then the ownership of the stakeholder should be transferred to the beneficiary who bought it. In this function, it describes the process of transferring a stakeholder address that is stored inside the User struct to the beneficiary user. The function check for updates if the owner of the stakeholder has already been transferred. If yes, then the stakeholder address is added to the new owner address of the stakeholder and deleted from the old one. To check for the owner of a stakeholder, it requires to look into the instance of that stakeholder contract (Marketplace).

```
function updateStakeholders(address _id) public {
        uint userId = getUserIdFromAddress(_id);
        User storage user = users[userId];
        for (uint i = 0; i < user.stakeholders.length; i++) {
            address a = user.stakeholders[i];
            Marketplace m = Marketplace(a);
            address conManager = m.getManager();
            if (conManager != _id) {
                // owner of contract has changed
                delete user.stakeholders[i];
                addStakeholderToOwner(conManager, address(m));
            }
        }
}
```

Marketplace Contract

In this smart contract, all the specific features for a stakeholder are implemented such as to list this stakeholder for sales, to buy this stakeholder, to make an offer to this stakeholder. The technical details are described below.

Marketplace Contract Constructor In the CreateProduct contract earlier, when newProduct function is called, the function creates a new instance of stakeholder by creating new Marketplace contract. The constructor for this Marketplace contract is called with the following parameters: description of the stakeholder, amount to recoup from this stakeholder, amount already been recouped from it, the percentage of the share, movie id that the stakeholder belongs to, the stakeholder type and the owner (creator) address on this stakeholder contract. The parameters of the constructor are then set and stored in the attributes inside this stakeholder contract. The initial parameters are also initialised as can be seen that offer related are set to the initial state. Also, when this stakeholder is created, it is not ready to be listed for sales yet, therefore, Product Condition is set to not deployed which means it has not been listed for sales on the market. Moreover, since this stakeholder is not sold yet, so the purchased boolean attribute is set to false initially.

The enum StakeholderType specifies the list of stakeholder types in FilmChain platform. These types describe the revenue waterfall and the share of the stakeholder.

```
enum ProductCondition { //Deploy means available for sales
    DEPLOYED,
    NOTDEPLOYED
}

enum StakeholderType {
    GrossPercentage,
    NetPercentage,
    CappedgrossPercentage,
    CappedNetPercentage,
    ProfitParticipation,
    Expense
}

// Attributes for stakes
    address public manager;
    uint public movieId;
    string public productDescription;
    ProductCondition public productCondition;
    bool public purchased;
```

```
21      address public newProductOwner;
22      StakeholderType public stakeholderType;
23      uint public fixedAmountToRecoup;
24      uint public amountRecouped;
25      uint public percentage;
26
27  function Marketplace(string description, uint amountToRecoup, uint
        amountAlreadyRecouped, uint percentageVal, uint movieIdVal, uint
        stakeholderType, address creator) public {
28          manager = creator;
29          productDescription = description;
30          productCondition = ProductCondition.NOTDEPLOYED;
31          fixedAmountToRecoup = amountToRecoup;
32          amountRecouped = amountAlreadyRecouped;
33          percentage = percentageVal;
34          movieId = movieIdVal;
35          stakeholderType = StakeholderType(stakeholderType);
36
37          // Set highest offer and bidder to initial state
38          highestBidder = address(0);
39          highestOfferPrice = 0;
40          purchased = false;
41          newProductOwner = address(0);
42  }
```

Listing the Stakeholder for Sales If the owner of the stakeholder wishes to list this stakeholder for sales on the market, this function can be called with a specify price. The function will change the condition of the attributes for this stakeholder to deploy, and this would make it possible for other clients buying or make an offer to this stakeholder. The price is also set to the listedPrice attributes to specify the price that seller desires for this stakeholder. This function is strictly restricted that it can only be called by the owner of this stakeholder. This means that only the owner can sell this stakeholder.

```
1   uint public listedPrice;
2
3   function listForSales(uint price) public restricted {
4       listedPrice = price;
5       productCondition = ProductCondition.DEPLOYED;
6       emit StakeholderListedForSales(price);
7   }
```

Emit Event when the Stakeholder is Listed When stakeholder is listed onto the market, an event is emitted to the blockchain to state that this stakeholder is available for sales on the marketplace.

```
1   event StakeholderListedForSales(uint price);
```

Buy Now In this project, we provide 2 features to support a client in buying a stakeholder. The first feature is to immediately buy now. Below is the implementation for the buy now (buyEquity function). The function is payable which means that in order to proceed, an amount of money needs to be specified when making this function call(transaction). The function first checks whether the given money is greater than the pricePlusFee, which is the total amount of money that the client needs to be including the processing fee for this stakeholder. More details on processing fee will be described below in this section. If it is, then the system only takes the amount of that is listed for the product and refund the extra money given. Also, the stakeholder is only available for sales if it has not been sold yet. After the transaction is done, the ownership of this stakeholder is transferred to the new beneficiary that bought this stakeholder. Once the transaction is successful, the event of BuyNow is emitted to the blockchain node.

```
1   // buy the listed equity directly
2       function buyEquity() public payable {
3           require(msg.value >= pricePlusFee, "Not enough money is inserted.");
```

```
        require(productCondition == ProductCondition.DEPLOYED, "This product is
    already deleted");
        require(purchased == false, "This product is already sold out");

        // refund the amount that is more than expected listed price
        if (msg.value > pricePlusFee) {
            uint refund = msg.value - (pricePlusFee * 1000000000000000000);
            msg.sender.transfer(refund);
        }

        highestBidder = msg.sender;
        highestOfferPrice = listedPrice;
        purchased = true;
        offerOwnership(msg.sender);
        claimProductOwnership();
        emit BuyNow();
}
```

Making Offers The second feature in this marketplace is that clients are also allowed to make an offer to a specific stakeholder at a desired price. When the function is called, the new offer is then stored in the Offer struct which stores information about the offer id, address of the client who made this offer, the offer price and the offer message. This new offer will then be pushed to the offers array which stores the list of all the offers that are made to this stakeholder. If the offer value is greater than the current highest offer, it first checks whether there exists a previous highest bid. This is because in the platform, the highest offer will be shown at the top of the table in the View Offers page and only the highest offer can be accepted by the seller. Therefore, if there exists a previous highest offer, then the system would refund the previous highest bidder's address first if the current new offer is higher than that. Then the new offer details are set to the attributes that store values for the highest offer, the address of the bidder, and the offer message for the new highest offer. Afterwards, the OfferCreated event is emitted to the blockchain node to signal that new offer, with price, address and message is created.

```
struct Offer {
    uint offerId;
    address offerAddress;
    uint offerValue;
    string offerMessage;
}

    // parameters for offers
    address public highestBidder;
    uint public highestOfferPrice;
    string public highestOfferMessage;
    uint public highestOfferId;
    uint public offerCount = 0;
    Offer[] public offers;
    mapping(uint => uint) offerMaps;
    mapping (address => uint) amountToRefund;

function makeOffer(string description) public payable {

        Offer memory newOffer = Offer({
            offerId: offerCount,
            offerAddress: msg.sender,
            offerValue: msg.value,
            offerMessage: description
        });

        offers.push(newOffer);
        offerMaps[offerCount] = msg.value;

        // Check whether it is the highest offer
        if (msg.value > highestOfferPrice) {
            if (highestBidder != address(0)) {
            // Refund the prevois highestBidder
            amountToRefund[highestBidder] = highestOfferPrice;
            }
            highestOfferPrice = msg.value;
```

```
37              highestOfferMessage = description;
38              highestBidder = msg.sender;
39              highestOfferId = offerCount;
40          }
41
42          offerCount+=1;
43          emit OfferCreated(address indexed bidder, uint price, string description);
44  }
```

Accepting Offer On the platform, to be able to accept the offer for a specific stakeholder, it is strictly required that only the owner of the stakeholder can undergoes that action. Therefore, it is specified as a restricted function in this case. Also, only the highest offer can be accepted therefore, if the current highest bidder is not address(0) (0x0) which is the initial setup, then the highest offer exists and this offer can be accepted. After the offer has been accepted, the ownership of this stakeholder will be transferred to the new beneficiary which is the address of the account who made the highest offer.

```
1 function acceptOffer() public restricted {
2         require(highestbidder != address(0));
3         purchased = true;
4         offerOwnership(highestBidder);
5         claimProductOwnership();
6 }
```

Withdraw Offer Funds Offers can be withdrawn in the platform. Also, if the client's offer does not get accepted by the seller, then the funds will be released to be withdrawn when the sales has finalised. After withdrawing function is called, it will check for the amount that can be withdrawn and send the money to the beneficiary's account. After the withdraw process is successful, the amount that can be refunded will be set to 0 to prevent multiple refund requests from a single client.

```
1 // Can call withdraw when you deleted the offer or you did not win the bid
2     function withdraw() public {
3         uint refund = amountToRefund[msg.sender];
4         address adressToRefund = msg.sender;
5         if (refund > 0) {
6             // set the refund amount to 0 to prevent user to try to refund again
7             amountToRefund[msg.sender] = 0;
8             adressToRefund.transfer(refund);
9         }
10    }
```

Transferring Ownership When the sales has been finalised, either by buying now or offer is accepted, these 2 functions will be called. The function offerOwnership will set the new stakeholder attributes to the highestBidder attributes which will be set when the offer is accepted or buy now is called. Then claimProductOwnership is called in order to transfer the funds for the stakeholder that stakeholder will receive after deducting all processing fees to the seller and transfer ownership of the stakeholder to the new beneficiary. More description on processing fee will be described below. Additionally, all the initial setup will be set in the contract so that this stakeholder can be listed for sales in the future if needed.

```
1 // call this function when the creator accepts the offer or product is purchased
2     // will be private
3     function offerOwnership(address beneficiary) public {
4         require(newProductOwner == address(0), " This product already has a new
      owner");
5         require(beneficiary == highestBidder);
6         newProductOwner = highestBidder;
7     }
8
9     function claimProductOwnership() public {
10        manager.transfer(priceIncludeFee * 1000000000000000000);
11        manager = newProductOwner;
```

```
        newProductOwner = address(0);
        productCondition = ProductCondition.NOTDEPLOYED;
    }
```

Retrieve Stakeholder Information The information that is stored inside this stakeholder can be retrieved by calling getProductInfo() function where it would return all the necessary information for rendering every important information for this stakeholder.

```
function getProductInfo() public view returns (address, uint, string,
    ProductCondition, bool, address, StakeholderType, uint, uint uint, uint) {
        return (
            manager,
            listedPrice,
            productDescription,
            productCondition,
            purchased,
            newProductOwner,
            stakeholderType,
            fixedAmountToRecoup,
            amountRecouped,
            percentage,
            movieId
        );
}
```

Processing Fee Calculation In FilmChain marketplace platform, every transaction will be included as a processing fee for using the platform. 2% of every transaction made will be added to the subtotal price and show on the checkout page. This 2% fee will not be able to refund even if your offer is not accepted. The 2% processing fee is calculated both for the seller when listed a stakeholder for sales and client when buy now or offer is made. For sellers, the 2% processing fee will be deducted from the listed price for sales. This means that the price that the seller would get is the price after the deduction. For buyers, the total price that needs to be paid is the offer price or the listed price plus the 2% processing fee from that price. In the function below, it can be seen that fraction of 49/50 and 51/50 are used to represent 0.02 processing fee. This is because in Solidity, it is not possible to have floats, therefore fraction is used instead.

```
function calculateSubTotalBuyNowForSeller(uint price) public pure returns (uint) {
        uint total_price = price * 49 / 50;
        return total_price;
}

function calculateSubTotalBuyNowForBuyer(uint price) public pure returns (uint) {
        uint total_price = price * 51 / 50;
        return total_price;
}
```

6.3.2 Compile and Deploy

After the smart contract is implemented, it needs to be compiled and deployed in the blockchain network. On this platform, the contract will be deployed in the Rinkeby test network via infura.

Compile.js Script

When compiling the contract, the compiled contracts are compiled to the /build directory inside the project folder with .json files. This is specified in the compile script using fs.ensureDirSync(buildPath). Additionally, since there exist multiple contracts to be compiled, every contract is compiled at once and output as a json file to the /build directory.

```
const path = require('path');
const solc = require('solc');
const fs = require('fs-extra');
```

```
4
5  const buildPath = path.resolve(__dirname, 'build');
6  fs.removeSync(buildPath);
7
8  const marketplacePath = path.resolve(__dirname, 'contracts', 'Marketplace.sol');
9  const source = fs.readFileSync(marketplacePath, 'utf8');
10 const output = solc.compile(source, 1).contracts;
11
12 fs.ensureDirSync(buildPath);
13
14
15 for (let contract in output) {
16   fs.outputJsonSync(
17     path.resolve(buildPath, contract.replace(':', '') + '.json'),
18     output[contract]
19   );
20 }
```

ABI output files

In the output json files, 2 important modules are compiled to which are the ABI and the Bytecode. Examples of an abi file are as follows:

```
1  [{
2      "constant":true,
3      "inputs":[],
4      "name":"offerCount",
5      "outputs":[{
6          "name":"",
7          "type":"uint256"
8      }],
9      "payable":false
10     "stateMutability":"view",
11     "type":"function"
12 },{
13     "constant":false,
14     "inputs":[{
15         "name":"_id",
16         "type":"address"
17     }],
18     "name":"addFactContract",
19     "outputs":[],
20     "payable":false,
21     "stateMutability":"nonpayable",
22     "type":"function"
23 },{
24     "constant":false,
25     "inputs":[],
26     "name":"claimProductOwnership",
27     "outputs":[],
28     "payable":false,
29     "stateMutability":"nonpayable",
30     "type":"function"
31 }
```

As can be seen, the ABI will describe the name of the function, the payability which is whether the function is payable, its output value and the state whether the function is read or write only.

Deployment with Infura to Rinkeby Network

Web3 instance is used to facilitate the deployment Grider, n.d. process. web3 instance is created and it will communicate with the provider where this provider will tell the instance what network it is going to communicate with. Mnemonic account is passed to the provider as the source of ether for the deployment where it comes from the 12 words password that is generated when a MetaMask account is created. When attempting to deploy to the Rinkeby network, it should be ensured that some node in the Rinkeby network exists to deploy the contract to. Infura API is used

to get access to a node that's hosted on the Rinkeby network by infura. Infura is the portal beyond web3 into the Rinkeby network.

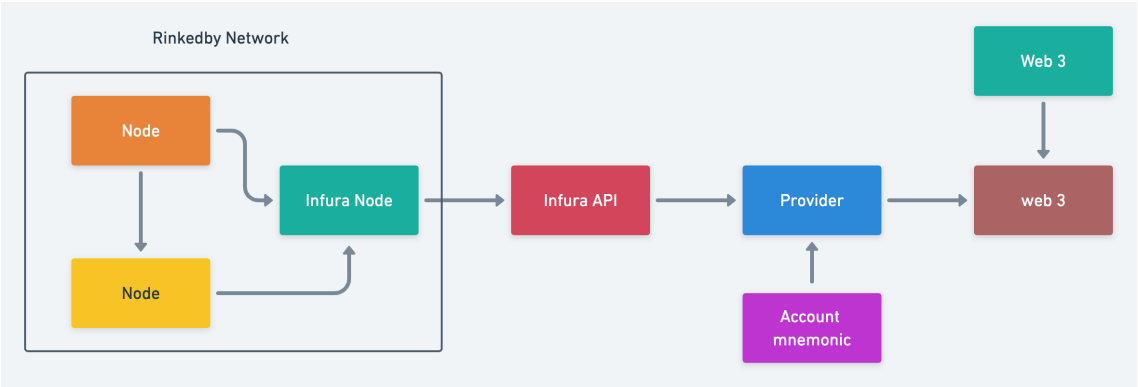

Figure 6.2: Deploy Flowchart

Deploy.js Script

In the deployment process, Truffle HD Wallet Provider is used as the account mnemonic and the 12 words password and the infura key are specified inside as the input parameter. The ABI (contract interface) is used to parse the instance of the contract and deployed using its bytecode.

```
const HDWalletProvider = require('truffle-hdwallet-provider');
const Web3 = require('web3');
const compiledCreateProduct = require('./build/CreateProduct.json');

const provider = new HDWalletProvider(
  'genre jacket still pill vocal yellow surround rubber discover cash artist
    traffic',
  'https://rinkeby.infura.io/v3/8586865a669c410592a233683a10ddbf'
);

const web3 = new Web3(provider);

const deploy = async () => {
  const accounts = await web3.eth.getAccounts();

  console.log('Attempting to deploy from account', accounts[0]);

  const result = await new web3.eth.Contract(JSON.parse(compiledCreateProduct.
    interface))
    .deploy({ data: '0x' + compiledCreateProduct.bytecode })
    .send({from: accounts[0], gas: '5000000' });


  console.log('Contract deployed to', result.options.address);
};
deploy();
```

Phase 2: Marketplace with FilmChain Token

Now the basic features of the marketplace in the backend are ready, ERC-20 token can now be included in the platform. The standard package of ERC-20 token is installed from openzeppelin library and the Marketplace smart contract is changed to is Ownable. The ERC-20 token is used as a pricing or the payment method in the marketplace where FCT20 is FilmChain ERC-20 token.

Constructor In the Marketplace contract, the tokenAddress object is defined. This holds the address of the deployed token contract and it is set in the constructor function.

```
FCT20 public tokenAddress;

function Marketplace(string description, uint amountToRecoup, uint
    amountAlreadyRecouped, uint percentageVal, uint movieIdVal, uint
    stakeholderType, address creator, address _tokenAddress) public {
        manager = creator;
        productDescription = description;
        productCondition = ProductCondition.NOTDEPLOYED;
        fixedAmountToRecoup = amountToRecoup;
        amountRecouped = amountAlreadyRecouped;
        percentage = percentageVal;
        movieId = movieIdVal;
        stakeholderType = StakeholderType(stakeholderType);
        setTokenAddress(_tokenAddress);

        // Set highest offer and bidder to initial state
        highestBidder = address(0);
        highestOfferPrice = 0;
        purchased = false;
        newProductOwner = address(0);
}
```

Buy Now with Token In the process of buying now, the process is similar to the logic inside one without token only that it is first checked whether their client is buying using ether or FTC20 token as the currency.

```
// buy the listed equity directly
    function buyEquity(uint _value, FTC20 _currency) public payable {
        require(productCondition == ProductCondition.DEPLOYED, "This product is
    already deleted");
        require(purchased == false, "This product is already sold out");

        // Use Ether
        if (address(_currency) == address(0)) {
            // refund the amount that is more than expected listed price
            if (msg.value > pricePlusFee) {
            uint refund = msg.value - (pricePlusFee * 1000000000000000000);
            msg.sender.transfer(refund);
            }
            require(msg.value = _value);
        } else {
            // This listing is in FTC20 token
            require(msg.value == 0, "Ether is not required");
            require(_currency.transferFrom(msg.sender, this, _value), "Transfer is
    not possible");
        }

        highestBidder = msg.sender;
        highestOfferPrice = listedPrice;
        purchased = true;
        offerOwnership(msg.sender);
        claimProductOwnership();
        emit BuyNow();
}
```

Make Offer with Token In the process of making an offer, the logic is similar to the process inside the normal Marketplace without using tokens. However, new FTC20 token attribute is added into the struct to specify as a currency and it is passed as a parameter when makeOffer function is called. When the function is called, since there is a new option for the listing to be in FTC20 tokens as well, so before the payment, it needs to check whether it is paid using normal ether or using FTC20 tokens. If client chooses to make an offer using FTC20 token, no ether is required to be passed in the msg.value.

```
struct Offer {
        uint offerId;
        address offerAddress;
        uint offerValue;
```

```
        string offerMessage;
        FTC20 currency;
}

function makeOffer(string description, uint _value, FCT20 _currency) public payable {

        Offer memory newOffer = Offer({
            offerId: offerCount,
            offerAddress: msg.sender,
            offerValue: _value,
            offerMessage: description,
            currency: _currency;
        });

        offers.push(newOffer);
        offerMaps[offerCount] = _value;

        // Use Ether
        if (address(_currency) == address(0)) {
            require(msg.value = _value);
        } else {
            // This listing is in FTC20 token
            require(msg.value == 0, "Ether is not required");
            require(_currency.transferFrom(msg.sender, this, _value), "Transfer is not possible");
        }

        // Check whether it is the highest offer
        if (msg.value > highestOfferPrice) {
            if (highestBidder != address(0)) {
            // Refund the previous highestBidder
            amountToRefund[highestBidder] = highestOfferPrice;
            }
            highestOfferPrice = _value;
            highestOfferMessage = description;
            highestBidder = msg.sender;
            highestOfferId = offerCount;
        }

        offerCount+=1;
        emit OfferCreated(address indexed bidder, uint price, string description);
}
```

set the address of the FilmChain token contract The setTokenAddress function is called to set its parameter to the public attributes of tokenAddress to specify the address of the deployed token contract.

```
function setTokenAddress(address _tokenAddress) public restricted {
    tokenAddress = FCT20(_tokenAddress);
}
```

6.4 Front End for Decentralised Application

The front end of this project is implemented using React Component, Javascript and Semantic UI React using JSX syntax.

React components help reduce duplication inside the javascript classes. Components that might occur in several areas in the project can be implemented once and reused throughout the project. In this project, the Menu bar and the Header appear on almost every page. Therefore, it is implemented inside a components directory only once and get called in the other files of the project. An example of the code snippets for the Menu bar is as follows:

```
export default () => {
  return (
    <Menu style={{ marginTop: '10px' }}>
```

```
4       <Link route="/">
5         <a className="item">
6           <Header>FilmChain Marketplace</Header>
7         </a>
8       </Link>
9       <Menu.Item>
10        <Input className="icon" icon="search" placeholder="Search..." />
11      </Menu.Item>
12
13      <Menu.Menu position="right">
14        <Link route="/">
15          <a className="item">
16            Home
17          </a>
18        </Link>
19
20          <a href="https://www.imdb.com/news/movie" target="_blank" className="item
    " >
21            News
22          </a>
23
24
25        <Dropdown item text='About'>
26          <Dropdown.Menu>
27            <Link route="/how-it-works">
28              <a className="item">
29                How it works?
30              </a>
31            </Link>
32            <Link route="/reviews">
33              <a className="item">
34                Reviews
35              </a>
36            </Link>
37            <Link route="/articles">
38              <a className="item">
39                FAQ
40              </a>
41            </Link>
42          </Dropdown.Menu>
43        </Dropdown>
44
45        <Dropdown item text='Contact Us'>
46          <Dropdown.Menu>
47            <Dropdown.Item color="facebook">
48              <Icon name="medium" />
49              <a href="https://medium.com/@BigCouch" target="_blank">
50              Medium
51              </a>
52            </Dropdown.Item>
53            <Dropdown.Item color="twitter">
54              <Icon name="twitter" />
55              <a href="https://twitter.com/bigcouchfilms" target="_blank">Twitter</
    a>
56            </Dropdown.Item>
57            <Dropdown.Item color="mail">
58              <Icon name="mail" />
59              <Link route="/contact-us">
60                E-mail
61              </Link>
62            </Dropdown.Item>
63          </Dropdown.Menu>
64        </Dropdown>
65
66        <Dropdown item text='Account'>
67          <Dropdown.Menu>
68            <Link route="/user/show">
69              <a className="item">
70                View Your Account
71              </a>
72            </Link>
73            <Link route="/reviews">
74              <a className="item">
```

```
                Settings
              </a>
            </Link>

          </Dropdown.Menu>
        </Dropdown>

        <Link route="/products/new">
          <a className="item" color="red">
            Sell
          </a>
        </Link>
      </Menu.Menu>
    </Menu>
  );
};
```

Figure 6.3: Menu Bar

6.4.1 Homepage

The final implementation of the homepage is shown in the figure below.

When accessing the web application, the first landing page that the client arrives will be this homepage. This homepage can be returned to when the button associated to the menu "Home" and "FilmChain Marketplace" is clicked. The menus mentioned are associated to the Link component from the React Router. Those links will direct client this homepage.

In rendering all the available stakeholders on the market, the listed of products are retrieved in the getInitialProps function. The stakeholders address is mapped and get render inside a single card. More information about a specific stakeholder will be displayed when clicking the View Product anchor. This will Link user and direct to the View Project Page. In each card, the stakeholder address is displayed instead of information about each specific card because to retrieve information for each card, it requires creating an instance of that contract address which means it needs to call another contract which can be done but will result in large amount of gas consumption or huge delay when loading to the homepage.

```
static async getInitialProps() {
    const products = await newproduct.methods.getDeployedProducts().call();

    return {
      products: products
    };
  }

  renderProducts() {
    const items = this.props.products.map((address) =>  {
      return {
        header: address,
        description: (
          <Link route={`/products/${address}`}>
            <a>View Product</a>
          </Link>
        ),
        fluid: true
      };
    });

    return <Card.Group items={items} />;
  }
```

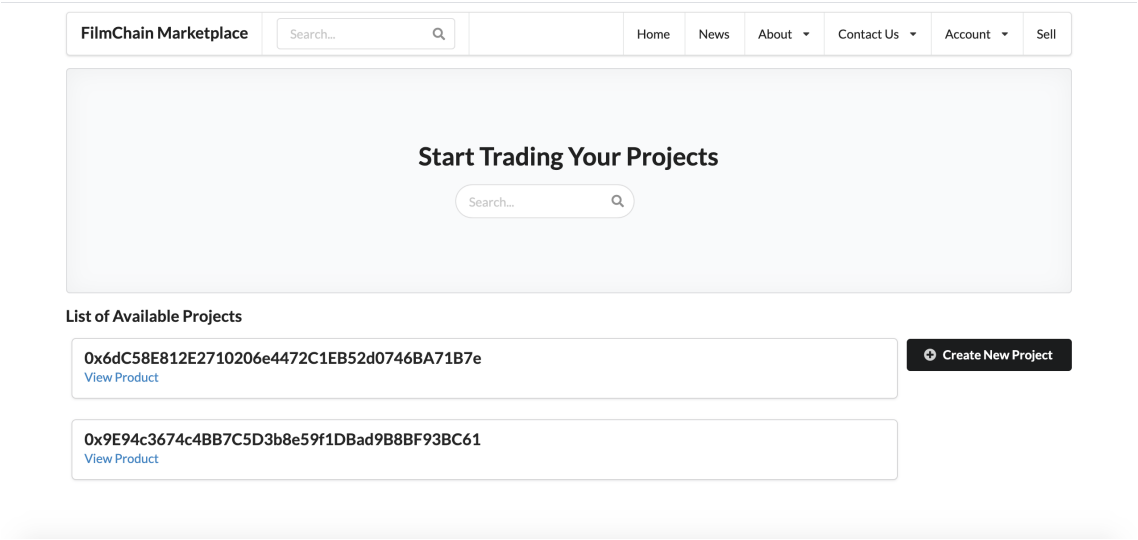

Figure 6.4: Homepage Implementation

6.4.2 Project Show Page

The implementation of the Project Show page is shown below.

The information renders inside this page is retrieved from the getInitialProps function at the top of the file by creating an instance of the marketplace contract.

```
<Table.Row>
              <Table.Cell width={4}>Product Condition</Table.Cell>
              <Table.Cell>productCondition</Table.Cell>
            </Table.Row>
            <Table.Row>
              <Table.Cell>Stakeholder Type</Table.Cell>
              <Table.Cell>stakeholderType</Table.Cell>
            </Table.Row>
            <Table.Row>
              <Table.Cell>Fixed Amount To Recoup ($)</Table.Cell>
              <Table.Cell>fixedAmountToRecoup</Table.Cell>
            </Table.Row>
            <Table.Row>
              <Table.Cell>Amount Recouped ($)</Table.Cell>
              <Table.Cell>{amountRecouped}</Table.Cell>
            </Table.Row>
            <Table.Row>
              <Table.Cell>Percentage</Table.Cell>
              <Table.Cell>percentage</Table.Cell>
            </Table.Row>
```

6.4.3 List for Sales Page

The final implementation for List for Sales page is shown below.

The page retrieves information of the stakeholder by getting information from the getInitialProps function. The function creates an instance of the marketplace contract and call the getProductInfo function from the contract.

```
static async getInitialProps(props) {
    const marketplace = Marketplace(props.query.address);

    const productInfo = await marketplace.methods.getProductInfo().call();

}
```

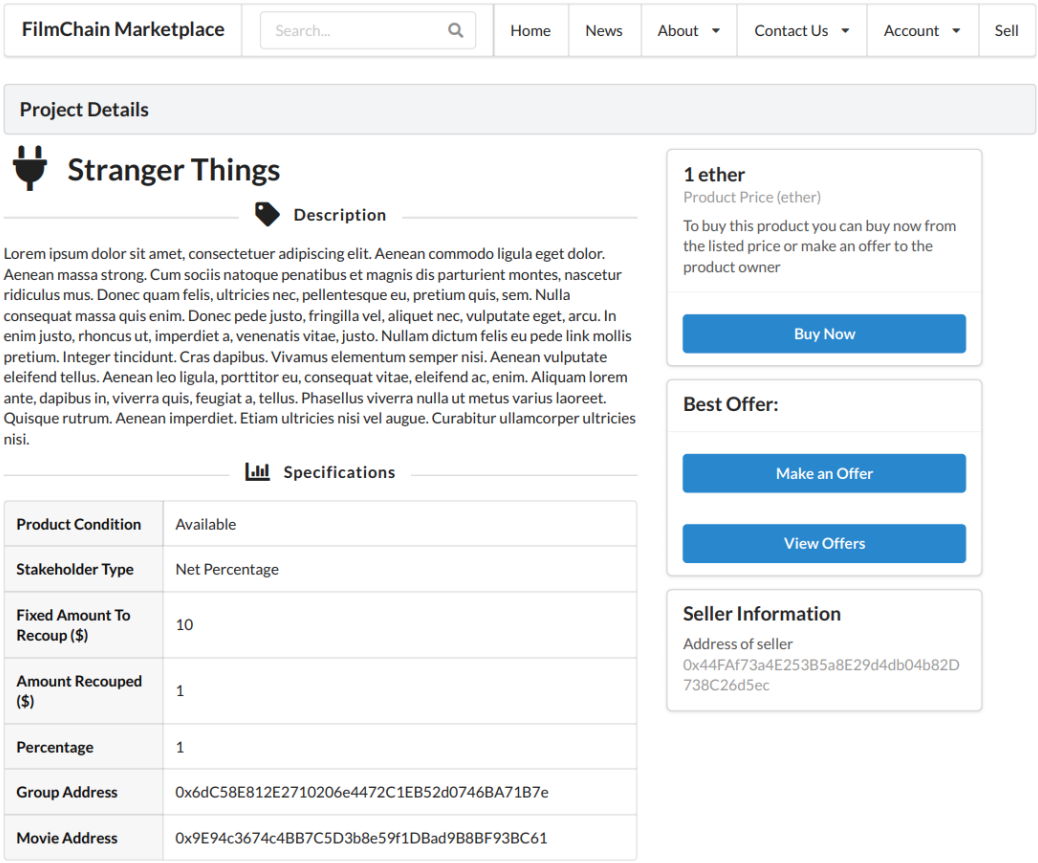

Figure 6.5: Project Show Page Implementation

6.4.4 View Offers Page

The implementation of View Offers page is shown below.

This page renders all the offer inside a table using table jsx and display a button only to the top row, which is the row that contains the highest offer. The code below shows the implementation to list the highest offers and all the relevant information associated with it inside the renderRow function.

```
renderRow() {
    return <OfferRow
      highestBidder={this.props.highestBidder}
      highestOfferPrice={this.props.highestOfferPrice}
      highestOfferMessage={this.props.highestOfferMessage}
      address={this.props.address}
    />;
  }
```

6.4.5 Make Offer Page

The final implementation of the Make Offer page is shown below.

The Make an Offer page is essentially a form that the client can insert the amount of money that is desired to offer for this stakeholder inside the input label. The address of the stakeholder is first retrieved inside the getInitialProps function. When the inputs are filled and Make Offer button is pressed, it will trigger the onSubmit function where it will send a request for transaction by calling the makeOffer function inside an instance of the marketplace contract. If the transaction

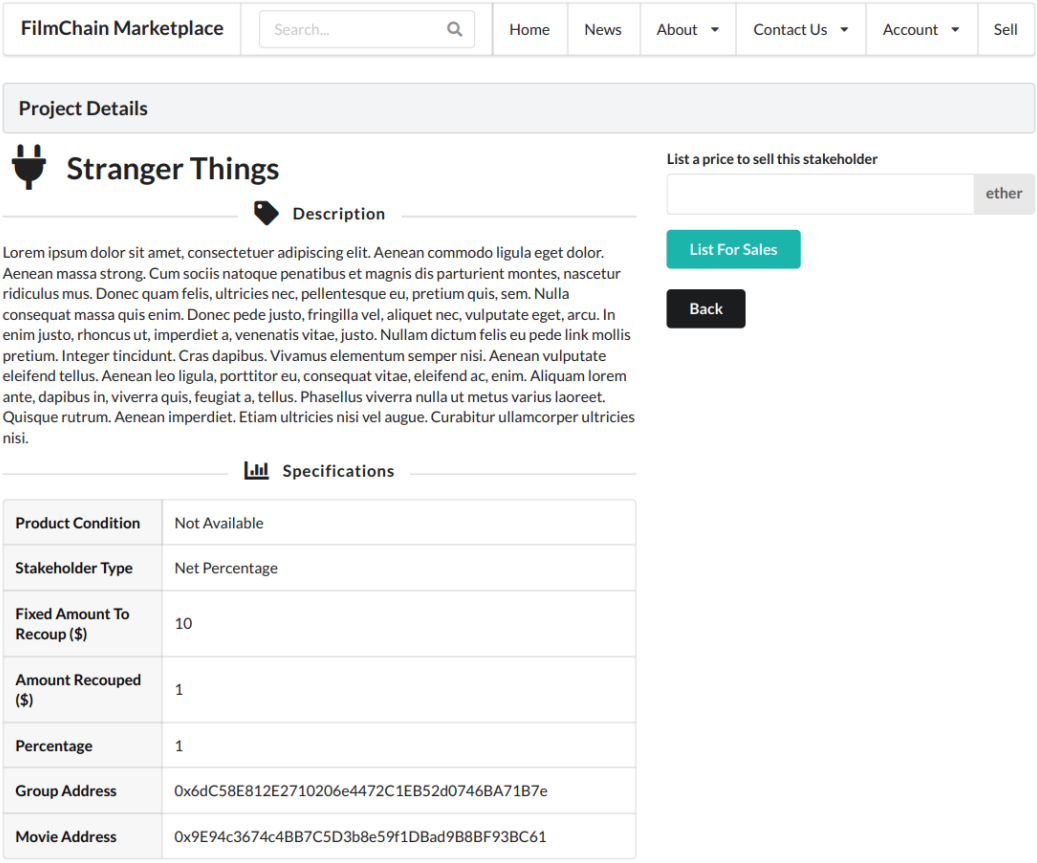

Figure 6.6: List for Sales Page Implementation

got accepted by MetaMask and is successful, then it will automatically route clients back to the Project Show page.

```
static async getInitialProps(props) {
    const { address } = props.query;

    return { address };
}

onSubmit = async (event) => {
    event.preventDefault();

    const marketplace = Marketplace(this.props.address);

    this.setState({ loading: true, errorMessage: '' });

    try {
      const accounts = await web3.eth.getAccounts();
      await marketplace.methods.makeOffer(this.state.description).send({
        from: accounts[0],
        value: web3.utils.toWei(this.state.value, 'ether')
      });

      Router.pushRoute(`/products/${this.props.address}/offers`);
    } catch (err) {
      this.setState({ errorMessage: err.message });
    }

    this.setState({ loading: false, value: '', description: '' });
};
```

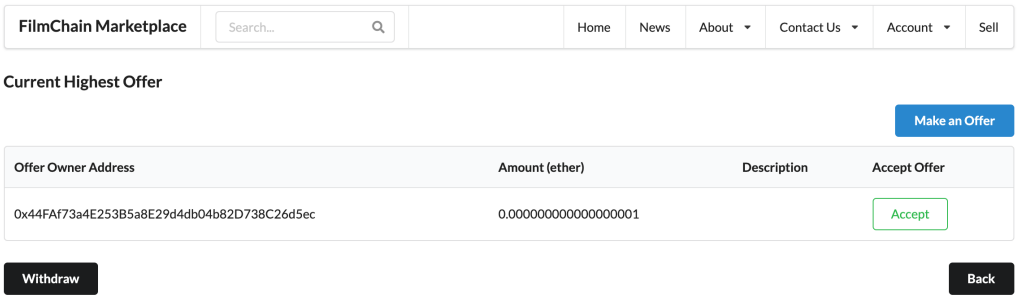

Figure 6.7: View Offers Page Implementation

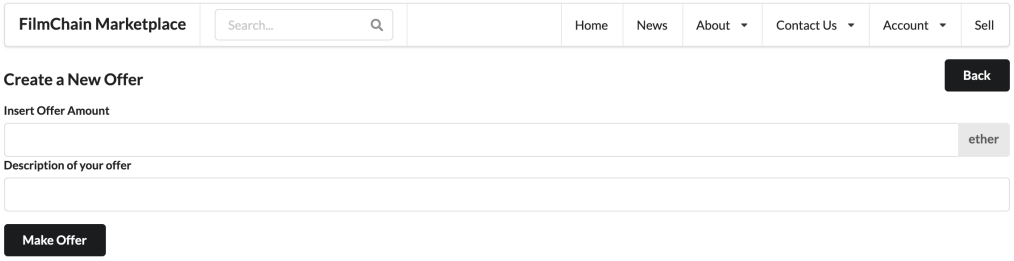

Figure 6.8: Make Offer Page Implementation

6.4.6 Buy Terms & Conditions Page

The final implementation of the page is shown below.

This page uses Grid to list the segments on the center of the page. This page links to the Checkout Summary page if the client agrees to the terms & conditions by clicking I understand.

```
    <Grid.Column width={10} style={{ marginTop: '20px' }}>
            <Header as='h2'>
              <Icon name='settings' />
              <Header.Content>
                Summary of How it Works
              </Header.Content>
              <br />
              <Header.Subheader>
                For more details on
                <Link>
                  <a> Terms & Conditions</a>
                </Link>
              </Header.Subheader>
            </Header>
            <Segment.Group>
              <Segment padded='very'>
                Make and Offer or Buy Now to immediately purchase the project
              </Segment>
              <Segment padded='very'>
                Once an offer is accepted, we will authorise the payment
              </Segment>
              <Segment padded='very'>
                The project ownership rights will be directly transferred to the
beneficiary account
              </Segment>
            </Segment.Group>
```

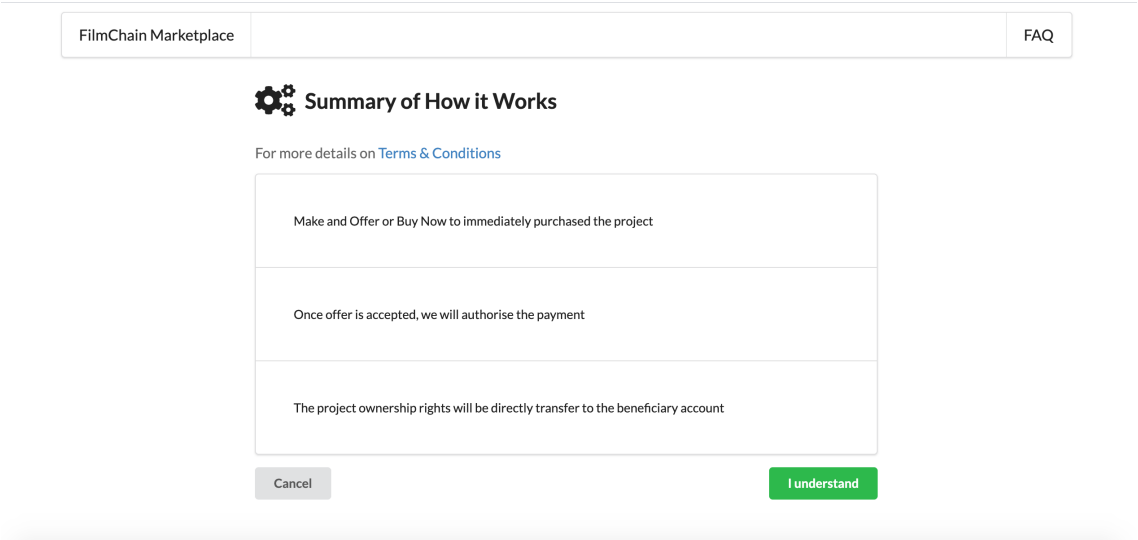

Figure 6.9: Buy Terms & Conditions Page Implementation

6.4.7 Checkout Page

The final implementation of the Checkout page is shown below.

The page includes a summary of the stakeholder that is retrieved from the props. Also rendering the price, processing fee and the subtotal price that needs to be paid in total. These logics are called in the getInitialProps function via creating an instance of the marketplace contract.

6.4.8 How it works? Page

The final implementation for How it works? Page is shown below.

An example of one of the components renders inside the page where it renders a header, then call a function renderBuyCards() that return a list of cards to be rendered for the section. Underneath, a button "Learn More About Buying" is linked using routers and direct to the More On Buying Page as can be seen on the route=‘./articles/more-on-buying‘.

```
renderBuyCards() {
    const items = [
      {
        header: 'BUY IT NOW / OFFER',
        description: 'Can immediately buy it now at the current listed price or
    make an offer that the seller can accept',
        fluid: true
      },
      {
        header: 'OFFER WILL BE SHOWN TO SELLER',
        description: 'After an offer is made, only the current highest offer will
    be displayed to the seller who can choose to accept it',
        fluid: true
      },
      {
        header: 'SUCCESSFULLY BUY NOW / OFFER ACCEPTED',
        description: 'The stakeholder will be automatically transferred to the
    beneficiary that buy the stakeholder',
        fluid: true
      }
    ];


    return <Card.Group items={items} />;
 }
        <Container>
          <Header as='h1'>
            <Icon name='plug' />
```

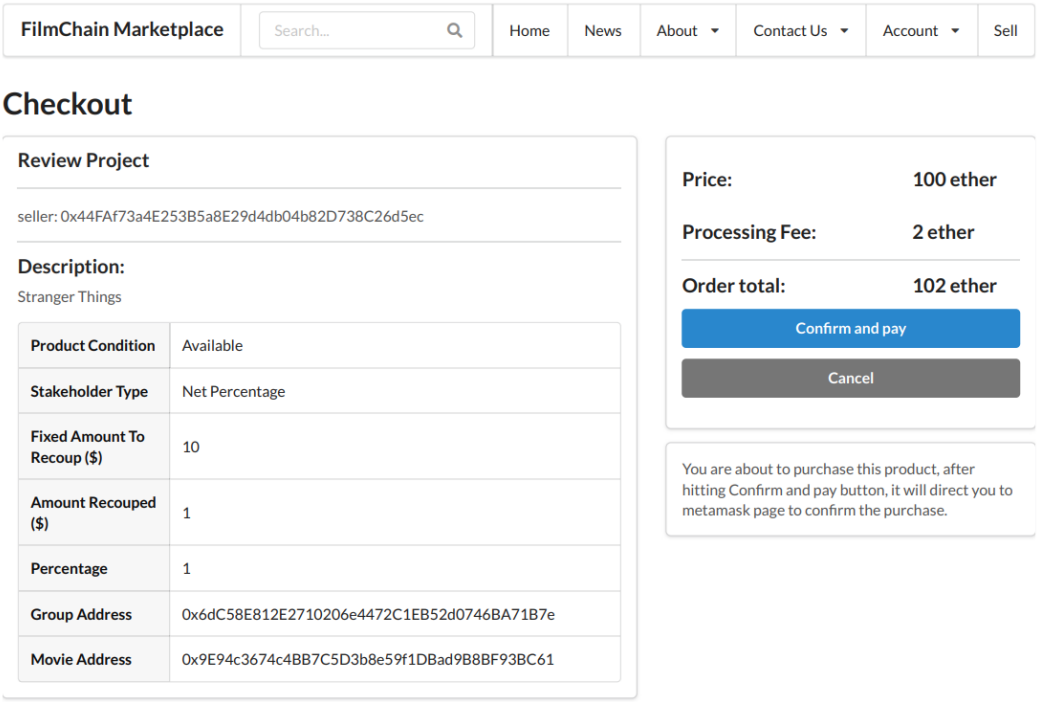

Figure 6.10: Checkout Page Implementation

```
26              <Header.Content>How to Buy?</Header.Content>
27            </Header>
28            {this.renderBuyCards()}
29            <br />
30            <Link route={'./articles/more-on-buying'}>
31              <a>
32                <Button color='teal' style={{ marginTop: '10px' }}>Learn More About
      Buying</Button>
33              </a>
34            </Link>
35          </Container>
```

6.4.9 More on Buying Page

The final implementation for More on Buying page is shown below.

A blog post is used in this page to display text information about buying to clients as well as providing useful links on the column on the right of the page that links to different articles.

```
1  renderBlogPost() {
2      return (
3
4        <div>
5          <Header as='h2' attached='top'>
6            Buying a Project
7          </Header>
8          <Segment attached>
9            <p>
10             There are 2 ways to buy in this trade marketplace where you can
11             either make an offer to the seller or immediately purchase by hitting
12              the Buy Now button.
13           </p>
14           <br />
15           <p>
16             Making an offer will show your interest in the listed project and
```

```
            an intention to buy at a certain price. All the offers will be listed
            from highest to lowest on the Offers page. You can also cancel or
            update your offer before your offer has been accpeted by the seller.
            Only the highest bid will contain the approve button which it allows
            the seller to approve the highest offer. After a deal has been made,
            all previous offers funds will be released to the beneficiary accounts
            but the owner of the account would have to hit Withdraw Funds Button
            to collect their money back.
          </p>

          <p>
            A deal will be made once the transaction has been accepted by either
            the buyer hit Buy Now button or the offer is accepted by the seller.
            If you don't want to risk the offers, just simply hit Buy Now Button
            to purchase the project immediately to ensure that you will be the next
            rights owner. The only accepted payment method is in ether.
          </p>
        </Segment>
      </div>
    );
  }
```

6.4.10 More on Selling Page

The final implementation for the More on Selling page is shown below.

This page uses blog post to display information similar to the More on Buying page and also provides links to other articles on the right hand column of the page.

6.4.11 Reviews Page

The final implementation of the Reviews Page is shown below.

In the Review page, list of reviews are rendered to display image and the text feedback of a client. The code snippets below show the render methods for this page which divides the comments into 2 sections. One for the reviews from important person and the section below for general users. The information for each can be retrieved inside each method.

```
render() {
    return (
      <Layout>
      <div>
        <Header as='h2' icon textAlign='center'>
          <Icon name='users' circular />
          <Header.Content>Reviews</Header.Content>
        </Header>
      </div>
      <Segment.Group stacked>
        <Segment padded='very'>
          <Header>
            HERE'S WHAT PEOPLE SAYS ABOUT US!
            <Header.Subheader>
            These reviews are a perfect glance of what we do and how our live
            trading marketplace has changed the film industry! Please also share
            your love and support of our platform below.
            </Header.Subheader>
          </Header>

        </Segment>
        <Segment padded='very'>
          {this.renderItems()}
        </Segment>

        <Segment padded='very'>
          {this.renderPeopleCards()}
        </Segment>

      </Segment.Group>
```

```
      </Layout>
);
```

6.4.12 FAQ Page

The final implementation of the FAQ page is shown below.

In this page, list of containers are displayed in order to show the title for each of the articles. These containers can be clicked and it will link clients to the page specify by the title. One of the implementations for a container is shown below where the routes direct client to a specified page.

```
  <Segment padded='very'>
              <Link route={'/articles/more-on-selling'}>
                <a>
                  <Header>How to List a Product for Sales?</Header>
                </a>
              </Link>
  </Segment>
```

6.4.13 Selling Fee Page

The final implementation of the Selling Fee page is shown below. This page renders a blog post describes the processing fee when selling stakes. List of other useful questions are linked and shown on the right column of the page.

6.4.14 Buying Fee Page

The final implementation of the Buying Fee page is shown below. This page is similar to the Selling Fee page, but describes the processing fees when buying instead. The page is rendered using blogs and containers and links.

6.4.15 List Project for Sales Article Page

The final implementation of the List Project for Sales Article page is shown below. The page uses render blog to describe how to add a listing for sales.

6.4.16 Contact Us Page

The final implementation of the Contact Us page is shown below. The page is essentially a form that will trigger when the submit button is clicked and it will send a message to the FilmChain team member.

6.4.17 Login Page

The final implementation of the Login page is shown below.

To login, clients must provide the account address and the password for that account into the specified fields where the password will be hidden. On submission, the system will signal the onSubmit function and sends login request to the backend where the address and password are checked. If the login is successful, it will automatically route client to the profile page, otherwise an error message will appear instead.

```
onSubmit = async (event) => {
    event.preventDefault();

    console.log(this.state.userAddress);
```

```
5
6       console.log(this.state.password);
7       this.setState({ loading: true, errorMessage: '' });
8
9       try {
10        console.log('reach try');
11        const accounts = await web3.eth.getAccounts();
12        await newproduct.methods
13          .login(this.state.userAddress, this.state.password)
14          .send({
15            from: accounts[0]
16          });
17        Router.pushRoute(`/user/${this.props.userAddress}/show`);
18      } catch (err) {
19        this.setState({ errorMessage: err.message });
20      }
21
22      this.setState({ loading: false });
23    };
```

6.4.18 Sign up Page

The final implementation for the Sign up page is shown below.

When signing up, clients are required to fill in first name, last name, email address and password. When Sign Up button is clicked, the onSubmit function will be called and it will send requests to the back end whether the account is possible to create. If the account address is already taken, then an error message would appear. The code snippets below also show the implementation for the password fields that provide placeholder and to hide the password while typing.

```
1  onSubmit = async (event) => {
2      event.preventDefault();
3
4      this.setState({ loading: true, errorMessage: '' });
5
6      try {
7        const accounts = await web3.eth.getAccounts();
8        await newproduct.methods
9          .createUser(this.state.email, this.state.firstname, this.state.lastname,
       this.state.password)
10         .send({
11           from: accounts[0]
12       });
13
14       Router.pushRoute('/user/login');
15     } catch (err) {
16       this.setState({ errorMessage: err.message });
17     }
18
19     this.setState({ loading: false });
20
21   };
22
23
24  render() {
25     return(
26         <Form.Field>
27             <label>Password</label>
28             <Input
29               icon='lock'
30               iconPosition='left'
31               placeholder='Password'
32               type='password'
33               value={this.state.password}
34               onChange={event => this.setState({ password: event.target.value })}
35             />
36           </Form.Field>
37     )
38 }
```

6.4.19 User Profile Page

The final implementation of the User Profile page is shown below.

The client can view the user page if it is already logged in by clicking View Your Account menu from the Account Menu bar. If the client has not logged in to the platform yet, it will first direct client to the Login Page.

The getInitialProps function fetched the user object and passed down. The information to be rendered onto the page is then returned from the getInitialProps function.

```
static async getInitialProps(props) {
    const accounts = await web3.eth.getAccounts();
    console.log(accounts);
    const userInfo = await newproduct.methods.getUser(accounts[0]).call();
    console.log(userInfo);

      return {
          address: props.query.address,
          accounts: accounts[0],
          userId: userInfo[0],
          email: userInfo[1],
          firstname: userInfo[2],
          lastname: userInfo[3],
          stakeholders: userInfo[4]
      }
  }
```

In rendering the list of cards of the available stakeholders that this user owns, the map function is used to loop through all the addresses of stakeholder from the stakeholders props. When clicking the View Info, it will direct clients to the View Info page for each of the stake where the stakes can be listed for sales at a certain price.

```
renderStakeholdersCard() {
    const items = this.props.stakeholders.map(address => {
      return {
        header: 'Stakeholder',
        meta: address,
        description: (
          <Link route={`/user/${address}/view-info`}>
            <a>View Info</a>
          </Link>
        ),
        fluid: true
      }
    });
    return <Card.Group items={items} />;
  }
```

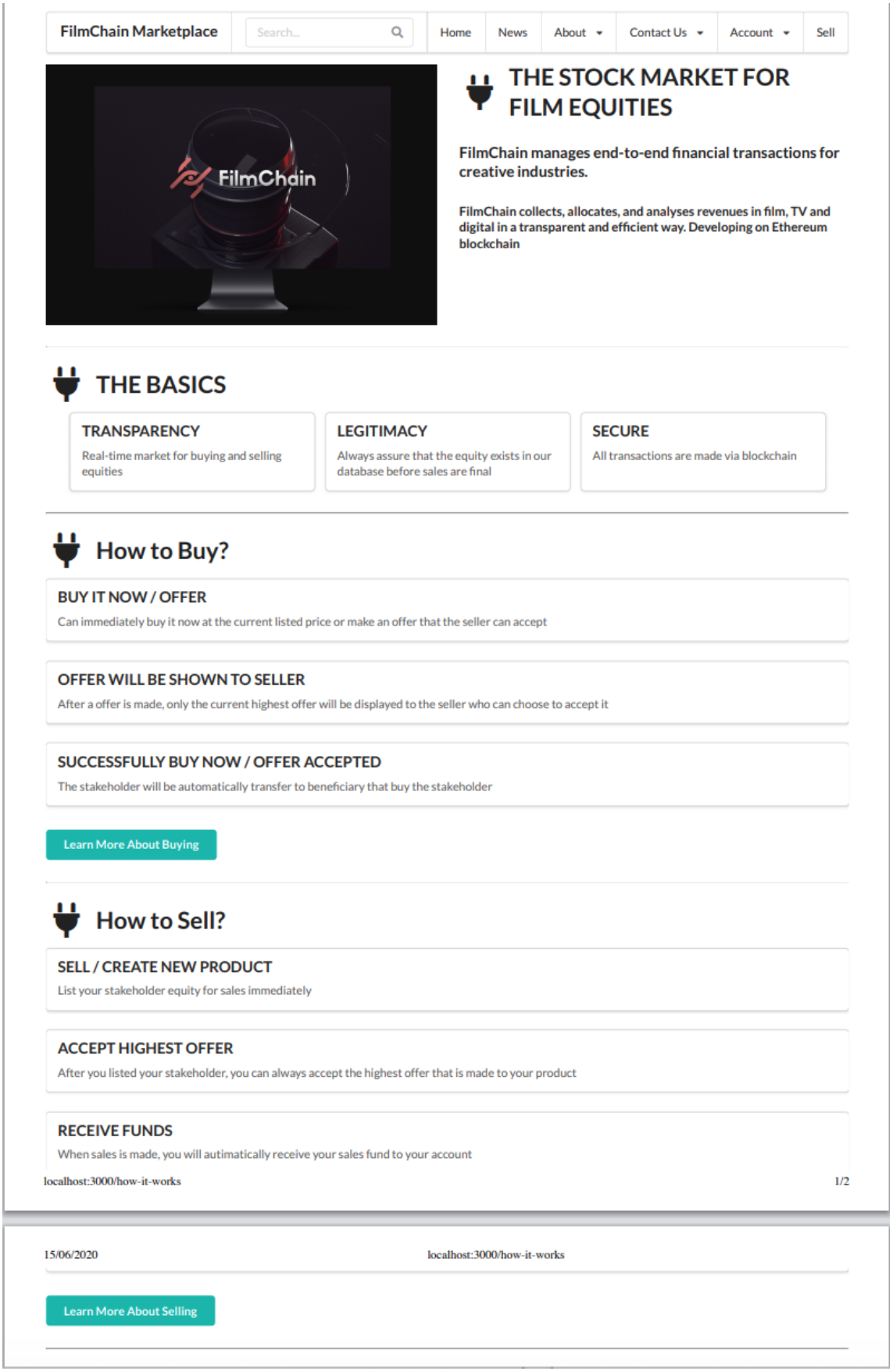

Figure 6.11: How it works? Page Implementation

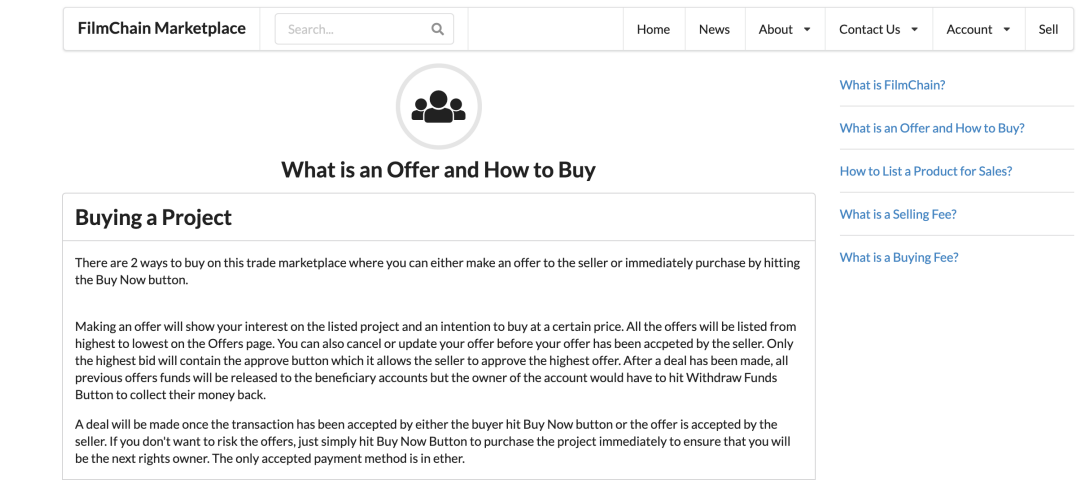

Figure 6.12: More on Buying Page Implementation

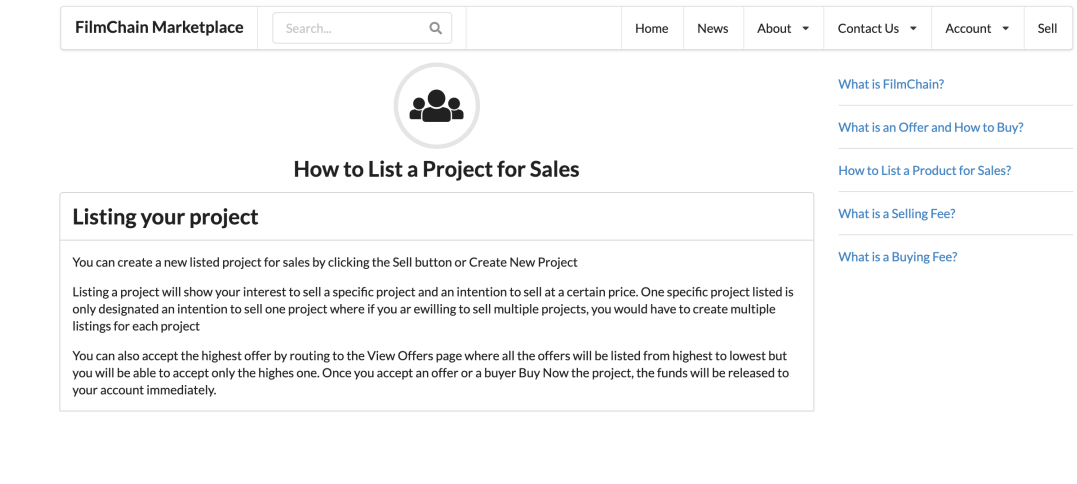

Figure 6.13: More on Selling Page Implementation

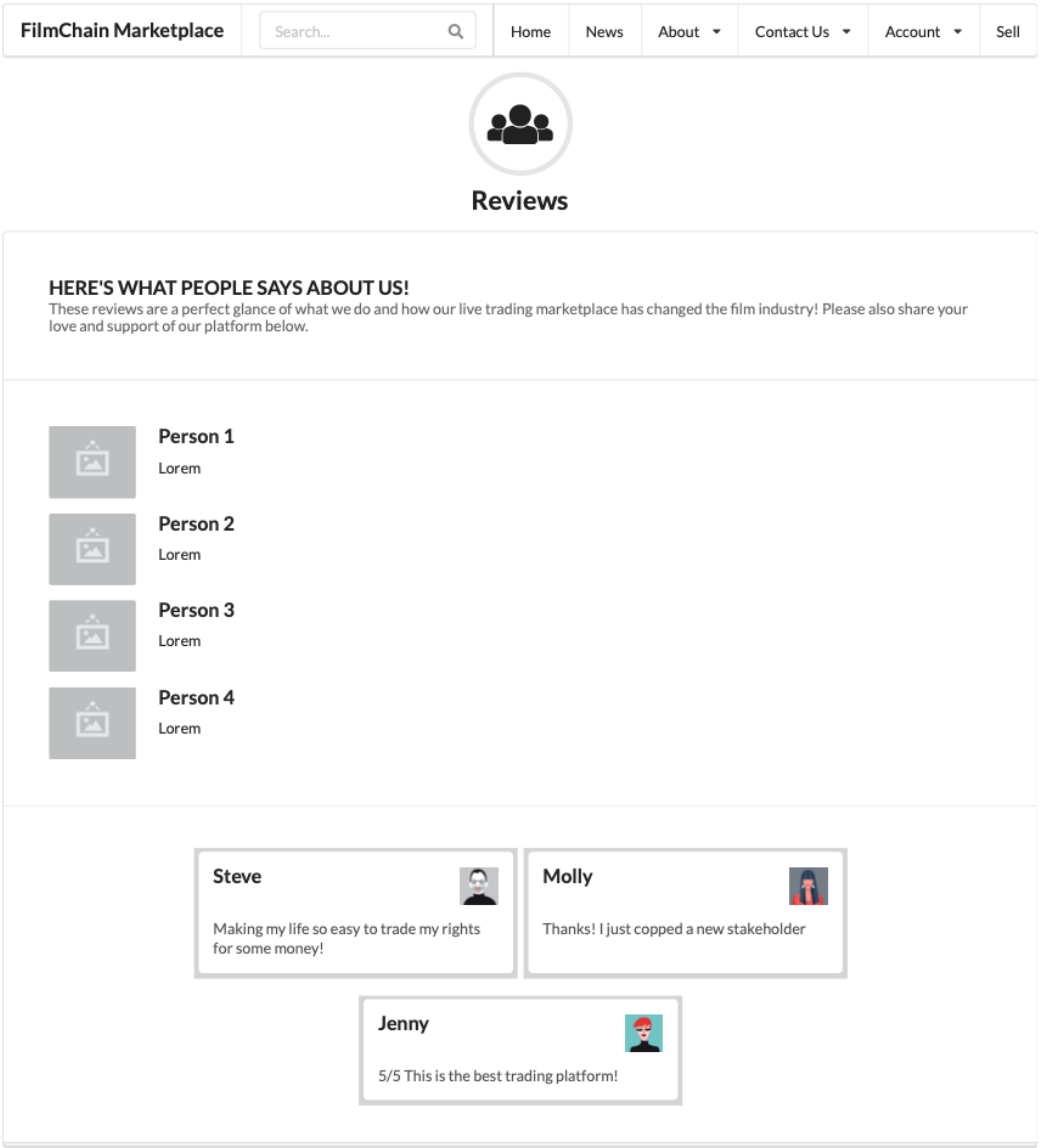

Figure 6.14: Review Page Implementation

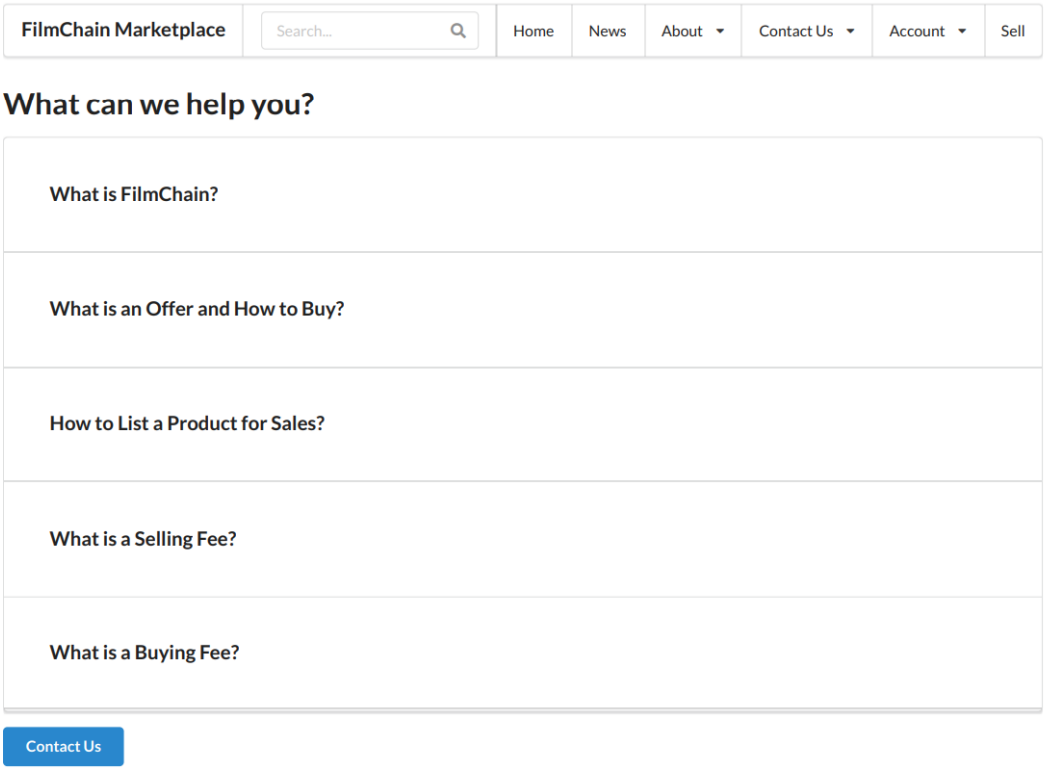

Figure 6.15: FAQ Page Implementation

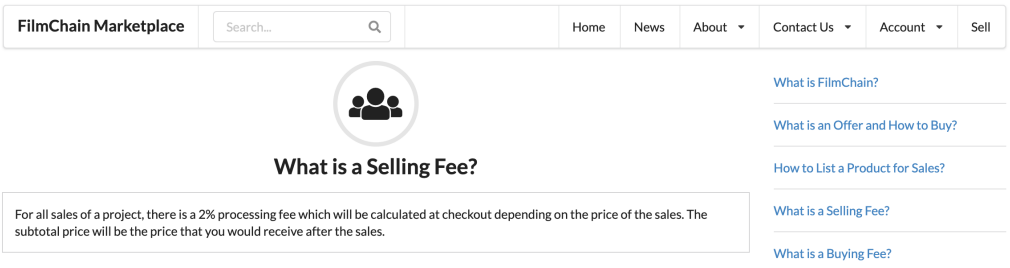

Figure 6.16: Selling Fee Page Implementation

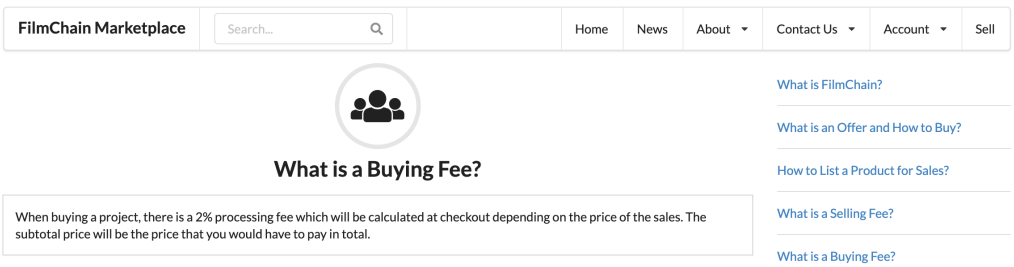

Figure 6.17: Buying Fee Page Implementation

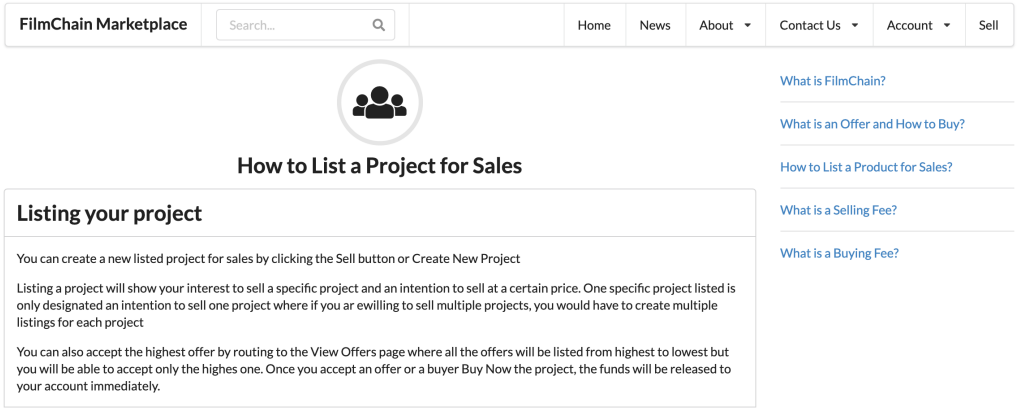

Figure 6.18: List Project for Sales Article Page Implementation

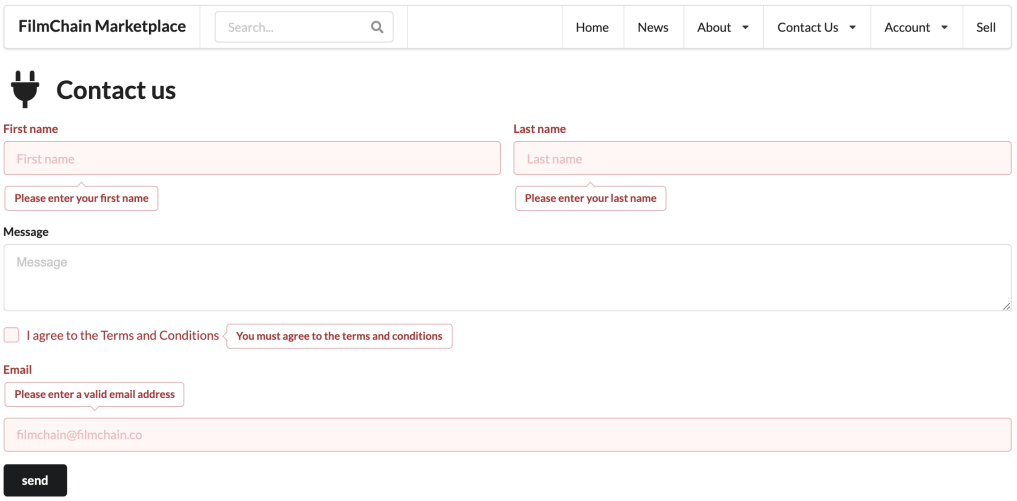

Figure 6.19: Contact Us Page Implementation

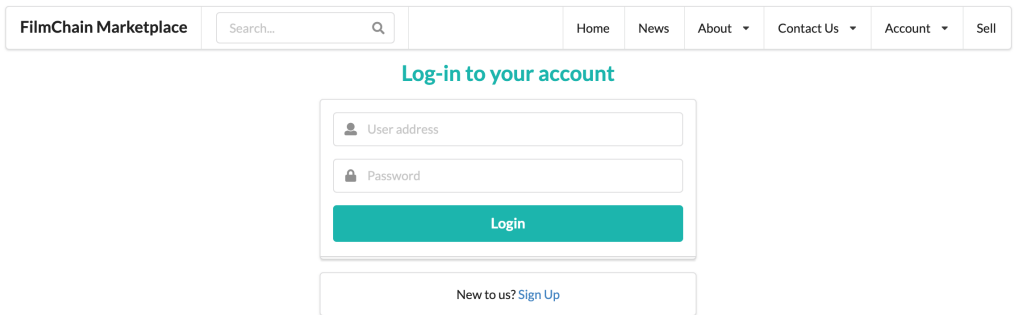

Figure 6.20: Login Page Implementation

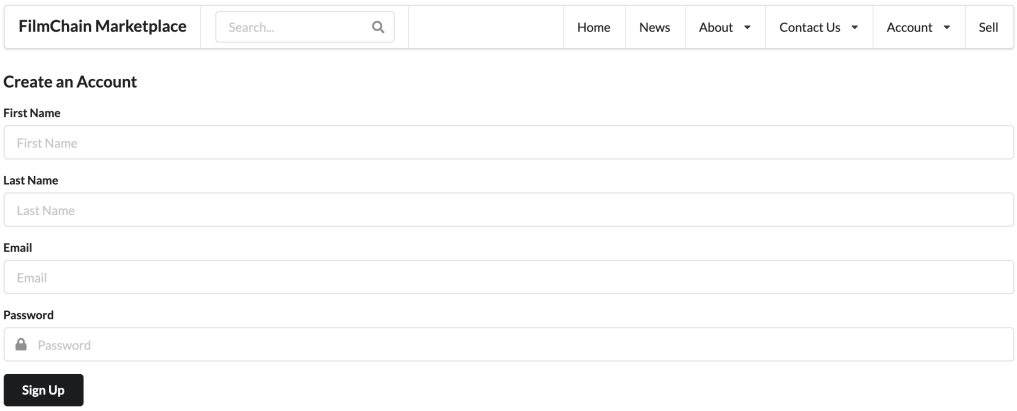

Figure 6.21: Sign Up Page Implementation

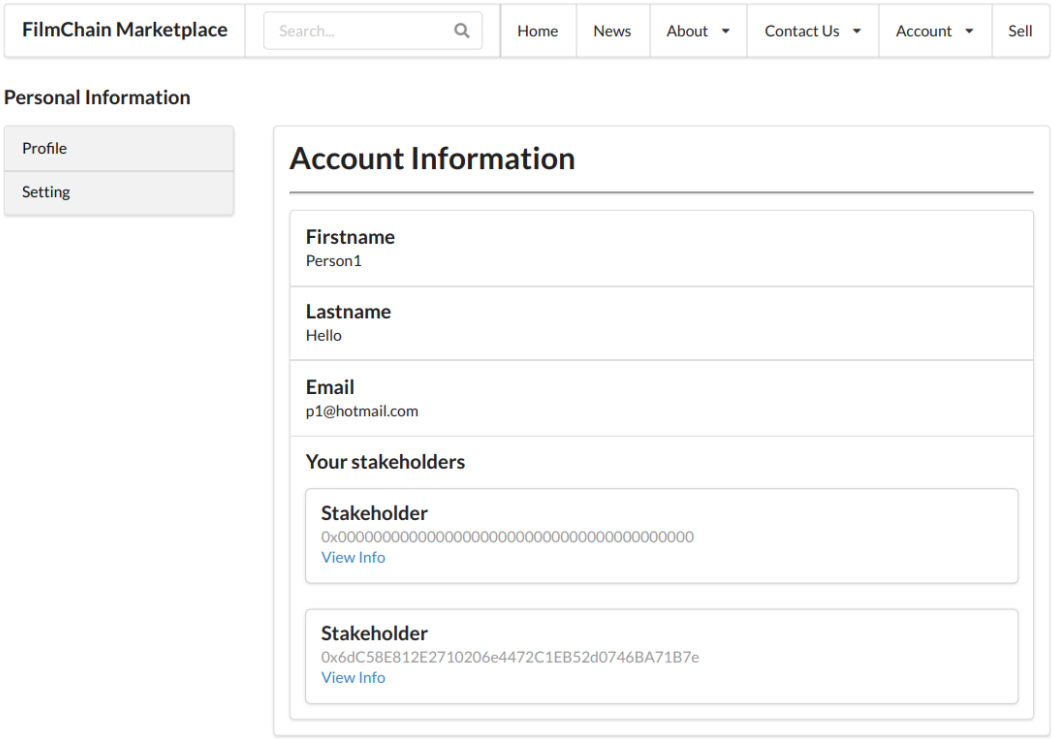

Figure 6.22: User Profile Page Implementation

Chapter 7

Evaluation

In this chapter, strength and weaknesses in each part of the project, including Front End and Back End implementations are discussed. Also, testing framework is used to test the features of the implementation. Lastly, challenges that occur throughout the projects are discussed.

7.1 Front End

The decentralised application provides full support of functionalities that are required in order to support all the features in a marketplace in consideration with making a well compatible with other FilmChain software products. Communication with FilmChain member has been smooth and continuously getting feedback from the team member allows platform the achieve how the web application should look visually. Testings are also undertaken to ensure that the functionalities are achieved as desired.

7.1.1 Strengths

Positive contributions has been feedback from FilmChain UX/UI team throughout the project. In every meeting, current status of the webpage will be displayed to a team member in order to get some comments or feedback about what should be required in each part of the progress. Then the requirements for each week will be recorded in the Trello channel to keep track of what to do in that week and at the end of the week, an updated version of the page will be shown to the member until a desired layout is achieved.

It is thought that one of the most important aspect of a web application is that it works as expected and easy to use. This web application platform provides a clean user interface with all highlighted information just for rendering a specific page without showing any duplication. Additionally, a plain color is used as a pattern for the whole project because it provides a more calm and relaxing environment as well as aid legibility and make the important part pops up. However, in the case of approving, accepting, rejecting, a specific colour that helps user to understand quickly will be used. For example, green button for accepting button and a red one for rejecting button.

Repetition pattern is kept for the overall design of the project to strengthen the overall looks and helps clarify that client is still using the same application even when browsing to different pages in the web application. Using repetition pattern makes different components in a page tie up together and organised. Additionally, since all the pages pattern remains consistent, clients can immediately recognise the unique property from each page and know what the page is for. For example, a Project Show page has descriptions and table on the left column and buy buttons on the right, but the View Offers page would have a table in the middle of the page showing a list of offers instead.

The linking from one page to another page also assures client to have a clear understanding of what is going to happen and what the user needs to do. For example, when client view a stakeholder that is for sales on the Project Show page and decides to checkout using buy now, the

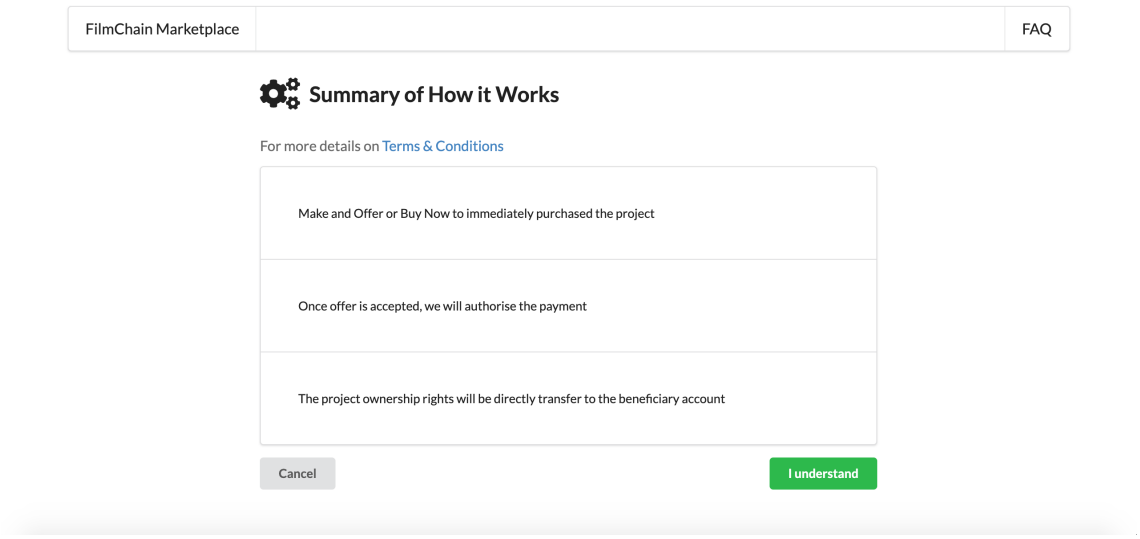

Figure 7.1: Term & Conditions page

page would direct client to the term & conditions page first where this page justifies the procedures and give a short summary for buying a stakeholder. Clients can also visit the full term & conditions page if one might want to read the full spec.

This platform provides with good user experience. After trials and errors, many feedbacks are also given by FilmChain team members and at the end including implementation of the Menu Bar with a unique platform name on the left and all the linkable menus on the right side. Also a big search bar is attached to the homepage to help client direct to a specific desired stakeholder as well as one small search bar in the menu bar. Also in the Project Show page, it lists both options for clients to choose which are Buy Now and Make Offer at the same place as well as View Offers' button that link to the list of offers page directly underneath the Make Offer button to help client browse to a desired page more easily.

A How it works page are made specifically for clients to have a broader understanding about the processes of essentially trading a stakeholder from the beneficiary(seller) side and the client(buyer) side. On the page it also has an option to Learn More and get an in depth information about the selected option of buying and selling. A FAQ page renders a list of frequently asked questions in articles and provides clients with answer to those questions within its own page. The platform also provides full support when needed by having a Contact Us page in case that a specific question arises and might not be answered in the FAQ or with some general questions.

This front end project is made by React Components which are similar to what other FilmChain web pages are built on, which makes the front end of this project compatible to FilmChain's platform and makes scalability with other FilmChain's platform possible. Also the CSS modules are also attached and can be used in the future with other platform.

In every transaction, platform will prompt update status to the user. A loading button will replace a normal button when the transaction is in progress and will revert back to normal button when the transaction is completed. Also error messages will be shown on every page if there is an event of an error occurring. For example, in an event of list a stakeholder for sales, it requires an input of a price(integer), however, if a client inserts a string of text instead, after trying to list at this price, the page would display a red error messages and states that it has type error of integer from the input section. This will notify the client that a price must be in an integer form and to retry to list the stakeholder with an integer price.

7.1.2 Weaknesses

In order to render each page, since the data that needs to be rendered onto the page are essentially attributes inside smart contracts and each stakeholder has different smart contracts address,

therefore, in rendering for example a Project Show page that needs to display all the information about a stakeholder, it requires to call a contract address to retrieve information from that contract which this process consumes an amount of gas. Also, on most of the pages it requires to retrieve information from the contracts, therefore, when browse to some pages client might suffer from a little delay in rendering.

Every transaction requires an approval of MetaMask and the communication between MetaMask and the blockchain network takes up a very long time. The platform already provides the user the status of the transaction, such as spinning button when the transaction is in progress, however, individual transaction takes up to 15 to 30 seconds where the user might suffer from it.

7.2 Back End

A smart contract infrastructure is successfully implemented using Solidity. It's features are tested with a Mocha testing framework to ensure that the contract provides with the correct functionalities. However, there are certain limitations for complex calculation and looping which result in some adjustment instead of what is preferred.

7.2.1 Strengths

Data structure of FilmChain's user data is studied before structuring the back end. Therefore, the same data types and data structures of user data in this project and in other FilmChain's project, which makes this project easily integrate with other FilmChain's platform if needed.

Design for having each stakeholder has a unique contract address makes it easier to understand a token where each token is essentially a contract address relating to stakeholder. Since each stakeholder can refer to as a token, buying and selling of stakeholder is essentially transferring a token address to one another in exchange with money in ether.

As this project is a separated project from other FilmChain's product, integration of this project and FilmChain's software still has not been tested. However, a significant amount of research has been undertaken in order to develop this project to make it as compatible as possible. Ranging from data structure to technologies in developing front and back end.

7.2.2 Weaknesses

When compiling the contract, the amount of gas for deployment is restricted to a certain limitation. This is due to Ethereum because it only allows a limitation of gas to be used when deploying contracts, therefore, not many complex functions can be

The restriction of gas is also a major concern for this project smart contract because in the data structure, every stakeholder is essentially a contract address. In order to interact with a stakeholder address, it requires a call to the contract address which this process consumes a significant amount of gas. Therefore, a number of contract call is limited in order to be able to run the contract smoothly. Contract call also cannot be used in loop since this will double up the gas consumption. Therefore, an alternative way can be used by calling function separately for each cycle.

Also, loops consume a lot of gas. As can be seen on the front end that in the View Offers page, offers are not arranged in order from highest to lowest, only the highest offer is shown at the top and other offers are of the order of the array mappings. It is possible to make the order from highest to lowest using for loop, but because the project code is very large and already contain many other more important functionalities, this makes the deployment exceeds gas limit. This is due to the fact that every cycle of a loop requires a certain amount of gas and in a practical world, if there are 100 total offers, the for loop would run 100 times to render an offering table. This would consume a very significant amount of gas and might take a long execution time.

This project is built and compiled with a certain solidity version. However, since Ethereum is a continuously developing project, new features and version are regularly updated. This would

result in compatibility problem because one version might not be compatible to compile and deploy with another version resulting in code breaking or the need to rewrite some functions in the smart contract.

7.2.3 Testings

In order to test the functionalities of the features inside smart contract, the solidity files must be compiled and deployed first. After compilation by using command line node compile.js inside the directory, an output json file is built inside the /build directory. In a built json file, 2 built modules will be shown. The first module is the ABI (Application Binary Interface) which is essentially the contract interface the describes the contracts and its function. This allows a function call from a contract to be possible. The second module output is the bytecode. The bytecode is interpreted by EVM (Ethereum Virtual Machine) and deployed on a local blockchain test network through Ganache CLI. Ganache provides tools of sets of unlocked accounts that can be used in local network testing. To access the local Ethereum node, Web3 libraries are used.

Mocha testing framework, a javascript testing framework is used to execute tests for the compiled smart contracts. In the test files, assertion libraries, ganache, web3, interface and bytecode are required to setup the test. Additionally, the provider, a tool used for communicating between instance of web3 contract and the Ganache local test network must be setup in order to get information about the accounts.

```
const assert = require('assert');
const ganache = require('ganache-cli');
const Web3 = require('web3');
const web3 = new Web3(ganache.provider());
```

Mocha framework makes it possible to write asynchronous test. Since interacting with smart contract is an asynchronous action, the core features of the contract, such as list for sales, buy now, make an offer, etc., are tested using Mocha. Additionally, before testing the functionalities of the contract, the contract needs to be assured that it has been deployed correctly by checking to see whether the deployed address exists.

In testing, the test file starts with a beforeEach() hook to get the instance of the deployed contract and do the initial set up. List of accounts of the Ganache are retrieved using the ethereum web3 library by calling web3.eth.getAccounts().

```
beforeEach(async () => {
  accounts = await web3.eth.getAccounts();

  createProduct = await new web3.eth.Contract(JSON.parse(compiledCreateProduct.
    interface))
    .deploy({ data: compiledCreateProduct.bytecode })
    .send({ from: accounts[0], gas: '2000000'});

  await createProduct.methods.newProduct('Stranger Things', '5', '1')
  .send({
      from: accounts[0],
      gas:  '5000000'
    });

  [marketplaceAddress] = await createProduct.methods.getDeployedProducts().call();
  marketplace = await new web3.eth.Contract(
    JSON.parse(compiledMarketplace.interface),
    marketplaceAddress
  );
});
```

A short code snippet of some of the test functions from deploying contract and some features of the contract is listed down below.

```
describe('Marketplace', () => {
  it('deploys create product and marketplace', () => {
    assert.ok(createProduct.options.address);
    assert.ok(marketplace.options.address);
  });
```

```
6
7   it('caller of the product is marked correctly', async () => {
8     const manager = await marketplace.methods.manager().call();
9
10    assert.equal(accounts[0], manager);
11  });
12
13  it('allows people to buy product', async () => {
14    await marketplace.methods.listForSales('5').call();
15    await marketplace.methods.buyEquity().send({
16      value: '5',
17      from: accounts[1]
18    });
19    const manager = await marketplace.methods.manager().call();
20    assert.equal(accounts[1], manager);
21  });
22
23  it('allows people to make offer', async () => {
24    await marketplace.methods.listForSales('5').call();
25    await marketplace.methods.makeOffer('Please').send({
26      value: '1',
27      from: accounts[1]
28    });
29
30    const highestOfferPrice = await marketplace.methods.highestOfferPrice().call();
31    assert.equal(highestOfferPrice, 1);
32  });
33 });
```

The snippets of the test results for main functionalities are as follows:

```
> filmchainethereum@1.0.0 test /Users/theerathat2008/filmchainethereum
> mocha


  Marketplace
    ✓ deploys create product and marketplace
    ✓ caller of the product is marked correctly (44ms)
    ✓ allows people to buy product
    ✓ allows people to make offer
    ✓ if multiple offer then only the highest appear top
    ✓ if someone bid higher then refund to the previously highest bidder
    ✓ allows to withdraw
    ✓ offer product ownership to the right


  8 passing (3s)

➜  filmchainethereum
```

Figure 7.2: Mocha Test Results

7.3 Challenges

The first major challenges in the beginning of the project were to get a clear understanding of the tasks to accomplish. Familiarise with FilmChain platform requires to study from objectives, data structures, back end to the front end. Each of the parts also has its own contributing technologies, for example, Solidity smart contract and React front end. The platform is still hosted on the local blockchain, therefore, FilmChain's project need to be run locally so a trial to run the whole new concept software and getting started is required. Additionally, complicated software code needs to be understood throughout repositories in order to understand the project and design a compatible and scalable project to FilmChain's platform. This has no doubt that it consumed a large amount of time.

FilmChain data structure is very complicated. Since it also contains all the information about a client and infrastructure for the revenue waterfall structure which is way too complicated to be used in this project. Therefore, a simpler version of the data structure is used solely for this project. In contrast, many trials of data structure have been made throughout the project because many requirements are concerned. The simpler data structure must be compatible with FilmChain platform and must be able to scale in the future where testing of many trials and error experiment are conducted until the best solution is concluded. This process is very time consuming and requires smart contract code to be rewritten multiple times.

In the back end, most of the functionalities are implemented inside smart contracts. Solidity and blockchain is a new concept and language for me so I have to get started by learning the languages which is one of the most time consuming modules. At the beginning, smart contract is implemented with a certain number of functions inside Remix, an online Ethereum IDE and compiled, deployed and test the functionalities inside it. However, since this project requires a connection with a front end, Remix is not suitable for the situation. Therefore, all the implementations are moved to use atom and Visual Studio Code. In this step, local development environment is set up including compiling, deploying and testing the smart contracts. Creating a compile and a deploy script requires knowledge of understanding process of compiling contracts into an ABI bytecode. This bytecode is then used to create an instance of a contract and deploys onto a local test network. To interact with the smart contract, web3 provider must be implemented to provide service to call functions inside the smart contract. Multiple methods have been tried until a successful connection is established because this project structure is not very similar to another project structure that can be found in the tutorials. Error debugging process is also a hurdle because in the case of failing compilation, meaningless error messages are given out and might not lead to the correct solution. Some of the compiling errors can be solved by trying on Remix but this is time consuming to run the same contract in multiple places. Additionally, only a specific version for each of the technologies can be used together as a whole in order for the compilation to be successful. For example, the truffle hd wallet provider must only be version 0.0.3 in order to be bale to successfully deploy the project. Additionally, Solidity compilers have various different versions, where different versions use different syntax and some parameter types are only available in one version. For example, in a version of solidity, string and bytes are similar and accessing each character is possible. However, in another version that results in an error.

React Components front end is decided to use in this project. However, javascript and React components are a new concept for me. This is the first time to develop a front end using React so a deep understanding of javascript and React components must be fulfilled in order to write an efficient front end algorithm. Also, linking and routing each page require an external knowledge of web3 and require to implement its own implementation. Additionally, an integration with MetaMask requires the project to be able for both clients that have MetaMask and that does not have one so both cases needed to be implemented and check before rendering each page and this requires an in depth server structure studies.

Connecting front and back end is also a hurdle. The need to understand and wire the connection from contract compiled code into a bytecode and used it to deploy on the local blockchain network. The ABI from the compiled contract is used as an interface to create an instance of a contract using web3 and connect each javasript page from the instance of the contracts. In order to understand the whole connection process, several data from each step need to be understandable and converted from one type another similar type when it's in different tool. For example, an address is converted to a string.

Multiple technologies are essential for a marketplace blockchain platform. MetaMask wallet and infura sets up and structure is quite complex at the back end because a key of access is required in order to process a transaction, therefore, the keys needs to be put carefully at the right place otherwise the transaction would fail. Also, the communication process between MetaMask to infura to blockchain needs to be understood in order to establish a fully connected transaction. As can be seen in the project that every time when a transaction is going to happen, MetaMask will pop up for clients to approve it.

During the time of the project, a pandemic event of the spread of corona virus occurs. As an international student, it is required for me to return to my home country before a certain date. Preparations and gathering all documents to provide at the airport is important and a quarantine

of at least 14 days is required after arriving in the home country. During the quarantine time, I lived in a separate flat and do not have access to a good internet service. Therefore, during those times, a limited 3g network is used throughout the time. Additionally, due to the closure of stores and restaurants, food supply was very hard to find at the time. During these times I need to prepare myself with all living necessities and only have a small amount of time per day to continue on this project.

A considerable amount of time is used for an end-to-end testing since a transaction of buying, making an offer of a stake, etc takes up to at least 15 seconds each for completion. One end-to-end manual testing of functionalities could take up to a considerable amount of time just for approving the transaction and wait for it to be successful when several end-to-end tests are conducted.

Chapter 8

Conclusion

Through this project, an end-to-end marketplace is implemented, providing users with functionalities to list stakeholder for sales which is essentially a token address of the ERC-20 token, buy stakeholder now, make an offer, etc. while ensuring a safe and transparent marketplace environment that meets the initial objective of the project. Including a secure Solidity smart contract that are deployed to the blockchain network and also a web interactive front end that helps user with trading options.

This project's contribute to FilmChain's vision to scale its platform further to provide a token economy secondary market for its client to be able to trade ownership tokens.

A research is undertaken to first understand how to develop a back end with a similar data structure to FlimChain's platform which leads to an experimental approach that results in a simpler data structure version, but remains with all necessary information in order to scale to FilmChain's platform in the future and accomplish this project. Marketplace smart contract infrastructure where the platform provides clients with substantial functionalities in the back end aspects. Ranging from listing stakeholder for sales, buy stakeholder now to make an offer, where a successful connection is established between MetaMask and infura inorder to process the transaction to the blockchain as well as providing supports for clients from providing articles that are about frequently asked questions and a Contact Us page that client can message FilmChain team directly if a question might arise. Also, algorithm to calculate the subtotal price to be paid when making and offer and buying now including processing fee for using FilmChain's marketplace is integrated which is essentially the amount of money needed paid as a commission fee of the platform. The platform also allows client to withdraw or update their sales listing price or offer price when it is made. Additionally, the platform also collects user data and store information about stakeholders as well as all the stakeholders that are owned by a client. New users can also create a new account by providing an email address, first name, last name and a MetaMask account address.

The rest of the time is used for developing a good user experience web application. The web application is fully connected to MetaMask, infura as well as the Solidity smart contract back end and is compiled and deployed on the blockchain network. These processes required a substantial amount of time to fully understand multiple programming languages and different technology specifications. Multiple user experience options are tried and finalised with a Menu bar and a big search bar in the home landing page to help user to browse more easily. Also, compact pages are made to display all the relevant information that user needs for each action from Project Show page that renders information about each specific stakeholder including stakeholder type which tells the recoupment structure, it's percentage, it's movie, etc to View Offers page that display all offers that are made to this sales stakeholder and seller being able to accept the offer if agrees. Additionally, the platform supports individual user to visualise account information as well as stakeholders that are owned by this account. The account Profile page also gives features of being able to view more information about each stakeholder and to create a new listing for sales from the View Info page. Not only that, the platform also provides useful data about the platform, including information about how to use the platform and to learn more about buying and selling via this application which are very important for new clients. Clients are also able to give reviews where it will be displayed on the Reviews page where this would give other clients an insight of a feedback about

this platform as well as give the confidence to use the platform. Moreover, the platform provides full user support by having a Contact us page to being able to directly contact FilmChain team if a question or a problem might arises.

8.1 Lessons Learnt

At the beginning of the project, variety of in depth independent researches are undertaken especially for blockchain and smart contract. With no prior knowledge, a substantial amount of knowledge is achieved by understanding the concept of blockchain and how blockchain technology can be used to provide a secure and transparent web application without the need for a central server. Additionally, the use of msg.sender in Solidity smart contract language makes it possible to keep track of the current account information that is communicating with the blockchain. As developing project, more practical use cases of blockchain technology arise, therefore, more substantial research is taken along the way including understanding compilation and deployment processes. Additionally, exploring Truffle and Ganache to represent information about accounts and make use for interacting with a local blockchain network from sending to receiving ether is used and a new testing framework, Mocha and Web3.js are explored to develop an end-to-end tests from interacting with smart contract for testing its functionalities.

In developing this project, wide range of programming languages, technologies and tools are used. With no prior experience in developing web application, many prerequisites such as getting familiar with javascript and React needs to be achieved. The use of components in React makes it possible to overcome any repetitions of rendering components and this saves up time from having to code similar duplicate codes. For example, the menu bar or articles list on each article page can be implemented once and save as a file in component and call from other classes when it is needed.

Not only programming experiences are achieved, a good engineering practices must be ensured throughout the project. This project is developed iteratively and having calls with FilmChain team member give early user feedback for what to alter and approach next as a short term focus for each week. During the development process, understanding that FilmChain's requirements can be dynamic and evolve over time. User requirements are kept on track using Trello and Git version control. Additionally, information that is rendered inside a page maximises clarity for each page and unused information are hidden. Also, implementations are continuously refactor in order to keep the coding environment clean and useful comments are written for a clearer understanding for viewers.

FilmChain is considered to be a startup company, but consists of many talented members. Attending meetings as well as individual calls provide me with an insight of working in the real blockchain working environment and the current trends of the industry.

8.2 Future Work

Beyond the scope of this project, the following extensions can be expected as a future work

8.2.1 Integrate with FilmChain Real Client Data

In this project, mock users' data are used as a data source for testing end-to-end marketplace. However, this could be extended by extending the simpler version data structure to support full user structure used in FilmChain platform. Then real user data can act as a database and test an end-to-end marketplace real user experience with real user data.

8.2.2 Upgrade Front End

In this project, a simple and compact version of web page is designed in order to keep the web pages clean. However, this could be extended using extra external javascript and CSS libraries to make a more flashy version web page if desired. Research can be conducted on how making the Offer page to be extended to render a list of offers in order from highest to lowest by using the small gas consumption. An individual profiles page can be extended to support profile editions and settings.

8.2.3 New Payment Methods

In this project, ether is used as the main currency for trading stakeholders. In order to gain ether, clients need to exchange it with real fiat currency first using other software. This could be extended to provide full support of any currency. For example, the platform could be able to support the transaction in usd, gbp, eur, jpy, rmb, fr, etc and able to exchange currencies when a transaction in conducted in a target currency real time. Also, other payment methods such as credit cards and paypal can be researched into in order to provide client with an easier payment solution.

8.2.4 Recommendations

On this project's homepage, the list of available stakeholders are displayed. This provides enough information for clients to browse to each stake. However, the machine learning model can be used to display different filters as an extension. For example, recommended stakeholders that a specific client might be interested in or best seller movie's stakeholder that most users are interested in.

8.2.5 Price Determination

In this project, all the price determination for each stakeholder is only driven by the listed price of the holder and the offer price from the clients. However, in this way, it is not possible to determine the real worth value for each stakeholder. Therefore, a machine learning model can be implemented to recommend a real market price that each stakeholder might worth considering attributes such as percentage, stakeholder type, recoupment structure, etc.

8.2.6 Deployment to Public Blockchain

In this project, the marketplace is currently deployed to the Rinkeby Test Network, which is a blockchain network that acts similar to the main network but no real money is used in each transaction. However, once a full service is ready, it could be deployed to the main public network, which would allow all clients to be able to access to this platform including non FilmChain's client. This would give more possibilities of new users interacting with this platform more widely.

Appendix A

First Appendix

A.1 Project Plan

The following are my project plans that I would like to achieve throughout the terms along with key deadlines.

- January: Research the difference between fungible and non-fungible tokens
- January: Propose appropriate tokens for representing shares in the revenue streams of films (ERC-20 vs ERC-721)
- January: Research how to represent tokens for ownership rights
- January: Set up Solidity and truffle framework
- February: Build ERC-20 token using Solidity
- 6 February 2020: Interim report deadline
- February: Collect requirements for features that should be the basic of the secondary market.
- February: Start initial basic implementation for secondary market with smart contracts integrate with smart tokens and ownership rights.
- February: Explore the source code for ERC-20 and ERC-721 that is relevant to support my project
- 28 February 2020: Project reviews with supervisor (Knottenbelt, William) for discussion about the progress of the project and initial thoughts
- March: Implement test to validate the code for smart contracts.
- March: Start on the design for the decentralised application for clients to be able to trade ownership rights
- Initial implementation of the decentralised app using React and Node JS
- April: Connect the smart contract with the decentralised app using Truffle, Gnache and Metamask.
- April: Full stack testing from front end to back end
- 11-15 May 2020: Project Health Checkup with my supervisor (Knottenbelt, William) regarding the status of the project and plan a roadmap listing what needs to be done to complete the project successfully
- May: code clean up
- End May and Early June: UG Project Fair at Imperial College London
- 15 June 2020: Final report submission day

Bibliography

BitDegree (n.d.) MetaMask Wallet Review. Available from: https://www.bitdegree.org/tutorials/metamask-wallet-review/

Chen, Y. (2018) Blockchain tokens and the potential democratisation of entrepeneurship and innovation. Available from: https://reader.elsevier.com/reader/sd/pii/S0007681318300375?token=F7A78669F75E2939D97DD18846FE73D92C6EF166B8EAFFB81F74ABF2B0FE802C1ABA86AC664D333500104B73DC28

Chevet, S. (2018) Blockchain Technology and Non-Fungible Tokens: Reshaping value chains in creative industries, 6–35. Available from: DOI:https://poseidon01.ssrn.com/delivery.php?ID=684088071004065010104108096080002109026012051033042091108126110074072028068073109121101122062000120EXT=pdf.

Choi, S. (n.d.) What is MetaMask? Really... What is it? Available from: https://medium.com/@seanschoi/what-is-metamask-really-what-is-it-7bc1bf48c75

coinhouse (n.d.) What is a token. Available from: https://www.coinhouse.com/learn/ethereum/what-is-a-token/

Copeland, T. (n.d.) What is Infura? Available from: https://decrypt.co/resources/what-is-infura

Curran, B. (n.d.) What is Ethereum's Infura? Scalable Access to Ethereum and IPFS. Available from: https://blockonomi.com/ethereum-infura/

Fernandez, P. (n.d.) What is Solidity and What is a Smart Contract. Available from: https://medium.com/@peter.fernandez/what-is-solidity-and-what-is-a-smart-contract-214efbbe2bf4

Grider, S. (n.d.) Ethereum and Solidity: The Complete Developer's Guide. Available from: https://www.udemy.com/course/ethereum-and-solidity-the-complete-developers-guide/learn/lecture/9020484#overview

Liehuang Zhu Keke Gai, M. L. (2019) Blockchain Technology in Internet of Things, 10–20. Available from: DOI:https://link-springer-com.iclibezp1.cc.ic.ac.uk/content/pdf/10.1007%2F978-3-030-21766-2.pdf.

Michael J.W. Rennock Alan Cohn, J. R. B. (2018) Blockchain Technology and Regulatory Investigations. *The Journal*, 36, 38. Available from: DOI:https://www.steptoe.com/images/content/1/7/v3/171269/LIT-FebMar18-Feature-Blockchain.pdf.

Sahu, M. (n.d.) What is Truffle Suite? Features, How to Install, How to Run Smart Contracts. Available from: https://www.upgrad.com/blog/what-is-truffle-suite/

Ting, Z. (2005) Hollywood Stock Exchange method applied to automobile industry: Online trade platform of marketing forecast for future automobile model sales, 5–15. Available from: DOI:https://search-proquest-com.iclibezp1.cc.ic.ac.uk/docview/305360784?pq-origsite=primo.

Vercel (n.d.) Next.js The React Framework. Available from: https://nextjs.org

Zoup (2018) Why most definitions of "non-fungible" are incorrect. Available from: https://nonfungible.com/blog/why-most-definitions-of-non-fungible-are-incorrect