NTNU
Norwegian University of
Science and Technology

The Music Industry on Blockchain Technology

Andreas Fougner Engebretsen
Hallvard Kristoffer Boland Haugen

Master of Science in Communication Technology

Submission date:	June 2018
Supervisor:	Danilo Gligoroski, IIK
Co-supervisor:	Chris Carr, IIK

Norwegian University of Science and Technology
Department of Information Security and Communication Technology

NTNU – Trondheim
Norwegian University of
Science and Technology

The Music Industry on Blockchain Technology

Andreas Fougner Engebretsen
Hallvard Kristoffer Boland Haugen

Submission date:	June 2018
Responsible professor:	Danilo Gligoroski, IIK
Supervisor:	Chris Carr, IIK

Norwegian University of Science and Technology
Department of Telematics

Title: The Music Industry on Blockchain Technology

Student: Andreas Fougner Engebretsen
Hallvard Kristoffer Boland Haugen

Problem description: The music industry is a complex network of creators and administrative roles. Artists, composers, songwriters, producers, publishers, record labels, and collectors all want a piece of the revenue. The distance between the creators of music and the consumers is large. The money travels through a long chain of third-party companies before they reach the creators. Slow royalty payouts make it difficult for artists to rely on income from this alone.

The industry also faces different challenges with standards and transparency. Contracts and split sheets between the different players are hidden between each joint in the chain. There is no global copyright register and no common platform for licensing creative work. This makes it difficult for music consumers and potential customers to obtain the rights to use.

In our thesis we focus on the problems related to transparency, and slow and reduced royalty payouts as a result of excessive middlemen in the industry. Our goal is to create a proof of concept (POC) demonstration of music copyright registration and licensing through smart contracts on the Ethereum blockchain.

We spent the last semester researching the music industry and its complex composition of various players. For our master project we will attempt to develop a decentralized application on the Ethereum blockchain. The inherent nature of blockchain technology provides a set of attractive properties which seems to solve the problems described in the paragraph above. More specifically, we will use the Solidity programming language to write the back end logic for our application. This logic will define licensing features with automatic payouts to appropriate right holders as well as how music copyrights will be issued in the form of sell- and tradeable Ethereum tokens.

Given that we are satisfied with the back end, we will develop a web application for users to interact with it. This will provide a graphical user interface for copyright registration and licensing. The application will be created in a JavaScript-based framework (Angular 2 / React). In order to read and write data(code) from the blockchain we will use the web3.js library.

Responsible professor:	Danilo Gligoroski, IIK
Supervisor:	Chris Carr, IIK

Abstract

The topic of revenue streams in the music industry has been frequently discussed since the transition from sales to streaming started when Spotify launched in 2008. Even though revenues in the industry have reached new heights, musicians express dissatisfaction with lower royalty payouts. Moreover, it has become increasingly more difficult to understand the royalty calculations. With today's complicated licensing agreements, money flows through a complex chain of third parties before it reaches the musicians. The industry struggles with transparency and efficiency, and the musicians are paying the price. Meanwhile, blockchain technology has developed since its first implementation with Bitcoin in 2008. Today, more advanced blockchains can run decentralized transparent applications that utilize the technology's efficient transaction system. With the industry issues and the promises of blockchain in mind, we investigate how blockchain technology can be applied to solve value chain problems within the music space.

In this thesis, we identify core issues in the music industry, propose a decentralized application (dApp) that attempts to solve these issues and implement the proposed solution. We develop the business logic using smart contracts on the Ethereum blockchain and make an associated web application using a JavaScript framework. The dApp works as a global copyrights database where musicians can register and license musical works. We exploit Ethereum's efficient transactional system to manage license purchases. Furthermore, we discuss the advantages and disadvantages of blockchain based solutions.

Sammendrag

Inntektsstrømmer i musikkbransjen har blitt hyppig diskutert siden overgangen fra salg til streaming som startet da Spotify ble lansert i 2008. Selv om inntekter i bransjen har nådd nye høyder er musikere utilfreds med lavere royaltyutbetalinger. Det er også blitt stadig vanskeligere å forstå beregningene bak utbetalingene. Med dagens kompliserte lisensavtaler flyter penger gjennom en kjede av tredjeparter før det når musikerne. Næringen sliter med åpenhet og effektivitet og det er musikerne som betaler prisen. I mellomtiden har blokkjedeteknologien utviklet seg fra den første implementeringen av Bitcoin i 2008. I dag kan mer avanserte blokkjeder kjøre desentraliserte og transparente applikasjoner som benytter seg av teknologiens effektive transaksjonssystem. Med bransjens problemer og blockchains løfter i i fokus, undersøker vi hvordan blokkjedeteknologi kan brukes til å løse problemene i musikkindustrien.

I denne masteroppgaven identifiserer vi kjerneproblemer i musikkbransjen, foreslår en desentralisert applikasjon (dApp) som sikter på å løse disse problemene, for så å implementere den foreslåtte løsningen. Vi utvikler forretningslogikken ved hjelp av smarte kontrakter på Ethereum-blokkjeden og lager en tilhørende webapplikasjon ved hjelp av et JavaScript-rammeverk. Den desentraliserte applikasjonen fungerer som en global opphavsrettsdatabase der musikere kan registrere og lisensiere musikalske verk. Vi utnytter fordelene ved Ethereums transaksjonssystem for å håndtere lisenskjøp.

Preface

This Master's Thesis is written by Andreas Fougner Engebretsen and Hallvard Kristoffer Boland Haugen as part of the Communication Technology Program at the Department of Information Security and Communication Technology, Norwegian University of Science and Technology. It is the final deliverable for the program, and was conducted during the spring semester of 2018. The thesis builds on a pre-study conducted by Andreas and Hallvard in the fall of 2017 that uncovers issues in the music industry.

Acknowledgements

First of all, we would like to express our gratitude to Chris Carr, our supervisor, and Danilo Gligoroski, the responsible professor for the guidance and valuable feedback throughout the semester. They believed in our idea and helped us define our own subject for this thesis making necessary adjustments along the way. Secondly, we would like to thank the Ethereum Community for the available tutorials and articles online. It would be impossible to learn Solidity and smart contract deployment without them. Finally, we would like to thank our family and friends for asking tough questions and providing input along the way.

Contents

List of Figures

Glossary

Ethereum Ethereum is a blockchain based software platform that enables smart contracts and decentralized applications (dApps) to be built and run without any downtime, fraud, control or interference from a third party.

Ethereum Virtual Machine The Ethereum Virtual Machine is the runtime environment for smart contracts in Ethereum.

Fiat Money Fiat money is a currency without intrinsic value that has been established as money, often by government regulation.

Gas The cost for running a transaction on the Ethereum blockchain. It represents the "work" a transaction requires to be processed..

Right Coin The official ERC721-token for Rights Done Right. It has the abbreviation RCN..

Smart Contracts Smart contracts are self-executing contracts with the terms of the agreement between buyer and seller being directly written into lines of code.

Trustlessness Trustlessness is a model that does not require trust to safely interact and transact.

List of Acronyms

ASCAP	American Society of Composers, Authors and Publishers.
BaaS	Backend-as-a-Service.
BMI	Broadcast Music, Inc..
CLI	command line interface.
dApp	decentralized application.
ETH	Ether.
EVM	Ethereum Virtual Machine.
GRD	Global Repertoire Database.
ICO	initial coin offering.
IMDb	Internet Movie Database.
NTNU	Norwegian University of Science and Technology.
POC	proof-of-concept.
PRO	performance rights organization.
RCN	Right Coin.
SESAC	Society of European Stage Authors and Composers.

Chapter 1

Introduction

In this chapter we explain the motivation for the research project and state our objectives and research question. We then explain the method for how we will answer the research question as well as outline the structure of the thesis.

1.1 Motivation

Blockchain technology combines decentralization with cryptographical hashing algorithms to establish trustless economic transaction systems. Some claim that it is the most important technological invention of our age [1]. Even though blockchain has a lot of attractive properties and promises, few companies, organizations, and governments have embraced its potential in real practical applications. Today, almost ten years after blockchain's birth with the release of bitcoin's white paper [2], more people are starting to understand and develop innovative solutions that go beyond cryptocurrencies [3]. Bitcoin was the first blockchain application that demonstrated the ideas of trustless cryptographically secured transaction systems. Newer second-generation blockchains, such as Ethereum, attempts to facilitate autonomous trustless decentralized applications (dApps) that can potentially disrupt conventional economic systems. The music industry is one example of such a financial system. Blockchain technology seems to offer solutions to problems that musicians have complained about regarding transparency, efficiency and fairness along the industry value chain [4] [5] [6]. We find it compelling to research the possibilities of applying blockchain technology to existing issues in today's music industry. People within the music space have also expressed their interest. Nick Mason, one of the founding members of the rock band Pink Floyd, expressed his interest in the foreword on the Blockchain For Creative Industries Research Cluster Middlesex University report [4]:

"*If blockchain technology is going to be the future, we need to dig in and make it happen.*"

- Nick Mason

1.2 Goals and Research Question

Based on the motivation, we define the following goals:

G1: Identify the core problems in the music industry and understand how blockchain technology can be utilized to resolve the issues.

The music industry is a complex hierarchy of different players. We define G1 to understand the structure of the industry and how revenues flows through the value chain before it reaches the musicians. This knowledge is vital in order to understand how the promises of blockchain can apply in music industry applications. G1 also involves researching which blockchain properties are meaningful and relevant to the discussion. In addition to the mentioned analytical goal, we also define a more specific practical goal:

G2: Demonstrate that musicians and other industry players can benefit from transparent blockchain based decentralized applications.

Blockchain technology seems to have a lot of promises in many different industries and applications. The Ethereum platform enables developers to make decentralized applications residing on the blockchain. We define G2 to get a better understanding of how to develop such a decentralized application and to provide an actual solution that addresses authentic problems. In our case, the goal is to create a Proof-of-Concept (POC) demonstration that can offer players in the music industry valuable features.

To reach the defined goals, we address the following research question:

RQ: How can blockchain technology be utilized in order to solve problems related to transparency, efficiency, and fairness along the music industry value chain?

1.3 Research Methods

In this thesis, we investigate the potential of applying blockchain technology in the music industry by following the design science approach. We do this by first analyzing the relationship between industry players to identify the core problems. We then proceed to implement a POC music copyright database and licensing solution, utilizing the Ethereum blockchain platform. We evaluate the results and discuss benefits and challenges for blockchain based solutions.

1.4 Outline

The structure of the thesis is split into three distinct parts:

1.4.1 Background Analysis

The first part is the background analysis found in the second chapter:

- **Chapter 2: Background** explains the structure of today's music industry and exposes the core issues within the music space. The chapter also describes the basics of blockchain technology and the Ethereum platform. Moreover, it discusses the most important properties and implications of trustless decentralized applications.

1.4.2 Planning and Developing the Proof-of-Concept

The next part consists of chapters 3 and 4, which contain planning and implementation of the proof-of-work solution.

- **Chapter 3: Application System Design** demonstrates the use cases for our POC, proposed system architecture, and relevant technologies.
- **Chapter 4: Implementing a Decentralized Application** presents the technical implementation of the Ethereum-based back-end logic as well as the front-end code.

1.4.3 Contributions, Discussion and Conclusion

The last three chapters make up the final part of the thesis. Here we present the project's contributions which primarily includes the results of the POC. We evaluate and discuss the results before offering our conclusion.

- **Chapter 5: Results** presents our contributions including the results from the background analysis and the POC solution. The project's open source nature also contributes to the blockchain development community.
- **Chapter 6: Discussion and Evaluation** highlights the advantages and disadvantages for Ethereum-based dApps compared to traditional centralized systems. Furthermore, we evaluate our implementation and point to specific areas with improvement potential.
- **Chapter 7: Conclusion** summarizes the presented work by answering the research question. It also presents future work.

Chapter 2 Background

This chapter forms the background analysis of the master project. We present a detailed description of today's music industry players and the relationship between them. Furthermore, we analyze how these relationships provide a basis for complex industry issues. This chapter also includes a high-level description of blockchain technology and its most important properties. Finally, we give an overview for the Ethereum blockchain and explain how it facilitates decentralized applications.

2.1 The Music Industry

In this section, we will give an overview of the structure of today's music industry. The process of writing and recording music, and getting it out to consumers involves a network of players - each with different roles and responsibility. We argue that it is essential to have an understanding of who does what to grasp the critical industry problems we attempt identify.

2.1.1 Songs Are Two Different Pieces of Work

When most people talk about a song, they typically refer to a recording that one can hear on the radio or streaming services. However, in the music industry the term describes more than what is heard. A song consists of two elements; the musical composition and the recording of a musical composition.

Musical Composition

Most modern music starts with the act of writing notes or lyrics down on a piece of paper. We define everything about a song which can be written down as a musical composition. That includes notes, chord progressions, arrangement, melodies and so on. A composer is a person who writes musical compositions in different music genres. In most modern music that contains lyrics, we typically use the term songwriter. A songwriter can either write the instrumental composition or the lyrics for a song.

Songwriters obtain copyrights for their work once it is created. If a song has several authors, they agree on how many percent of the copyrights they each own. That is often the case when a person (or group) writes the instrumental composition, and a different person writes the lyrics.

A music publisher is a company that represents songwriters. Songwriters often sign a publishing contract in which they transfer the copyrights in their work to their publishing company. Signing such a contract typically involves a signing bonus or publisher advance to the songwriter. There are different types of contracts. Most of them leave the publishing company with two main tasks. The first is to connect songwriters existing songs with suitable recording artists in order to get the songs professionally recorded in a studio. The publisher also promotes songs to media production companies such as film studios and advertising agencies. In addition to promoting songwriter's work, the second task is to ensure that songwriters get paid. That involves licensing songs, monitor where songs are used, and distribute collected royalties to songwriters. There are many different types of licensing agreements, each of which generates a different type of royalty. We will discuss these types more in-depth in a later section. A standard publishing contract states that the publisher distributes 50% of the collected royalties for a given song to the appropriate songwriter(s) while keeping 50% as payment for its services [7].

Sound Recording

A person who performs a musical composition in a studio is called a recording artist. In the previous paragraph, we explained how it is common practice for songwriters to be represented by a publishing company. Similarly, recording artists are often represented by a record label. They sign a record label contract defining different relationship terms, such as the time the contract is valid between the two. The artist sometimes enjoys a record label signing bonus for entering into such an agreement. The main task of a record label is to develop their affiliated artists by providing a set of different services. For instance, a label helps with marketing, promotion, touring and distribution. Additionally, a record label facilitates studio time for their artists and connects them with music producers. During studio sessions, a producer may take on many different roles, such as suggesting changes to a song's arrangement or coaching the artists to perform in a specific way. In addition, a producer is often responsible for the technical engineering part of the recording process such as controlling the actual audio recordings, mixing the different recordings together, and sometimes help with the final mastering of the track. Both the artist and the producer are considered authors of a sound recording, yet the former typically enjoys a bigger piece of the pie when the track is used commercially.

The result of a finished studio recording session is the master recording file.

Sound recording copyrights protect this file. Note that sound recording copyrights are separate from the musical composition copyrights protecting the musical composition being recorded. The record label owns both the digital master file and the copyrights that protect it according to a standard record label contract. In return, the record label is responsible for licensing this sound recording and distribute the collected royalties to appropriate artists and producers. Today, there are three major record labels. The so-called "Big Three" are Universal Music Group, Sony Music Entertainment, and Warner Music Group. Most record labels outside of these are subsidiaries which live under one of the Big Three's corporate umbrella. In fact, the three major labels account for almost 80% of the whole music market today [8].

Independent Musicians

Not every musician has a company representing them. This is often the case for little-known artists trying to make a name for themselves in the business. These are called independent (indie) artists, and they often follow a do-it-yourself approach. That involves writing their own songs and record them in their own studio. Independent artists enjoy certain benefits such as more freedom and complete ownership of their work. However, the road to success may present more challenges for indie musicians. An independent record label is a record label that operates without any funding or advances to indie artists when signing a contract. They typically take care of music production and distribution in return for a percentage of any future profits the music provides. Signing with an independent record label may also provide some benefits in terms of networking and promotion, and many indie artists choose to begin their career this way. Some artists who write and record their own music never sign with a major label nor an indie label. These self-released musicians typically sign with a music aggregator that takes care of distributing music to on-demand streaming services. Two well-known music aggregators are CD Baby and TuneCore. Small indie labels can also sign up with aggregators for distribution.

The Merlin Network is a global digital rights agency that operates in the independent record label sector. They represent tens of thousands indie labels from 53 countries. "Merlin represents by far the most commercially significant set of rights outside of the three major labels" according to their website[1]. Many consider them the fourth big label because of this. Their main focus is digital distribution. "Merlin licenses digital music services seeking to offer their users the widest possible range of independent music from around the globe. Merlin has struck deals with the world's most important new-generation digital music services"[2]. These services include Spotify, Google Play, YouTube, and others.

[1]http://www.merlinnetwork.org/what-we-do - [Accessed: 22-04-2018]
[2]http://www.merlinnetwork.org/whowelicense - [Accessed: 22-04-2018]

2.1.2 License and Royalty Types

Like live concerts, digital and retail sales, merchandise and signing bonuses, licensing royalties is a vital source of income for artists and songwriters. Licensing royalties are fees that companies pay for a license to use music in their business. The type of license required depends on many different factors. Most licenses and corresponding royalties fall into four main categories.

1. **Public performance** royalties are paid to copyright holders for a license to perform songs publicly. Music played over the radio, in a restaurant or a bar is considered a public performance. More specifically, a public performance "occurs when a song is sung or played, recorded or live, on radio and television, as well as through other media such as the internet, live concerts, and programmed music services."[3]

   Public performance royalties apply to both musical compositions and sound recordings. Thus, if an online radio station wants to stream a song, they have to obtain a public performance license from all copyright holders (usually a publisher and a record label).

2. **Synchronization (sync) royalties** are paid to copyright holders for a license to synchronize music with some visual media output. Some examples are film, television shows, advertisements, video games, accompanying website music, movie trailers, etc.

   Sync royalties apply to both musical compositions and sound recordings. Thus, if an ad agency wants to use a song in a commercial, they have to obtain a sync license from all copyright holders.

3. **Mechanical royalties** are paid to copyright holders when musical compositions are copied and sold on a "mechanically reproduced" medium. Traditionally, record labels paid these royalties to songwriters when mechanically manufacturing albums and vinyl for their sound recordings which is where the name stems from. Nowadays, streaming services also have to pay streaming mechanical royalties (also known as streaming mechanicals) to songwriters when their musical compositions are streamed on their platform.

4. **Print royalties** are the least common type. They apply to musical compositions which are transcribed to a print piece and then distributed. "Print royalties are paid to the copyright holder based on the number of copies made for the printed piece."[4]

[3]https://www.bmi.com/faq/entry/what_is_the_difference_between_performing_right_royalties_mechanical_r - [Accessed: 18-03-2018]

[4]https://www.musicbed.com/knowledge-base/4-types-of-music-royalties/109 - [Accessed: 18-03-2018]

2.1.3 Royalty Collectors

We mentioned earlier that publishers and record labels monitor where their musicians' work is used as well as collecting appropriate royalties based on this. That is true to some extent. However, this job requires a tremendous amount of work. For publishers and record labels to focus on their core business, other organizations specialize in the space of royalty collection. Each country (sometimes world territory) has their own group of copyright collecting agencies. They issue various licenses, monitor where musical works are used and collect and distribute royalties to right holders. In many cases, this structure of intermediaries makes the licensing process easier. For instance, a digital radio station does not have to negotiate deals with every publisher and record label when playing popular music. The relationship between publishers, record label, and collectors are slightly different for each world territory. Some have different collectors for different royalty types.

Performing Rights Organizations

Agencies that collect public performance royalties are known as Performing Rights Organizations (PROs). In the US market, there are three big PROs. They are called American Society of Composers, Authors and Publishers (ASCAP), Broadcast Music, Inc. (BMI) and Society of European Stage Authors and Composers (SESAC). A PRO is responsible for collecting public performance royalties for musical works. As mentioned in Figure 2.1, this includes fees paid by radio stations, restaurants, concert venues, bars, nightclubs, etc., which grants a public performance license allowing musical works to be performed in their public environment. All the musical compositions the publisher represents become a part of the PRO's catalog when signing a contract. Thus, companies only need to buy a license from one PRO to get permission to utilize a big catalog of musical compositions. The PROs keep a percentage of the royalties for their service before they distribute the money to appropriate right holders. TONO is the only PRO in Norway. They offer the same services to songwriters and publishers in Norway as ASCAP, BMI and SESAC offer in the US.

SoundExchange is another PRO in the US which collects royalties for sound recordings. That includes a fee paid by non-interactive digital radio services that broadcast music over internet, satellite or cable. The fee grants a statutory license allowing for streaming sound recordings in their public environment. Note that SoundExchange is not eligible to collect royalties for sound recordings played over the FM/AM radio network. That is due to a US copyright law resulting from terrestrial broadcasters arguing that performers benefit from free promotion through radio plays [9]. SoundExchange is currently fighting to close this "seemingly unfair" loophole[5]. SoundExchange is a non-profit organization. They distribute 50% of collected royalties to record labels and 50% to artists.

Mechanical Royalty Collectors

The Harry Fox Agency (HFA) was established by the National Music Publishers' Association in 1927. HFA is the leading provider of rights management and royalty services in the US mechanical licensing space with over 48,000 publisher clients[6]. They negotiate mechanical licensing deals with small and major record labels as well as on-demand streaming services.

Collecting Sync and Print Royalties

Sync and print licenses are in almost all cases negotiated on a case-by-case basis directly between the licensee and the publisher. For instance, if a company wants to release a guitar book containing chords and lyrics of famous country songs, they would have to contact the publishers who hold the copyrights in the musical compositions and negotiate terms such as the number of printed books and price per copy.

Royalty Flow

Royalties for musical compositions and sound recordings travel through several players before musicians enjoy their share. Figure 2.1 and 2.2 illustrate how the money flows through a chain of intermediaries before it reaches songwriters and recording artists respectively.

[5]https://www.soundexchange.com/advocacy/closing-the-amfm-radio-royalty-loophole/ - [Accessed: 19-03-2018]

[6]https://www.harryfox.com/publishers/why_affiliate.html - [Accessed: 17-03-2018]

Royalty Flow for Musical Compositions

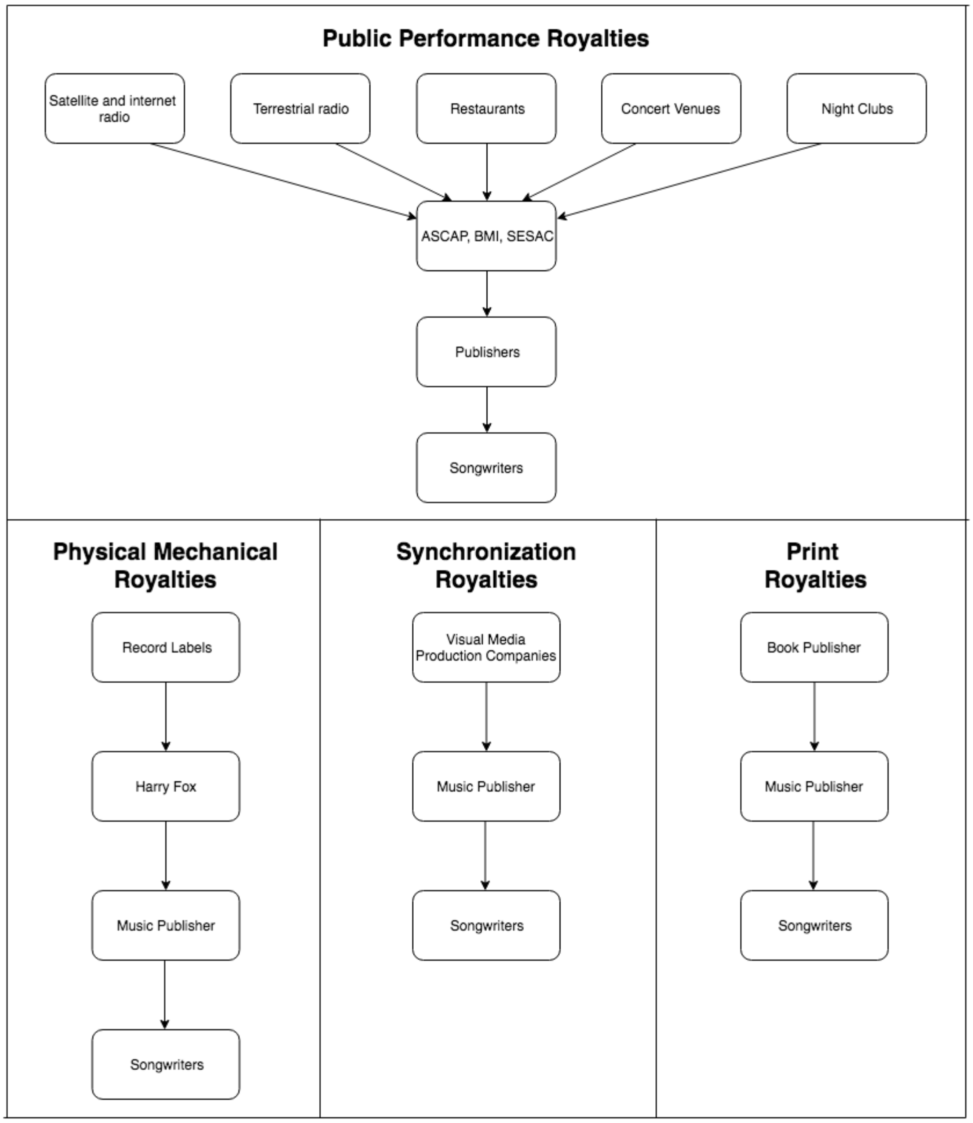


Figure 2.1: How musical compositions royalties flow for different license categories. Figure is created by us, but inspired by Kirstin Thomson [10].

Royalty Flow for Sound Recordings

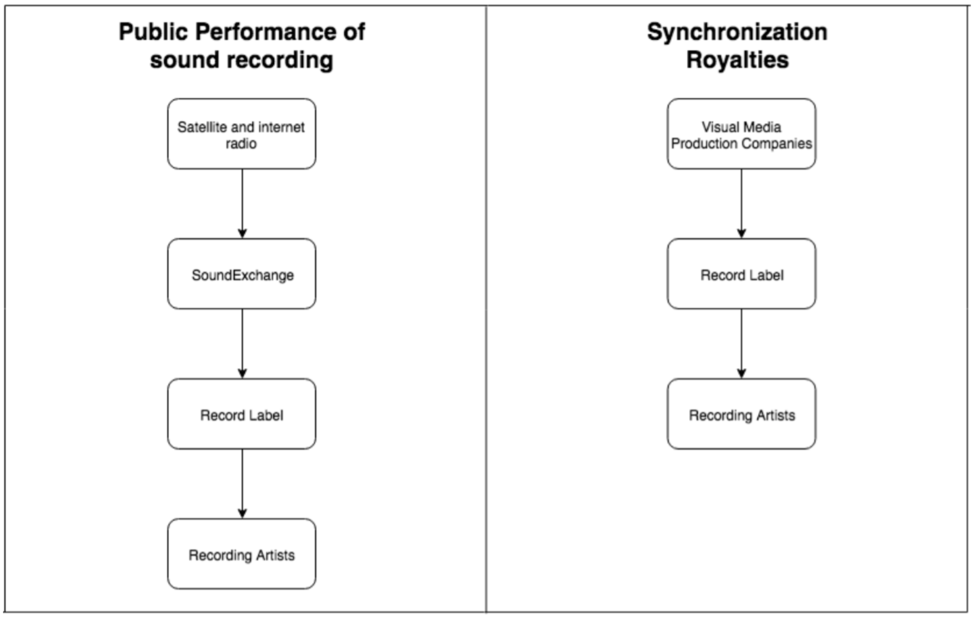

Public Performance of sound recording
Satellite and internet radio
SoundExchange
Record Label
Recording Artists
Synchronization Royalties
Visual Media Production Companies
Record Label
Recording Artists

Figure 2.2: How sound recording royalties flow for different license categories. Figure is created by us, but inspired by Kirstin Thomson [10].

2.1.4 On-Demand Streaming Services

On the topic of different licensing and royalty types, we also need to talk about on-demand streaming services. On-demand means that end users can choose what music to stream. Some examples are Spotify, Tidal and Apple Music. Numbers show that these services are becoming the primary platforms where regular people consume music. "Streaming music platforms accounted for almost 2/3rd of total U.S. music industry revenues in 2017 and contributed nearly all of the growth." [11] The substantial growth of these industry revenue sources makes them important to study.

Spotify is the leading player in the music streaming space with over 150 million monthly users and 75 million paying subscribers[7], making it the most popular on-demand streaming service in the world. They use a standard freemium business model strategy earning money through advertisements and payments from premium subscribers. Premium users enjoy on-demand uninterrupted high- quality music streaming with options like listening to their playlist in offline mode. Non-paying

[7]https://investors.spotify.com/financials/press-release-details/2018/Spotify-Technology-SA-Announces-Financial-Results-for-First-Quarter-2018/ - [Accessed: 24-03-2018]

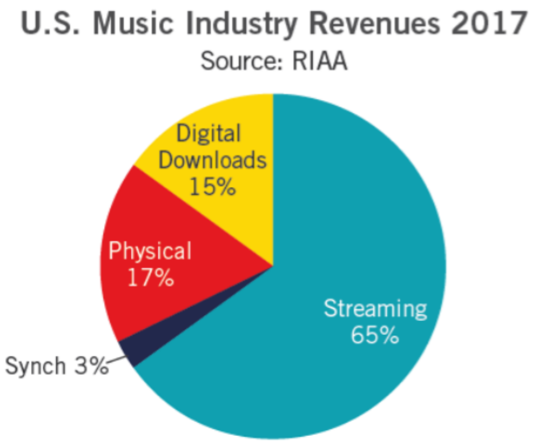


Figure 2.3: Streaming makes up the majority of revenue in the US music industry. Figure source: 2017 RIAA Revenue Statistics [11]

users have access to the same content, but with some limitations which include advertisements, lower sound quality and no option for offline streaming. Most on-demand music streaming services offer more or less the same subscription-based service with minor differences. In addition to similar business models, they also share the same licensing requirements in order to offer music through their digital platforms. What do these license agreements look like? We will use Spotify as an example for answering this question.

Streaming Royalties

Spotify has to compensate both songwriters and recording artists for the music they offer in their catalog. They do this by paying streaming royalties. The streaming royalty type does not directly fall into any of the four main categories mentioned in the previous chapter. A single stream is considered a hybrid of a digital mechanical reproduction and a performance of a musical work [12]. Furthermore, it also requires permission to play the actual master recording itself. Thus, Spotify pays royalties in three different categories; mechanical and performance royalties to songwriters and master recording usage to recording artists. However, these royalties are not paid directly to musicians. Spotify negotiates licensing agreements with big companies that represent the majority of music copyrights. Figure 2.4 illustrates the companies of which streaming royalties flow through before they reach musicians.

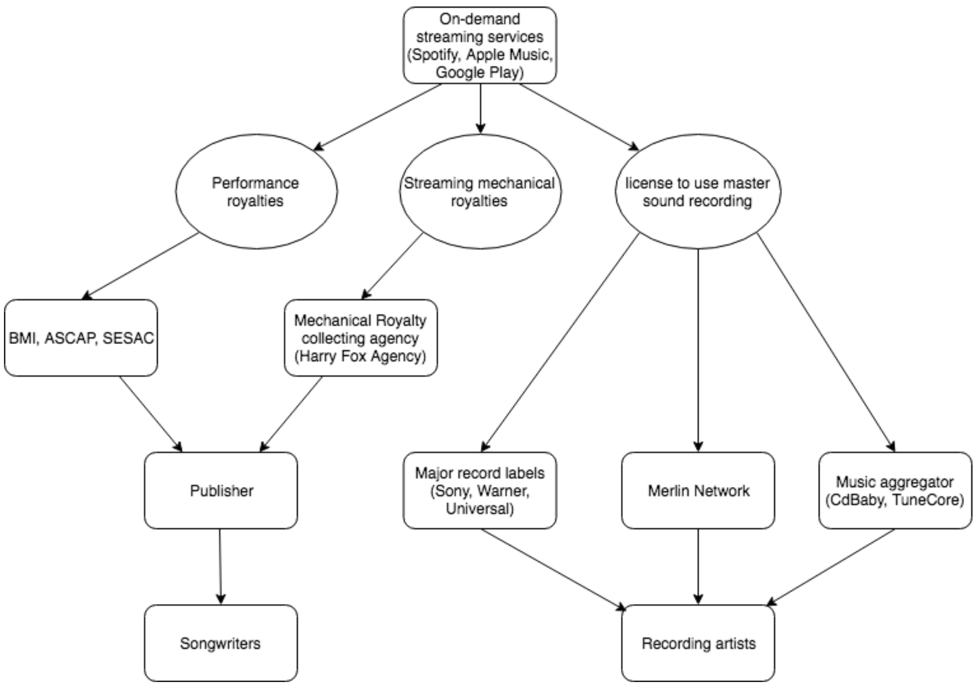

On-demand streaming services (Spotify, Apple Music, Google Play)
Performance royalties
Streaming mechanical royalties
license to use master sound recording
BMI, ASCAP, SESAC
Mechanical Royalty collecting agency (Harry Fox Agency)
Publisher
Major record labels (Sony, Warner, Universal)
Merlin Network
Music aggregator (CdBaby, TuneCore)
Songwriters
Recording artists

Figure 2.4: How streaming royalties flow for different license categories. Figure is created by us, but inspired by Kirstin Thomson [10].

Other Expenses

Streaming royalties are not the only expenses for Spotify. Swedish news reported that in order to secure necessary music rights, Spotify more or less gave away 18% of its company stocks. “When Spotify was launched in October 2008, the service was pulled out of fresh rights agreements with Sony BMG, Universal Music, Warner Music, EMI, and Merlin. The record companies bought in Spotify at the time - for a mocker. They received 18 percent of Spotify shares for almost 100,000 SEK” [13]. (Translated by us. 100,000 SEK was approximately $13,500 at the time). Furthermore, Spotify also pays annual administrative fees to get permission to utilize the major record labels’ music catalog.

2.2 Issues in the Music Industry

While investigating the structure of the industry, different issues and areas of friction became apparent to us. The various problems affect many industry players, but the hardest impact is on the musicians. Even though the music market is growing each year due to an increase in paying consumers [11], musicians point to lower and unfair payouts [6]. During our research, we learned that the root cause of this claim is the transition from sales-based to license-based industry revenues. 10 years ago, physical sales (albums and vinyls) accounted for the majority of industry revenues. By looking at the sales numbers, it was easier for musicians to understand where royalties came from. Today, most people listen to music through streaming and digital radio services. These modern ways of music consumption is available as a result of secret licensing deals between powerful players in the industry. These deals often involve complex royalty calculations that cause musicians to get less insight into their earnings. Furthermore, we discovered that there are two main causes for most of the problems today:

1. The hierarchical structure of players and the relationship between them
2. Lack of information and standards

A report titled Fair Music: Transparency and Payment Flows in the Music Industry [5] written by Rethink Music in 2015 points to the same industry problems as we found in our background analysis. Rethink Music is a project group initiated by the Berklee College of Creative Entrepreneurship that discusses the future of music. There are dozens of issues with today's industry. Rethink Music lists many problems - some of which are more profound than others. In this section, we will discuss the most critical issues that our proposed solution attempts to solve.

2.2.1 Industry Structure and the Relationships Between Players

In this subsection, we discuss concrete problems that arise as a result of today's hierarchical industry structure and the relationships between the different players.

Transparency

To "follow the money" is easier said than done in today's nontransparent music industry. Nontransparency refers to secret contracts and split sheets that are kept hidden at each joint in the revenue chain. Most arrangements are non-disclosure agreements (NDAs) unavailable for anyone but the contractual parties. The secret agreements within the industry hierarchy ultimately make it nearly impossible for a musician to know precisely how much money their creative work is worth and generating. The value of a song or a recording is not set by the musician, but rather an intermediary. Spotify passed iTunes' revenue in Europe as early as in the first quarter of 2014 [14] with the same percentage (70%) of their revenue being passed on to right holders [5]. However, even though total industry revenues have increased, musicians continue to complain about low royalty payouts. Where is the money going? As mentioned, music licensing through streaming services and digital radio leads to a complex economic model. One example is Spotify's pro rata model which dynamically values a per-stream rate based on disclosed factors [15]. Hidden variables in royalty calculations make transparency an issue.

For a streaming service to grant access to a major label's repertoire, the two parties negotiate a NDA licensing contract. These licensing deals may include advance payments, administration fees, preferred advertising slots on the platform and equity in the service, all of which are direct benefits for the label. For artists, however, the additional assets may not end up in their pockets. One can imagine the label's benefits result in lower streaming royalties for the artists. In a leaked contract between Sony Music Entertainment and Spotify from 2011, the advance payments totaled to $42,5 million for the following three years [16]. An advance payment is paid to cover future royalty payments. It is unknown how much of this payment is shared with the label's artists. Large publishing companies have made similar deals. Another leaked contract between Sony/ATV (a major publishing company) and DMX (a background music provider) unveiled a 30% lower royalty rate in exchange for a more substantial administration fee [5]. This deal had a critical impact on the musicians' profit. Given the leaked contracts' years of publication, it should be worth mentioning that these additional fees have lowered over the years as streaming services' market share has grown significantly [17].

The example above is just one of many agreements made in the industry that does not directly benefit the musicians. The lack of transparency at every joint of the income chain, where every party takes a cut, is the main reason why musicians never gain full insight into how much revenue their work is generating.

Efficiency

Royalties for streams, radio plays, album sales, and commercial use can take as long as three years from a recording is consumed by a listener until they reach the artist [5]. Payouts are delayed due to the numerous transactions that have to take place before the publisher or record label can pay the musicians. As stated previously, today's music industry relies heavily on a network of administrative, third-party companies. The licensing transition has added players to the chain, making the distance between musicians and music consumers significant. It is a hierarchy created at a time when internet or digital music platforms did not exist. Fifteen or twenty years ago, these additional administrative roles were necessary so new albums would reach the consumers. Today it is possible for musicians to release music and reach an audience without companies representing them. Given the platforms we have now some of the mentioned middlemen leads to more bureaucracy. For every player in the chain of payments, the less efficient the industry becomes.

International performances is another phenomenon that leads to efficiency issues. When a musician's work is performed in another country than the country that the musician resides in, the royalty claim will be made by a local PRO, extending the income chain with another administrative party. The process begins when a PRO, residing in the country or territory where the performance took place, calculates and receives a claim on the musician's PRO's behalf. The money is then sent to the musician's PRO where the same process repeats. In other words, two PROs will process the royalty claim before they send the money further down the chain. To speed up the process, BMI, together with 12 other copyright societies created FastTrack[8]. This system is a decentralized network of copyright documentation that can be accessed by collective societies around the world. With a shared database, collective societies can spend less time on locating musical works from other societies when claiming on their behalf. It also contributes to more accurate calculations and less friction when more societies share one database. The database does not, however, include every society meaning it is not a fully adopted database and millions of creative works are missing. The database is only available to member societies and payments are not made directly from a local PRO to the right owners leaving much potential for further efficiency measures.

2.2.2 Lack of Information and Standards

In this subsection, we discuss how the music industry lacks an official public copyright database. Additionally, we discuss the implications of the fact that there are no adopted standards for reporting nor a general international licensing regulation.

No Public Registry

There is no verified global registry for music, so it can be challenging to figure out who owns what in order to obtain various licenses. A musical work is copyrighted as soon as it is written down, recorded or edited in any form. A song might involve several songwriters, producers, and recording artists, which can potentially lead to multiple publishing companies and record labels sharing the copyrights. To use the music commercially in a legal manner, one needs to get consent from every copyright holder. That can be a tedious process when there is no common registry to find them.

[8]http://www.fasttrackdcn.net/our-products/cis-net/ - [Accessed: 21-05-2018]

The movie and visual content industry have managed to create and adopt the International Movie Database (IMDb), but the music industry is still running without a shared database. Several attempts have been made to create a universal music registry that every copyright owner can upload copyrights to. The Global Repertoire Database (GRD) is one attempt. The EU commissioner launched the project in 2008 with the ambition that a group of legal, administrative and technological experts would fix some of the problems in the music industry today. It was backed by the world's biggest record labels, technological companies, and collective societies. Unfortunately, the project failed when key players withdrew their funding fearing that their operational costs would surpass their revenue. It left the GRD group $13.7 million in debt [18]. The fear of high restructuring costs amongst the intermediaries keeps the industry from adapting to a more efficient system.

Collective societies around the world have managed to create a shared database for their repertoires, as discussed earlier in this section. However, only member societies can access it, excluding millions of creative works.

Standards

The music industry has partly agreed upon a set of identifiers for both sound recordings (International Standard Recording Code) and musical compositions (International Standard Musical Work Code) in order to uniquely identify a musician's work. However, these have not been widely adopted as right holders often insist on defining their own data reporting standards. Some of these reporting standards are based on artist names or names of tracks, making human errors an issue when calculating royalties. Rethink Music uncovered multiple royalty calculation mistakes in one artist's quarterly report solely due to spelling errors. Such mistakes is one of the reasons that royalties do not make it to their rightful owner. In fact, Rethink Music estimates that anywhere between 20-50% of music payments never make it to their rightful owners [5].

The transition to subscription based platforms has made music more accessible. Today, music is consumed at a global scale. However, the licensing agreements that facilitate easy access are often negotiated at a regional level. There are no global standards for how music should be licensed as different regions have different licensing regulations. A stream in the United States might not be worth the same as a stream in Europe. Digital radio stations pay different amounts of royalties depending on their licensing terms with the right holders. Differences in licensing agreements ultimately affect the musicians. Their work's value is dependent on different factors, such as location and platform, which makes it challenging to verify that the pay is actually correct.

2.2.3 Status Quo

With today's technology, there are no technical boundaries for making changes for the better for the musicians. Nevertheless, fundamental changes have not been made as powerful players in the complex hierarchy of the music industry would not benefit from too much change. More accurate royalty calculations, public copyright databases, and more transparent processes might unveil redundant jobs and unsympathetic deals. Despite few incentives for dominant players to change, some improvements have recently been made, and some are on their way.

Songwriter Credits

Earlier this year, Spotify made it possible to view credits for every song on their desktop app getting the songwriters and producers one step closer to getting the acknowledgment they deserve[9]. This update will increase songwriter and producer recognition and facilitate more collaborations and job opportunities.

Increased Royalty Rate

After a US court ruling in January, streaming services are now required to pay a higher percentage of their revenue to songwriters. By the end of 2022, streaming services will pay 15.1% of their revenue to PROs, an increase from today's 10.5% [19].

Music Modernization Act

The US government is soon expected to change their copyright laws. Royalty payouts to songwriters have dropped throughout the digital music age, especially since the rise of streaming services [20]. The Music Modernization Act (MMA) is a newly suggested bill[10], that has already been passed unanimously in the House of Representatives, may change that. Some of the key changes include:

- Market-based rates for songwriters
- Establish a collective society for publishers and songwriters to track credits and distribute royalties when digital services use their work
- Establish a publicly accessible database with copyright information for every song
- Services are no longer liable for making sure every right holder is paid their fair royalty share

[9]https://artists.spotify.com/blog/spotify-now-displays-songwriter-credits - [Accessed: 11-03-2018]

[10]https://musictechpolicy.files.wordpress.com/2017/12/fact-sheet.pdf - [Accessed: 11-03-2018]

- Right to recognition for adjunct musicians (producers, sound engineers, mixers)
- Royalty protection for pre-1972 performances.

The bill has received criticism for not addressing several other issues such as lack of performer royalties to recording artists for terrestrial radio airplay. Independent songwriters are also criticizing the complex collecting structure described in the suggested bill claiming that more money will ultimately end up in the hands of the major publishers and not its rightful owners [21].

2.2.4 Accessibility and Competition

Another reason for seemingly lower revenues is tougher competition among musicians. Technology has opened many doors for aspiring musicians and songwriters. A song can now be written, produced, recorded and distributed from anywhere at any time. Neither prior music knowledge, production experience nor money, is required to enter today's industry. At the same time, new music today is more available than ever before. Digital sales platforms (iTunes, Amazon, etc.) limit consumers to the songs that they have purchased, making exploring new content an expensive affair. The rise of the streaming services has made it possible for consumers to listen to any song available on their platform at no extra cost. As a result, even though the music industry is generating more revenue than earlier, royalties are paid to a broader spectrum of artists. This paper will not continue to discuss this issue, as it is a natural consequence of the technological advancements made over the past decades.

2.3 Blockchain

In this section, we will give a short explanation of what blockchain is and how it works. There are several well-written articles and white papers explaining implementations and technicalities in depth. This paper will not focus on the details, but rather give an overview of the technology and discuss its most important properties. Our research question focuses on utilizing blockchain technology to solve the music industry problems mentioned above. This question is mainly discussed in part two and part three of this thesis. We consider a fundamental knowledge about blockchain's most critical aspects and their implications a prerequisite to engaging in the further discussion.

2.3.1 Overview

The idea of blockchain was first presented in 2008 when a person (or group) under the alias Satoshi Nakamoto released the white paper for Bitcoin [2]. Nakamoto's core idea was to make a peer-to-peer version of electronic cash without the need of a trusted third party responsible for processing and verifying transactions. Today, financial transactions usually take place through a bank or another payment provider. With the current method, everyone has to trust that the bank or supplier performs the transactions as it should. Nakamoto suggested replacing these intermediaries with a public decentralized distributed database called the blockchain. Blockchain stores data in blocks that are linked together to make a chain. Hence, the name blockchain. This chain is the very foundation of the technology and is sometimes called the digital ledger. Each node in the network stores a copy of the same blockchain. Every time a new block of data is processed, all nodes verify the new block and then updates their ledger. There is no central «master» copy of the ledger, and no nodes in the network are trusted more than other nodes. The invention of the blockchain enabled Bitcoin to become the first electronic currency without the use of a trusted authority [22].

With no trusted third party, several questions arise. If Bob sends Alice $10, how can we be sure that Bob has $10 available? Can we be sure that he does not send "the same" $10 to more receivers simultaneously? What if Bob tries to go back and change the record of an earlier transaction? Without a centralized node, the blockchain has to be designed in such a way that false transactions will not be accepted and it should not be able to alter previous entries on the ledger. The system has to assure everyone that their copy of the ledger is accurate. This trust is established through a consensus algorithm. This algorithm is the basic mechanics for how everyone on the network agrees on which block to accept to keep their copy of the ledger in sync as transactions occur. Bitcoin currently uses the proof-of-work consensus algorithm which is briefly explained in the next paragraph.

2.3.2 The Bitcoin Protocol

Today, there are many different blockchains. Bitcoin and Monero are two examples focusing on electronic peer-to-peer cash systems. Ethereum and EOS are two other examples with a more general-purpose. The different blockchains have different objectives and predetermined rules. They also differ in what data they store. However, their core idea concerning trust establishment without intermediaries is very similar. In this paragraph, we use the Bitcoin blockchain as the example to illustrate the network's consensus algorithm.

The Bitcoin blockchain organizes transactions into blocks. In addition to a set of transactions, blocks also contain a sequence number, a timestamp, a nonce (random number used only once), the hash of the previous block and some other metadata.

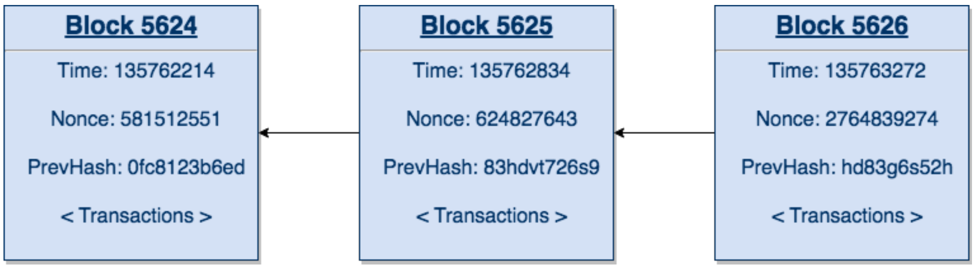


Figure 2.5: The data contained in a Bitcoin block. The backward arrows illustrate how each block include the hash of its predecessor. The figure is created by us.

Proof-of-work is the algorithm used in Bitcoin that allows for decentralized consensus. This algorithm establishes consensus for the next valid block the network should accept and add to the chain. The process is called mining. Miners are transaction processing nodes that compete to solve a difficult proof-of-work puzzle. The puzzle is a complicated mathematical problem, but its solution is easy to verify. Mining nodes store pending transactions locally in their memory. They use special software to select a collection of said transactions for which they attempt to build a block. "A block is a valid addition to the chain if its nonce is chosen so that the block's hash is less than a target value" [23]. That is, the concatenation of all data within the block needs to be hashed to an output level starting with a certain amount of 0s. Miners solve this problem using a brute force method that selects different nonce values. The difficulty level of the proof-of-work puzzle is "adjusted periodically by an adaptive algorithm based on the recent blockchain history(...)". The goal is to "(...) to maintain the long-term invariant that one new block will be mined every ten minutes on average" [23]. In other words; Bitcoin aims to mine a new block of transactions every 10 minutes. This is also described as a block time of 10 minutes.

When a miner solves the puzzle, he broadcasts the block to the network, and the other miners verify that the solution is valid before they append it to their chain.

2.3.3 Defining a Distributed and Decentralized Network

In its purest form, a blockchain is just a data structure that makes up a database. The unique thing about it is that no central entity owns and governs the system. The blockchain is completely maintained and publicly owned by the nodes that make up the network. The previous chapter explained how the Bitcoin blockchain establishes consensus in a decentralized fashion and spreads the data across a distributed network of nodes that processes and verifies the validity of blocks. There are many indecisive definitions for the two phrases decentralized systems and distributed systems. They are somewhat related, but not identical. Vitalik Buterin, the founder of the Ethereum Blockchain, explained in an article written in 2017 [24] that decentralization can be categorized into different types. He illustrates three separate axes of centralization/decentralization:

1. "**Architectural (de)centralization** — how many physical computers is a system made up of? How many of those computers can it tolerate breaking down at any single time?"
2. "**Political (de)centralization** — how many individuals or organizations ultimately control the computers that the system is made up of?"
3. "**Logical (de)centralization** — does the interface and data structures that the system presents and maintains look more like a single monolithic object, or an amorphous swarm? One simple heuristic is: if you cut the system in half, including both providers and users, will both halves continue to fully operate as independent units?"

Where does blockchain fit into the three axes? Buterin explains it like this: "Blockchains are politically decentralized (...), architectural decentralized (...), but they are logically centralized (...) [24]." Figure **??** illustrates where Buterin places blockchain in a centralized/decentralized combination diagram.

It is worth mentioning that these placements are highly debateable. However, we agree with Buterin's definitions. Considering Buterin's placements of the blockchain on the three mentioned axes, we can discuss each of them in more detail:

1. Architectural decentralized implies that data is processed and stored across a network of physical nodes. This architecture is the reason we also call blockchains distributed systems.

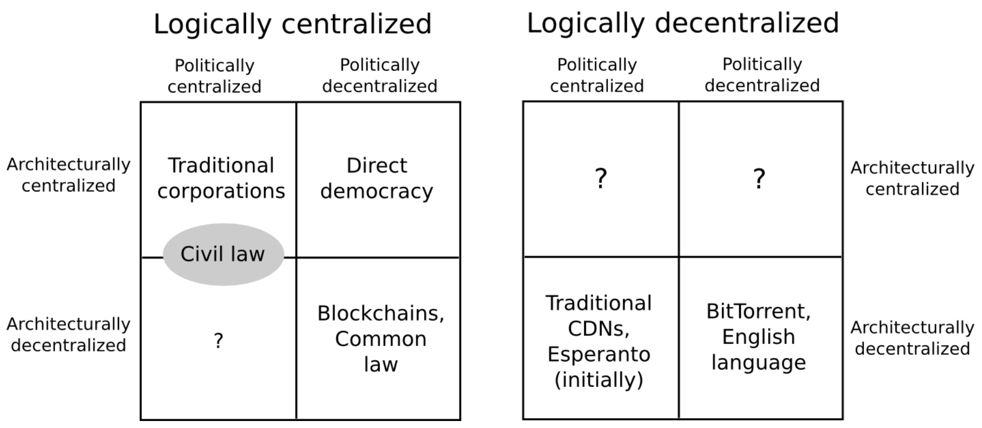

Logically centralized
Politically centralized
Politically decentralized
Architecturally centralized
Architecturally decentralized
Traditional corporations
Direct democracy
Civil law
?
Blockchains, Common law
Logically decentralized
Politically centralized
Politically decentralized
?
?
Architecturally centralized
Architecturally decentralized
Traditional CDNs, Esperanto (initially)
BitTorrent, English language

Figure 2.6: Vitalik illustrates blockchain's placements in a chart that includes the three dimensions. Figure source: Vitalik Buterin [24]

2. Politically decentralized implies that no single entity controls the system. This is where decentralized consensus, explained in the previous section, becomes relevant. The consensus algorithm enables continuously common agreement about the current state of the blockchain without a trusted central entity.
3. Logically centralized implies that there is only one agreed upon state of the system. The blockchain looks and behaves like one single computer system.

To summarize we can conclude that blockchains are distributed networks inherently characterized by several decentralized elements. However, from a logical point of view, they operate as a centralized system to the outside world.

2.3.4 Properties of Distributed and Decentralized Networks

Decentralized and destributed systems provide several attractive properties. Security, immutability and transparency are the three most relevant examples in our discussion.

Security

In the same article, Buterin argues that the decentralized parts of blockchains provide several security properties:

1. "Fault tolerance — decentralized systems are less likely to fail accidentally because they rely on many separate components that are not likely"(to fail at the same time).

2. “Attack resistance — decentralized systems are more expensive to attack and destroy or manipulate because they lack sensitive central points that can be attacked at much lower cost than the economic size of the surrounding system.”

3. “Collusion resistance — it is much harder for participants in decentralized systems to collude to act in ways that benefit them at the expense of other participants, whereas the leaderships of corporations and governments collude in ways that benefit themselves but harm less well-coordinated citizens, customers, employees and the general public all the time.”

Immutability

Blockchain’s nature also provide the immutability property. Immutability is defined as something that is unable to be changed. Once a block is accepted by the network and added to the chain, its data can never be altered. The explanation for how immutability is ensured is that all blocks include the calculated hash of its predecessor. If an attacker changed some data within block number n that the network would accept, he would have to recalculate a valid block hash for that block. However, if the hash of block n changes, then block (n+1) will reference the old block n hash which does not exist anymore. This mismatch breaks the entire chain of blocks [25]. If an adversary can recalculate each block followed by the tampered block, the attacker could potentially replace the data for the entire blockchain. However, this attack is highly implausible as it requires a tremendous amount of hashing power. The immutability property is vital as the blockchain can prove data integrity and all users can be confident that the data will not alter in the future.

Transparency

The fact that all data held by the blockchain is publicly available and managed only by the network itself ensures a level of transparency that conventional centralized transaction systems do not offer. The data in question varies depending on the blockchain we study. In the case of the Bitcoin blockchain, the data constitutes monetary transactions between entities. On the Ethereum blockchain, the data may also include complex scripts that automatically execute if certain conditions are met. The next section includes a more in-depth discussion regarding Ethereum and its properties. Generally speaking, blockchain technology ultimately provides a clear reality which is fundamentally different from what traditional centralized systems offer.

2.4 Ethereum

2.4.1 Platform for Decentralized Applications

Ethereum is a public blockchain-based computing platform for decentralized applications (dApp) that launched in July 2015. Vitalik Buterin first described the system in a white paper [26]. Some refer to Ethereum as a “second-generation” blockchain as it is not solely a cryptocurrency system like Bitcoin. Buterin himself explained in an interview with the newspaper The Pioneer [27] that his team “focuses on creating a new blockchain architecture that’s designed to be general-purpose so that you can use it for any applications that might benefit from being on a blockchain.” There are several types of applications that can benefit from blockchain technology. Some examples are voting systems, domain name registries, financial exchanges, crowdfunding platforms, company governance, or intellectual property such as musical works [3].

2.4.2 Smart Contracts

Ethereum features something called smart contracts. These contracts are scripts that can be added to the Ethereum blockchain. Due to blockchain’s immutability, a smart contract can never be altered once it is deployed to the network. As a result, there exists pieces of code that all users can interact with on the blockchain. Smart contracts are similar to classes in object-oriented languages. They define state variables and functions. We call the former state variables because they define the internal state of the contract. These variables can be of many different types such as integers, arrays, strings and mappings (similar to HashMap in Java). Functions can either read values from state variables or modify them.

2.4.3 Two Types of Accounts

Ethereum features two types of accounts: smart contract accounts and externally owned accounts (regular Ethereum-wallets). Both types have an Ethereum address and a balance. An externally owned account is controlled by an associated private key held by the owner of that Ethereum wallet. In contrast, smart contract accounts are controlled by their own code.

2.4.4 Ethereum Virtual Machine

Ethereum is described as a blockchain with an integrated Turing-complete programming language. The part of the protocol that handles the execution and the internal state of smart contracts is referred to as the Ethereum Virtual Machine (EVM). From a practical standpoint, the EVM can be thought of as a large decentralized computer managing millions of objects (smart contracts). To avoid infinite computational loops,

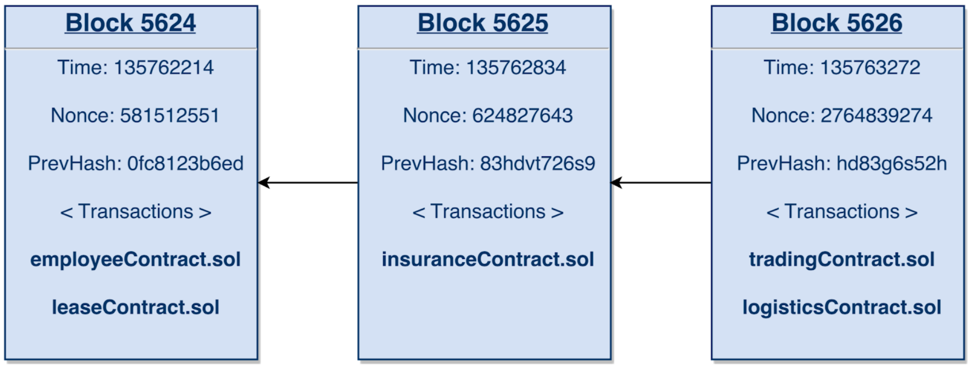


Figure 2.7: The data contained in an Etheruem block. Notice that smart contracts are part of the data contained in the block. These are computer programs that anyone can interact with. The figure is created by us.

DDOS attacks, or network spam by deploying infinite worthless smart contracts, the EVM is powered by gas called ether (ETH). Gas is paid to the nodes running the EVM. The amount of gas a user has to pay is determined by the amount of computation or storage an execution requires.

2.4.5 Consensus and block time

Even though Ethereum and Bitcoin both currently use the proof-of-work consensus algorithm, there are slight differences to the protocol. Block time is one example. Block time refers to how long it takes for the network to complete the mathematical puzzle and assign a new valid block to the blockchain. While the Bitcoin network aims for a block time of 10 minutes, the Ethereum aims for a block time of 12 seconds [28]. However, the actual average block time seems to be closer to 15 seconds[11]. Ethereum has a relatively fast block time as the platform is supposed to facilitate a variety of different applications, many of which require transactions to be confirmed more responsively. To achieve this, the difficulty level of the proof-of-work puzzle is significantly lower than Bitcoin's.

2.4.6 Token system

We often read about coins and tokens in the cryptocurrency space. These terms should not be confused with one another. A coin is defined as cryptocurrency used as a currency, e.g., Bitcoin, Litecoin, and Monero. A token represents a tradeable asset or a utility used in dApps. [26]

[11]https://etherscan.io/chart/blocktime - [Accessed: 16-05-2018]

Ethereum has defined several token systems. ERC20 is the name of the most well-known standard. Everyone can create their own ERC20 token, but dApp projects mostly use them for raising funds through something called an initial coin offering (ICO). REP is one example of an ERC20 utility token offered by the Augur project through their ICO [29]. The token is used to create stake shares on Augur's Ethereum-based decentralized prediction market platform. The ERC20 standard represents fungible tokens. Fungible refers to something that can be replaced by another identical item. A dollar is an example of a fungible item. One can always exchange it for a different dollar and keep the same value.

The ERC721 is a different token standard that represents non-fungible assets. Non-fungible tokens are inherently unique and cannot be replaced. It is the exclusive characteristics that give the ERC721 token value, rather than the number of tokens owned. One ERC721 token represents one unique non-fungible item such as a house or a rare collectible. There is no limit for how many ERC721 tokens an Ethereum account can hold. The ERC721 standard merged (was officially accepted by the Ethereum Foundation as a standard token system) in March 2018[12]. The game CryptoKitties was the first popularly-adopted implementation of the ERC721 standard [30].

2.4.7 Transparent Trustless Decentralized Applications

The big potential of smart contracts is that they can potentially serve as the back-end code of trustless dApps. In the blockchain space, the word trustless is defined as something that does not require a trusted third party to work safely and correctly. Thus, trustless decentralized applications means that there is no requirement for a trusted third party to handle the communication with the app because all interaction and transactions would be transparent and determined only by the app's code. We have mentioned the phrase decentralized applications throughout this paper without clearly expressing what it means. Investopedia gives this definition[13]: "Decentralized applications (dApps) are digital applications or programs that exist and run on a blockchain or P2P network of computers instead of a single computer, and are outside the purview and control of a single authority."

We agree with this explanation. However, we argue that dApp projects additionally should be open source. When smart contracts deploy to the blockchain, they are stored as bytecode. Anyone can view the bytecode since the blockchain is transparent. If a dApp project is open source, anyone can compile the solidity contracts themselves and verify that the resulting bytecode is identical to the bytecode on the Ethereum

[12]https://github.com/ethereum/EIPs/pull/841 - [Accessed: 24-04-2018]

[13]https://www.investopedia.com/terms/d/decentralized-applications-dapps.asp - [Accessed: 25-04-2018]

blockchain. Assuming this is the case, the public is offered complete insight into a dApp's inner workings.

The combination of blockchain's inherently transparent nature and open source back-end systems is what makes decentralized applications trust-free and fully transparent [3]. One can be confident the EVM will execute all incoming requests in a deterministic manner predetermined only by the contract code. Besides, all transactions between accounts can be traced and examined. Ultimately, interacting with a dApp will never involve malicious computing or invisible bias during transactions.

2.5 Related Work

Our project is not the first application that attempts to solve issues in the music industry. In this section, we will briefly talk about other organizations and companies that have similar goals. Even though three of the following four projects are blockchain based, they are all slightly different from our contribution.

2.5.1 Ujo Music

Ujo Music[14] is an online music platform with a similar business model as iTunes. Users can purchase digital recordings and download them. Unlike iTunes, Ujo makes it possible for all artists to release music directly without labels or aggregators. Additionally, it runs on the Ethereum blockchain making all sales royalties go directly to the musicians. Listeners can also pay a certain price to get access to exclusive content. In addition, listeners get to view every transaction made and see who gets paid and at what percentage share. As we have mentioned in the music background section, most people consume music through streaming services and various types of radio nowadays. We do not believe that digital sales is the future and consider Ujo Music's business model a bit outdated.

2.5.2 Musicoin

Musicoin[15] is a platform for publication and consumption of music. The service is similar to soundcloud.com - a free online streaming and download service. However, it is built on Ethereum and uses smart contracts to automatically collect and distribute royalty payments. Musicoin.org was introduced together with a new token called $MUSIC (MC) which acts as a digital currency. In particular, they have a pay-per-play (PPP) smart contract. That means that consumers spend a specific number of MC every time they stream a song. Musicoin challenges today's streaming services such as Spotify and Apple Music by lowering the entry barrier for indie artists. Also,

[14]https://ujomusic.com/ - [Accessed: 18-02-2018]
[15]https://musicoin.org/how-it-works - [Accessed: 19-02-2018]

they claim to pay more per play to musicians than their competitors. The team has currently a beta running at musicoin.org.

2.5.3 SongFreedom

SongFreedom[16] is a platform for cinematographers and other musicians who want to license music for their work. SongFreedom (soon to be Fyr-Fly) makes it easier for people to buy licenses from an ever-growing music library. The platform is not blockchain-based but it does simplify the process of getting a legal license for musicians.

2.5.4 Blokur

By using artificial intelligence and blockchain technology, Blokur[17] gathers music copyrights from different sources and places them in a single blockchain state. It is a platform for right holders where they can register their copyrights and shares. For instance, Blokur allows its users to register performing rights, sales rights (mechanical), commercial rights (synchronization), etc. from country to country. If there is a conflict of rights data in a specific region, their algorithm will spot it and request the user to solve it. Blokur also has a licensing ecosystem under development that will enable close to instant payments to the different stakeholders.

[16]https://www.songfreedom.com/about - [Accessed: 18-02-2018]
[17]https://www.blokur.com/-[Accessed: 02-06-2018]

Chapter 3 Application System Design

The second part of the thesis includes Chapter 3 and Chapter 4. In these chapters we explain the system design and the technical implementation of our solution.

In this chapter, we apply the properties of blockchain to the music industry issues described in the background analysis. To be more specific, we explain why we use the Ethereum blockchain as the underlying infrastructure for our dApp. Moreover, we discuss different planning aspects of our POC. These aspects include use cases, proposed system architecture, data storage and relevant technologies.

3.1 Rights Done Right

To test the potential of Ethereum and smart contracts, we have developed a proof-of-concept Ethereum-based copyright database and music licensing dApp. We call it Rights Done Right. The goal of the POC demonstration is to show that blockchain technology can be used to securely implement essential use cases both from the perspective of copyright holders (songwriters, artists, record labels and publishers) and licensees (users who are granted a license to use music legally in some manner). In this section, we discuss why we use the Ethereum blockchain and which use cases our dApp implements.

3.1.1 Choosing the Etheruem Blockchain

In section 2.3 and 2.4, we discussed aspects and properties of blockchain technology. While some apply exclusively to Ethereum, others hold true for blockchain in general (including Ethereum). Our dApp benefits from many of the mentioned qualities in the following ways:

- Security
    - As mentioned in 2.3.4, blockchains are fundamentally fault-, attack- and collusion resistant. As a result of these security properties, the back-end system of our dApp will have zero downtime (unless an attacker is able to simultaneously compromise more than 50% of the nodes in the entire Ethereum network as explained in section 2.3.4). Today, modern web applications have to be careful about deployment configurations and choice of web host as they need to watch out for attacks such as distributed denial of service (DDOS) attacks. Luckily, applications on the Ethereum blockchain do not have to be equally worried about such attacks as the fundamental underlying infrastructure has built-in security measures. In our case, security properties are incredibly important because our dApp may deal with serious licensing contracts that involve huge amount of royalties. In these situations, it is extremely important that the back-end system carrying those transactions never goes down or gets compromised by an adversary.
- Trust and Transparency
    - Our dApp assures a high level of trust and transparency to all users. This property addresses the issue regarding musicians' lack of royalty insight. The music industry structure makes it very hard to know whether royalties flow through all intermediaries in a correct and fair manner. As mentioned in the background chapter, between 20-50% of music payments do not make it to the rightful owners. Missing royalties would certainly not be an issue when licensing music through our platform. The trust-free and open-source nature of our dApp assure users that licensing contracts indeed execute only as they are programmed to do. Furthermore, the Ethereum transactions carrying the licensing royalties can precisely be traced from sender to recipient at all times.
    - The transparent nature of Ethereum also facilitates a global copyright database, accessible to everyone. As there are no access restrictions for who can write or read data from the Ethereum blockchain, our dApp can indeed function as a global public copyright database.
- Efficient smart contracts
    - All functionality offered by our dApp, such as copyright registration and music licensing, is done through smart contracts. One feature that makes smart contracts desirable building blocks for dApps is their ability to instantly execute code and enforce transactions automatically when certain conditions are met. Generally, One can think of smart contracts

as accelerated replacements for traditional third-party intermediaries. In our case, this enables users to get a copyright proof once they register a musical work to the database. More importantly, it enables users to purchase a music license and the corresponding royalty payments will be transferred to the relevant rightful copyrights owners right away. This automated payment process is more efficient than conventional royalty payment processes.

- Token System
    - The ERC721 token interface can represent copyrights as a non-fungible asset on the Ethereum blockchain. The standard defines ownership for an asset as well as methods for transferring that ownership. By implementing the ERC721 standard, everyone can interact with the copyright token as any other ERC721 tokens. Thus, copyrights can be traded or sold on exchanges or marketplaces. Furthermore, it implies that users can keep track of (and transfer) their copyrights by holding their tokens in a digital Ethereum wallet that supports the ERC721 standard.
- Immutability
    - Blockchain's immutability property ensures that all data within our dApp is unaltered and can never change in the future. In particular, this implies that musicians can be confident that their entries to the copyright database is safe and immutable. Additionally, licensees can be confident that they deal with license contracts designed only by the people holding the copyrights for associated works. Generally speaking, one can think of this property as data integrity. It is considered a fundamental requirement in almost all applications.

The mentioned technical properties are not the only reason Ethereum is a preferred platform for dApps. In fact, there are other dApp blockchains with similar technology and features such as EOS, NEO, and Cardano. However, Ethereum's significant position in today's blockchain space is an essential reason for why we chose it as our platform. The whole Ethereum project involves strong leadership and a big group of people working on improving the platform. Instead of bragging about profits and economic growth, the foundation talks about a long-term vision that involves a clear scaling strategy and a strong commitment to keeping the network fully decentralized [31]. These things are important to blockchain start-ups and developers and may be the main reason why Ethereum has established itself as the leading dApp platform. Compared to other blockchain ecosystems, Ethereum is relatively mature with a big community and proper documentation for the programming language Solidity. We will discuss Solidity further in a later section.

3.1.2 Music Licensing Use Cases

We focus on three essential use cases on our POC dApp.

1. **Register a musical work:** Musicians should be able to register musical works to the public database. The process should be fast and easy and result in a digital copyright proof when successfully registered.
2. **Create license contracts:** Copyright owners should be able to create licensing contracts for the works they have registered in the database. As mentioned earlier, there are many types of licenses - each with different purposes.
3. **Search for musical works and purchase licenses from the database:** Every user should be able to search through the database to get information about registered works. This information should include details about the work and its owners. When inspecting a musical work, relevant licensing contracts should be displayed and available for purchase.

In addition to the described use cases, all users should also be able to see their related user data. For copyright owners, this includes an overview of all uploaded works and how much royalties these works have generated. For licensees, this includes receipts for all purchased licenses.

3.2 System Architecture

In this section, we will explain the architecture of our system. The dApp is divided into four separate physical nodes: client, web server, Ethereum blockchain, and Firebase server.

3.2.1 Client: Web browser

The client is a device running a web browser that displays the web application's view. Through the presented user interface, a client can pass data and requests to the application. Google Chrome is the desired browser since the application relies on a particular chrome-extension to communicate with the blockchain.

3.2.2 Web Server: Angular Application

A web server hosts the Angular web application written in HTML, CSS, and TypeScript (including all dependencies). These files are all executed on the client side to render the application's user interface. It passes input from the client to the appropriate functions in the back-end. The web application connects to two back-end systems: an external, centralized database called Firebase and the Ethereum blockchain. The latter connection is made possible through a javascript API called Web3.js which injects into the client's browser through the MetaMask Chrome extension. Both MetaMask and Web3.js will be discussed in detail later in a later section. Web3.js facilitates the JSON-RPC communication protocol used between the web app and the Ethereum network. The communication protocol between the web app and Firebase is HTTP TCP/IP.

3.2.3 Ethereum blockchain: Back-end Execution Environment

The Ethereum blockchain with deployed smart contracts functions as our main back-end. The functions written in the deployed contracts are responsible for all business logic, validation and vital data storage with regards to the registration of copyrights as well as servicing our licensing features. Due to high storing costs with Ethereum, only crucial data is stored on the blockchain. Supplemental data is stored in a regular, centralized database. A more in-depth discussion about the location of data can be found in Section 3.3.

3.2.4 Firebase Server: Data storage and Authentication

Firebase is a platform based on Google Cloud Services that works as a Backend-as-a-Service (BaaS) in our application. Using Firebase as a BaaS allows for a serverless application architecture. This design is beneficial as it removes the need for writing traditional server-side components. Firebase is responsible for three different tasks in our application:

1. User authentication
2. Data Storage
3. File Storage

3.2.5 Application Deployment

Figure 3.1 shows a deployment diagram of our dApp. The diagram follows basic Unified Modeling Language (UML) syntax. It illustrates the different hardware components (nodes), what software components (artifacts) run within each node, and the communication protocols that connect the nodes together.

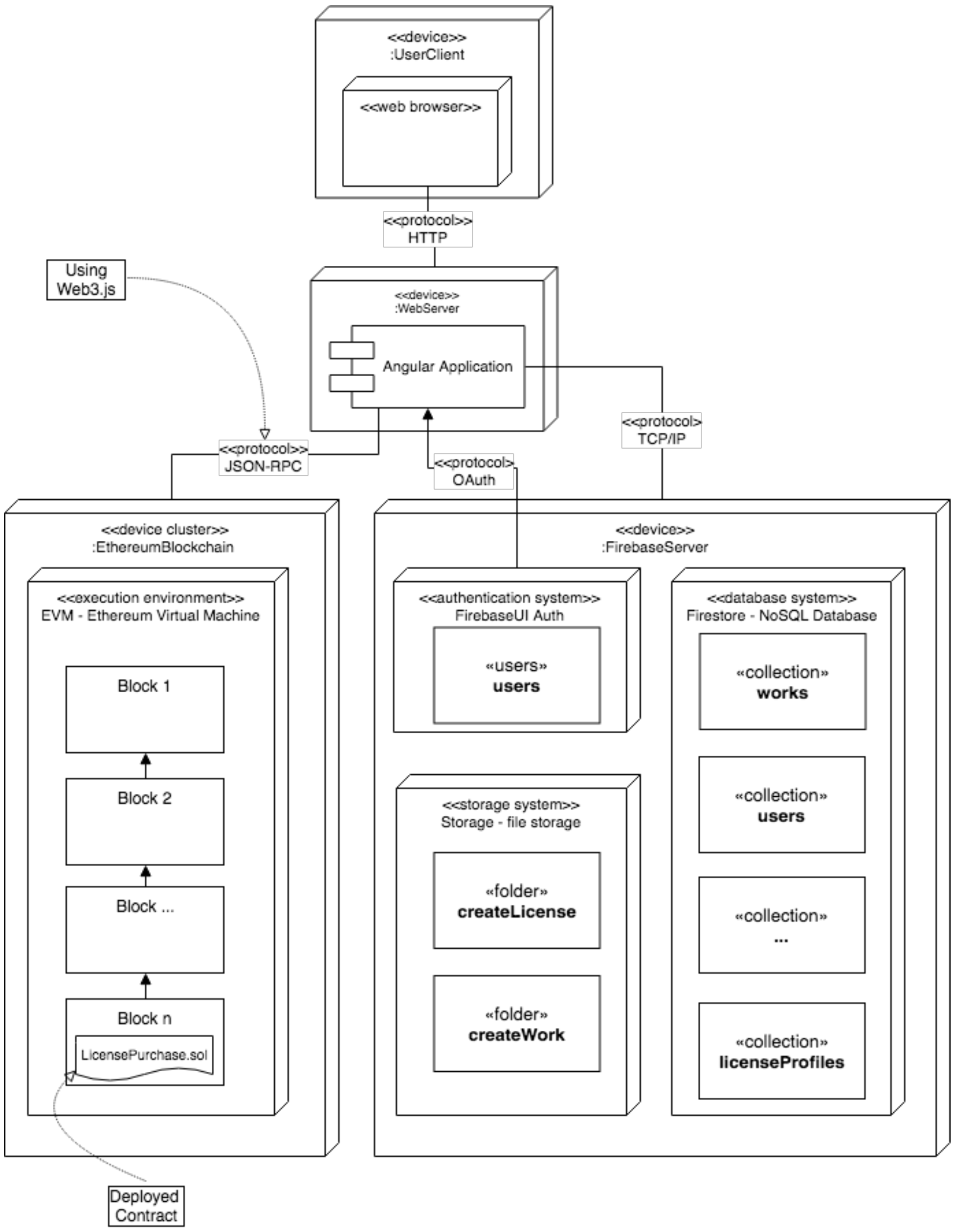


Figure 3.1: Deployment diagram showing the architecture of the system. A web server hosts the web application and communicates with a dual backend: business logic and vital storage on the blockchain and authentication and storage with Firebase services.

3.3 Application Storage

Our application stores data both to the Ethereum blockchain and Firebase. This section will discuss why this is the case. The application handles different data objects for different use cases. For the sake of simplicity, we will use the "register musical work" use case to outline the data storage.

A musical work is defined as an object with multiple attributes:

- Title
- Type of work (song, recording, composition, lyrics, etc.)
- Description
- Time of registration
- List of all contirbutors and their respective share of ownershop
- A file embodying the work itself

All these attributes describe the musical work in different aspects. However, we argue that some of them are more critical than others. For instance, the file of the work itself is enough to identify the musical work. The list of contributors and their share of ownership identifies the work's split sheet. These are the only two essential work attributes used in the back-end business logic. For that reason, these are the only properties we need to store on Ethereum. However, storing data/files on Ethereum is very expensive. The following calculation shows how expensive it can be store a 5MB file:

According to the yellow paper of Ethereum, storing 256 bits(8 bytes) on the blockchain will cost approximately 20,000 in gas [32]. The calculation below is done with values from ETH Gas Station[1] and CoinMarketCap[2]:

Price of storing 8 bytes on the blockchain with a given gas price[3]:

$$20,000 \times 12\ GWEI = 240,000\ or\ 0.00024\ ETH$$

For a 1MB file (1 000 000 Bytes) the total cost of gas will reach:

$$\frac{1,000,000}{8\ Bytes} \times 0.00024\ ETH = 30\ ETH$$

[1]https://ethgasstation.info/
[2]https://coinmarketcap.com/currencies/ethereum/
[3]Value as of the 30th of May 2018 at 22:00 CEST

The cost of 1 ETH at the time[4]: $550.42

$$30\ ETH \times \$550.42 \approx \$16,500/MB$$

The total cost of gas in USD for storing a 1 Megabyte file on the Ethereum blockchain would (at the given date and time) cost approximately $16,500.

As blockchain storage is costly, our application aims to minimize the number of bytes processed and stored on Ethereum. To achieve this, we do not store any files on the blockchain. Instead, we store the MD5-hash-representation of the file. This digital fingerprint (also known as the file checksum) is only 128 bits of data. Storing a hash of a file instead of the file itself is known as checksum-based storage. This design is common practice for Ethereum-based dApps as it allows for cheap storage of a large amount of data. Conclusively, the work object stored on Ethereum contains the following properties:

- The checksum of the file
- List of all contributors and their respective share of ownership
- Time of registration

The entire work object (as defined at the beginning of this section) is stored on Firebase. This data has two purposes

1. The information is displayed to clients when querying works
2. The data contains searchable keywords for the application's database search function.

Figure 3.2 demonstrates the sequence for the "create work" use case. It shows how data sequentially flows between the different components. As the user selects a file to register, it is immediately pushed to Firebase. Once the upload is complete, the file's metadata returns to the application. This data includes the hash of the file among other attributes. When the user has provided all required properties for the work and pushes the "Create Work" button, the application sends a request to Ethereum to process and store the contributors, split sheet and the hash of the file. If the work is successfully stored on the blockchain, the application pushes the entire work object (including all properties and the file itself) to Firebase.

[4]Value as of the 30th of May 2018 at 22:00 CEST

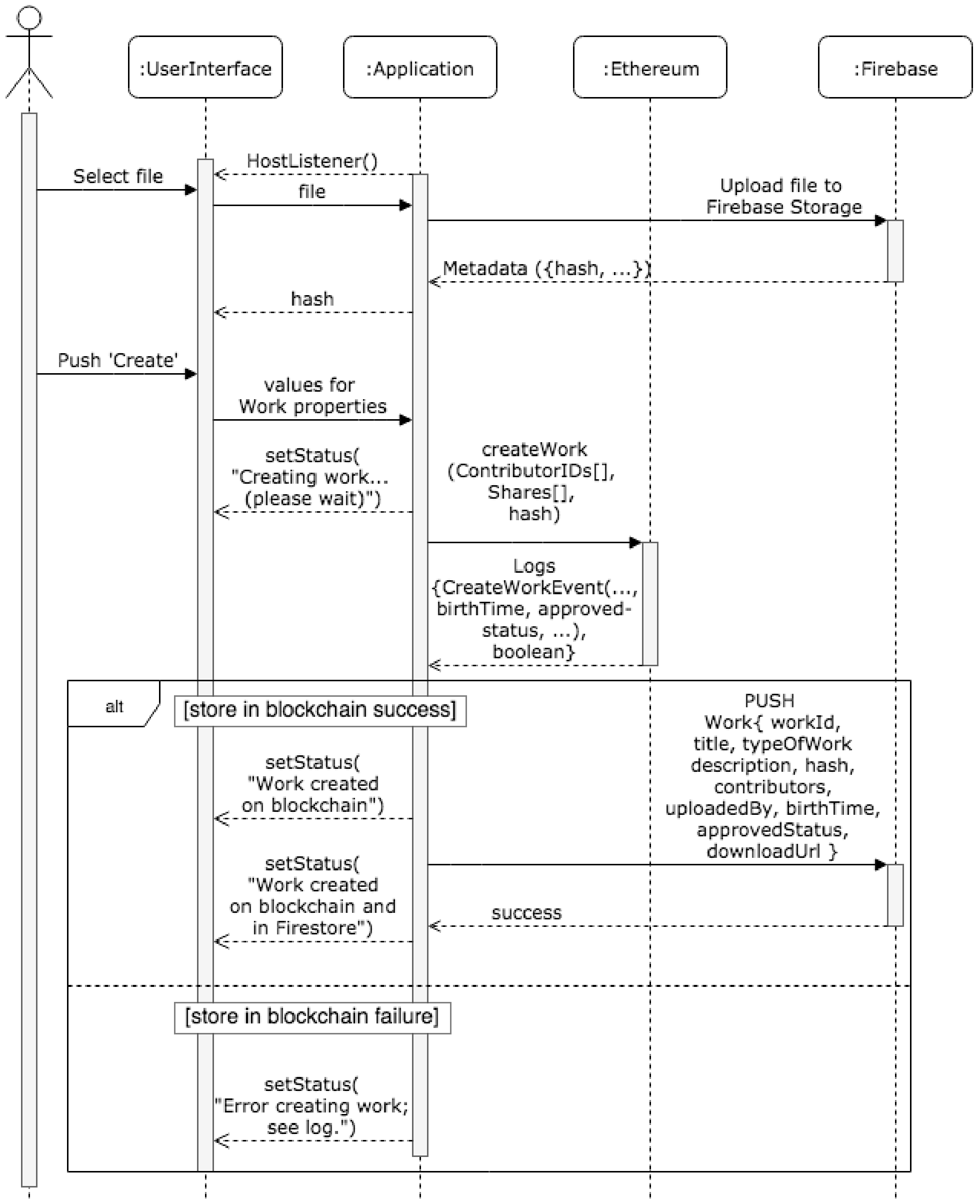

:UserInterface
:Application
:Ethereum
:Firebase
Select file
HostListener()
file
Upload file to Firebase Storage
Metadata ({hash, ...})
hash
Push 'Create'
values for Work properties
setStatus("Creating work... (please wait)")
createWork (ContributorIDs[], Shares[], hash)
Logs {CreateWorkEvent(..., birthTime, approved-status, ...), boolean}
alt
[store in blockchain success]
setStatus("Work created on blockchain")
PUSH Work{ workId, title, typeOfWork description, hash, contributors, uploadedBy, birthTime, approvedStatus, downloadUrl }
setStatus("Work created on blockchain and in Firestore")
success
[store in blockchain failure]
setStatus("Error creating work; see log.")

Figure 3.2: A simplified sequence diagram to illustrate the data flow in the application. The selected copyright file is stored in Firebase before the user submitting the work form in our application. The work with only essential properties is stored on the blockchain first, and if successful pushed to Firebase with additional properties.

3.4 Technologies

This section discusses the different tools and technologies used to develop our solution. We have split them into two categories: development environment and application dependencies.

3.4.1 Development Environment

Some of the technologies we have used are not vital parts of our application and have only had assistive roles throughout development. The following technologies do not serve a purpose once the dApp is officially deployed, but they have been beneficial for testing purposes.

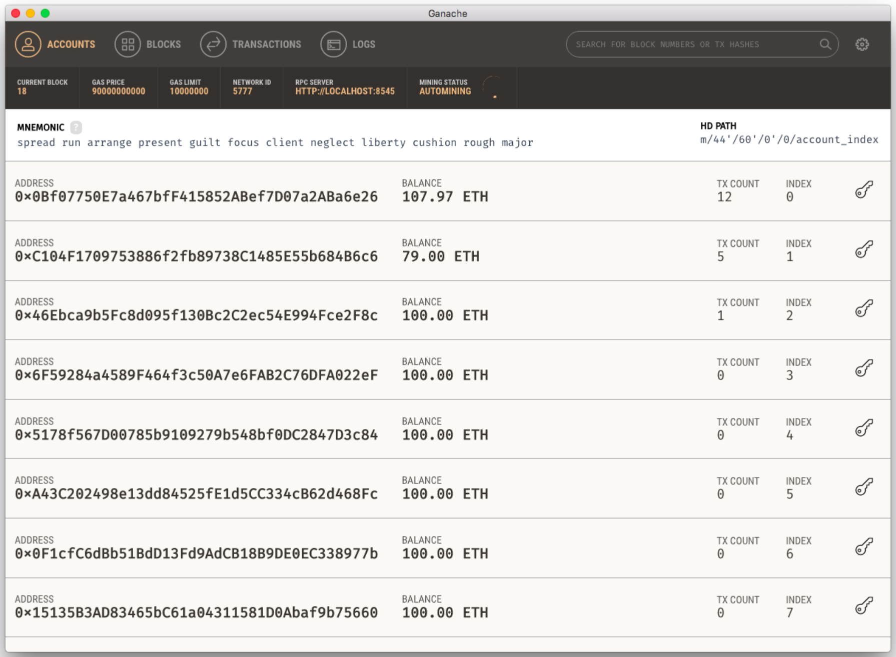
Visual Studio Code (v1.23.1)

All code is written in Visual Studio Code. This text editor is optimized for web application development. Additionally, Visual Studio Code makes it easy to import support extensions and packages that handles issues such as compilations errors. For instance, we used the JuanBlanco.solidity package that features syntax highlighting and code completion for the Solidity programming language.

Ganache (v1.1.0)

This desktop application is a locally hosted Ethereum blockchain that comes with 10 test accounts with a pre-filled balance of a 100 ETH. Ganache is a JavaScript implementation of the actual Ethereum blockchain[5]. The blockchain is automatically initialized once the application is open. Every transaction and block can be reviewed and examined, giving us meaningful insight into how our smart contracts' functions affect the state of the chain. Ganache's quick setup, simple user interface and integration with the Truffle framework (explained in next paragraph) made it ideal for hosting the contracts used in our web application.

[5]http://truffleframework.com/ganache/



CURRENT BLOCK	GAS PRICE	GAS LIMIT	NETWORK ID	RPC SERVER	MINING STATUS
18	90000000000	10000000	5777	HTTP://LOCALHOST:8545	AUTOMINING

MNEMONIC
spread run arrange present guilt focus client neglect liberty cushion rough major

HD PATH
m/44'/60'/0'/0/account_index

ADDRESS	BALANCE	TX COUNT	INDEX
0×0Bf07750E7a467bfF415852ABef7D07a2ABa6e26	107.97 ETH	12	0
0×C104F1709753886f2fb89738C1485E55b684B6c6	79.00 ETH	5	1
0×46Ebca9b5Fc8d095f130Bc2C2ec54E994Fce2F8c	100.00 ETH	1	2
0×6F59284a4589F464f3c50A7e6FAB2C76DFA022eF	100.00 ETH	0	3
0×5178f567D00785b9109279b548bf0DC2847D3c84	100.00 ETH	0	4
0×A43C202498e13dd84525fE1d5CC334cB62d468Fc	100.00 ETH	0	5
0×0F1cfC6dBb51BdD13Fd9AdCB18B9DE0EC338977b	100.00 ETH	0	6
0×15135B3AD83465bC61a04311581D0Abaf9b75660	100.00 ETH	0	7

Figure 3.3: Test accounts with their corresponding balance on a running Ethereum client.

Truffle Framework (v4.1.0)

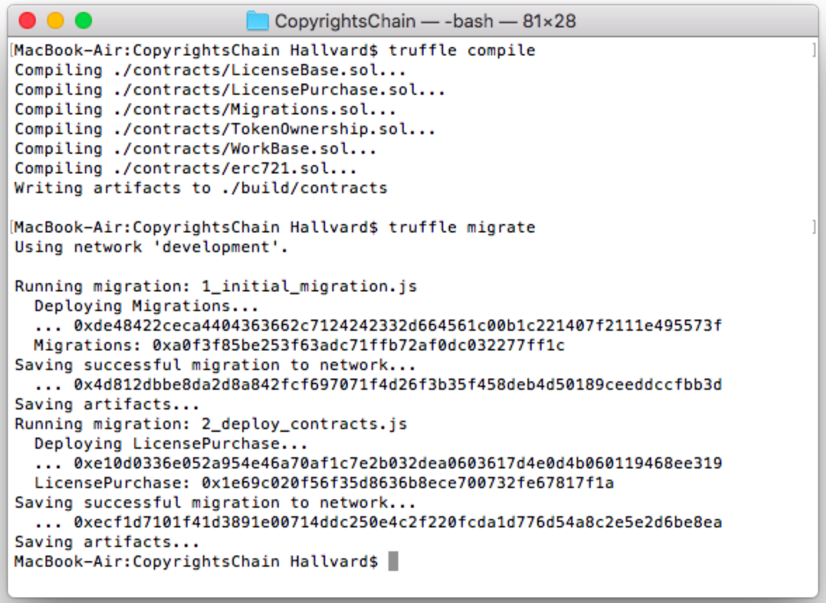
Truffle, an Ethereum development framework, describes itself as the “Ethereum swiss army knife”. Its purpose is to make dApp development more accessible for developers by simplifying compilation and migrations of smart contracts written in Solidity. Initializing a project using truffle gives us a specific project structure, as well as a set of predefined folders and scripts. Once installed, we are provided with a handful of terminal commands that makes building and deploying very simple:

"Truffle compile": The framework compiles our Solidity contracts and creates separate JSON-files as artifacts (one for each contract). These artifacts are used during the deployment of the contracts and the files should not be altered once they have been created[6].

"Truffle migrate": The framework also allows us to deploy the contracts to a running blockchain. It gives us two prewritten deployment scripts as a two-step migration process. When we run the command in a terminal window, the two deployment scripts are executed, and our contracts are deployed to our running local Ethereum blockchain.

Figure 3.4 shows successful compilation and migration of the contracts. We can view the four transactions made above on our running Ganache blockchain. This is illustrated in Figure 3.5.

[6]http://truffleframework.com/docs/getting_started/compile - [Accessed: 01-02-2018]

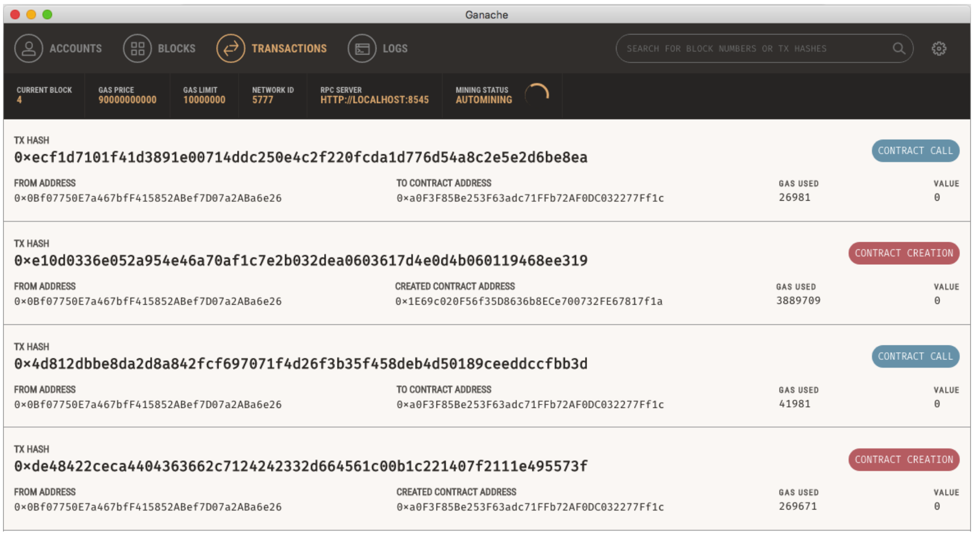


```
[MacBook-Air:CopyrightsChain Hallvard$ truffle compile
Compiling ./contracts/LicenseBase.sol...
Compiling ./contracts/LicensePurchase.sol...
Compiling ./contracts/Migrations.sol...
Compiling ./contracts/TokenOwnership.sol...
Compiling ./contracts/WorkBase.sol...
Compiling ./contracts/erc721.sol...
Writing artifacts to ./build/contracts

[MacBook-Air:CopyrightsChain Hallvard$ truffle migrate
Using network 'development'.

Running migration: 1_initial_migration.js
  Deploying Migrations...
  ... 0xde48422ceca4404363662c7124242332d664561c00b1c221407f2111e495573f
  Migrations: 0xa0f3f85be253f63adc71ffb72af0dc032277ff1c
Saving successful migration to network...
  ... 0x4d812dbbe8da2d8a842fcf697071f4d26f3b35f458deb4d50189ceeddccfbb3d
Saving artifacts...
Running migration: 2_deploy_contracts.js
  Deploying LicensePurchase...
  ... 0xe10d0336e052a954e46a70af1c7e2b032dea0603617d4e0d4b060119468ee319
  LicensePurchase: 0x1e69c020f56f35d8636b8ece700732fe67817f1a
Saving successful migration to network...
  ... 0xecf1d7101f41d3891e00714ddc250e4c2f220fcda1d776d54a8c2e5e2d6be8ea
Saving artifacts...
MacBook-Air:CopyrightsChain Hallvard$ 
```

Figure 3.4: Truffle commands successfully executed in terminal.



```
CURRENT BLOCK   GAS PRICE     GAS LIMIT   NETWORK ID   RPC SERVER              MINING STATUS
4               90000000000   10000000    5777         HTTP://LOCALHOST:8545   AUTOMINING

TX HASH
0×ecf1d7101f41d3891e00714ddc250e4c2f220fcda1d776d54a8c2e5e2d6be8ea                                          CONTRACT CALL
FROM ADDRESS                                  TO CONTRACT ADDRESS                           GAS USED   VALUE
0×0Bf07750E7a467bfF415852ABef7D07a2ABa6e26    0×a0F3F85Be253F63adc71FFb72AF0DC032277Ff1c    26981      0

TX HASH
0×e10d0336e052a954e46a70af1c7e2b032dea0603617d4e0d4b060119468ee319                                          CONTRACT CREATION
FROM ADDRESS                                  CREATED CONTRACT ADDRESS                      GAS USED   VALUE
0×0Bf07750E7a467bfF415852ABef7D07a2ABa6e26    0×1E69c020F56f35D8636b8ECe700732FE67817f1a    3889709    0

TX HASH
0×4d812dbbe8da2d8a842fcf697071f4d26f3b35f458deb4d50189ceeddccfbb3d                                          CONTRACT CALL
FROM ADDRESS                                  TO CONTRACT ADDRESS                           GAS USED   VALUE
0×0Bf07750E7a467bfF415852ABef7D07a2ABa6e26    0×a0F3F85Be253F63adc71FFb72AF0DC032277Ff1c    41981      0

TX HASH
0×de48422ceca4404363662c7124242332d664561c00b1c221407f2111e495573f                                          CONTRACT CREATION
FROM ADDRESS                                  CREATED CONTRACT ADDRESS                      GAS USED   VALUE
0×0Bf07750E7a467bfF415852ABef7D07a2ABa6e26    0×a0F3F85Be253F63adc71FFb72AF0DC032277Ff1c    269671     0
```

Figure 3.5: The four transactions made during migration to the blockchain

Angular CLI (v1.7.4)

Angular CLI is a Command Line Interface that automatically sets up an Angular project with its necessary files, configurations, and folders. It allows developers to generate code through the CLI that follows the best practices of Angular development. Angular CLI uses the module bundler webpack.js to set up a local web server which hosts the web application.

3.4.2 Application Dependencies

The following are technologies that our dApp depends on and would not function without.

Solidity (v0.4.19)

Solidity is the official language for smart contracts on the Ethereum blockchain. According to the documentation[7], it is a language that is influenced by C++, Python, and JavaScript, and runs on top of the Ethereum Virtual Machine (EVM). All business logic that dictates the rules of the dApp is written in Solidity. The programming language is still in its early days which means that developers are provided with fewer options and more limitations than what they usually can expect from a more mature programming language like Java. Our contracts are compiled with compiler-version 0.4.19. The low version number reveals how 19 minor changes have been released over the past two years while a major version release has yet to be made. The latest release at the time of writing this paper is 0.4.24.

In the very beginning of developing our smart contracts, we used an online Solidity IDE called Remix[8]. This online environment made it easy to learn the syntax and concepts of the language. However, Remix becomes a liability when writing more complex smart contracts do to browser memory issues. Thus, the browser IDE does not facilitate complex dApp projects.

Angular (v6.0.0)

One of the most commonly used JavaScript frameworks is Angular. With its large community and continuous improvements and development, it is an excellent option for web application development. “Angular combines declarative templates, dependency injection, end to end tooling, and integrated best practices to solve development challenges.”[9] Angular projects are single page applications where multiple components are dynamically loaded on a single page rather than routing between

[7]http://solidity.readthedocs.io/en/v0.4.24/index.html - [Accessed: 21-01-2018]
[8]https://remix.ethereum.org/ - [Accessed: 23-01-2018]
[9]https://angular.io/docs - [Accessed: 08-01-2018]

entire pages. This improves the user experience by faster transitions between the views.

We use Angular v.6.0.0 which is the latest version of the framework at the time of writing this paper. TypeScript (a superset of JavaScript) is the primary language of the framework. Angular makes it easy to import external libraries and handle dependencies. In our application we use Bootstrap and Angular Material for styling, AngularFire2 for connecting to Firebase, and of course Web3.js to communicate with our blockchain to mention a few. The project consists of one module with multiple components and services. Some key features of the Angular application will be explained in detail later in this paper.

Web3.js (v1.0.0-beta.34)

After doing some research on how we could reach the functions of our deployed Solidity contracts, we landed on web3.js as a great provider for a web-based solution. Web3.js is a "collection of libraries which allows us to interact with a local or remote Ethereum node, using an HTTP or IPC connection"[10]. It is a JavaScript API backed by the Ethereum Foundation that gives us some predefined functionality that our front-end code can benefit from.

For us to understand how we use Web3.js in our Angular project we used a tutorial[11] for inspiration. This tutorial was a great way to learn how to use Web3.js and how we connect the front-end of the application to the deployed smart contracts.

MetaMask (v4.6.1)

Regular web browsers are not initially equipped with the right tools to communicate with a blockchain. As previously stated, Web3.js handles this type of communication, but this framework is not available unless it is injected into the browser. This is where MetaMask comes in. This Google Chrome extension gives us access to our blockchain by injecting Web3.js into the application's JavaScript context.[12]

MetaMask enables the users of our web application to read from and write to the blockchain. Reading from the blockchain is free of charge, but writing to the blockchain requires gas. Using their Ethereum address and corresponding private key, users can log in to MetaMask and approve or reject transactions from their address. The extension opens a window automatically when executing "write-functions" to the deployed contract.

[10]https://web3js.readthedocs.io/en/1.0/ - [Accessed: 25-01-2018]
[11]https://github.com/Nikhil22/angular-truffle-starter-dapp - [Accessed: 27-01-2018]
[12]https://metamask.io/ - [Accessed: 26-01-2018]

Firebase

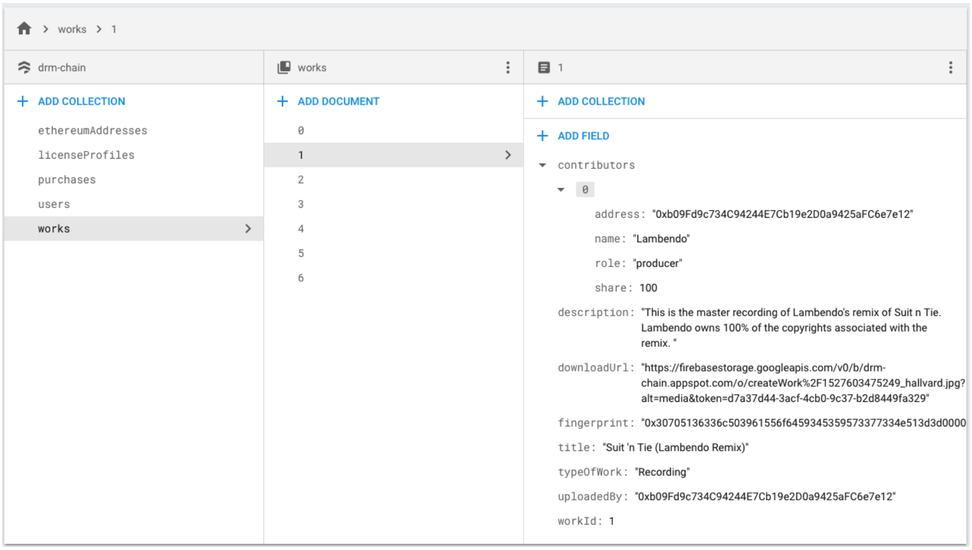
Firebase is a development platform backed by Google. The platform allows for web applications with serverless architecture. Its connection with our front-end web application is established through the official library[13] for Angular. The serverless architecture paradigm allows us to pass data from JavaScript directly to a set of predefined service endpoints without having to set up a server and create methods for GET, POST, etc. In our application, we use three separate Firebase features: authentication, database, and storage.

Firebase Authentication: The Authentication SDK by Firebase provides us with email and password based authentication. Users on our platform can authenticate themselves by an email and a password and receive a token that lasts for the duration of their session. Authentication SDK is made available through the AngularFire2 library in our web application.

Firebase Firestore: Another feature we use is the Firebase database, more specifically the Cloud Firestore NoSQL database. Firestore is flexible as it does not enforce every document in a collection to have the same fields. The database stores incoming objects as documents in collections. A document can have multiple fields, each of which maps to a value. Figure 3.6 displays a snapshot of work one from the 'works' collection in our Firestore database.

Cloud Storage: The Firebase Cloud Storage is a file storage service backed by Google. We use this service whenever a file is uploaded to the web application. It stores files in an appropriate folder, e.g., /uploadedWorkFiles. Cloud Storage is compatible with Firebase's Authentication SDK and can easily be scaled to handle much traffic.

[13]angularfire2 v5.0.0-rc.7



```
ethereumAddresses
licenseProfiles
purchases
users
works
```

```
0
1
2
3
4
5
6
```

```
contributors
  0
    address: "0xb09Fd9c734C94244E7Cb19e2D0a9425aFC6e7e12"
    name: "Lambendo"
    role: "producer"
    share: 100
description: "This is the master recording of Lambendo's remix of Suit n Tie.
             Lambendo owns 100% of the copyrights associated with the
             remix. "
downloadUrl: "https://firebasestorage.googleapis.com/v0/b/drm-
             chain.appspot.com/o/createWork%2F1527603475249_hallvard.jpg?
             alt=media&token=d7a37d44-3acf-4cb0-9c37-b2d8449fa329"
fingerprint: "0x30705136336c503961556f6459345359573377334e513d3d0000
title: "Suit 'n Tie (Lambendo Remix)"
typeOfWork: "Recording"
uploadedBy: "0xb09Fd9c734C94244E7Cb19e2D0a9425aFC6e7e12"
workId: 1
```

Figure 3.6: Screenshot of our Cloud Firestore database. On the left is a list of our collections, followed by a list of documents with a key equal to the Work ID. On the right is an example of a stored musical work with all the corresponding attributes.

Chapter 4

Implementating a Decentralized Application

In this chapter, we explain the technical implementation of our decentralized application. The back-end logic is written in a collection of smart contracts. The front-end user interface is written in an Angular web application project. Developing the dApp is what has kept us occupied for the majority of time during the master project. We encourage interested readers to visit our public repository[1] on GitHub to inspect the code. Also, the repository includes a "README-file" explaining how to download, run, and test the dApp locally on any computer. In this chapter, we discuss the most critical parts of the code. This includes important data structures, functions, communication protocols, and other essential aspects of the dApp. We also justify some of our technical design decisions and explain how different designs impact the gas cost of interacting with the dApp.

4.1 Implementing Smart Contracts

This section describes the implemented smart contracts that dictate the rules of the application. There are five distinct smart contracts to keep things organized. Logically related code is bundled together in separate files instead of one giant file. Like Java and other object oriented programming languages, Solidity supports code inheritance. Our inheritance system looks like this:

- contract **WorkBase**
- contract **ERC721**
- contract **TokenOwnership** is **WorkBase**, **ERC721**
- contract **LicenseBase** is **TokenOwnership**
- contract **LicensePurchase** is **LicenseBase**

[1]https://github.com/hkbhaugen/RightsDoneRight/

Thus, LicensePurchase inherits all data and functions defined in its parent contracts. Ultimately, this is the deployed contract our application points to. We will discuss only the most essential parts of each of these contracts. Most of the details are explained using in-line comments in the code.

4.1.1 WorkBase.sol - The Application's Core Contract

This contract defines the most fundamental code used throughout the back-end system. It includes constant data types, state variables, and functions for managing the state variables. When a contract deploys to the Ethereum blockchain, the EVM dedicates storage space where values for internal state variables are stored.

Storing Musical Work

Workbase contains a lot of the core data of the app. Firstly, it defines a musical work as a struct. A struct is a custom data type that may include several attributes of different types:

```
    //every musical work is represented by a struct with a set of
        variables
    struct Work {

        //timestamp from the block when this work was registered
        uint64 birthTime;

        //the MD5-hash of the uploaded file
        bytes32 fingerprint;

        //list of contributors Ethereum address
        address[] contributors;

        //list of percentage ownership assigned to each contributor
        uint[] splits;
    }
```

The contract then declares an array that holds Work structs:

```
    //An array containing the work struct for all registered works in
        existence.
    //the index of the struct in the array is the ID of the work
    Work[] public workDB;
```

The workDB is the application's master database that holds all the registered musical works. Whenever a user registers a new work, it is added to this array. Its index position in the array becomes the work's unique ID. This is shown in the createWork() function:

```
    //function for registering a work
    //This will update the work list of all contributors and assign RCN
        -tokens according to the _splits[] argument
    function createWork (bytes32 _fingerprint, address[] _contributors,
         uint[] _splits) public returns (bool) {

        require(_contributors.length == _splits.length);

        //get the timestamp for when the current block is mined
        uint64 _birthTime = uint64(now);

        //A work struct with info about the work. PS: this data is
            permanent
        Work memory _newWork = Work({
            birthTime: _birthTime,
            fingerprint: _fingerprint,
            contributors: _contributors,
            splits: _splits
        });

        //we push the newly created work-struct to the workDB
        //Its ID (index in the workDB array) is assigned the newWorkId
            variable
        uint _newWorkId = workDB.push(_newWork) - 1;

        //set the work to inactive until all contributors has approved
            the splits
        //NOTE: this is also the default value, so vi may skip this
            code-line
        workIdToApproved[_newWorkId] = false;

        //emit the createWork event
        CreateWork(_newWorkId, _birthTime, _fingerprint, _contributors,
             _splits);

        //return that the function executed successfully
        return true;
    }
```

Mappings

The mapping data type in solidity plays an important role in many smart contracts. It can be thought of as a hash table that stores data by connecting a key with a value. It is declared as the following:

```
mapping(_KeyType => _ValueType)
```

WorkBase declares eight different mappings. To avoid confusion, we will only discuss a few of them. One of the mappings keeps track of whether a work has been approved or not. It is defined as the following:

```
//mapping from workId to a boolean approved value
mapping(uint => bool) public workIdToApproved;
```

Notice how the workIdToApproved mapping is updated in the createWork function described above. A work is not approved until all involved contributors have approved it. When a work is approved, it is considered valid, and the smart contract will issue copyright tokens to the contributors as proof of ownership of that work. The following function does this:

```
    //Function for issuing RCN-tokens to contributors of a work.
    //Called only after _allContributorsApproved returns to true
    function _issueRcnTokens(uint _workId) internal returns (bool) {
        //variable containing the relevant work struct
        Work memory _work = workDB[_workId];

        //list of all contributors
        address[] memory _contributors = _work.contributors;

        //list of splits
        uint[] memory _splits = _work.splits;

        //Loop over all contributors
        for (uint i= 0; i < _contributors.length; i++) {
            address _contributor = _contributors[i];

            //update mapping from contributors to their worklist
            //meaning this contributor has one or more tokens related
                to this work
            addressToWorkList[_contributor].push(_workId);

            //update mapping from workId to tokenHolders
            workIdToTokenHolders[_workId].push(_contributor);

            //Loop from value 0 to _splits[i] (_splits[i] = number of
                tokens for _contributors[i])
            for (uint j= 0; j < _splits[i]; j++) {

                //create new rcn-token and push it to the master rcnDB
                //its value is the related workId and its index in the
                    array becomes the tokenId
                uint _newTokenId = rcnDB.push(_workId) - 1;

                //Fill the tokenID-list in workIdToTokenList
                workIdToTokenList[_workId].push(_newTokenId);

                //call the internal transfer function for assigning
                    token ownership
                _transferToken(0, _contributor, _newTokenId);
            }
        }
    }
```

4.1.2 TokenOwnership.sol - Copyrights as Tokens

The _issueRcnTokens function shown above creates and issues tokens to contributors. These tokens represent copyrights for an approved work. The work's splits-attribute determines how many tokens each contributor is issued. We define one token to represent 10% ownership of a work's overall copyrights. Thus, the _issueRcnTokens function issues 10 tokens when executed. Theoretically, we could define one token to be 1% (issue 100 tokens for one approved work). 100 tokens would allow for finer granularity and more flexible split-options. However, issuing that many tokens would require significantly more EVM-computations and result in a more expensive transaction. While experimenting with different percentages ownership per token, we learned that the cost scales linearly with the number of issued tokens. That is, issuing 100 tokens instead of 10 tokens results in a 10 times more expensive transaction.

The copyright token has the token symbol "RCN" which is an abbreviation for RightCoin. Hence, the function name _issueRcnTokens. We stored all issued tokens in a public array declared like this:

```
//A master database of all issued copyright tokens.
//Whenever a work is created and approved, a number of tokens are
    issued and added to this array.
//A tokens index in the array is its tokenId.
uint[] public rcnDB;
```

We use a mapping to keep track of which tokens are owned by what Ethereum account. It is declared like this:

```
    // mapping from tokenID to the address of its owner
    mapping(uint => address) public tokenIdToOwner;
```

The following _transferToken function updates the tokenIdToOwner mapping. Notice how this function is called at the end of the _issueRcnTokens function mentioned earlier. _transferToken is an internal function that is responsible for actually assigning ownership of a unique token to a person. In our case, internal functions means that they are used in different use cases by different components of the back-end code.

```
//function for token ownership assignment
//used in _issueToken() and transfer()
function _transferToken(address _from, address _to, uint _tokenId) internal {
    //increase the token count associated with the _to address
    addressToTokenCount[_to]++;

    //for "newborn tokens", we insert the (token=>address) key-value pair in the tokenIdToOwner mapping
    //for existing tokens, we update the address-value of a tokenID-key to the _to address
    tokenIdToOwner[_tokenId] = _to;

    //For existing tokens, the _from address is not 0x0
    //This is the case in transer(), and we have to to more checks and operations
    if (_from != address(0)) {

        addressToTokenCount[_from]--;

        //clear approved transfer allowance for the token
        delete tokenIdToApproved[_tokenId];

        //get the workId associated with the _tokenId
        uint _workId = _getWorkIdfromTokenId(_tokenId);

        //check if _to address already has the workId in his workList
        if (_workIdNotInWorkList(_workId, _to)) {

            //add the workId to worklist associated with the _to address
            addressToWorkList[_to].push(_workId);

            //add the _to address in workIdToTokenHolders mapping
            workIdToTokenHolders[_workId].push(_to);
        }
    }
}
```

We have previously mentioned Ethereum's ERC721 token standard which represents unique non-fungible items. Our copyright RCN-token implements this standard. One can see from TokenOwnership's contract definition that it inherits from the ERC721 interface contract:

```
contract TokenOwnership is \textbf{ERC721}, WorkBase {}
```

The TokenOwnership contract is required to fill in implementation for the functions declared by the ERC721 standard because of this inheritance. The ERC721 standard is defined as the following:

```
/// @title Interface for contracts conforming to ERC-721: Non-Fungible
    Tokens
/// @author Dieter Shirley <dete@axiomzen.co> (https://github.com/dete)
contract ERC721 {
    // Required methods
    function totalSupply() public view returns (uint256 total);
    function balanceOf(address _owner) public view returns (uint256
        balance);
    function ownerOf(uint256 _tokenId) external view returns (address
        owner);
    function approve(address _to, uint256 _tokenId) external;
    function transfer(address _to, uint256 _tokenId) external;
    function transferFrom(address _from, address _to, uint256 _tokenId)
         external;

    // Events
    event Transfer(address from, address to, uint256 tokenId);
    event Approval(address owner, address approved, uint256 tokenId);

    // Optional
    // function name() public view returns (string name);
    // function symbol() public view returns (string symbol);
    // function tokensOfOwner(address _owner) external view returns (
        uint256[] tokenIds);
    // function tokenMetadata(uint256 _tokenId, string
        _preferredTransport) public view returns (string infoUrl);

    // ERC-165 Compatibility (https://github.com/ethereum/EIPs/issues
        /165)
    function supportsInterface(bytes4 _interfaceID) external view
        returns (bool);
}
```

We are not going to discuss how our TokenOwnership contract implements all these functions. The full implementation can be found in our GitHub repository (link in the beginning of chapter 4).

The fact that all ERC721 tokens follow the same standard and all ERC721-functions are public facilitates a standard way for users to interact with RCN-tokens as any other ERC721 tokens. That means that one can transfer the ownership of specific copyrights to another Ethereum-account using an Ethereum-wallet that supports the ERC721 standard. Toshi[2] is an example of such a wallet. Additionally, it is possible to sell copyrights tokens on a marketplace or an exchange.

4.1.3 LicenseBase.sol - Storing License Contracts on Ethereum

The LicenseBase contract is similar to WorkBase. It includes code for storing license profiles on the blockchain that can be viewed and purchased through our web application. A license profile defines data about a licensing agreement. It is defined as a custom struct data type.

```
    //A struct containing the birthTime and fingerprint of a license
        profile
    struct LicenseProfile {
        //The block number for when the license profile is created
        uint birthTime;

        //price (in wei)
        uint price;

        //MD5-hash of the profile-document describing the terms for the
             profile
        bytes32 fingerprint;
    }
```

When created, license profiles are stored in an array defined like this:

```
    //An array containing the licenseProfile structs for all registered
         licenseProfiles in existence
    //the index of the struct in the array is the ID of the
        licenseProfile
    LicenseProfile[] public licenseProfileDB;
```

LicenseBase also implements the function createLicenseProfile that will create a licenseProfile struct and push it to the licenseProfileDB array. This database array contains all licenseProfile structs in existence. We will not show the createLicense-Profile function as it is very similar to the createWork function described in the previous section.

We keep track of which contracts belong to which works in a mapping defined like this:

```
   //mapping from licenseProfileIdToWorkId
    mapping (uint => uint) public licenseProfileIdToWorkId;
```

[2]https://itunes.apple.com/us/app/toshi-ethereum-wallet/id1278383455?mt=8

4.1.4 LicensePurchase.sol - Handling License Purchases

The LicensePurchase contract defines the function used when buying a specific license for a work. It is implemented like this:

```
    function buyLicense(uint _licenseId) public payable returns (bool success) {

        //License profile has to be activated in order to buy it
        require(activatedProfiles[_licenseId]);

        //license profile struct variable
        LicenseProfile memory _licenseProfile = licenseProfileDB[_licenseId];

        //the amount of weis the purchaser sent in the transaction
        uint _amount = msg.value;

        //validate transaction value
        require(_amount == _licenseProfile.price);

        //associated workId
        uint _workId = _getWorkIdByLicenseProfileId(_licenseId);

        //list of tokens associated with the work associated with _licenseId
        uint[] memory _tokenList = _getTokenListFromWorkId(_workId);

        //the value of which we increase each tokenBalance
        //compared to what we would expect
        uint incrementValue = (msg.value / _tokenList.length);

        //loop over all tokens belonging to the work associated with _licenseId
        for (uint i = 0; i < _tokenList.length; i++) {
            //update the tokenIdToBalance mapping increasing the balance of the relevant tokens
            tokenIdToBalance[_tokenList[i]] += incrementValue;
        }

        //uppdate the addressToPurchasedLicenseIds mapping adding the _licenseId to the list value
        addressToPurchasedLicenseIds[msg.sender].push(_licenseId);

        //increment the count of which a license profile has been purchased
        licenseProfilePurchaseCount[_licenseId]++;

        //Emit LogPurchase event
        LogPurchase(uint64(now), _licenseId, _workId);

        return true;
    }
```

This function is quite complicated as it contains multiple function calls and statements. Notice the following details:

- The function definition includes the "payable" keyword. That allows the function to receive ETH as a part of the transaction.
- The function receives a licenseProfile as input. That is the licenseProfile that the sender of the transaction wants to purchase.
- The amount of ETH sent with the transaction has to be equal to the price of the licenseProfile input.

As mentioned earlier, smart contracts are accounts with its own ETH balance. When users execute the buyLicense function, they send ETH to the smart contract increasing the balance of the contract. That may seem strange as it looks like the contract owns all the paid royalties. However, this is not the case. The smart contract defines a mapping named tokenIdToBalance. It is defined like this:

```
//mapping from a tokenId to its balance
    //the sum of all token balances is equal to the contract's balance
    mapping (uint => uint) public tokenIdToBalance;
```

This mapping specifies how much of the contract's balance each token is entitled to. In fact, the entire contract balance is the sum of all tokens' aggregated entitlement as defined in the mapping. Notice how the buyLicense function updates tokenIdToBalance to increase the balance for each relevant token when somebody purchases a license. This concept is illustrated in the following figure:

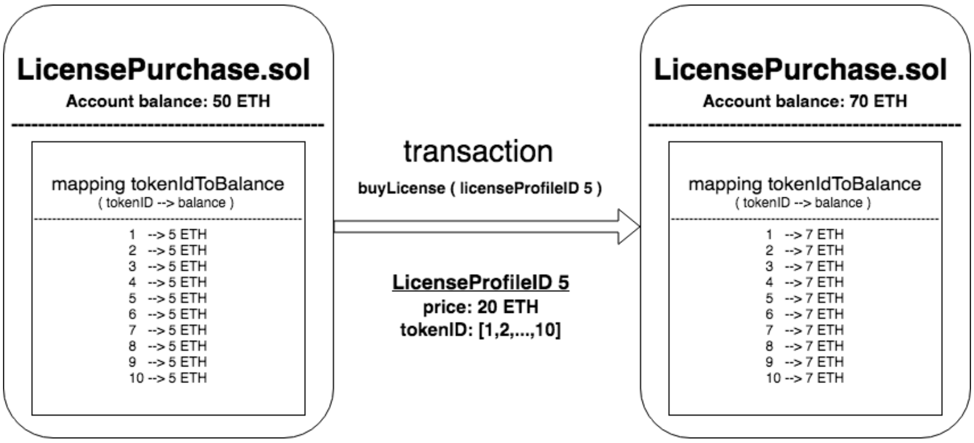


Figure 4.1: Illustrates the concept of token balances when a license is purchased.

Figure 4.1 demonstrates how the state of the smart contract changes when somebody purchases a license. For the sake of simplicity, we assume that there only exists one work (associated with tokens [1,2,..10]) on the application. The initial contract balance is equal to 50 ETH. That is also the total sum of the values in the tokenIdToBalance mapping. The transaction represents a purchase of a licenseProfile with ID 5. This license costs 20 ETH and is associated with tokens [1,2,..10]. When the EVM executes the transaction, the contract's final balance increases to 70 ETH. At the same time, the smart contract code updates its tokenIdToBalance mapping. Tokens [1,2,..10] are now entitled to 7 ETH each.

The reason we implemented the described balance control scheme is to save gas. A more expensive approach would be a forwarding design. In such a scheme, the licensePurchase contract would receive ETH from the purchase transaction and immediately forward it to appropriate token owners. However, every forwarded transaction cost gas which the initial purchaser ultimately would have to pay.

The LicensePurchase contract also defines a function named withdrawFromWorkId(). This function enables token owners to withdraw the generated royalties (held by the contract) for a given work. The code for this is the following:

```
     //function for withdrawing the balance of all owned tokens associated with one work
    function withdrawFromWorkId(uint _workId) public returns (bool success) {

        //list of tokens owned by msg.sender associated with the input _workId
        uint[] memory _ownedTokens = getTokensFromWorkIdAndAddress(_workId, msg.sender);


        uint _totalWithdraSum = 0;

        //iterate over owned tokens
        for (uint i = 0; i < _ownedTokens.length; i++) {
            uint _tokenId = _ownedTokens[i];

            //increase the withdraw sum with the balance of _tokenId
            _totalWithdraSum += tokenIdToBalance[_tokenId];

            //reset the balance associated with _tokenId
            delete tokenIdToBalance[_tokenId];
        }

        //transfer the sum from the contract balance to msg.sender
        msg.sender.transfer(_totalWithdraSum);

        LogWithdrawFromWorkId(msg.sender, _workId, _totalWithdraSum);

        return true;
    }
```

The function takes a workID as input and evaluates which relevant tokens msg.sender owns (msg.sender is the Ethereum address of the user who initiated the transaction). For these tokens, the function calculates the aggregated token balance sum. This sum is then sent from the smart contract account to the account of msg.sender.

4.2 Building a Web Application with Angular

In this section, we will discuss the source code of our web application. That includes a brief walkthrough of the Angular project structure, essential components, services, and models. We would like to provide some insight into how our application functions and justify some of the decisions we have made during development.

4.2.1 Project Structure

The Angular project consists of two main folders: the node modules folder and the source folder. The former contains all the JavaScript libraries on which our application depends. Some examples are Web3.js, AngularFire2, and RxJS (a reactive programming library).

- node_moduels
- src
  - app
    - auth
    - blockchain-services
    - firestore-services
    - models
    - components
      - create-work
      - create-work.component.css
      - create-work.component.html
      - create-work.component.ts
      - home
      - login
      - ...
    - app-routing.module.ts
    - app.component(.ts, .html, .css)
    - app.module.ts
  - index.html
  - ... config files, global stylesheet, ...

The second folder is our source code. It contains an app folder with all our services, components and models. It also defines our single web page index.html where the HTML-body is dynamically loaded with the different component views. Additionally, the source folder contains some configuration files, an environment specification, and global stylesheets. In the next section, we will review the contents of the app folder.

4.2.2 Authentication and Authorization

Like any other user based application, our application needs to have some form of authentication. The official Firebase library for Angular can provide us with this. In the auth folder, we have an authentication service that checks if a user is logged in. If that is the case, the service returns the user's details from a collection in Firestore. The current user is set as an observable, meaning that other components and services can subscribe to this user object and have access to the user's details. The authentication service also features methods for user sign in, sign up, and sign out.

On our platform, we do not wish to give the same access to every user. Users will have access to different views depending on their needs when signing up. Angular features so-called Route Guards. A guard is a class used to restrict access to one or multiple views in the application. We use route guards to provide users with only relevant functionality. For example, copyright owners may want to register a copyright for musical works that they own. The same does not apply to licensees who simply want to purchase licenses for different works. Another example is that non-registered users of the platform should only be able to query the copyright database and view the results, but not purchase licenses.

4.2.3 Services

In Angular, services are classes with a specific task or purpose. Unlike components, they do not have a user interface or any templates related to them. Services can serve multiple components, and they are often used to fetch data from (and pass data to) other servers. “Service providers can be injected directly into components as dependencies.”[3]. The application has two separate sets of services: one set for blockchain related tasks and one set for Firebase.

Blockchain Services

In our suggested solution, there are two services related to the blockchain: web3.service.ts and the ethereum.service.ts.

The first service is mainly for establishing a Web3.js provider so that we can connect our Angular application to the running blockchain. That is the service where MetaMask is set as the current Web3 provider.

The second service provides the components with access to the functionality of the deployed smart contract. Code snippet 1, taken from this service, illustrates how we manage to call the create work function of the deployed smart contract with specified parameters. This method is called once a user registers a work. Since this contract function writes to the blockchain, we also specify which Ethereum address (account) that will pay for the transaction as well as a total gas limit for the transaction. One could also specify the gas price, but we did not find this necessary as it can be adjusted in a MetaMask window before accepting the transaction.

```
createWork(account, fingerprint, contributors, splits): Observable<any>
     {
    return Observable.create(observer => {
        this.LicensePurchase.deployed()
            .then(instance => {
                // Calling the function 'createWork' from the contract
                    with respective parameters
                return instance.createWork(fingerprint, contributors,
                    splits, { from: account, gas: 6400000 });
            })
            .then(value => {
                // Passing the returned value to the observable
                observer.next(value);
                observer.complete();
            })
            .catch(e => {
                console.log(e);
                observer.error(e);
            });
```

[3]https://angular.io/guide/architecture

```
    });
}
```

Firebase Services

The second set of services is related to Firebase. We have a service for every collection in Firestore (license profiles, purchases, users, and works). The complexity of these services is generally low. They have a push method and one or two get methods.

The user service has more complicated methods than the other services. While the other services push data to a new document in their respective Firestore collection, the user service gets the logged in user's document and modifies its attributes. Retrieving the user document is done by subscribing to the current user from the authentication service. The method displayed in the code below is the method that updates every contributor's user document.

```
pushUnapprovedWorksToUsers(contributorIds, workId) {
    // Looping through all contributors
    contributorIds.forEach(uid => {

        // Get the user document for the current contributor
        const doc = this.afs.doc(`users/${uid}`).ref;

        // Run a transaction to prevent others from writing to the same document at the same time.
        this.afs.firestore.runTransaction(transaction => {
            return transaction.get(doc).then(snapshot => {

                // Directly appending to the value of array property in Firestore is not possible.
                // We do for that reason have to take a snapshot of the stored array and push to it locally.
                let unapprovedWorksArray = snapshot.get('unapprovedWorks');
                if (typeof unapprovedWorksArray !== 'undefined') {
                    unapprovedWorksArray.push(workId);
                } else {
                // Create a new array if the array does not already exist in the document
                    unapprovedWorksArray = new Array<number>();
                    unapprovedWorksArray.push(workId);
                }

                // Replaces the current value of 'unnaprovedWorks' with the local array variable
                transaction.update(doc, 'unapprovedWorks',
                    unapprovedWorksArray);
                });
        });
    });
}
```

The method takes in an array of user IDs related to the proposed musical work and a work ID. We iterate over the contributor IDs and modify the user document of every contributor of the work. The user service has similar methods for the approved work array as well as for both inactivated and activated license profile arrays. These methods are all triggered by successful registration on the blockchain.

4.2.4 Models

The application has a few models that are used as data types containing a set of attributes in our components. These models are defined in separate interfaces and can be used by any component or service in the application. Not only do models ensure better code quality, but they also enforce object storage with identical property names.

One example is the work model as seen in the code below. This model is used as a data type describing a work object. Some of the properties are optional (as seen by the question mark behind the name) and do not have to be set when a Work object is instantiated.

```
import { Contributor } from './contributor';

export interface Work {
    workId?: number;
    title: string;
    typeOfWork: string;
    description: string;
    contributors: Array<Contributor>;
    fingerprint: string;
    uploadedBy?: string;
    birthTime?: number;
    approvedStatus?: boolean;
    downloadUrl?: string;
}
```

The work model is used In the create-work component. The value of the form is set to an instance of the work model type when the form is submitted. All the mandatory properties are set in the form while other methods set the optional properties. The reason for this is because some of the properties can only be set once an event from the Solidity contract is returned and provides the component with some additional data. Other properties like the ‘downloadUrl’ and ‘uploadedBy’ depend on information from Firestore.

4.2.5 Components

Every view of our application is a component. The components consist of a TypeScript file and related templates in HTML and CSS. The HTML and CSS files define the user interface, and the TypeScript file handles the data flow both to and from the user interface.

The TypeScript file imports models and services to handle this flow. By using services and models, components function as a control center for data flow between the user interface and the services that communicate with the back-end.

All Angular applications require a root component “that connects a component hierarchy with the page DOM.” (https://angular.io/guide/architecture). Our root component (app.component.ts) checks if a user is logged in and renders the application’s navigation bar. The root component will load other components to the view based on the current path (URL).

The create-work component displays a form that a copyright owner can fill out and submit to register a work. The instance of the work model is set to the form value. As mentioned, only a few attributes of the work object are added to the blockchain when the work is submitted. The sequence is explained in Figure XX. The following code shows how this is implemented in the create-work component:

```
createWork = () => {

    // Displaying current status for the user
    this.setStatus('Creating work... (please wait)');

    // Pushing vital properties to the Blockchain. Since the logged in user pays for the
    // Ethereum transaction the 'user.ethereumAddress' is sent as a parameter as well.
    this._ethereumService.createWork(this.user.ethereumAddress, this.fingerprint,
      this.contributorsToChain, this.splitsToChain)
      .subscribe(eventCreatedWork => {

        // If the transaction is successfully completed and the work has been
        // stored on the blockchain.
        if (eventCreatedWork.logs[0].type === 'mined') {

          this.setStatus('Work stored on blockchain.');

          // Sets the ID that the work was given on the blockchain to a property of the Work instance.
          // tslint:disable-next-line:radix
          this.work.workId = parseInt(eventCreatedWork.logs[0].args.workId);

          // Sets a property of the Work instance equal to the download
              URL from the returned
          // metadata object upon file upload completion to Firebase.
          this.work.downloadUrl = this.metadata.downloadURL;

          // Overwriting the contributors property of the Work instance
              to an array of more complex objects
          this.work.contributors = this.contributorsToFirestore;

          // Pushing the new Work to the 'works' collection in
             Firestore and updating fields in every
          // contributor's user document in the 'users' collection.
          this.pushToFireStore(this.contributorIds, this.work);

        } else {
          this.setStatus('Not mined');
        }
      }, e => {
        this.setStatus('Error creating work; see log.');
      });
    }
```

4.2.6 Modules

The basic building blocks of an Angular application are modules. In our application, there are two modules: the routing module and the root module. The root module imports all components, dependencies and services and launches the application. The routing module defines all possible routes in the application, specifies the paths and places necessary guards. The routing module is imported into the root module.

Chapter 5
Results

The third and final part of the thesis includes chapter 5-7 where we outline the results of our investigation, discuss those results, and present our conclusion where we answer the research question.

In this chapter, we outline the results of our research. It can be split into three main contributions: background analysis, back-end smart contracts, and front-end web application.

5.1 Background Analysis

Research and Identify Music Industry Problems

Our research provides a detailed description of the complex modern music industry. Furthermore, it delineates key issues in the industry and uncovers the fundamental reasons behind them. As we gained knowledge of the overall hierarchy of the music industry and how it has changed over the years, we were able to discover several issues and group them into categories and subproblems:

1. The industry structure and friction between the players
    - Non-transparent value chain and NDA licensing contracts between players
    - Slow non-automatic royalty payment system
2. Lack of information and standards
    - No official global copyright database
    - No international standards for licensing and reporting

Applied Blockchain Concepts

During our research, we connected blockchain properties with the mentioned issues. We learned about the Ethereum blockchain and realized that the platform facilitates trustless decentralized application. While conventional applications require users to trust a third party, Ethereum-based applications don't have this requirement. This distinction has profound consequences in terms of transparency and trust. Furthermore, we explain how these elements can be applied in the music space to increase hierarchical transparency, and make the whole music industry more fair for musicians. The modern music industry relies heavily on trusted entities. A blockchain based music industry would be a transparent community with publicly available transactions and licensing contracts. Our research also suggests that blockchain, as an economic transaction system, can solve problems related to today's inefficient money flow. By using blockchain, fast global royalty payments can become a reality.

5.2 Smart Contracts

Our master project's second contribution is the back-end code implemented in smart contracts. The results serve as a proof-of-concept demonstration and not as a finished product. For that reason, the implementation itself is not deployed, and people cannot easily interact with it. However, we have demonstrated how to compile and deploy the smart contracts to a local Ethereum test network. The code solves concrete problems and works as intended.

5.2.1 Functioning Proof-of-Concept

Our POC successfully implements four important components:

1. An open, transparent global music copyright and licensing database that contain.

   – Split sheets for musical works

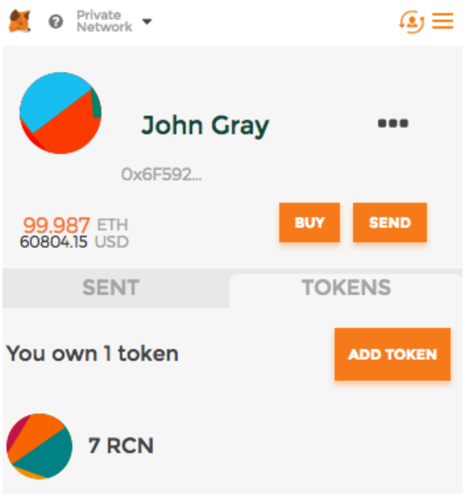
   – License contracts for musical works

   We believe that the open database may facilitate unified licensing standards that could be internationally accepted.

2. A correct working implementation of the standard ERC721 token system which enables copyrights to be represented by unique tradeable tokens. Figure 5.1 and 5.2 shows the Metamask wallet for the two users John Grey and Andreas Engebretsen (Lambendo). The Metamask wallet partly supports the ERC721 standard. It can show the number of RCN-tokens held by the user, but it does not show the unique ID of the tokens. Moreover, MetaMask currently does not support functions for sending RCN-tokens to other Ethereum wallets (users). .

3. Users can create and purchase license contracts for single musical works. Figure 5.3 shows a transaction representing a user buying a license. Notice that it carries 8 ETH (the price of the license) to the smart contract.

4. Musicians get real-time information about generated license royalties. These royalties can be withdrawn at any time. Figure 5.4 shows a transaction where a user requests the smart contracts to withdraw generated royalties. Figure 5.5 shows the user's balance after the withdrawal request. Notice how it has increased to 106.3 ETH from its initial default balance (100ETH).



John Gray

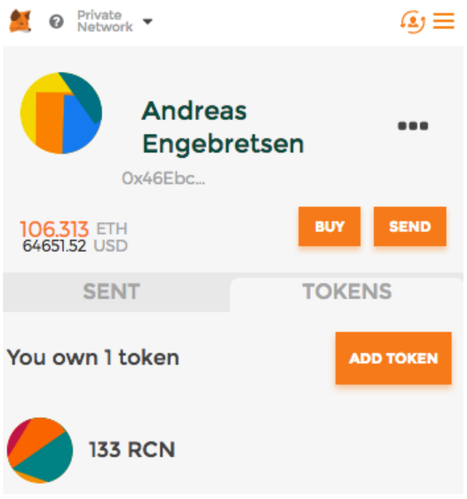
0x6F592...

99.987 ETH
60804.15 USD

You own 1 token

7 RCN

Figure 5.1: The MetaMask Ethereum wallet shows that John Grey holds 7 RCN-tokens that represent unique copyrights in various musical works



Andreas Engebretsen

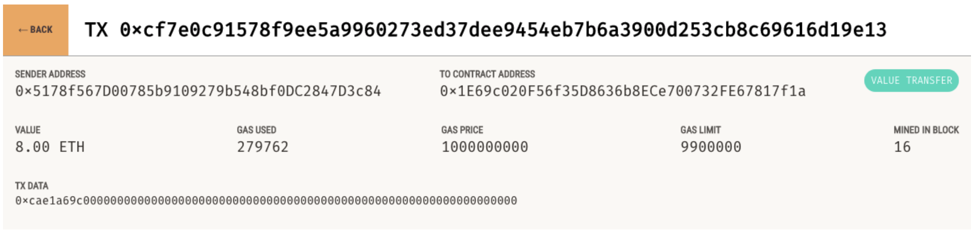
0x46Ebc...

106.313 ETH
64651.52 USD

You own 1 token

133 RCN

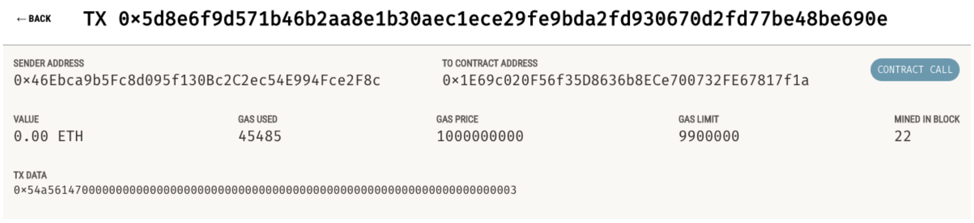
Figure 5.2: The MetaMask Ethereum wallet shows that Andreas holds 133 RCN-tokens that represent unique copyrights in various musical works



TX 0×cf7e0c91578f9ee5a9960273ed37dee9454eb7b6a3900d253cb8c69616d19e13

SENDER ADDRESS	0×5178f567D00785b9109279b548bf0DC2847D3c84
TO CONTRACT ADDRESS	0×1E69c020F56f35D8636b8ECe700732FE67817f1a
VALUE	8.00 ETH
GAS USED	279762
GAS PRICE	1000000000
GAS LIMIT	9900000
MINED IN BLOCK	16
TX DATA	0×cae1a69c0000000000000000000000000000000000000000000000000000000000000000

Figure 5.3: User with address 0x5178f567D00785b9109279b548bf0DC2847D3c84 buys a license. The transaction carries 8 ETH to the smart contract address 0x1E69c020F56f35D8636b8ECe700732FE67817f1a

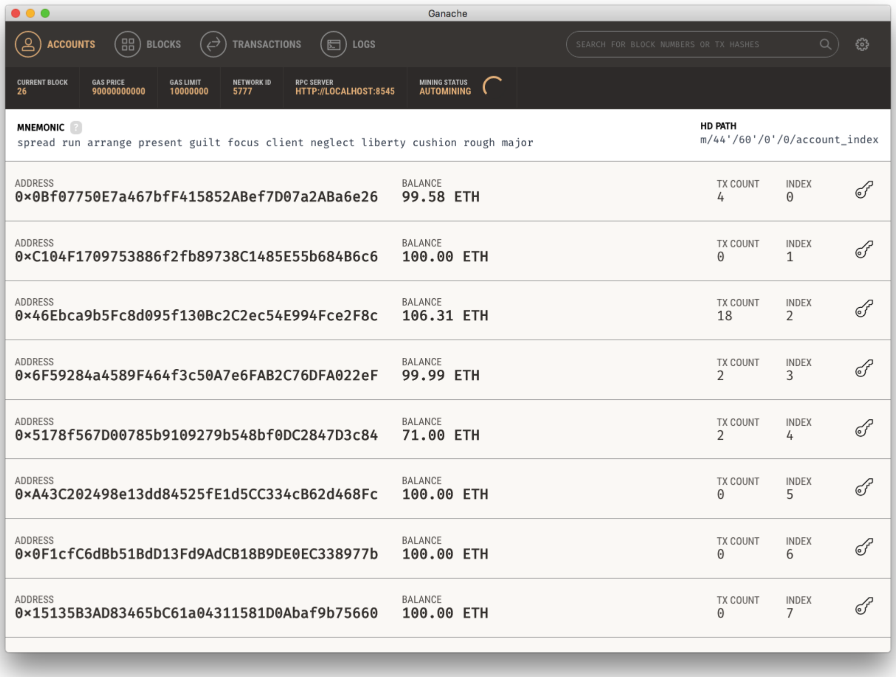

← BACK TX 0x5d8e6f9d571b46b2aa8e1b30aec1ece29fe9bda2fd930670d2fd77be48be690e

SENDER ADDRESS	TO CONTRACT ADDRESS	
0x46Ebca9b5Fc8d095f130Bc2C2ec54E994Fce2F8c	0x1E69c020F56f35D8636b8ECe700732FE67817f1a	CONTRACT CALL

VALUE	GAS USED	GAS PRICE	GAS LIMIT	MINED IN BLOCK
0.00 ETH	45485	1000000000	9900000	22

TX DATA
0x54a561470000000000000000000000000000000000000000000000000000000000000003

Figure 5.4: A transaction where user 0x46Ebca9b5Fc8d095f130Bc2C2ec54E994Fce2F8c sends a requests transaction to withdraw royalties for one of the user's work.



CURRENT BLOCK	GAS PRICE	GAS LIMIT	NETWORK ID	RPC SERVER	MINING STATUS
26	90000000000	10000000	5777	HTTP://LOCALHOST:8545	AUTOMINING

MNEMONIC
spread run arrange present guilt focus client neglect liberty cushion rough major

HD PATH
m/44'/60'/0'/0/account_index

ADDRESS	BALANCE	TX COUNT	INDEX
0x0Bf07750E7a467bfF415852ABef7D07a2ABa6e26	99.58 ETH	4	0
0xC104F1709753886f2fb89738C1485E55b684B6c6	100.00 ETH	0	1
0x46Ebca9b5Fc8d095f130Bc2C2ec54E994Fce2F8c	106.31 ETH	18	2
0x6F59284a4589F464f3c50A7e6FAB2C76DFA022eF	99.99 ETH	2	3
0x5178f567D00785b9109279b548bf0DC2847D3c84	71.00 ETH	2	4
0xA43C202498e13dd84525fE1d5CC334cB62d468Fc	100.00 ETH	0	5
0x0F1cfC6dBb51BdD13Fd9AdCB18B9DE0EC338977b	100.00 ETH	0	6
0x15135B3AD83465bC61a04311581D0Abaf9b75660	100.00 ETH	0	7

Figure 5.5: Eight of the test accounts on our local Ethereum network. Notice how the third user (with address 0x46Ebca9b5Fc8d095f130Bc2C2ec54E994Fce2F8c) has just withdrawn royalties from the work into his personal account, giving him a total balance of 106.31 ETH.

It is worth mentioning that there are problems and use cases within the music licensing space that our POC does not implement. Below are listed some unresolved problems:

- A concrete suggestion for an international standard defining how licensing should be regulated.
- Single license contracts that involve several musical works. Two examples that most often involve huge music catalogs are streaming contracts and public performance contracts.

5.2.2 Ethereum Community and Education

In addition to solving specific industry issues, our back-end code also serves as a contribution to the Ethereum dApp development community due to the project's open source nature. Our code and documentation can provide specific answers and ideas for other dApp projects in the Ethereum space. Furthermore, several educational institutions are becoming more interested in the potential of blockchain technology. It is not unthinkable that our implemented smart contracts (and concepts provided throughout our thesis) can be used for educational purposes. In fact, we encourage teachers, professors, and students to study the technicalities of our code and apply the same ideas to solve other problems in other industries.

5.3 Web Application

The third contribution of our master's project is the front-end web application. We developed a graphical user interface (GUI) for two reasons:

1. Showcase the systems architecture and functionality provided by the back-end smart contracts.
2. Make it easy for users to interact with the blockchain without dealing with a command line terminal

The web application is a platform where users can search a copyright database containing all registered works on the blockchain. Registered users can also register new works, create license profiles for those works, and buy licenses for different musical works. The rest of this section demonstrates the dApp's use cases with a description and a screenshot of the corresponding GUI.

Home

The home view is minimalistic and straightforward. It contains a search field where both registered and non-registered users of the platform can search the database to get specific details about musical works. The home view (as with every other view) also includes a menu button for users who have created an account and logged in. Clicking this button enables the user to navigate to different views.

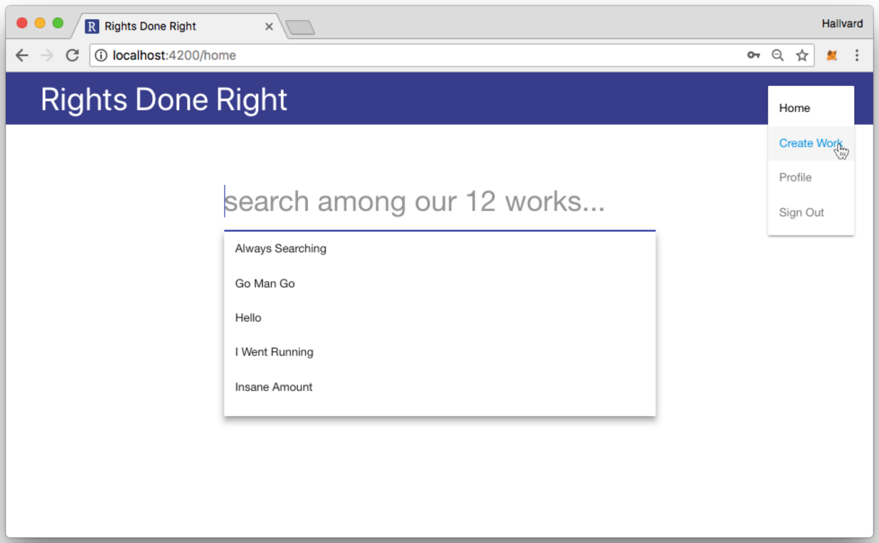
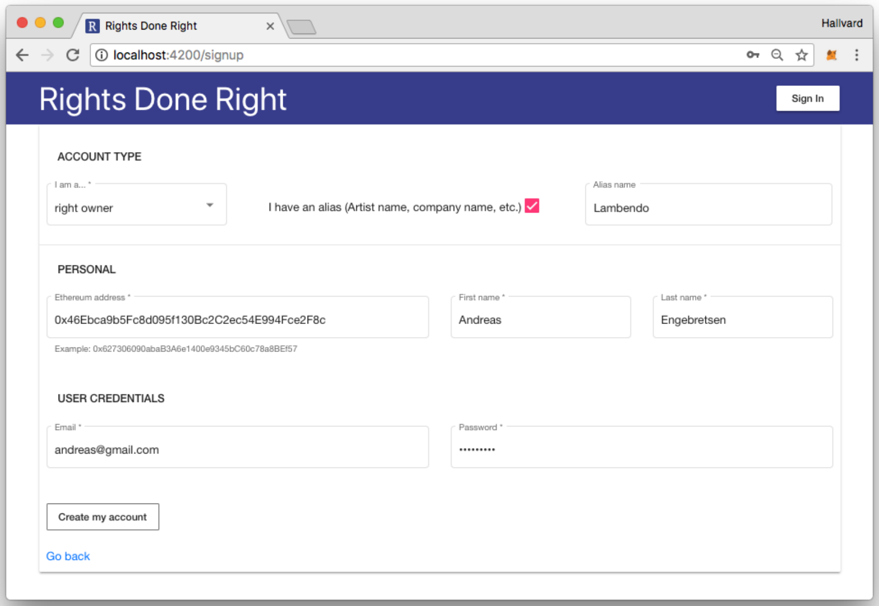


Figure 5.6: This is the home view. It includes a simple search bar and a profile menu button for logged in users.

Sign up

Users who want to register music to the database and license their work must sign up by creating an account. The same goes for users that wish to purchase licenses for the use of music. Creating an account is similar to any other user registration processes. It involves filling out a form with information such as name and Ethereum address. We also require a unique username (email address) and password for login and authentication purposes.



Rights Done Right

ACCOUNT TYPE

I am a... *: right owner

I have an alias (Artist name, company name, etc.) ✓

Alias name: Lambendo

PERSONAL

Ethereum address *: 0x46Ebca9b5Fc8d095f130Bc2C2ec54E994Fce2F8c

Example: 0x627306090abaB3A6e1400e9345bC60c78a8BEf57

First name *: Andreas

Last name *: Engebretsen

USER CREDENTIALS

Email *: andreas@gmail.com

Password *: •••••••••

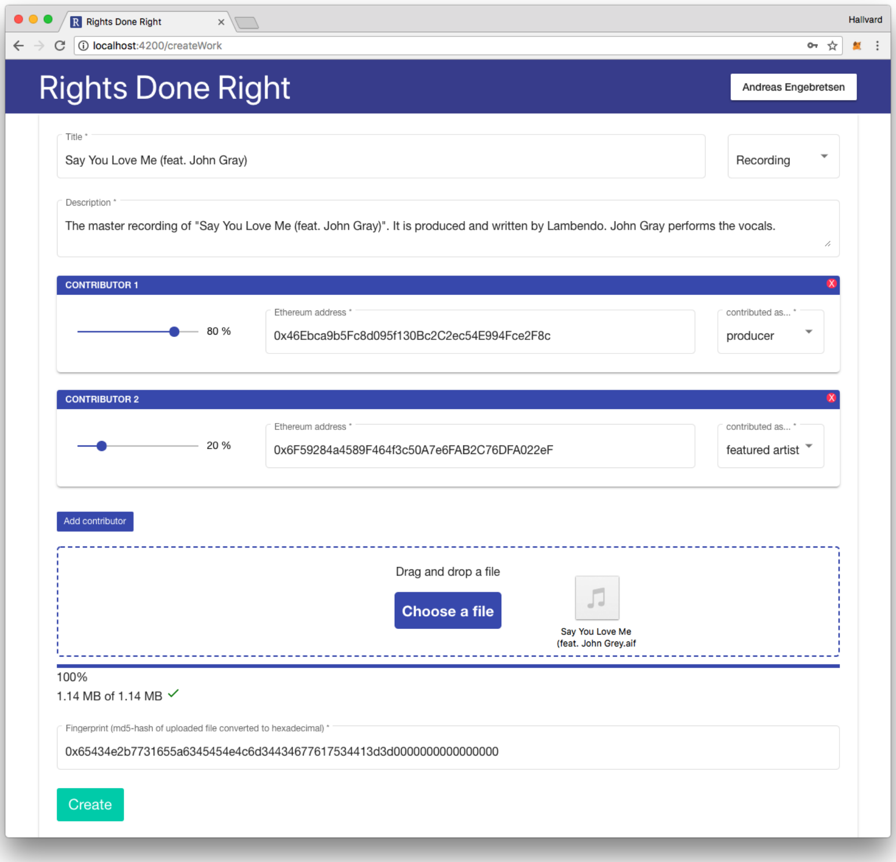
Create my account

Go back

Figure 5.7: This is the sign up form when creating an account on the platform. The picture shows the independent producer, Lambendo, signing up on the platform.

Create Work

This use case is available to every user who wants to register a musical work to the blockchain. As mentioned, some examples of musical works are recordings, lyrics, song compositions, notes, melodies, and full symphony arrangements. There are three pieces of information required to register a work: metadata (title, type of work, description), contributors and their share of the copyrights of the work, and the actual file embodying the work the user wants to copyright.



Rights Done Right

Title *
Say You Love Me (feat. John Gray)
Recording

Description *
The master recording of "Say You Love Me (feat. John Gray)". It is produced and written by Lambendo. John Gray performs the vocals.

CONTRIBUTOR 1
80 %
Ethereum address *
0x46Ebca9b5Fc8d095f130Bc2C2ec54E994Fce2F8c
contributed as... *
producer

CONTRIBUTOR 2
20 %
Ethereum address *
0x6F59284a4589F464f3c50A7e6FAB2C76DFA022eF
contributed as... *
featured artist

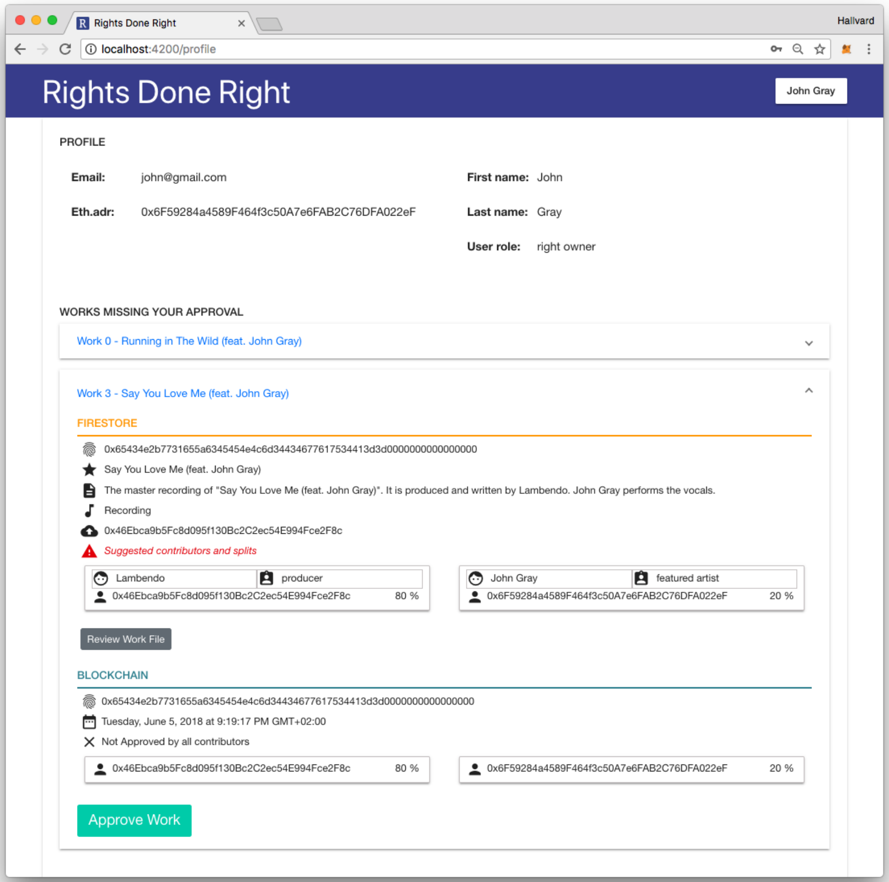
100%
1.14 MB of 1.14 MB ✓

Fingerprint (md5-hash of uploaded file converted to hexadecimal) *
0x65434e2b7731655a6345454e4c6d34434677617534413d3d0000000000000000

Figure 5.8: This is the user interface when registering a musical work. The picture shows Lambendo registering his track "Say You Love Me (feat. John Grey). The contributors split their royalties in the following manner: Lambendo will have 80% ownership and the vocalist, John Grey, gets 20%.

Approve work

When a work has successfully registered on the blockchain and in the database, a notification is sent to all contributors who were listed in the registration process. This notification is visible in the contributors' personal profile view. From here, users can review the uploaded file and verify that the data for the work is indeed correct. Approving a work implies that the user acknowledges the work information and accepts the suggested split sheet.



PROFILE

Email:	john@gmail.com	**First name:**	John
Eth.adr:	0x6F59284a4589F464f3c50A7e6FAB2C76DFA022eF	**Last name:**	Gray
		User role:	right owner

WORKS MISSING YOUR APPROVAL

Work 0 - Running in The Wild (feat. John Gray)

Work 3 - Say You Love Me (feat. John Gray)

FIRESTORE

0x65434e2b7731655a6345454e4c6d3443467761753441 3d3d0000000000000000

Say You Love Me (feat. John Gray)

The master recording of "Say You Love Me (feat. John Gray)". It is produced and written by Lambendo. John Gray performs the vocals.

Recording

0x46Ebca9b5Fc8d095f130Bc2C2ec54E994Fce2F8c

Suggested contributors and splits

Lambendo	producer	John Gray	featured artist
0x46Ebca9b5Fc8d095f130Bc2C2ec54E994Fce2F8c	80 %	0x6F59284a4589F464f3c50A7e6FAB2C76DFA022eF	20 %

BLOCKCHAIN

0x65434e2b7731655a6345454e4c6d3443467761753441 3d3d0000000000000000

Tuesday, June 5, 2018 at 9:19:17 PM GMT+02:00

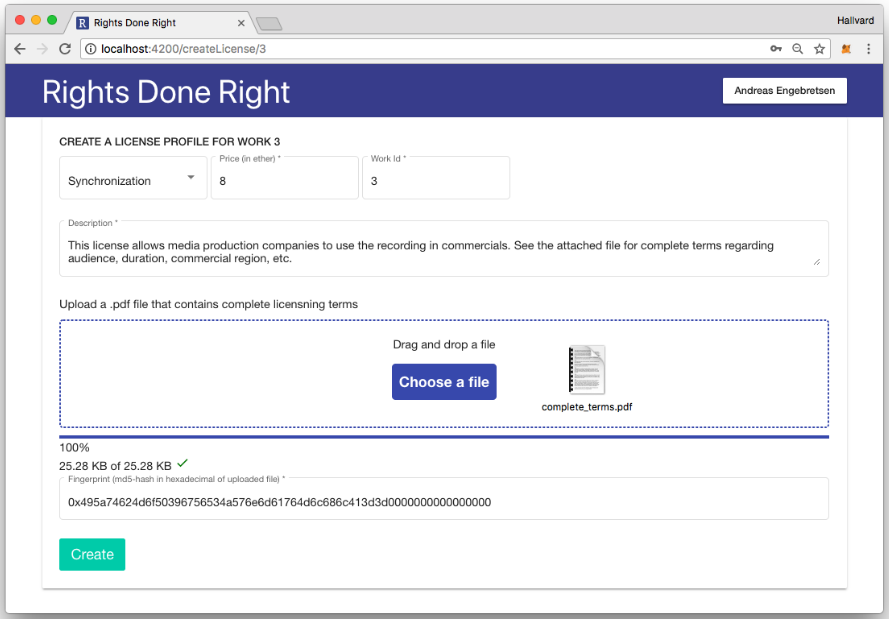
Not Approved by all contributors

0x46Ebca9b5Fc8d095f130Bc2C2ec54E994Fce2F8c	80 %	0x6F59284a4589F464f3c50A7e6FAB2C76DFA022eF	20 %

Figure 5.9: This is the user interface showing a contribution notification on John Grey's personal profile view. Both Lambendo and John Grey have to approve this information before the work becomes a valid entry in the database on the blockchain

Create License

When all involved contributors approve a work, it is ready to be licensed. There are many different types of licenses. A synchronization license allows for using the musical work in a video production. A public performance license allows for playing the song publicly (for instance in a restaurant or through radio). As mentioned earlier, the application makes it possible to create something we define as license profiles. A license profile contains different licensing data. It includes the following: metadata (the type of license, short description, The ID of the musical work to which the license applies), price (in ETH), and a file containing complete licensing terms. There are no requirements for how detailed these terms have to be. In some cases, it would be sufficient to specify for what region and for how long the license is valid. In more severe cases, these terms would be defined by high-level management and lawyers. This would be the case if a big record label would license music for a major film studio or a streaming service such as Spotify.



CREATE A LICENSE PROFILE FOR WORK 3

Synchronization

Price (in ether) *: 8

Work Id *: 3

Description *: This license allows media production companies to use the recording in commercials. See the attached file for complete terms regarding audience, duration, commercial region, etc.

Upload a .pdf file that contains complete licensning terms

100%

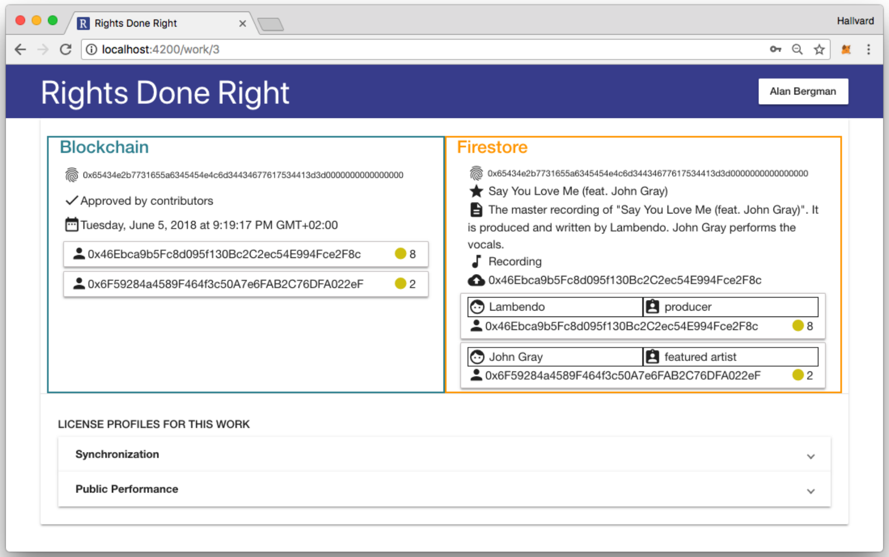
25.28 KB of 25.28 KB ✓

Fingerprint (md5-hash in hexadecimal of uploaded file) *: 0x495a74624d6f50396756534a576e6d61764d6c686c413d3d0000000000000000

Figure 5.10: This is the user interface when registering a license profile. The picture shows a synchronization profile allowing video production users to use the musical work (work ID 3) in a commercial. The attached file is a standard Master Use (MU) License for Advertising.

View Work

When a musical work is registered and approved by all involved contributors, it becomes a valid entry in the public database on the blockchain. Anyone can query this database and retrieve information about specific musical works. All data, including the split sheet between the contributors, is open and will be displayed on the application. As we've mentioned previously, our dApp stores data to two different back-end systems: the Ethereum blockchain and in a centralized database (Firebase). The boxes illustrate this by showing where the different data is stored.



Blockchain

0x65434e2b7731655a6345454e4c6d34434677617534413d3d0000000000000000

✓ Approved by contributors

Tuesday, June 5, 2018 at 9:19:17 PM GMT+02:00

0x46Ebca9b5Fc8d095f130Bc2C2ec54E994Fce2F8c ● 8

0x6F59284a4589F464f3c50A7e6FAB2C76DFA022eF ● 2

Firestore

0x65434e2b7731655a6345454e4c6d34434677617534413d3d0000000000000000

Say You Love Me (feat. John Gray)

The master recording of "Say You Love Me (feat. John Gray)". It is produced and written by Lambendo. John Gray performs the vocals.

Recording

0x46Ebca9b5Fc8d095f130Bc2C2ec54E994Fce2F8c

Lambendo	producer
0x46Ebca9b5Fc8d095f130Bc2C2ec54E994Fce2F8c	● 8

John Gray	featured artist
0x6F59284a4589F464f3c50A7e6FAB2C76DFA022eF	● 2

LICENSE PROFILES FOR THIS WORK

Synchronization

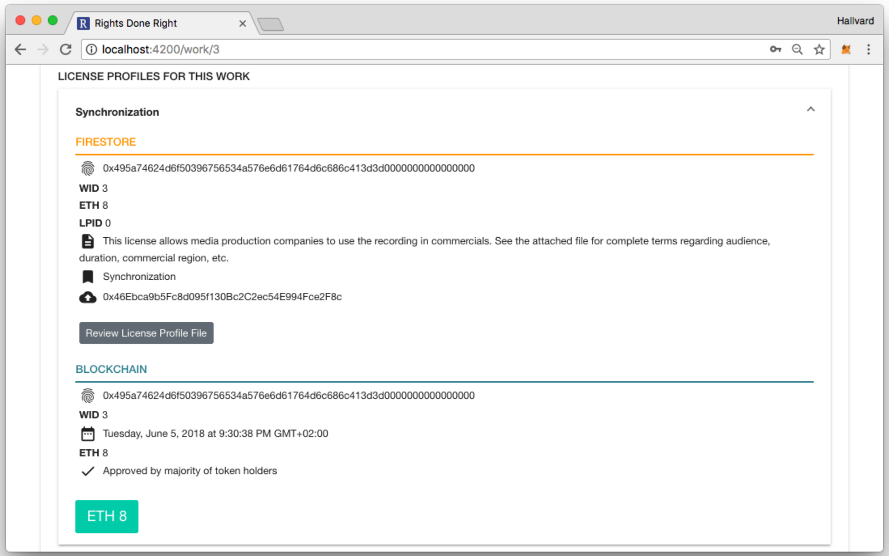
Public Performance

Figure 5.11: This is the user interface when viewing a musical work. The picture shows the information for "Suit 'n Tie (Lambendo Remix)". Notice how both boxes include an identical digital fingerprint as their first field. When these fingerprints are equal, users can be confident that the displayed information is accurate. A mismatch between the displayed fingerprints signals that the centralized storage (Firebase Storage) has been compromised.

Purchase License

Registered users have the availability to purchase licenses for different license profiles. One example: a restaurant that wants to use music on their premises. The application allows users to search and license the music they need in a matter of seconds.

In the previous view, we saw the user interface when viewing data for a musical work. Below the information boxes, there is a list containing the different license profiles that have been created for that particular musical work. Clicking on one of these will cause it to expand showing more information about each license profile. From here, it is also possible to purchase a license.



LICENSE PROFILES FOR THIS WORK

Synchronization

FIRESTORE

0x495a74624d6f50396756534a576e6d61764d6c686c413d3d0000000000000000

WID 3

ETH 8

LPID 0

This license allows media production companies to use the recording in commercials. See the attached file for complete terms regarding audience, duration, commercial region, etc.

Synchronization

0x46Ebca9b5Fc8d095f130Bc2C2ec54E994Fce2F8c

BLOCKCHAIN

0x495a74624d6f50396756534a576e6d61764d6c686c413d3d0000000000000000

WID 3

Tuesday, June 5, 2018 at 9:30:38 PM GMT+02:00

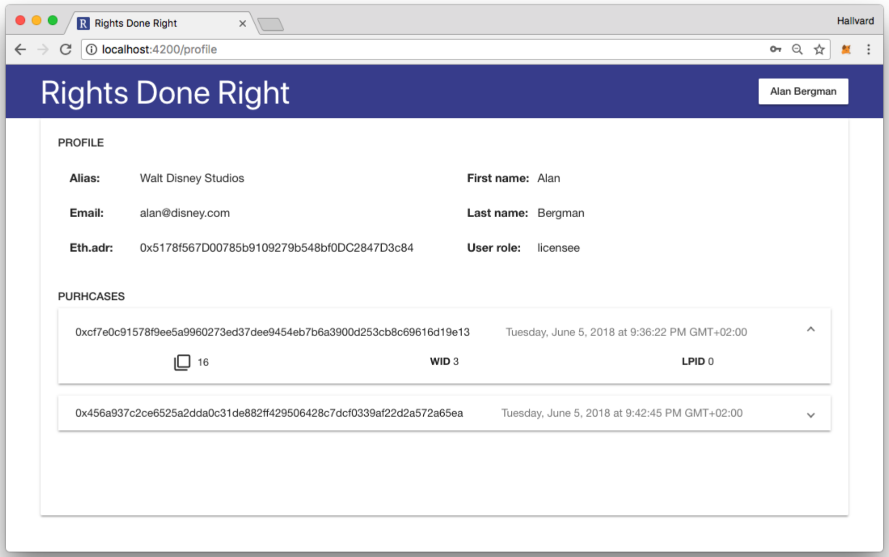
ETH 8

✓ Approved by majority of token holders

Figure 5.12: When clicking on one of the license profiles below the information boxes, the user can view more information about it. The image shows the user interface for the synchronization profile associated with the song "Suit 'n Tie (Lambendo Remix)" shown on the previous view. Purchasing the license is as simple as clicking the "ETH 8" button at the bottom of the view and confirm the transaction in the MetaMask wallet.

Purchase Receipts

Licensing music traditionally involves signing a document (manually or digitally) that serves as a receipt that confirms the agreement. Our application provides a somewhat different proof. Buying a license through our application involves sending a transaction to the blockchain. This transaction carries the money and some other necessary data required to process the purchase. When the transaction is finished (when the block containing the transaction is successfully mined), the user receives a purchase notification which is available on the purchaser's personal profile view. The receipt includes the following information: The unique hash of the transaction that carried out the purchase, the block number and timestamp for when the transaction was confirmed on the blockchain, and the ID of the musical work for which the license applies. This information is sufficient to prove a valid license purchase. Using services such as etherscan.io[1], it is easy to check and verify the transaction that carried out the license purchase.



PROFILE

Alias:	Walt Disney Studios	**First name:**	Alan
Email:	alan@disney.com	**Last name:**	Bergman
Eth.adr:	0x5178f567D00785b9109279b548bf0DC2847D3c84	**User role:**	licensee

PURHCASES

0xcf7e0c91578f9ee5a9960273ed37dee9454eb7b6a3900d253cb8c69616d19e13 Tuesday, June 5, 2018 at 9:36:22 PM GMT+02:00

16 **WID** 3 **LPID** 0

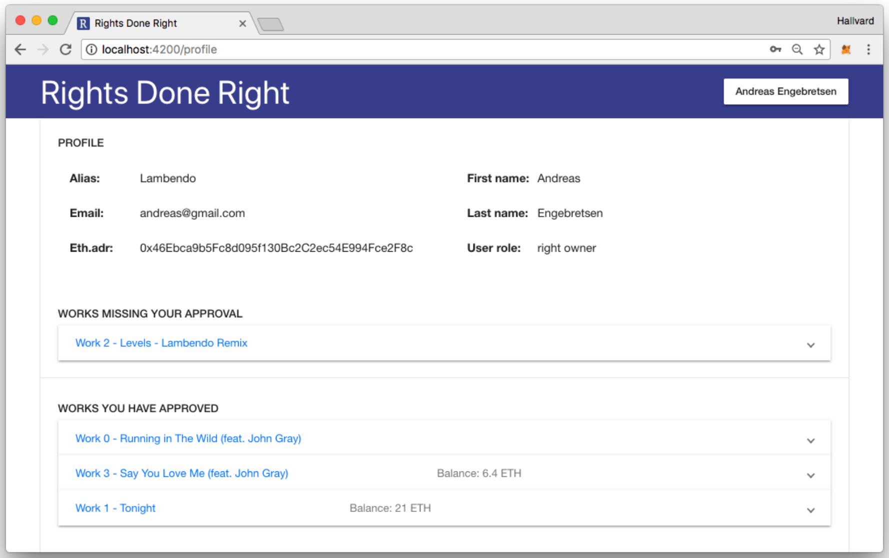
0x456a937c2ce6525a2dda0c31de882ff429506428c7dcf0339af22d2a572a65ea Tuesday, June 5, 2018 at 9:42:45 PM GMT+02:00

Figure 5.13: this is the user interface for viewing purchase receipts. The picture shows Walt Disney Studio's profile. They have currently licensed two musical works for use in their movies. Clicking on a receipt will cause it to expand and reveal more information about the purchase.

[1]https://etherscan.io/

Generated Royalties

The personal profile view keeps a list of all the works in which the user has complete or partial ownership. When users license any of those works through the application, the balance for that particular work will update in real time. Automatic updates to the balance enable musicians to keep track of how much money their work is generating.



PROFILE

Alias: Lambendo
First name: Andreas
Email: andreas@gmail.com
Last name: Engebretsen
Eth.adr: 0x46Ebca9b5Fc8d095f130Bc2C2ec54E994Fce2F8c
User role: right owner

WORKS MISSING YOUR APPROVAL

Work 2 - Levels - Lambendo Remix

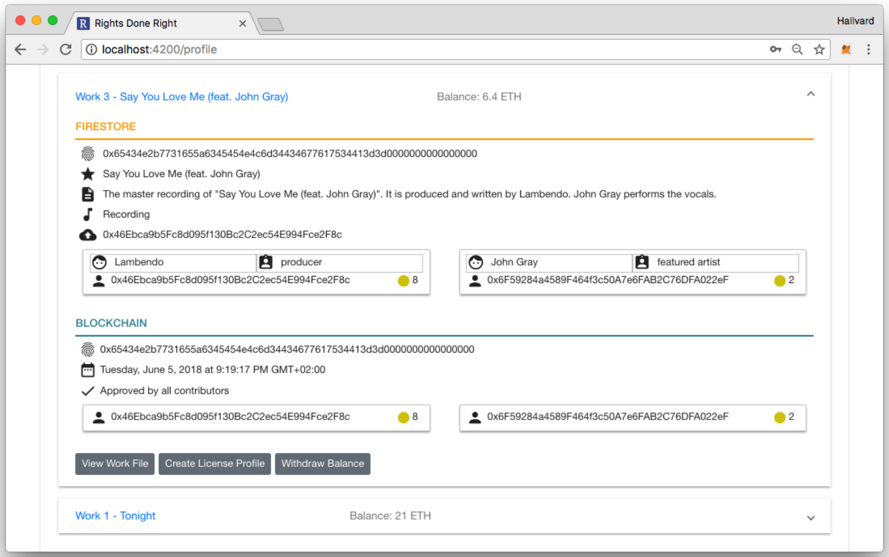
WORKS YOU HAVE APPROVED

Work 0 - Running in The Wild (feat. John Gray)
Work 3 - Say You Love Me (feat. John Gray) Balance: 6.4 ETH
Work 1 - Tonight Balance: 21 ETH

Figure 5.14: The user interface that shows how users can keep track of their royalties in real-time. The picture shows Lambendo's personal profile view containing a list of all works that he has ownership in/of. The list keeps a real-time sum of how much royalties each of the tracks have generated. The track "Suit 'n Tie (Lambendo Remix)" is completely owned by Lambendo. It has generated 13 ETH of royalties which means that it has been licensed for that exact amount through our application.

Withdraw Royalties

The previous view illustrated how a user can keep track of financial data for each of the works they have ownership of. As mentioned, one of the application's smart contracts holds all generated royalties until copyright owners withdraw it. By clicking on any given work on their personal profile view, users can view more information about it and withdraw any royalties to their personal Ethereum wallet.



Work 3 - Say You Love Me (feat. John Gray) Balance: 6.4 ETH

FIRESTORE

0x65434e2b7731655a6345454e4c6d34434677617534413d3d0000000000000000
Say You Love Me (feat. John Gray)
The master recording of "Say You Love Me (feat. John Gray)". It is produced and written by Lambendo. John Gray performs the vocals.
Recording
0x46Ebca9b5Fc8d095f130Bc2C2ec54E994Fce2F8c

Lambendo | producer
0x46Ebca9b5Fc8d095f130Bc2C2ec54E994Fce2F8c ● 8

John Gray | featured artist
0x6F59284a4589F464f3c50A7e6FAB2C76DFA022eF ● 2

BLOCKCHAIN

0x65434e2b7731655a6345454e4c6d34434677617534413d3d0000000000000000
Tuesday, June 5, 2018 at 9:19:17 PM GMT+02:00
✓ Approved by all contributors

0x46Ebca9b5Fc8d095f130Bc2C2ec54E994Fce2F8c ● 8
0x6F59284a4589F464f3c50A7e6FAB2C76DFA022eF ● 2

Work 1 - Tonight Balance: 21 ETH

Figure 5.15: This is the user interface found on the personal profile view showing information for musical works the user has ownership of. The bottom right "Withdraw Balance"-button enables a user to instantly send the sum of generated royalties for that work to its private Ethereum wallet.

Chapter 6
Evaluation and Discussion

In this chapter, we evaluate the results and discuss relevant topics that we encountered during our research and development. The following discussions cover themes that apply to most blockchain-based projects. We clarify those themes and analyze how they relate to our decentralized application and the music industry in general.

6.1 Challenges with Blockchain Technology

Blockchain technology has existed for around ten years, but today's most advanced blockchain platforms still face numerous critical limitations. Even though we have not focused on these issues throughout our thesis, we want to discuss them and point out that they restrain adoption. In this section, we will present some of the main challenges for the Ethereum blockchain and what implications they have on our dApp.

6.1.1 Scalability

Ethereum's scalability issue is their primary challenge. Currently, the network can process 15-20 transactions per second. Compared with centralized e-commerce systems, that number is significantly low. Scaling the platform is the number one priority for the Ethereum foundation. We are not going to discuss the different measures they currently work on to increase the transaction throughput. The scaling issues affect all deployed dApps on the platform. The consequence is non-responsive applications which are generally not acceptable for most users. In our case, basic use cases, such as to register a work, withdraw royalties, or purchase a license, would take undesirably long time whenever the Ethereum network would experience congestion.

6.1.2 Block Time

The average block time on the Ethereum network is 15 seconds. Thus, users of any Ethereum dApp would have to wait on average 15 seconds every time they send a transaction. Without going into details, this limitation is somewhat related to the previous one. Compared to today's responsive applications, a 15 second waiting time is substantially higher than what most users accept. Despite this limitation, we believe that the advantages that we have pointed out throughout the thesis can make the lack of responsiveness worth it.

6.1.3 Transaction Fees

Another obvious drawback is the cost of communicating with Ethereum dApps. In a centralized world, companies pay hosting companies to rent servers and computing resources. These costs are likely priced into the products and services offered. However, those costs are usually hidden from the consumers as they do not pay directly every time they use an application. On the Ethereum blockchain, computational power and data storage come at a price. Users have to pay this price directly in the form of gas to miners that maintain the Ethereum network. There are two elements to this discussion:

1. The price the user has to pay for a transaction. The fee is determined by two factors: number of required gas-units and the cost of each unit. The gas cost fluctuates depending on multiple factors such as the network traffic and ETH price.

2. The principle of paying for transactions. Most users are not used to continuously pay micro amounts when interacting with applications on the internet.

We did some calculations to figure out how many USD it would cost to register a work. We learned that the fee for registration of one work would be between \$0.10 and \$0.60, depending on the gas price. For an independent right holder, this amount may not be too high, as the copyright registration is a one time process. However, for a major label with millions of works in their catalog, the cost is significantly higher. We recognize the issue. Whether or not users will accept it needs to be further investigated.

6.1.4 Programming Language

We also want to point out that developing on blockchain platforms is a challenge when compared to other more established platforms such as Java or .NET. We were presented with several problems when working with Solidity (the programming language for the Ethereum blockchain). Although its community and documentation are the biggest within the blockchain space, the language is characterized by several shortcomings compared to more mature languages. While developing the smart contracts we encountered many technical limitations. These are a few examples:

1. No built-in string functions.
2. Functions cannot have nested arrays as inputs.
3. Functions cannot return dynamic arrays.
4. Custom types (structs) cannot contain attributes of the mapping type.

The limitations of the language made development trickier than expected. They forced us to create front-end JavaScript functions to convert user input strings to other compatible Solidity types. That was particularly a problem when dealing with MD5-hashes for uploaded files. We had to convert the hash strings to byte-arrays before the back-end Solidity functions would process the data.

6.2 Is the Music Industry Ready for Blockchain?

Although our POC demonstrates that blockchain can bring clear value propositions to musicians, we would like to question if major industry organizations are ready to change their traditional corporation. Generally speaking, we have yet to witness any industry adopt these types of trustless decentralized applications. To continue this debate, we would like to discuss whether or not there are enough incentives for the industry as a whole to adopt blockchain technology.

When evaluating the results of the master project, our dApp indeed fulfills its goal of demonstrating the potential of blockchain based solutions. We illustrate how smart contracts can increase industry transparency and resolve the slow royalty payment system. Moreover, we show how copyrights can be stored in a transparent distributed database instead of in closed company systems. For those reasons, we argue that decentralized applications have the potential to disrupt the music industry. However, it seems like the music industry is not ready for blockchain technology. We do not have clear answers to whether blockchain solutions can co-exist with current contractual and copyright systems. Adoption by industry players depends on their ability to identify clear value propositions offered by innovations such as Ethereum dApps. Our thesis mainly points out how musicians would benefit from blockchain based solutions. However, it is not as easy to imagine that big record labels and royalty collecting societies would be motivated to transform their current business operation radically. To be more specific, it is unlikely that powerful players would welcome a licensing platform that is beyond their own control.

Blockchain technology is not going to turn the music industry upside down any time soon. We predict a slow transition with incremental changes that eventually might result in a drastic industry shift. Ultimately, we imagine that collaborations between high-profile artists, songwriters, and blockchain start-ups will have a significant influence on how the industry transforms the next 5-10 years.

6.3 Necessity of Blockchain

The third discussion we would like to address is whether or not we need blockchain technology to solve the issues in the music industry. Even though our dApp indeed provides solutions to current issues, it is interesting to discuss why some of them not yet have been solved using centralized solutions. Let us look at the two problem categories we defined in the background analysis (from Chapter 2.2):

1. The hierarchical structure of players and the relationship between them
2. Lack of information and standards

Blockchain technology definitely seems to provide solutions to problems within the first category. In fact, we argue that blockchain innovations will be the driving force that ultimately solves the issues. We cannot see how the music space would become more transparent and fair as long as a small group of dominant companies continues to control the industry.

That said, we argue that it is indeed possible to create centralized solutions to problems within the second category. There are no technological limitations for creating a universal registry for copyrights and adapting to international licensing standards. However, several attempts to solve these issues have already been made. One example is the EU commissioned Global Repertoire Database (GRD) as mentioned in the Background chapter. This project failed - not because of technological limitations, but due to major players fearing for their positions and their operational costs. Perhaps blockchain technology will be the breaking point forcing the industry to change.

6.4 Blockchain as an Economic Transaction System

Ethereum and other blockchains use either cryptocurrencies or tokens when transferring value across the network. In some aspects, one can argue that these transaction systems deliver well on their promises as values are transferred faster than traditional financial systems and in a correct manner. However, the fiat value of these cryptocurrencies and tokens is very volatile. Building financial systems based on highly volatile building blocks, such as ETH, makes little sense. Today's modern economics exclusively revolve around government regulated fiat currencies that everyone ultimately relies on in their daily life. Thus, we argue that blockchains at the moment do not provide a sound basis for financial transaction systems. However, one might argue that the currency's volatility will drop if its popularity and adoption grows.

The volatility in the discussion has profound consequences for our dApp. With our current implementation, every license profile is set to cost a specific amount of ETH. This value is static as it is stored on the blockchain. The consequence is that the fiat price for a license contract will change significantly on a daily basis. One possible solution to this problems is to save license prices in US dollars instead of in ETH. However, this would require an "Ethereum Oracle" to calculate the corresponding ETH value every time a work is licensed. Oracles are trusted third-party data providers that feed smart contracts with necessary data. That said, the Ethereum community has not agreed whether or not the presence of third-party oracles weakens blockchain's core decentralization objectives. In the future, we need to research other pricing strategies that might work better while the cryptocurrencies are volatile.

6.5 Immutability

The blockchain immutability property provides both advantages and disadvantages regarding data on the blockchain. On the one hand, it provides data integrity as no stored records on the blockchain can be altered. This property is fundamental in most applications, including our dApp. One example is the fact that contributors of a musical work can be confident that their shares of ownership will not change once the work is registered to the blockchain. On the other hand, immutability limits the possibilities of changing data such as information about registered works or licensing terms. One can argue that these attributes should not be static as musicians may want to change specific parameters in the future.

Another issue regarding immutability is the challenge of patching vulnerabilities to a deployed contract. The inherent nature of blockchain limits the possibilities to easy update smart contracts and fix any validation or security-related bugs. From a developer's standpoint, not being able to patch vulnerabilities after deployment is perceived as a significant challenge. The contracts must be deployed with the utmost level of confidence that every edge case, loophole, and validation is handled. There is a "workaround" for this challenge. It involves "killing" the current contract and deploy a new updated one. Some developers in the community do not consider this a viable option as it may freeze considerable amounts of ETH held by the old contract. Furthermore, re-deploying a smart contract would weaken a blockchain company's reputation considerably.

6.6 Centralized vs. Distributed Storage

When developing a decentralized application, data storage becomes an issue in almost all cases due to blockchain's storage costs. Developers have to chose between two different storage systems: centralized data centers or distributed peer-to-peer systems. This section will discuss advantages and disadvantages for each of them.

Centralized storage is used in most applications today. We call it centralized because the data centers that stores the information are controlled and maintained by a third party. Centralized systems are popular because of their easy configurability and integration with any application. Today, giant tech companies such as Facebook, Google, Apple, Amazon and Microsoft control a huge share of all data on the internet. Their centers are targets for hackers as they are filled with huge amounts of sensitive information. Most consumers use services like Facebook and Google, and they store data on these platforms. However, history has shown that the giant technology companies cannot always protect themselves. Apple's iCloud system was hacked in 2014 [33], and Dropbox was hacked in 2012 [34]. Centralized data centers are single point of failure, and there is always a change of data theft. Security challenges is not the only concern in the discussion. Users ultimately have to trust the third party's intentions. Some users worry about their data being provided to governments, law enforcement agencies, or advertising agencies.

Distributed peer-to-peer (P2P) systems is an alternative to centralized storage systems. P2P storage systems spread data across a network a nodes, making them more secure, possibly faster, cheaper and censorship resistant. Distributed storage systems leverage the enormous amounts of unused storage space located on user's hard drive around the world. There are many distributed peer-to-peer storage systems operating today. Some of them address the issue of expensive storage on blockchains. Swarm[1] and Storj[2] are two work-in-progress cloud storage services built into the Ethereum web3 protocol. Key people in the Ethereum space have stated their interests in said decentralized storage systems. Vitalik Buterin explained that "distributed file storage systems like Storj have the potential to eliminate high markup costs and market inefficiencies and provide a much higher level of privacy, reliability and quality of service than we see today."[3].

Etheruem based storage systems are still in beta stage or not available at all. For that reason, we utilize Google's Firebase service to store data in our dApp. The data we store include:

[1]https://github.com/ethersphere/swarm - [Accessed: 15-03-2018]
[2]https://storj.io// - [Accessed: 25-05-2018]
[3]https://www.bitrates.com/coin/STORJ/overview/USD - [Accessed: 06-06-2018]

- Metadata for registered musical works (title, type, description)
- Metadata for registered license contracts (type, description)
- The file of all registered musical works and complete license contract terms

We recognize that this centralized solution contradicts our goals in terms of being a trustless application. If someone would be able to compromise Google's data centers, the hacker could easily change data within our app. This would have profound consequences as users could potentially purchase a "false license". Another trust issue with our current architecture is the fact that we rely on Google for retrieving the hash representation for files that we store on Ethereum. Since Firebase performs the MD5-hash algorithm and sends the results back to the application, we have to trust that the calculation is done correctly, and that the results accurately represents the file. However, the algorithm behind the hash output is a global standard and could be cross-checked manually. For the reasons explained in this paragraph, a small part of our solution is more centralized than we intended it to be. Yet, we consider the application to be overall decentralized, even though the current solution is not optimal.

6.7 Web Application

Our third contribution, the implemented web application, serves two important roles in our research. It contains a graphical user interface that is designed to be both educational and intuitive.

1. The web application is characterized by visual educational elements that illustrate the back-end functions executed by the deployed smart contract. These functions are triggered by different front-end buttons located throughout the web application. Additionally, the user interface also illustrate the overall physical architecture of the application. This is done by displaying where the different properties of a work are stored. For a regular user of the platform, clarification of the system architecture and a visual aid to back-end logic might not be of particular interest. However, we believe that educational institutions and other developers will benefit from a user interface that reflects technicalities that take place behind the scenes.
2. Be that as it may, we do focus on the application's intuitiveness. Despite the fact that our dApp serves only as a POC, the platform still has many user-friendly features such as a user system, search features, intuitive routing/navigation, drag-and-drop file upload, and a primitive push notification system. These are important features because they facilitate easy interaction with the Ethereum blockchain without having to deal with a command line interface.

If we were to focus solely on the user experience, some of the smart contract functionality would naturally be hidden from the interface. In a future stage of development, we should move our focus to strengthening the user-friendliness, and abstract the most complicated blockchain technicalities away from the users.

Forthcoming efforts should also investigate different pricing models for license profiles. One improvement would be to let registered users request specific licenses with suggested terms. Then based on the terms and a negotiated price, the right holders may or may not approve the license. We believe that this will increase the right holders' revenues and ease obtainment of licenses.

Chapter 7
Conclusion

In this chapter, we present our conclusion by elaborating on the objectives and answering the research question. Additionally, we summarize the most critical limitations that we encountered during our investigation and suggest how to overcome these limitations in future work.

Our research question is the following:

RQ: How can blockchain technology be utilized in order to solve problems related to transparency, efficiency, and fairness along the music industry value chain?

We conclude that blockchain technology can be utilized in multiple ways to meet specific problems in today's industry. Our research found problems related to transparency, efficiency, standards, and inaccessible copyright information. One way to utilize blockchain technology has been demonstrated through the development of our decentralized application built with these issues in mind. The application proves that properties of blockchain can be applied to real-world challenges. Ethereum, with its self-enforcing smart contracts, enables automated transactions tailored for specific scenarios.

Based on the issues uncovered in our background analysis, our application offers a number of improvements. It streamlines the money flow by putting the right holders in control of their own royalty payouts. Furthermore, it exposes transactions and split sheets for different works to the public, making the payment process more transparent. The application also collects copyright information to a public registry stored on the Ethereum blockchain. By using blockchain as the underlying infrastructure, we establish trust without the need for a third party.

In addition to answering the RQ in a positive way, our project revealed multiple disadvantages that makes blockchain technology less attractive for various industries.

As mentioned in Chapter 6, the technology has yet to achieve satisfactory results with regards to scalability and processing time. Moreover, most blockchains are not free to engage with nor built for data storage. The available development tools are relatively new and primitive. These disadvantages hinder dApp development.

Our application works well when testing it on a local host. However, it is difficult to conclude whether or not the results would be equally satisfactory in a real-world scenario. That would require dealing with real copyright material and licensing agreements on the main Ethereum network. As we do not have contacts in the music industry, neither artists nor music organizations have tested the application at the time of publishing this paper.

7.1 Future Work

When evaluating our decentralized application and its functionality, we identify several areas of improvements. In this section, we summarize the most critical aspects of our solution that should be investigated further in the future.

7.1.1 Legal Issues

There are several legal issues regarding our dApp that we do not discuss in our thesis. We are not confident that license contracts signed by two parties over the Ethereum blockchain are considered valid from a legal perspective. Moreover, to complicate the matter further, countries have different copyright laws and regulations that needs to be accounted for. Doing business through self-enforcing, autonomous applications controlled only by its own code is a new concept that brings new legal questions without clear answers. This thesis did not focus on legal issues regarding this topic. Future work should investigate the intersection between blockchain and law.

7.1.2 Higher Degree of Decentralization

Data storage is a serious limitation for all Ethereum based applications. We recognized early in our development phase of this project that Ethereum is not a good platform for storing much data. The current system architecture involves a centralized third party, Google's Firebase service, for data storage. Our architecture increases the degree of centralization as users have to trust Google's infrastructure and security. Future work should research the potential of migrating to a complete decentralized architecture. A migration would involve changing our current data storage platform to a decentralized storage provider such as Storj or Swarm. As mentioned in the discussion chapter, these are work-in-progress distributed cloud storage systems that integrate well with Ethereum dApps. Changing to decentralized storage would be an enhancement to our contribution, as it will strengthen our core objectives regarding trust.

7.1.3 Decentralied App Governance

Governance is another critical limitation in our suggested solution. The dApp implementation currently allows anyone to upload files and claim ownership for a particular work. There is no third party that governs the copyright registration process. A key point about dApps is that they should not be governed by a third party. We were not able to come up with a solution to this problem. Future work should investigate the possibilities for how users can prove that they only register musical works that they actually own.

7.1.4 Code Enhancements

Writing code is a continuous process that does not terminate once an application works. It is likely that cleaner and more robust code can be written for our smart contracts. We define two areas of improvement.

Optimization

The quality of Solidity code is determined by how gas-efficient the code is. We have limited the data input to the different contract functions to keep the data processing (gas usage) to a minimum. Fewer number of computations lead to less gas usage. However, there are certain operations in our contracts that require substantial amount of gas. One example is the function that issues tokens to the right holders once every right holder has approved the work. This high gas usage is due to a double for-loop which causes many operations; one for each token issued. The more granular the shares become, the more tokens will be produced, and gas usage will increase. The gas used by a transaction is paid by the initiator who is a user of our platform. To keep users costs low, future development should strive for code optimization throughout the deployed contracts.

Validation and Security

Due to blockchain's immutability property, it is of the utmost importance that our contract functions only for the purposes intended. Validation and security should be taken into account while developing contracts to ensure safe function calls. In our application, it is essential that our users only have access to their balance and their related work. Under no circumstance should other users be able to withdraw balances for other users nor exchange rights on their behalf. In our solution, some validation measures have been implemented. For instance, it is only the message sender's (the transaction initiator's) related work balances that is available for withdrawal. Be that as it may, the deployed contract may be not completely secure as our dApp is only intended to be a POC. Future work should identify security issues and update the code with stronger security and validation.

References

[1] J. Naughton, “Is blockchain the most important IT invention of our age?,” *The Guardian*, 2016. <www.theguardian.com> [online article accessed: 2018-05-01].

[2] S. Nakamoto, “Bitcoin: A peer-to-peer electronic cash system,” 2008. White Paper, <www.bitcoin.org/bitcoin.pdf> [accessed: 2018-04-01].

[3] R. Beck, J. S. Czepluch, N. Lollike, and S. Malone, “Blockchain - the gateway to trust-free cryptographic transactions,” in *24th European Conference on Information Systems, ECIS 2016, Istanbul, Turkey, June 12-15, 2016*, p. Research Paper 153, 2016.

[4] M. O’Dair, Z. Beaven, D. Neilson, R. Osborne, and P. Pacifico, “Music on the blockchain,” *Technical report 1 - Blockchain For Creative Industries Research Cluster*, 2016. Middlesex University.

[5] R. Music, “Fair music: Transparency and payment flows in the music industry,” *Recommendations to increase transparency, reduce friction, and promote fairness in the music industry.*, 2015. Berklee Institute for Creative Entrepreneurship.

[6] L. Marshall, “‘let’s keep music special. f—spotify’: on-demand streaming and the controversy over artist royalties,” *Creative Industries Journal*, vol. 8, no. 2, pp. 177–189, 2015. Taylor & Francis.

[7] B. Plumb, “The songwriter and music publisher relationship: Part iv,” *Royalty Exchange*, 2018. <www.royaltyexchange.com> [blog post accessed 2018-03-14].

[8] H. McDonald, “Music royalties explained,” *The Balance Careers*, 2017. <www.thebalancecareers.com/big-three-record-labels-2460743> [accessed: 2018-04-22].

[9] F. Staff, “Public performance rights for sound recordings,” *Future of Music Coalition*, 2018. <www.futureofmusic.org> [online article accessed: 2018-04-01].

[10] K. Thomson, “Music and how money flows,” *Future of Music Coalition*, 2015. <www.futureofmusic.org> [online article accessed: 2018-03-21].

[11] J. P. Friedlander, “News and notes on 2017 riaa revenue statistics,” *Strategic Data Analysis, Recording Industry Association of America (RIAA)*, 2017. <www.riaa.com> [accessed: 2018-04-19].

[12] M. Foreman, “Music royalties explained,” *The Pro Audio Files*, 2018. <www.theproaudiofiles.com/music-royalties-explained/> [accessed: 2018-04-22].

[13] M. Jerräng, “Så fick spotify skivbolagen med sig,” *International Data Group, Computer Sweden*, 2009. <www.computersweden.idg.se> [online article accessed: 2018-03-27].

[14] I. Lunden, “In europe, spotify royalties overtake itunes earnings by 13%,” *TechCrunch*, 2014. <www.techcrunch.com> [online article accessed: 2018-03-21.

[15] J. Muikku and P. Durgam, “Pro rata and user centric distribution models: A comparative study,” *Digital Media Finland*, 2017.

[16] M. Singleton, “This was Sony music’s contract with Spotify,” *The Verge*, 2015. <www.theverge.com> [online article accessed: 2018-03-21].

[17] P. Kafka, “Spotify’s new deal with universal music group means some albums will go behind a paywall,” *Recode*, 2017. <www.recode.net> [online article accessed: 2018-03-21].

[18] K. Milosic, “Grd’s failure,” *The Music Business Journal*, 8 2015. Berklee College of Music, <www.thembj.org> [online article accessed: 2018-04-05].

[19] A. Nicolaou, “Songwriters’ court victory deals a blow to spotify,” *Financial Times*, 2018. <www.ft.com> [online article accessed: 2018-05-02].

[20] P. Resnikoff, “My song was streamed 178 million times. i was paid $5,679. . . ,” *Digital Music News*, 2015. <www.digitalmusicnews.com> [online article accessed: 2018-03-02].

[21] P. Galdston and D. Wolfert, “The music modernization act misses the mark,” *Variety, Penske Media Corporation*, 2018. www.variety.com, [online article accessed: 2018-03-26].

[22] E. Staff, “Blockchains: The great chain of being sure about things,” *The Economist. Retrieved*, vol. 18, 2016.

[23] J. A. Kroll, I. C. Davey, and E. W. Felten, “The economics of bitcoin mining, or bitcoin in the presence of adversaries,” in *Proceedings of WEIS*, vol. 2013, p. 11, 2013.

[24] V. Buterin, “The meaning of decentralization,” 2017. <medium.com/@VitalikButerin/the-meaning-of-decentralization-a0c92b76a274> [online blog post accessed: 2018-05-20].

[25] A. Lewis, “A gentle introduction to immutability of blockchains,” *Bits on Blocks*, 2016. <www.bitsonblocks.net> [online article accessed: 2018-04-01].

[26] V. Buterin *et al.*, “A next-generation smart contract and decentralized application platform,” 2013. White Paper, <www.github.com/ethereum/wiki/wiki/white-paper> [accessed 2018-23-03].

[27] A. Khan, “Ethereum to create a new blockchain architecture that’s designed to be general-purpose,” *Pioneer News Group*, 2017. <www.dailypioneer.com> [online article accessed: 2018-04-11].

[28] V. Buterin, “Toward a 12-second block time,” *Ethereum Foundation*, 2014. <blog.ethereum.org/2014/07/11/toward-a-12-second-block-time> [online blog post accessed: 2018-04-02].

[29] J. Peterson, J. Krug, M. Zoltu, A. K. Williams, and S. Alexander, “Augur: a decentralized oracle and prediction market platform,” 2018. White paper, <www.augur.net/whitepaper.pdf> [accessed: 2018-05-03].

[30] Staff, “Cryptokitties: Collectible and breedable cats empowered by blockchain technology,” 2017. White Paper, <www.drive.google.com/open?id=1soo-eAaJHzhw_XhFGMJp3VNcQoM43byS> [accessed: 2018-06-13].

[31] G. Konstantopoulos, “How will ethereum scale? top talks from devcon3 summarized,” 2017. <https://medium.com/loom-network/how-will-ethereum-scale-top-talks-from-devcon3-summarized-f51f99ed4602> [online blog post accessed: 2018-05-23].

[32] G. Wood, “Ethereum: A secure decentralised generalised transaction ledger,” *Ethereum project yellow paper*, vol. 151, pp. 1–32, 2014.

[33] A. Charlton, “icloud accounts at risk of brute force attack as hacker exploits ’painfully obvious’ password flaw,” *International Business Times*, 2015. <www.ibtimes.co.uk> [online article accessed: 2018-06-05].

[34] S. Gibbs, “Dropbox hack leads to leaking of 68m user passwords on the internet,” *The Guardian*, 2016. <www.theguardian.com> [online article accessed: 2018-06-05].

Appendix A
GitHub Repository

All the source code for the Rights Done Right project can be found in the following repository:

- **https://github.com/hkbhaugen/RightsDoneRight/**

Follow the instructions in the README-file on how to setup the local blockchain and run the application.