Masters in Informatics Engineering
Dissertation/Internship
Final Report

Binder
Backend server and frontend web client for music collaboration

André Filipe Rocha Macedo
afmacedo@student.dei.uc.pt

Supervisors:
João Monteiro
Hugo Oliveira
September 5th, 2017

FCTUC DEPARTAMENTO
DE ENGENHARIA INFORMÁTICA
FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE DE COIMBRA

Abstract

Humans have the tendency to do activities together, and making music is no exception. To produce music together, people needed to be in the same place and play instruments near each other. Nowadays, it is possible for people around the world to work with each other in the production of music. However, the means to do that are rather rudimentary, with only a few tools available online to provide an effective service capable of facilitating the collaboration between multiple people.

At the moment, the process to produce music in the amateur scene involves the use of tools and services, such as *Dropbox, Google Drive,* and *Whatsapp*, that are not suited for music collaboration. Although there are already some collaboration platforms oriented for musicians, they are not widely used amongst them. This indicates that the former platforms lack the capacity of attracting amateur musicians. On the other hand, amateur musicians also have difficulty promoting themselves and finding new opportunities.

This project aims to solve both problems. The main focus is the development of an online platform, where musicians can collaborate with each other and find new projects to work on. The project's second objective is the versioning of the projects, in which the musicians can follow some of the software development principles and check the contents of the project at any point in time and even revert changes.

It was done an analysis of services that provide a collaborative ecosystem to better understand what already exists in the market that is capable of creating music, as well as to detect their flaws to try and conceive a platform that is capable of fulfilling the needs of the collaborative music creation.

The development process was Scrum, and a product backlog with the all the features to be implemented was created. This phase also included the definition of technologies used, risks of implementation and a plan for testing. Afterwards, a system architecture capable of satisfying the problems stated above was created. The implementation phase that followed consisted of developing a backend REST API server using Ruby on Rails and a frontend web client using React Redux.

Keywords

Music, collaboration, version control, multitracking

Table of Contents

Index of Figures

Index of Tables

Glossary

Commit A commit is a snapshot of a file or files at a certain moment. In software development, it is normally used to register the progress of the software being built.

Repository A repository is a place where software files are hosted. It can be located on a local computer, or on a remote server.

Branch: A branch is an independent line of development. It is used to isolate the developers work from each other.

REST API REST Stands for Representational State Transfer. It's a stateless client-server communications protocol. A REST API is an interface with several endpoints that allow external applications to perform operations in the system.

Digital Audio Workstation (DAW): A Digital Audio Workstation is a software application used for recording, editing and producing audio files, such as songs or sound effects [1].

Plug-ins: A plug-in can add or enhance audio-related functionality to a software application. In this particular case, a plug-in can add extra functionality to a DAW [2].

Stems: A stem is a group of audio sources, grouped together, that can be in mono, stereo or on multiple tracks.

Multitracking: Multitracking is a method that allows a set of different audio channels to play together synchronously. The instruments and vocals can all be recorded individually.

Trim: Trimming is the process of cutting pieces of a track. It allows removing unwanted short pieces of music from the track.

Loop: A loop is a repeating section of sound material. It is often used to create patterns.

Stereo panning: Is the distribution of a sound signal to a different channel. For instance, it allows changing sounds from the right channel to the left channel.

Track volume: Individual track volume permits the user to set a specific track with a lower or higher volume.

Equalizer: Is the process of adjusting the frequency response of an audio system, using linear filters.

Compression: Reduces the volume of loud sounds and amplifies quiet sounds through signal processing.

Metronome: Produces a sound at regular intervals. Often used by musicians to practice at a regular pulse.

Track bouncing: Is the method of combining multiple tracks into one, allowing room for more tracks.

Overdubbing: Is a technique in which the musician is capable of recording a new sound over an existing track, without replacing it. That it, the final result is a combination of the original track and the newly recorded sound.

Acronyms

VCS	Version Control System
RCS	Revision Control System
CVCS	Centralized Version Control System
DVCS	Distributed Version Control System
CVS	Concurrent Versions System
QA	Quality Assurance

1. Introduction

The present document is the result of the work conducted within the scope of the internship at Deemaze Software under the supervision of Hugo Oliveira, Ph.D. Professor at the Department of Informatics Engineering of the University of Coimbra, and Engineer João Monteiro, Co-Founder, and Software Engineer at Deemaze Software. The resulting work of this project and internship is called Binder.

This introductory chapter is divided into four sections. The first section presents the motivation of this work in the music collaboration area. The second section presents the goals of this project and internship. The third section contextualizes the proposed work in Deemaze Software. Finally, the fourth section describes the purpose of each chapter, as well as all the documents that append this report.

In this work, the software engineering challenge of creating and developing a real world product using the today's industry standards in product development will be undertaken.

1.1 Motivation

In the *Encyclopædia Britannica*, music is defined as *"art concerned with combining vocal or instrumental sounds for beauty of form or emotional expression"* [3]. Humans, as a social species, tend to do activities together, and making music is not an exception. To produce music together, people needed to be in the same place and play instruments near each other. Nowadays, with the advancements in technology, it is possible for people around the world to work with each other to produce music. However, the means to do that are rather rudimentary, with only a few tools available online to provide a service capable of facilitating the collaboration between multiple people, and very few can do it efficiently and for free.

After talking to several amateur and professional musicians, it became clear that most of them collaborate with other musicians using services such as *Whatsapp, Dropbox* and *Google Drive*, that do not have music collaboration as their primary goals. Very few used more specialized services in music collaboration, due to being overly complicated or not fitting their needs. Another issue noted was that it is very difficult for an amateur musician to show itself and to find partners to work with, outside of their social group.

As stated before, most music collaboration is done in a rather rudimentary process of sharing files in the cloud or sending them directly to each other. Although there are already some services that aim to facilitate the creation of collaborative music, most users still use the most rudimentary processes. This proves that a service that can truly solve this problem does not exist at the moment. Since the process is not optimized for music collaboration, the time spent producing and its quality can be greatly affected.

When comparing the process of creating music with professional development of software, the latter is being optimized ever since the first computer program. At the moment, it is unthinkable for a team of developers to develop software in the industry without version control tools or specialized tools to make the coding process faster. That is called collaboration. Open source projects are also an interesting topic to look at because they are projects open to everyone where people from all over the world can find them, contribute to and improve the final product.

Some of the principles used in software engineering can be useful to optimize the process of producing music. After analyzing the process of creating music by both professional and amateur musicians, we came to the conclusion that professionals already have a method for music creation and collaboration with other parties, since they heavily prioritize having complex music workstations, other than a more collaborative environment. Amateur musicians, on the other hand, have a much bigger need for the collaborative environment, because they did not assert their place in the industry yet.

1.2 Goals

The goals of this work can be split into two major parts:

- Internship, which focuses on experience and knowledge acquired by the intern.
- The project, which corresponds to the implementation of all the proposed features.

1.2.1 Internship

The goals for the internship are to consolidate knowledge about Software Engineering processes, as well as designing and conceiving a product, master the Ruby on Rails framework and React library, used in the development of web applications, and to gain experience in developing software in a business environment where teamwork is essential to produce software with quality.

1.2.2 Project

The main goal of the project is to produce an application capable of expediting the music collaboration process and solving the problem of amateur musicians finding new projects to work on. To accomplish this, the following features must be fully implemented and tested, to minimize the number of bugs:

- **Integration of music projects with version control**: Allow users to check the history of operations in their projects, as well as navigating back in time to check for a previous version.
- **Collaboration in music projects**: Allow users to search for music projects and apply for them, according to the user's skill set and the project needs.

1.3 Scope

The internship is taking place at Deemaze Software, a software company headquartered in Coimbra, Portugal, that is specialized in the development of mobile and web applications.

The Scope of the project is to implement the backend of the application as a REST API, in order to be usable by external applications (e.g. an Android application), and to implement a frontend client, in order to use the application on a web browser.

This internship is being done with Pedro Batista, also an intern in this case, and is responsible for developing an Android application for this problem.

1.4 Document Structure

The remainder of this document is divided into the following chapters:

- 2. State of the Art: presents an analysis of the most popular direct and indirect competitors, as well as their business models;
- 3. Approach: describes the approach followed in this internship, from the development methodology to the risks and testing plans
- 4. Architecture: describes the architecture of each part of the system, as well as a deployment view of the system and some of the technical decisions made.
- 5. Implementation: details what was implemented and how the implemented features work.
- 6. Conclusion: summarizes the work that was done and the future work planned.
- 7. Mockups: features mock-ups for the most important features of the project.
- 8. References: list of references used in the document.

2. State of the Art

The main goal of this chapter is to analyze the current collaboration tools available in the music scene that compete with the internship product. This analysis provides a better understanding of where the product will fit, as well as to identify the most common business models and attractive functionalities. This analysis is composed of the following sections:

- Version control concept in section 2.1
- Version control services in section 2.2
- Comparative analysis of version control systems in section 2.3
- Most popular direct competitors in section 2.4
- Most popular indirect competitors in section 2.5
- Comparative analysis of direct competitors in section 2.6

For each of these sections, a brief explanation of the competitors will be provided, as well as some of their main features and ratings (for mobile applications).

Three aspects must be clarified before proceeding to the next sections:

- The internship product was developed as a web application platform.
- Services are considered to be direct competitors if they provide a music collaboration tool as a web application.
- Services are considered to be indirect competitors if they provide a music collaboration tool on a platform other than the web.

2.1 Version Control

Version Control, or Source Control, is the ability to register file changes over time, so that the user may recall specific versions later. It allows to revert files back to a previous state, compare changes over time and see who made modifications to the files. It gives the possibility to recover from errors introduced by someone or something, by going back in time and tracing its origin. There are three different types of Version Control Systems (VCS): local, centralized, and distributed.

2.1.1 Local Version Control Systems

Local Version Control Systems are very simple to use. It usually consists of having several versions of a file stored in a local database. RCS (Revision Control System), is one of the first and most used local VCS. It is still used in some projects, but its usage is nowhere near the more modern VCS, due to its natural barrier for multiple developers to work on the same project, since it is bound to the local developer's workstation.

2.1.2 Centralized Version Control Systems

Centralized Version Control Systems (CVCS) were developed to mitigate the need to work with other developers on different systems. Developers would commit a change to a central system, usually a single server containing all the versioned files. This creates a dangerous single point of failure.

2.1.3 Distributed Version Control Systems

To avoid having a single point of failure, Distributed Version Control Systems (DVCS) were created. With these Version Control Systems developers mirror the full repository, which means that the repository can be copied backup in the case of a server failure. There are no disadvantages in using a DVCS instead of a CVCS because a DVCS have the same features a CVCS has, with the plus of being distributed and avoiding the single point of failure. In fact, DVCS can work as a CVCS.

2.2 Version Control Services

This section presents some of the most common version control services available online.

2.2.1 Github

Github[I] is a git repository hosting service, founded in 2008 by Tom Preston-Werner, Chris Wanstrath, and PJ Hyett and originally written in Ruby[II], using the Ruby on Rails[III] framework. Github offers plans for private and free repositories and it is the most well-known source of open source projects.

Figure 1 - Github logo

2.2.2 Bitbucket

Bitbucket[IV] offers the same service as Github. It was founded in 2008 by Jesper Noehr and it is written in Python, using the Django framework[V].

Figure 2 - Bitbucket logo

2.2.3 Gitlab

Gitlab[VI] is a repository hosting website, same as Github and Bitbucket. The main difference from the latter is that Gitlab is open source, and has a version operating under the MIT License, which allows for commercial use. This means that it is possible to have Gitlab running on a private server, for commercial purposes. It also has a paid version, with extra features.

Figure 3 - GitLab logo

[I] https://github.com/
[II] https://www.ruby-lang.org/en/
[III] http://rubyonrails.org/
[IV] https://bitbucket.org/
[V] https://www.djangoproject.com/
[VI] https://about.gitlab.com/

2.2.4 SourceForge

SourceForge[VII] is a web-based service that offers a centralized location to host open source projects. It was founded in 1999 by VA Software and the first to offer this service for free. SourceForge allows the usage of several types of version control tools.

Figure 4 - SourceForge logo

2.3 Version Control Services Comparative Analysis

Although SourceForge was the first service to offer free repositories to open source projects and has an external API, it only allows for open source projects, and it has been involved in controversy recently, as some applications were riddled with unwanted third party software. Github and Bitbucket support private repositories and also have an external API. Gitlab has the same perks as Github and Bitbucket, but because it is open source with a commercial license, it can be hosted on private servers, which is a clear advantage over the other Version Control Services, as shown in Table 1.

	Github	**Bitbucket**	**Gitlab**	**SourceForge**
External API	✓	✓	✓	✓
Open source with commercial license	×	×	✓	×
Private Repositories	✓	✓	✓	×

Table 1 - Version control systems comparative analysis

2.4 Direct Competitors

This section presents some of the most popular direct competitors currently operating.

2.4.1 Blend.io

Blend[VIII] is a cloud-based music platform founded in 2013. It integrates with the most popular Digital Audio Workstations (Ableton Live, FL Studio, etc). Additionally, it enables the distribution of music through some of the main known channels (e.g. streaming services such as Spotify, Beatport, Apple Music). Blend's business model consists in a marketplace, where artists can buy and sell stems at their own price.

Figure 5 - Blend.io logo

[VII] https://sourceforge.net/
[VIII] https://blend.io/

2.4.2 Gobbler

Gobbler[IX] is a platform for music collaboration whose main goal is to aid the process of media project management and all it entails (backup, versioning, file transfers and collaboration) and supports project files from popular Digital Audio Workstations. The project versioning is saved in the users own Google Drive account. It makes money by selling plug-ins in the marketplace. Gobbler has a web application to manage the project and a desktop application to keep project files synced in the cloud.

Figure 6 - Gobbler logo

2.4.3 Kompoz

Kompoz[X] is an online collaboration tool that allows users to contribute to existing projects by uploading tracks. The project owner creates open positions for the project and then decides who to incorporate or not. Their business model consists of subscription plans, where users can pay to have more private projects.

Figure 7 - Kompoz logo

2.4.4 Splice

Splice[XI] is a cloud-based collaboration platform that can integrate with the most popular Digital Audio Workstations. It has a version control mechanism where the user can check previous iterations of the project. It assumes that the user has a Digital Audio Workstation for music production. It also has a marketplace as a business model, where the user can buy stems and plug-ins.

Figure 8 - Splice logo

2.4.5 Soundtrap

Soundtrap[XII] is a music recording platform available not only for the web but also for iOS and Android. It enables artists to create, browse, record, edit and share tracks with other users. It has built-in sampled music instruments with a wide range of effects. It supports multitracking and an invite system for users to collaborate together. It is a freemium platform, where you can upgrade to premium to have access to more sound effects, as well as the ability to have a greater number of projects and tracks.

Figure 9 - Soundtrap logo

[IX] https://www.gobbler.com/
[X] http://www.kompoz.com/music/home
[XI] https://splice.com/
[XII] https://www.soundtrap.com/

2.5 Indirect Competitors

This section presents some of the most popular indirect competitors currently operating.

2.5.1 Spire

Spire[XIII] is a multitrack recorder application available for iOS that automatically adds a professional finish to the recordings through dynamic equalizer and compression. The multitracking supports 4 tracks and has a metronome to aid the user in the recording. For the music edition, it can change the individual volume of each track and do stereo panning. Spire's collaboration mechanism consists of sharing the music produced via email, Facebook or Twitter.

Figure 10 - Spire logo

2.5.2 Trackd

Trackd[XIV] is a music collaboration application available for iOS that supports multitracking and audio file uploads, as well as music recording. It has the ability to edit each individual track volume and control stereo panning. The music social feed provides the ability to team-up with other people.

Figure 11 - Trackd logo

2.5.3 Speazie

Speazie[XV] is a music recording platform available for both Android and iOS, that enables artists to create, browse, record, edit, and share tracks. It allows file uploading, overdubbing, and has a beats library. The produced music can then be shared in a social feed within the app.

Figure 12 - Speazie logo

2.5.4 Soundtrap

Soundtrap can be considered both a direct competitor and an indirect competitor, due to having a mobile (Android and iOS) and web platform. The mobile platform of Soundtrap is capable of doing everything the web counterpart can.

Figure 13 - Soundtrap logo

[XIII] http://www.spire.live/
[XIV] https://trackdmusic.com/
[XV] https://www.speazie.com/

2.6 Direct Competitors Comparative Analysis

This section compares the direct and indirect competitors regarding its most important features and the application developed in this project - Binder. The comparison resulted in Table 2, and the features being compared were chosen for being used in music collaboration, editing and keeping track of different versions.

	Splice	Soundtrap	Blend	Gobbler	Kompoz	Binder
Version Control	✓	✗	✗	✗	✗	✓
Project Search	✓	✓	✓	✓	✓	✓
Music Player	✓	✓	✓	✓	✓	✓
Cloud Services integration	✗	✗	✓	✓	✗	✗
Social Media integration	✗	✓	✗	✗	✗	✗
Multitracking	✓	✓	✓	✓	✓	✓
Audio upload	✓	✓	✗	✗	✓	✓
Music feed	✓	✓	✓	✗	✓	✓*
DAW integration	✓	✗	✓	✗	✗	✗
Has own studio	✗	✓	✗	✗	✗	✓+
Sample marketplace	✓	✗	✓	✗	✗	✗
Music marketplace	✗	✗	✓	✗	✗	✗
Music sharing/publish	✓	✓	✓	✗	✓	✓*
Desktop application	✓	✓	✗	✓	✗	✗
Message system	✓	✓	✓	✓	✓	✓*
Audio recording	✗	✓	✗	✗	✗	✓+
Trim	✗	✓	✗	✗	✗	✓-
Loop	✗	✓	✗	✗	✗	✓-
Stereo panning	✗	✗	✗	✗	✗	✓+
Track volume	✗	✗	✗	✗	✗	✓+

Table 2 - Comparative analysis of direct competitors

* Features are planned and will be added in a future iteration
+ Features present in the Android platform
- Features are planned and will be added in a future iteration of the Android platform

Table 2 shows that all collaboration services offer some sort of messaging system (either by direct messaging or by comments), as well as multi tracking. On the other hand, only Soundtrap offers Audio recording and some sort of music edition. If we look at version control, only Splice offers that. Most of them also do not offer integration with DAW and do not have a built-in studio.

By analyzing this table, we can infer that multitracking and a form of communication are essential features in a project of this scope. We can also conclude that since we want to direct the project to a more amateur crowd, the built-in studio and integration with DAW are not essential.

Going back to what was said in the introductory chapter, we can also establish that although most services do not offer version control, it would be beneficial to the project to have that feature, allowing project owners to have more control over their music projects.

This project aims to gather the best of every application and offer the user the ability to collaborate and create music. In order to achieve this goal features like version control and multitracking are highly prioritized, following basic music manipulation (stereo panning and track volume), and a messaging system to be implemented in the future. The web application's main focus is the management of music projects and finding new projects and project members. The Android counterpart is more focused in the music recording and basic music edition.

Gitlab will be used as a version control service since it has all the perks of other Version Control Services, is open source, has a commercial license, and an external API. The fact that it can also be run under a private server is a big plus.

3. Approach

As stated earlier, Binder aims to provide a platform for music collaboration among musicians. Being a software engineering project, different approaches can be followed in order to develop a product that solves the problem.

This chapter presents the approach followed in the internship. Section 3.1 describes the methodology used in the internship product development. Section 3.2 presents the tools and technologies used in the application development. Section 3.3 details the risks identified and its mitigation plans. Finally, section 3.4 describes testing of the application.

3.1 Methodology

This section describes the methodology used for the internship product development, Scrum. This methodology was chosen since it is the methodology used in all projects at Deemaze Software.

3.1.1 Scrum

As any other software development methodology, Scrum is a methodology for a team to work together and develop a product. It has an interactive approach, where each new piece is built upon previously developed pieces[4].

[5]As stated in Figure 14, scrum projects are split in cycles (usually between one and four weeks) called Sprints. Each Sprint represents a set of tasks to be completed. The list of features to be implemented is called the Product Backlog. At the beginning of each Sprint, the list of tasks for the current Sprint is set by the Product Owner. The selected tasks are transferred from the Product Backlog to the Sprint Backlog. During the Sprint period, the Scrum team has Daily Meetings to discuss the work done in the previous day and to define the agenda for the day. At the end of the Sprint, the Sprint Backlog must be fully implemented and reviewed and the cycle is repeated.

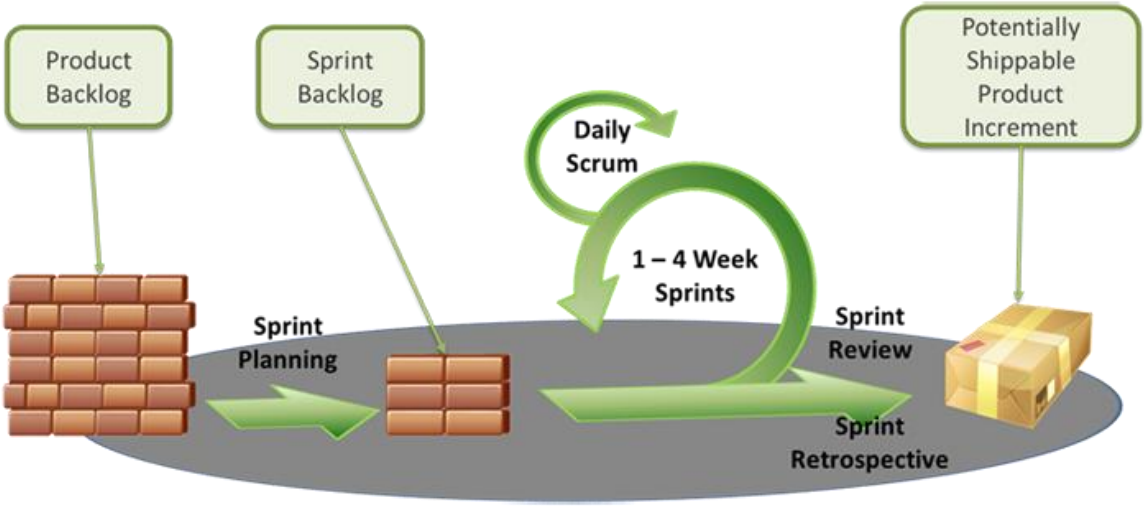

Figure 14 - Scrum methodology[5]

In this internship, we adopted a slightly modified version of Scrum, in order to match the company's development process. We have Sprints with the duration of one week, and on the

last day of the Sprint, a showcase is made, with supervisor João Monteiro, to show the progress of the week, talk about possible blockers and adjust the Sprint Backlog for the next week accordingly.

The Scrum process is being managed in a tool called *Trello*, where we have 8 lanes:

- **Backlog**: Where all the user stories are originally put.
- **To Do**: Where the user stories from the Sprint Backlog are put.
- **Waiting for Layout**: for tasks that need a design layout.
- **In Dev**: For every task that is currently being developed.
- **Code Review**: Where tasks that need to be reviewed by someone from the company are.
- **Quality Assurance (QA)**: Where the task is manually tested by another member of the company.
- **Done**: For tasks that pass this process.
- **Deployed**: For tasks that are deployed to the server.

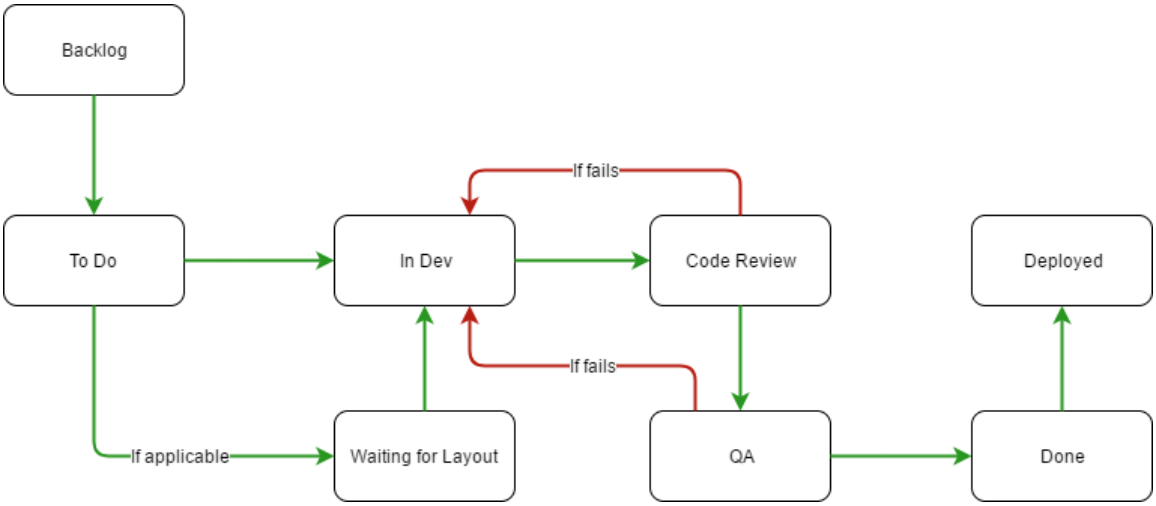

Figure 15 - Trello flow

Figure 15 describes the flow used in the development process. All the tasks are initially in the *Backlog* lane, where they are refined and moved to the *To Do* lane. When a task is ready to start being developed, it is moved to the *In Dev* lane. However, if it needs any type of design (usually tasks regarding the front-end web client), it first goes to the *Waiting for Layout*, and after the design is done, it is pushed to *In Dev*. After the development, the task goes under *Code Review*, where it is reviewed by a member of the company. If there is something to be rectified it returns to *In Dev*, otherwise it advances to *QA*. When in *QA*, the task is manually tested by another member of the company. If it fails the *QA*, it goes back again to *In Dev*, otherwise, it is moved to *Done*. At the end of the Sprint, all tasks that are placed in *Done* are deployed on the server.

3.1.2 Roles

This section lists the Scrum roles[6] present in this project.

- **Product Owner**: The Product Owner is the person inside the project with the responsibility to organize the tasks in the Product Backlog. The Product Owner is also responsible for clarifying all Product Backlog tasks to the Scrum Team.

- **Scrum Master**: The Scrum Master is responsible for keeping up the team productivity. The Scrum Master does this by removing any impediment that can obstruct the development process.
- **Scrum Team**: The Scrum Team is cross-functional - there are no roles within the team. They cross-train each other so no one becomes a bottleneck in the delivery of work.

Supervisor João Monteiro is the Product Owner, responsible for the prioritization of the requirements. However, as it is important for the intern to learn the Scrum methodology properly, some of the tasks are delegated to the Scrum Team. I and Pedro Batista constitute the Scrum Team since we need to constantly communicate and synchronize our work. I perform all the developer, testing and architect tasks, as well as any type of technical decisions. The Scrum Master role is performed by João Monteiro.

3.1.3 Planning

The Scrum methodology uses User Stories to define a requirement, containing a description of the action performed by the end-user. All user stories make up the Product Backlog. As this project was proposed by Deemaze Software as a curricular internship, after getting the details of the project from the Project Owner, talked to people that work in the music business, and performed brainstorm sessions, the User Stories started to be written.

Although the project was proposed by Deemaze Software, Pedro and I had the freedom to discuss and decide which were going to be the main features and which user stories were going to be the main focus.

3.1.4 User Stories

A User Story is a way to describe a software feature from the end user's perspective in everyday language. User Stories that include multiple features and can be subdivided in smaller User Stories are called epics.

Following *Mike Cohn's* template[7], a user story should contain the following information:

- **Role**: Describes who requires the story.
- **Goal**: Describes what is required.
- **Benefit**: Describes why this story is required.

A user story usually follows the following format:

- As a <**role**>, I want <**goal**>, in order to <**benefit**>.

In this project, the user stories fall into three categories:

- **Backend Server**: Implementation of features on the REST API server.
- **Front-end Client**: Implementation of features on the front-end web client.
- **DevOps**: Tasks that must be performed in order to configure servers or environments.

3.1.5 Product Backlog

The Product Backlog is a list of all the User Stories that need to be implemented until the end of the project. It contains 2 types of user stories:

- **Must have**: Features that must be implemented until the end of the project.
- **Nice to have**: Features that are not mandatory to be implemented until the end of the project, but would be a good addition to the project.

The following list contains the main User Stories in the project, with a brief description of its main tasks. The detailed full list of user stories can be consulted in Annex A.

Table 3 shows the main tasks regarding the user. They focus on the authentication process and showing and updating a profile.

Epic #1 – User
Log in and register
Recover password
Show user profile
Update user profile

Table 3 - Main features regarding the epic User

Table 4 describes the main operations that can be performed on a music project. It is possible to create, open, update and delete a music project, as well as downloading its music files.

Epic #2 – Project
Create a project
Open a project
Update a project
Delete a project
Download a project

Table 4 - Main features regarding the epic Project

Table 5 gives an overview of what can be done with a music track. A track belongs to a project and the operations listed above can be performed over that resource

Epic #3 – Tracks
Create a track
Open a track
Update a track information
Update a track file
Delete a track
View track history
Open previous version of a track

Table 5 - Main features regarding the epic Tracks

The collaboration tasks described in table 6 are the core of the platform, being responsible to allow the collaboration of different people in a single music project.

Epic #4 – Collaboration
Create a track application
Open a track application
Update a track application
Delete a track application
Search for open track applications
Submit a file to a track application (track submission)
Accept a track submission
Reject a track submission

Table 6 - Main features regarding the epic Collaboration

Table 7's music player tasks are responsible to play music files on the platform, whether it is a single track file or the set of track files present in the music project.

Epic #5 – Music Player
Play a project music
Play a track file
Play a previous version of a track file
Play a track submission

Table 7 - Main features regarding the epic Music Player

The epic 6, regarding server configuration and shown in table 8, incorporates tasks that mainly do any type of configuration in the project.

Epic #6 – Server Configuration
Install Gitlab
Configure Gitlab
Configure React-Redux
Configure REST API server environments
Set up domain and SSL for REST API server

Table 8 - Main features regarding the epic Music Player

3.1.6 Estimation

The estimation of user stories was made using points following the *Fibonacci* sequence. The points try to reflect the complexity of the task: tasks with a low amount points are easier to complete in comparison with tasks with a higher point count. The *Fibonacci* sequence was used because it helps to estimate larger tasks, where the uncertainty is greater[8]. The Product Owner supervised the task estimation process and discussed each estimation value with the scrum team and the rest of developers in the company.

3.1.7 Sprint

After the Product Backlog is defined, it is time to start defining the tasks to be performed in each Sprint. A Sprint is made of short duration milestones that allow the Scrum team to develop a small portion of the project and produce a demonstrable product increment, called the deliverable.

In this project, the Sprint duration was set to one week, with daily meetings and a showcase of the Sprint result in the last day, where the Sprint Backlog for the next week is also defined. The Sprint duration was set by the Scrum Master. The short duration of the Sprint implies that a reduced number of user stories are implemented in each increment, which also allows being on top of every problem that may occur since the Scrum Team does not have much experience.

3.2 Tools and Technologies definition

An important part of the project is the definition of what tools to use and why. Since the API is going to be used by the Android client, a decision was made to also develop a web client separate from the API server, in order to keep the system modularity and improve decoupling. With this, the system will also be easier to develop and maintain. The backend REST API server and the Frontend Web client development stack is described in the next subchapters.

3.2.1 Backend REST API server

Ruby on Rails was chosen as the backend stack, due to the company's experience and ability to give support. *Ruby on Rails* is a framework for web development that extends the *Ruby* language. Testing was done using the *Rspec*[XVI], the standard framework for unit testing in *Ruby on Rails*.

3.2.2 Frontend Web client

React-Redux[XVII] stack is going to be used as the Frontend client. *React* is a javascript library for building user interfaces. *React*[XVIII] is also part of the company's development stack, but since the project is going to need handling a great deal of data, and *React* is not really good at handling much data[9], we will use *React* with *Redux*. *React-Redux* is a stack that allows all the perks of *React*, with the ability to handle data easily. For the testing part, a javascript testing library created by *Facebook* named *Jest*[XIX] will be used, because it is recommended by the *React-Redux* documentation.

[XVI] http://rspec.info/
[XVII] http://redux.js.org/
[XVIII] https://facebook.github.io/react/
[XIX] https://facebook.github.io/jest/

3.3 Risks

Risks are uncertain events that may or may not occur, thus influencing the prospects of achieving the project goal. Risk management is an important process that improves the project's probability of success.

The following tables contain the risks that may affect the project, its impact, the probability of occurrence and its mitigation plans:

Table 9 shows that due to the limited experience in project management, I, the intern, might have difficulty managing a project of this size.

ID	1
Title	Limited project management experience
Description	The intern has a limited experience in project management, and it is possible to fail the estimations for the user stories
Impact	High
Probability	Medium
Mitigation Plan	Discuss with the Product Owner/Scrum Master and other members of the company that is familiar with the tools being used, the estimation for the user stories
Status	The mitigation plan was put to practice and some estimations were revised

Table 9 - Limited project management experience

Table 10 describes the risk of online servers being susceptible to failures. Having redundant servers help mitigate this risk.

ID	2
Title	Service Failure
Description	The physical server where the product is hosted goes offline
Impact	High
Probability	Low
Mitigation Plan	Set up a redundant server in a different physical server
Status	The system architecture ensures server redundancy

Table 10 - Service Failure

Due to the lack of experience, I, the intern might have trouble configuring the Gitlab server and its API, as shown in Table 11.

ID	3
Title	Inability to configure Gitlab
Description	The Gitlab server may not be able to properly accommodate all the project needs
Impact	High
Probability	Low
Mitigation Plan	Use an alternative to version control in the project. Save a new version of the music project every time a new track or subtract is changed
Status	To date of writing, the risk did not happen

Table 11 - Inability to configure Gitlab

In table 12 describes how online libraries are always susceptible to being taken down or being deprecated.

ID	4
Title	Third party libraries go down or become deprecated
Description	Third party libraries used in both the front end web client and backend API server may become deprecated or go down
Impact	High
Probability	Low
Mitigation Plan	Use libraries with a big community of users behind it, as it is easier to find alternatives when they become deprecated, or manually implement the libraries' functionality
Status	To date of writing, the risk did not happen

Table 12 - Third party libraries go down or become deprecated

Since we are working with binary files, some conflicts might occur during commits. We may need to force the latest version of the file into the repository as can be seen in table 13.

ID	5
Title	Conflicts when committing changes
Description	Upon making a new commit with a track edition in Gitlab, conflicts may occur
Impact	Low
Probability	Low
Mitigation Plan	Force the latest commit in the Gitlab repository
Status	To date of writing, the risk did not happen

Table 13 - Conflicts when committing changes

Although it is a nice to have feature, there might be issues with the libraries when implementing music edition, as shown in Table 14.

ID	6
Title	Inability to use a library to edit music
Description	The available libraries may not be able to implement all the music edition functionalities
Impact	Medium
Probability	Low
Mitigation Plan	Manually implement the missing functionalities
Status	To date of writing, music edition is not implemented and therefore, the risk did not happen

Table 14 - Inability to use a library to edit music

3.4 Testing

Software testing consists of any action intended to evaluate how a system meets its required results. It helps to keep the number of defects to a minimum while maintaining the code quality[10].

This section presents the types of tests that are going to be implemented and a brief description of how they are going to be implemented.

3.4.1 Unit testing

In software development, unit testing is the process of testing small parts of software individually and independently. It ensures that the software works as the developer intended, and continues to work as the developer continues to implement other features. It is usually an automated task, but it can also be done manually.

In this internship, unit testing was done to the web client and the REST API server modules, the two main components of the system, in order to guarantee that, given an input, the end results are as expected. Every user story will have at least two test cases: one to ensure the action in the user story can be executed, and one or more to check error conditions in the user story. *Jest* will be used to test the *React-Redux* (by recommendation of the Redux documentation) modules in the web client. *Rspec* will be used to test the *Ruby on Rails* API modules on the server side.

To measure the amount of code tested it was used the code coverage metric, that shows the percentage of lines of code covered by the unit tests made.

Since the development of the frontend client is not yet finished, the following code coverage applies only to the REST API server modules.

Total code coverage: 97.27%

Model	Code coverage %
User	100
Project	93.94
Track	85.37
Track application	90.91
Track submission	80.65
Instrument	100

Table 15 - Model code coverage

Controller	Code coverage %
User	95.65
Project	88.89
Track	85.37
Track application	88.24
Track submission	87.18
Instrument	100
Files	96.55
Search	100
Track history	100
Authentication	100

Table 16 - Controller code coverage

As we can see from table 15 and table 16, code coverage of unit testing is averaging nearly 100% for the REST API server, which guarantees with a fairly high confidence that everything works as intended

3.4.2 Fault Tolerance

The system must tolerate failures from the Ruby on Rails API server, Gitlab server and Database failures. Since there are two points of failure, the REST API server, and Gitlab server, a load balancer can be used to handle traffic distribution every time there is a failure in one of the REST API servers. The Gitlab will have a master-slave configuration (in different geographical locations), in which the slave has all the master repositories, and will be transformed into the master if a failure occurs on the server.

For this project, and since all servers are being hosted in DigitalOcean data centers (each server in a different location), the DigitalOcean's load balancing service will be used to guarantee that, if one of the REST API servers has a failure, the traffic is redirected to the online server.

3.4.3 Scalability

Scalability testing tests the performance of a system, a network, or a process when the number of requests changes in size or volume. To perform scalability tests, a tool named *rails-perftest* will be used to measure response times, the number of requests per second. *rails-perftest* simulates multiple requests to the API server and outputs the metrics and possible ways to improve them.

Scalability tests are planned to be developed after the development phase of the project is complete. Since that phase is not fully complete, scalability testing is going to be done in the future.

3.4.4 User Acceptance

User acceptance testing is the process of verifying that the product is accepted by the end user. It is usually performed by the owner or client of the product. However, in the scope of this internship, the product will be tested by outside users, preferably by amateur or semi-amateur musicians, to make sure that all requirements are met.

User acceptance tests are planned to be performed after the development phase of the project is complete. Since that phase is not fully complete, these tests are going to be done in the future.

3.4.5 Continuous Integration

Continuous integration is the practice of automatically running all automated tests every time a new feature is implemented or a new pull request is made in the git repository. This type of testing allows for errors to be detected more easily and earlier in the development process since all unit tests are run in every iteration of the project.

There are several services that provide continuous integration, without any meaningful difference between them. In this internship *wercker*[XX] will be used, because it is the one that I, the intern, have more experience with.

In this project, every time a new commit was pushed to the remote repository, *wercker* started a new build which consisted of executing the following:

- Rubocop – Ruby on Rails code analysis tool, to make sure the code is following the standard rules of good practices
- Rspec – Unit testing framework used in Ruby on Rails to run all back-end tests
- Jest – Unit testing framework used in React-Redux to run all front-end tests

If any of these commands failed to execute correctly, the build failed and it was not possible to merge the code with the master branch.

Every time code was merged into the master branch, *wercker* ran a slightly different build that consisted of executing a deploy to the online server, in addition to the previous commands. This way, when a new feature was developed and accepted, it would automatically be live after a few minutes.

[XX] http://www.wercker.com/

4. Architecture

The architecture is a fundamental part of a software project. It comprises software elements, the externally visible properties and the relationships among them[11].

This chapter is divided into the following three sections:

- **System**: presents the deployment view of the system.
- **Ruby on Rails**: describes the high-level overview and the workflow of the Ruby on Rails framework.
- **Gitlab**: presents a high-level overview of the Gitlab architecture.
- **React** + **Redux**: shows the workflow of a react + redux application.

4.1 System Deployment View

The deployment diagram (Figure 16) of the system shows what components exist and to whom they are connected. In this system, there are two types of clients – web client (React-Redux application) and Android client (out of the scope of this internship). Both clients make requests to the Ruby on Rails API server through a load balancer. The load balancer is a device capable of distributing network traffic across multiple servers, thus improving overall performance and availability. The load balancer will redirect network traffic to the appropriate machine, based on their workloads. After the request is received, the Back-end API server will process it and communicates with the Database and GitLab server's API.

To minimize system downtimes and as a means to mitigate risk with ID 2, we chose to have server redundancy on the Ruby on Rails API, a Database Master-Slave replication, as well as a Master-Slave replication in the Gitlab server.

This system is capable of mitigating the problems of music collaboration by introducing a control version system (Gitlab server) to store music files, and by creating a platform (composed by the Ruby on Rails server and the Web and Android client) in which users can create and cooperate with music projects.

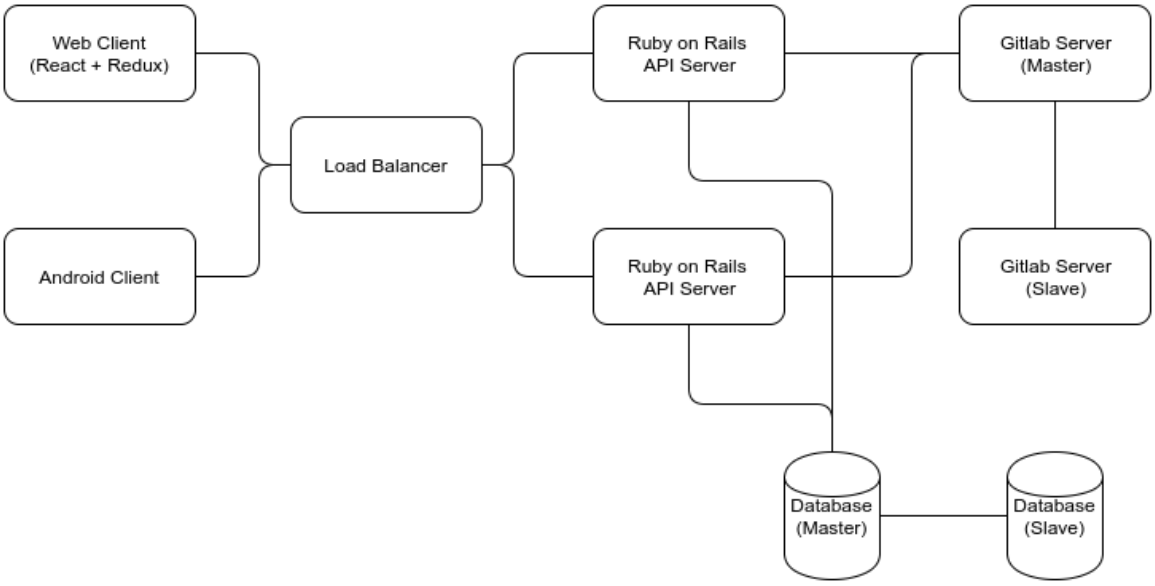

Figure 16 - System Deployment View

4.2 Ruby on Rails

The Ruby on Rails framework (Figure 17) follows the MVC architectural pattern, in order to improve the maintainability of the application.

The Model is used to handle the interaction with the corresponding elements in the database. It contains the business logic and the rules to manipulate data. The View is the front-end of the application. It communicates directly with the user, via the user interface. The Controller processes the incoming requests and communicates with the View and the Model. It processes the data that comes from the Model and passes it to the View.

Apart from the web server, Ruby on Rails modules fit in those three categories from the MVC pattern:

- **View**: includes the Action View, Action WebServices, and Action Mailer.
- **Controller:** incorporates the controller and the dispatcher.
- **Model**: where the Active Record fits.

In this project, the Ruby on Rails server is used as a REST API server. The list of endpoints and possible requests and responses are detailed in Annex B.

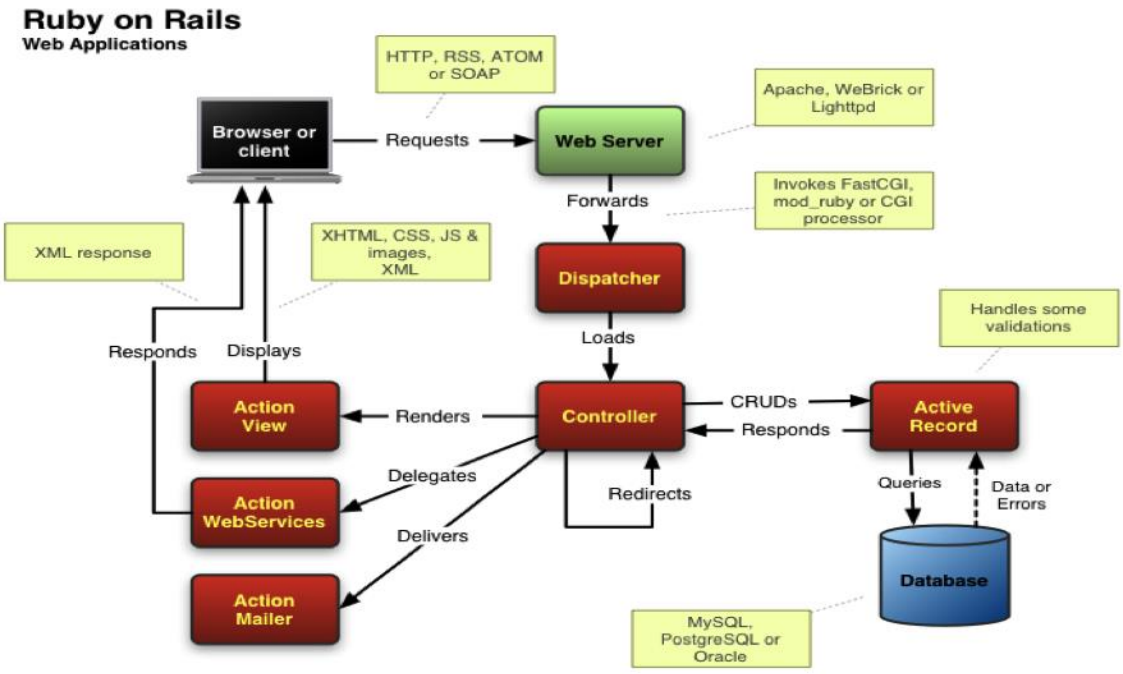

Figure 17 - Ruby on Rails Architecture[12]

4.3 Gitlab

Gitlab is a Ruby on Rails application and has two ways of communicating with each other: HTTP/HTTPS and ssh. To understand Gitlab's architecture, it is important to learn its components (Figure 18):

- **Nginx** is the web server and load balancer of the Gitlab system.
- **Gitlab workhorse** works as a task distributor, as it directs the request to the appropriate component.
- **Gitlab Pages** is a component that only serves static pages.
- **Redis** is a key-value store used to store information about tasks.
- **Unicorn** is a worker that handles quick tasks, such as checking user permissions or sending tasks to Redis.
- **Sidekiq** is a worker that performs tasks asynchronously and gets the task information from Redis.
- **Gitlab** shell is another worker, but it is only receiving requests from ssh.
- **PostgreSQL** is the database, where the repositories and information about users and their permissions are persisted.
- **Gitaly** is responsible for all git operations in the system.

In this project, only the Gitlab API was used, meaning that only the communication flow via HTTP/HTTPS was utilized.

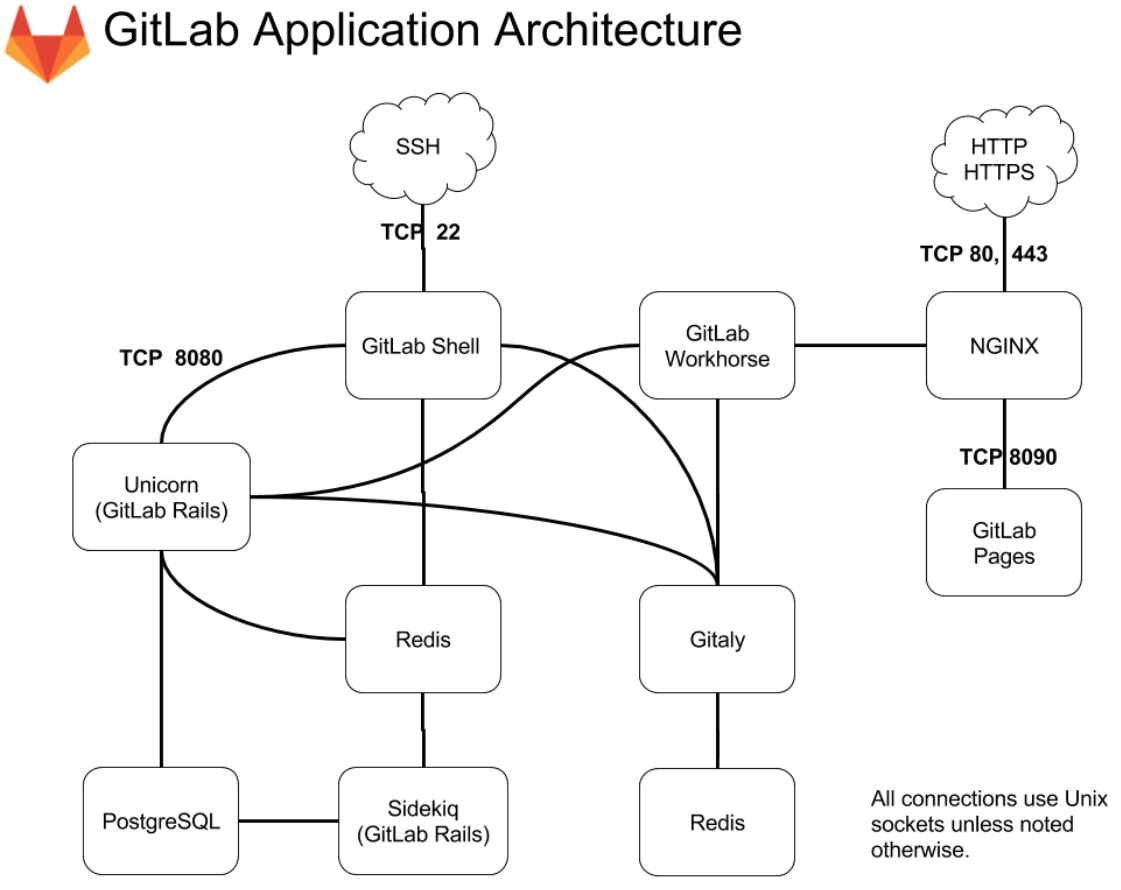

Figure 18 - Gitlab Architecture[13]

4.4 React-Redux

React-Redux's architecture (Figure 19) is straightforward. However, there is one big difference when compared with using *React.js* only – the Store, which is the "global state" of the application, where all the data is accessible in one place. With *React.js*, components had to fetch data on their own, leading to a lot of components having to make API requests. With React-Redux, all components can access the store.

In this architecture, when an action is triggered, it is forwarded to the dispatcher, which in turn does an operation with the Store and, after that renders the view with the result.

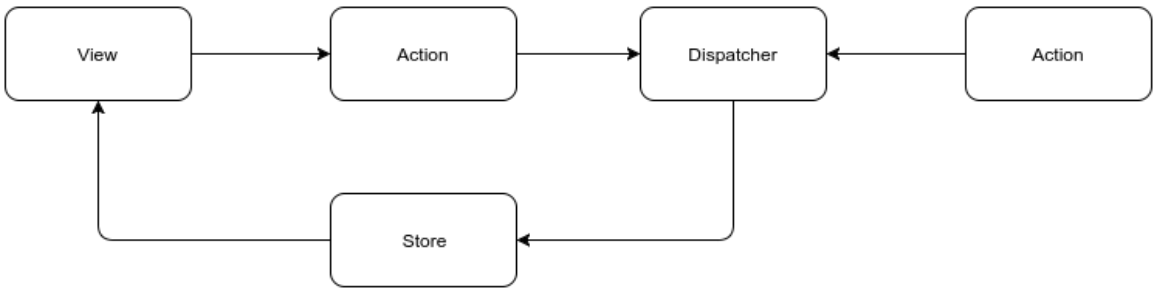

Figure 19 - React-Redux Architecture[14]

5. Implementation

5.1 REST API server

The REST API server is the main server of this project. The front-end client communicates only with the REST API server. However, the REST API server also communicates with the Gitlab server, where the git repositories are created and the music files are stored. This server is made up of the following major features:

5.1.1 Authentication

As stated before, a REST API is stateless by nature, which means that in each request the server needs to verify who is making the request and whether it is authorized or not. To mitigate this issue, JWT (JSON Web Token) was used. JWT is an open standard (RFC 7519) that defines a self-contained secure way of transmitting information between parties as a JSON object. The information can be verified and trusted because it is digitally signed. JWTs can be signed using a secret (with the HMAC algorithm) or a public/private key pair using RSA.[XXI]

In this project, the *JWT* contains an email and id to identify the user and is digitally signed by the server with a secret random string.

For every request made to the REST API, a *JWT* is expected in the request's header. The REST API will receive and decode the token in order to identify who is making the request and assert if it is authorized.

5.1.2 Log in & Register

Registrations are made with an email, password and a name. The username and password are then used to log in the user's account.

Every time an unauthenticated user logs in and/or registers an account, an authentication token is generated using *JWT's*. The authentication token is then used to make subsequent requests, thus identifying the user.

5.1.3 Projects

Projects are the place where collaboration can happen. A user is able to create, open, edit and delete a project.

The creation of a project is determined by two main operations: creating the project in the REST API server, and creating the project in the Gitlab server. The REST API server will contain information about the project name, description, its members, the number of tracks, track applications and track submissions as well as a reference to the files in Gitlab. The Gitlab server is the repository where the files are stored.

[XXI] https://jwt.io/

When a request is made to create a new project, the REST API server makes a request to the Gitlab server, and if the response is positive, the project is also created in the REST API server with a reference to the project repository.

To open or edit a project page, the requests are made only to the REST API server. On the other hand, to delete a project, the REST API server makes a request to the Gitlab server, and if the response is positive, the project is also eliminated in the REST API server.

The project page contains a music player to play the project music. When clicked, the files are retrieved from the Gitlab server in base64 format.

5.1.4 Tracks

Tracks are the individual components of a music project. Each track contains a music file made up of a single instrument.

Similar to the projects, tracks have information stored on both servers: the file is stored in the Gitlab server and the information about the track is stored in the REST API server.

When a track is created, the REST API server sends the file to the corresponding project repository in the Gitlab server and *commits* a message. If the response is positive, the REST API server stores the track information (name, creator, instrument) alongside with the reference to the repository file.

If a track file is updated (eg. a new version is uploaded), the Gitlab server receives the file from the REST API server and commits a new message. The track file deletion is identical to the project deletion, but removing only the file from the repository.

When opening a track page, the user can play the individual track file, by clicking on the play button. That action causes the Gitlab server to return the track file in base64 format.

5.1.5 Track Applications

Track applications are the core of the music collaboration platform. It is through them that it is possible for users to collaborate with other users in music projects. A track application presents itself in two different ways, according to the type of user: for the project owner, it will show a description and a list of track submissions, and for a normal user, it will show the description and a button to upload a track submission. Only the project owner is capable of creating, editing and deleting a track application. On the other hand, the project owner cannot submit files to its own track application.

When a normal user opens a track application and submits a file, the REST API server sends it to the project repository, where it is stored.

When the project owner opens a track application, a list of all track submissions is presented and they are capable of listening to the file individually or alongside the rest of the project tracks.

5.1.6 Track Submissions

Track submissions are the files submitted in track applications. After the user submits the file, the REST API server stores it in the project repository. The project owner can accept or reject the track submissions. If a track submission is rejected by the project owner, the file is deleted from the project repository. In opposition, if a track submission is accepted, the file

immediately becomes a part of the project as a new track, and the track submission owner is from then a member of the project. Any other submissions still pending are destroyed after that.

5.1.7 Track History

The track history shows the different versions of a track file. It holds all its updates, using the Gitlab server as its control version system. In addition to listing all the version of a file, it is also possible to open a previous version and play it. When that happens, the REST API server requests the different version of the file from the Gitlab server using its API.

5.2 Frontend Client

The front-end client is the Internet browser used to access the music platform online.

5.2.1 Reducers

Reducers are the components of what is called "the store". The store is where all the information regarding the web platform is at. For every operation performed, a certain reducer is populated, and the information becomes available for all React components (without reducers, react components need to fetch information individually or to be fed by other components).

This project includes the following reducers:

- Auth reducer – stores information about the user authentication
- User reducer – stores information about a user profile
- Project reducer – stores information about a project
- Track reducer – stores information about a track
- Track application reducer – stores information about a track application
- Track submission reducer – stores information about a track submission
- Track history reducer – stores the list of file changes of a track
- File reducer – stores the files of a project

5.2.2 Actions

Actions are the mechanism responsible for populating reducers. They contain all the front-end client business logic, thus separating the containers from the store. Data in the store can only be modified by actions and accessed by containers. This enforces a specific flow of communication between the multiple elements of the front-end client.

There are two types of actions: action creators and asynchronous actions. The first are simple actions that are performed right away in a sequential manner (e.g. replacing the current data in the project reducer when a new project page is opened). The former are actions that require fetching data from external sources, in this case, the REST API server. Asynchronous actions always contain action creators, in order to change data in the correspondent reducer.

5.2.3 Containers and Components

Containers are responsible for triggering actions and fetching data from reducers. Because of that, they are usually called "smart components". On the other hand, components are usually

called "dumb", because they are only responsible for rendering information in the browser and not having any type of conditional or logic operations that the containers usually have. For that reason, containers usually contain multiple smaller components.

6. Mockups

Mockups are a very useful tool to understand what an application will look like how its features will work. The following list shows some of the mockups for the most important features of this project, namely music project management, collaboration and version control:

Figure 20 shows the project details and its possible operations: create, edit, delete, go to the project track list and go to the track application list. It is also possible to see the project description and its contributors.

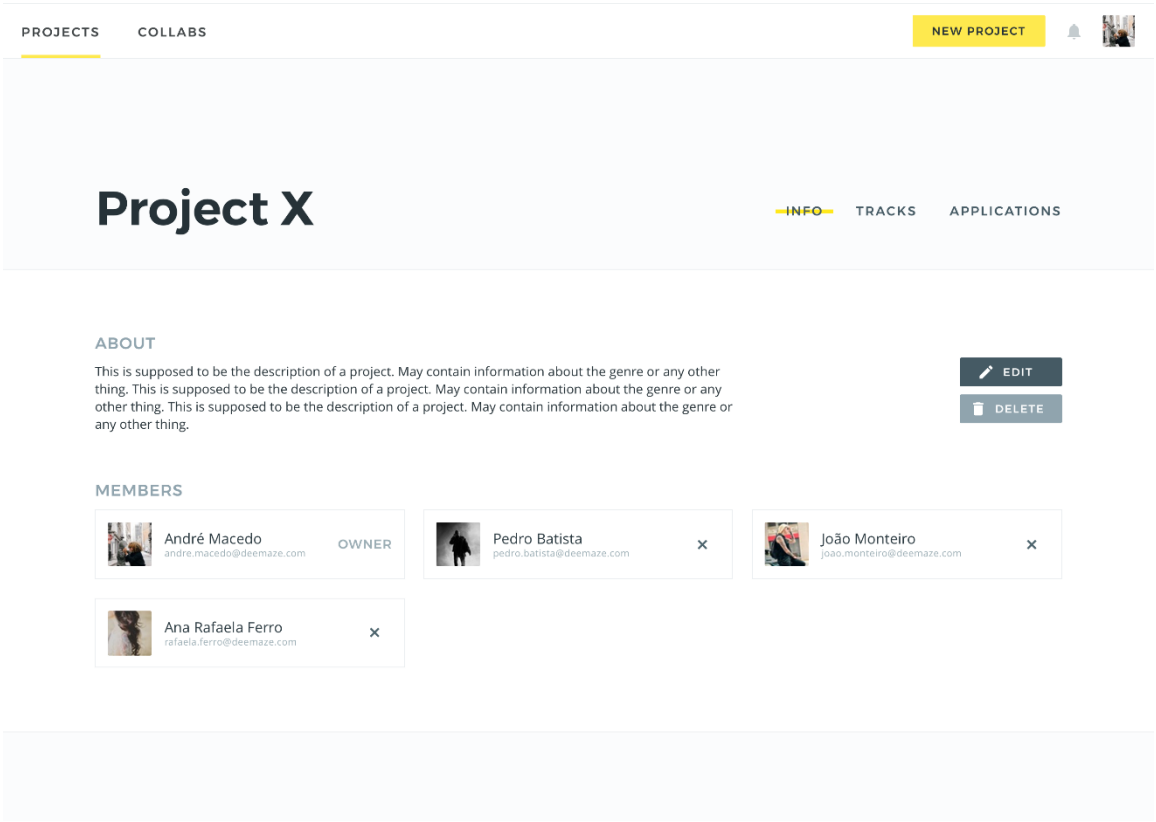

Figure 20 - Music project information

Figure 21 shows the track history and the possible operations over a track: Upload a new track, edit, delete, play the track and check the track history.

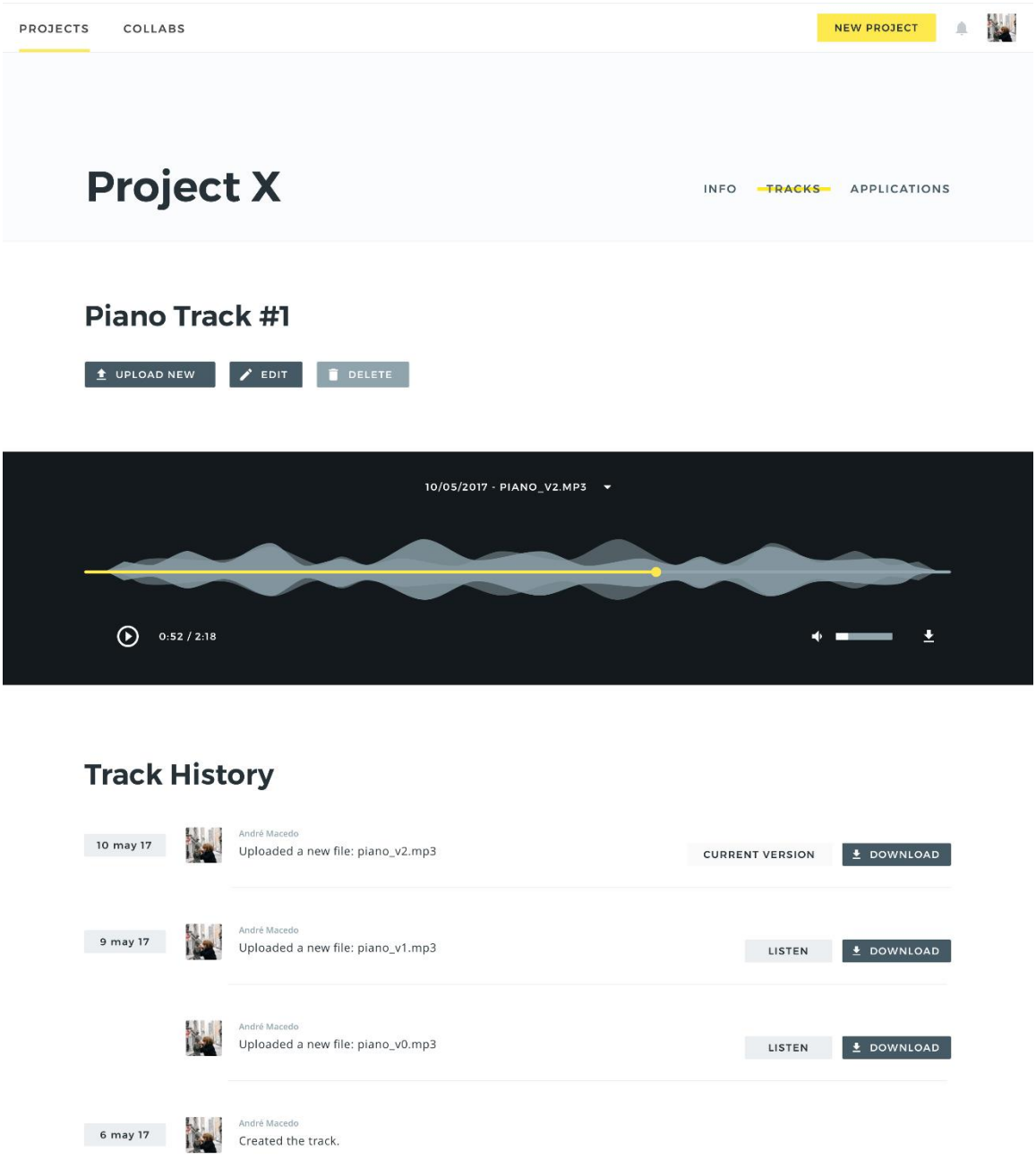

Figure 21 - Project track information

In Figure 22 it is possible to see the track application description and its possible operations: edit and delete. There is also a list of all track submissions and the ability to listen, accept or reject the submission.

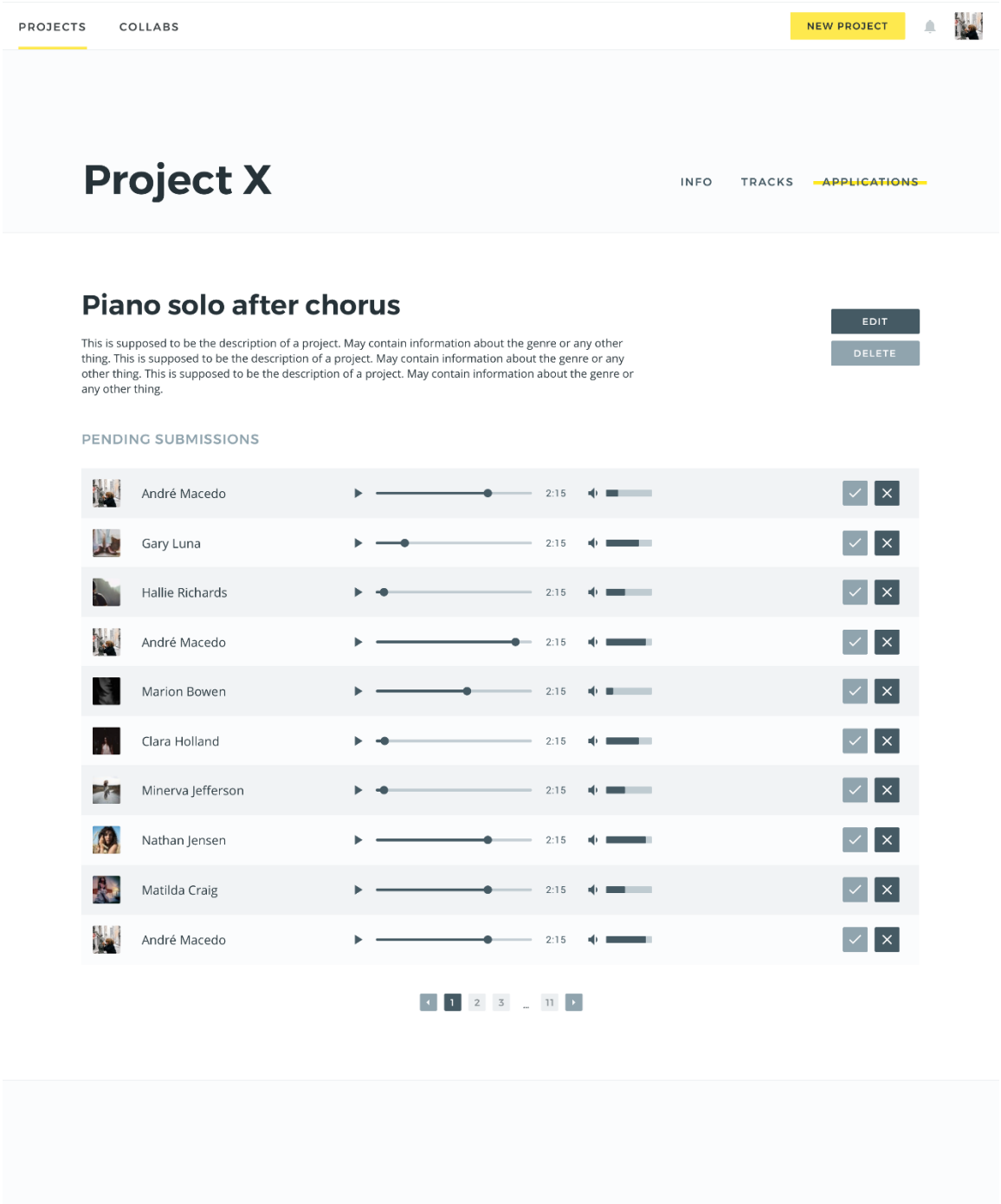

Figure 22 - Track application information

Figure 23 shows the mockup for the search projects with open track applications.

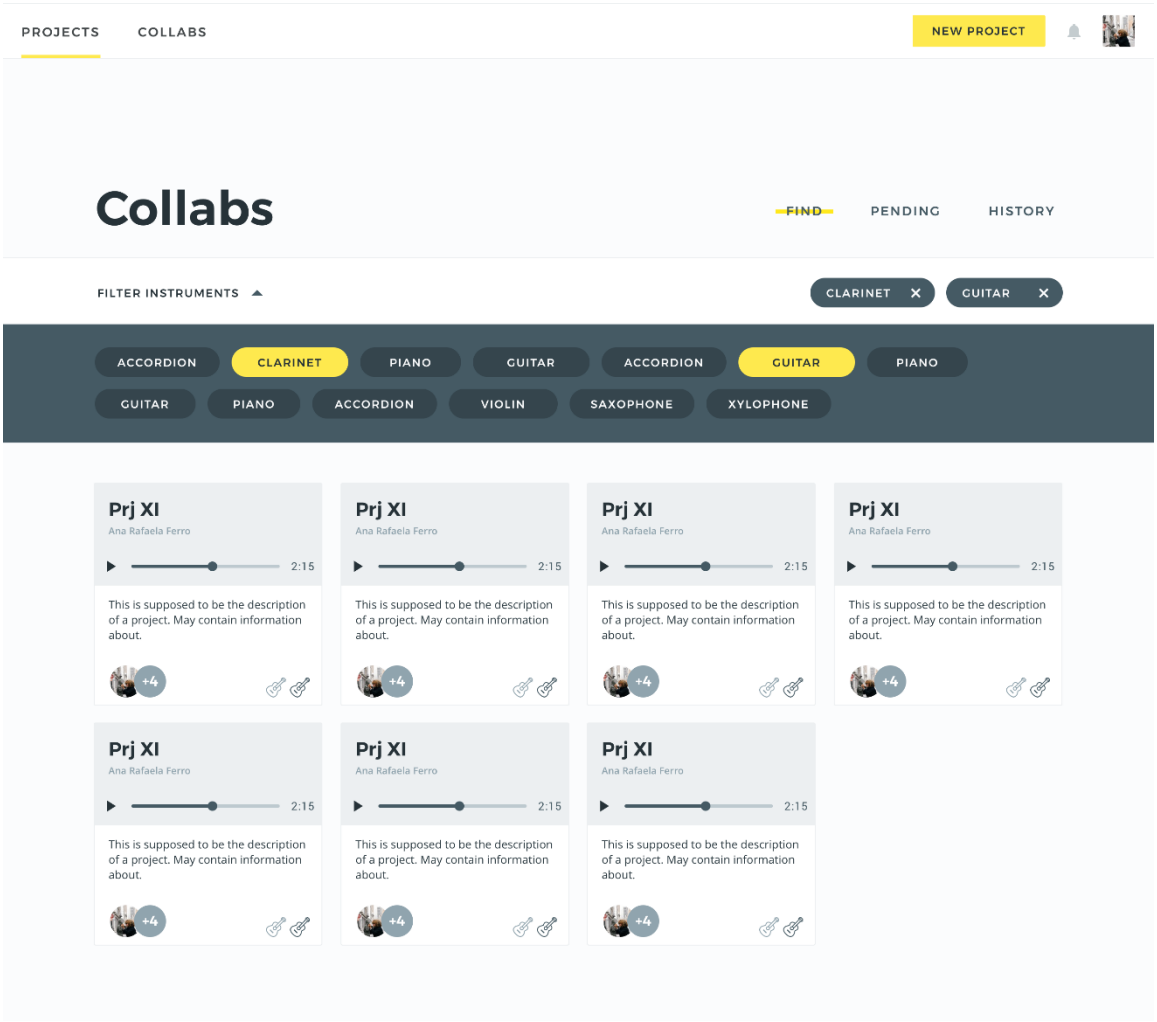

Figure 23 - Search project information

7. Conclusion

To conclude the final report, the following chapters will detail what is left to implement, future work and final remarks about the project.

7.1 Future Work

Some say that the development phase is never complete. This project is no exception, and there is still much to be done in future work.

Development-wise a few features are planned to be developed, such as the ability to download a music file with all the individual tracks combined and the ability to fully revert a track file to a previous version (right now it is only possible to listen to a previous version). The next iteration of the project will incorporate comments in music projects and its tracks.

After developing the above features, the next step is to write scalability and acceptance tests. Despite being extremely important, it was not possible to perform these tests sooner, because without completing the proposed features, the tests would not be very reliable.

6.2 Final Remarks

This internship started with the study of the market and its players. This study allowed me to learn who were the players in the industry with collaborative tools or services.

In the second stage, a study of the state of the art was done. This stage allowed me to better understand the music industry and the process of producing music by both amateur and professional musicians, as well as how they collaborate together.

After this stage, we started to define the approach for this internship, gathering the system requirements based on what we learned in the first two stages and defining the development methodology, alongside with risk identification and the definition of tools and technologies to be used. With the system requirements done, the next step was doing low-fidelity mockups with the company's designer, to better understand what this product would become. After that, the architecture was defined, the first Sprint Backlog set and the development stage began.

Right now, at the end of the development stage of the project and still with some features to be developed and tests to be performed, it is possible to look back and identify a few errors. Over optimistic estimations was one of the main errors, due to the inexperience in developing a real world product. The second most predominant error was not splitting the tasks in the backlog correctly and also having some smaller tasks not defined.

Overall, this project helped me to understand how a real world product is created and developed. It vastly improved my understanding of the development process and my skills in the languages and frameworks used.

8. References

[1] Kefauver, Alan P.; Patschke, David (2007-01-01). Fundamentals of Digital Audio, New Edition, A-R Editions

[2] Collins, Mike A. (2003). Professional Guide to Audio Plug-ins and Virtual Instruments. Burlington, MA: Focal Press

[3] https://www.britannica.com/art/music

[4] http://www.toolshero.com/project-management/scrum-agile-methodology/

[5] http://www.toolshero.com/wp-content/uploads/SCRUM-overview-resize.png

[6] https://www.atlassian.com/agile/scrum

[7] https://www.mountaingoatsoftware.com/agile/user-stories

[8] https://xbosoft.com/software-quality-blog/estimating-agile-story-points-using-fibonacci/

[9] https://www.quora.com/Why-should-I-use-Redux-when-I-can-just-keep-my-state-in-the-top-level-React-component/answer/Torsten-Engelbrecht

[10] https://users.ece.cmu.edu/~koopman/des_s99/sw_testing/

[11] Bass L.; Clements P.; Kazman R. Software Architecture in Practice 2nd Edition Reading, MA: Addison-Wesley, 2003

[12] http://adrianmejia.com/images/rails_arch.png

[13] https://docs.gitlab.com/ce/development/gitlab_architecture_diagram.png

[14] Based on https://github.com/markerikson/react-redux-links/blob/master/react-redux-architecture.md#redux-architecture

Masters in Informatics Engineering
Dissertation/Internship
Final Report

Appendix A
Approach

André Filipe Rocha Macedo
afmacedo@student.dei.uc.pt

Supervisors:
João Monteiro
Hugo Oliveira
September 5th, 2017

FCTUC DEPARTAMENTO
DE ENGENHARIA INFORMÁTICA
FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE DE COIMBRA

Table of Contents

Document Scope

This appendix contains the user stories referenced in chapter 3 of the main document.

Chapter 1 contains all the user stories in the Product Backlog.

1. User Stories

A user story is a way to describe a software feature from the end-user's perspective, in everyday language. It can be an epic user story, that includes multiple smaller user stories.

1.1 Categories

In this project, the user stories fall into three categories:

- **Backend**: Refers to the implementation of features in the REST API server.
- **Frontend**: Refers to the implementation of features in the frontend web client.
- **DevOps**: Refers to any type of task that must be performed in order to configure something (servers, production environments, etc).

1.2 Types of User Stories

The Product Backlog contains 2 types of user stories:

- **Must have**: Features that must be implemented until the end of the project.
- **Nice to have**: Features that are not mandatory to be implemented until the end of the project, but would be a good addition to the project.

1.3 Product Backlog

The Product Backlog is a list of all the user stories that need to be implemented until the end of the project. The following list contains all User Stories of the project. User Stories that have two categories will have the estimation points for both.

1.3.1 Epic #1 – User

Story ID: 1
Description: As an unregistered user, I want to register using my email and password, so I can create an account
Category: Backend server, frontend client
Type: Must have
Backend Estimation: 2
Frontend Estimation: 3

Story ID: 2
Description: As an unregistered user, I want to login using my email and password, so I can enter the platform
Category: Backend server, frontend client
Type: Must have
Backend Estimation: 2
Frontend Estimation: 3

Story ID: 3
Description: As a registered user, I want to recover my password, in case I forgot it
Category: Backend server, frontend client
Type: Must have
Backend Estimation: 3
Frontend Estimation: 3

Story ID: 4
Description: As an authorized user, I want to edit my profile, so I can update my info
Category: Backend server, frontend client
Type: Must have
Backend Estimation: 2
Frontend Estimation: 3

Story ID: 5
Description: As an authorized user, I want to open a user profile, so I can check the user info
Category: Backend server, frontend client
Type: Must have
Backend Estimation: 1
Frontend Estimation: 2

1.3.2 Epic #2 – Project

Story ID: 6
Description: As an authorized user, I want to create a new project, so I can create music
Category: Backend server, frontend client
Type: Must have
Backend Estimation: 3
Frontend Estimation: 3

Story ID: 7
Description: As an authorized user, I want to open a project, in order to check its info
Category: Backend server, frontend client
Type: Must have
Backend Estimation: 1
Frontend Estimation: 3

Story ID: 8
Description: As a project owner, I want to edit my project description and name, in order to update its info
Category: Backend server, frontend client
Type: Must have
Backend Estimation: 1
Frontend Estimation: 3

Story ID: 9
Description: As a project owner, I want to delete my project, because I no longer need it
Category: Backend server, frontend client
Type: Must have
Backend Estimation: 2
Frontend Estimation: 3

Story ID: 10
Description: As an authorized user, I want to download a project's music file, in order to play it on my computer
Category: Backend server, frontend client
Type: Nice to have
Backend Estimation: 5
Frontend Estimation: 3

1.3.3 Epic #3 – Tracks

Story ID: 10
Description: As a project owner, I want to create a track, in order to add music to the project
Category: Backend server, frontend client
Type: Must have
Backend Estimation: 2
Frontend Estimation: 3

Story ID: 11
Description: As a project owner, I want to edit my track, in order to update its info
Category: Backend server, frontend client
Type: Must have
Backend Estimation: 2
Frontend Estimation: 3

Story ID: 12
Description: As a project owner, I want to delete a track, because I no longer need it
Category: Backend server, frontend client
Type: Must have
Backend Estimation: 1
Frontend Estimation: 3

Story ID: 13
Description: As an authorized user, I want to open a project track, in order to check its info
Category: Backend server, frontend client
Type: Must have
Backend Estimation: 1
Frontend Estimation: 3

Story ID: 14
Description: As an authorized user, I want to update a track file, in order to update its contents
Category: Backend server, frontend client
Type: Must have
Backend Estimation: 3
Frontend Estimation: 3

Story ID: 15
Description: As an authorized user, I want to view the track history, in order to check its info
Category: Backend server, frontend client
Type: Must have
Backend Estimation: 3
Frontend Estimation: 2

Story ID: 16
Description: As an authorized user, I want to check the contents of a track at a specific point in time, in order to check its contents
Category: Backend server, frontend client
Type: Must have
Backend Estimation: 3
Frontend Estimation: 5

Story ID: 17
Description: As an authorized user, I want to edit a track's file settings, in order to update its player
Category: Backend server
Type: Must have
Backend Estimation: 2

Story ID: 18
Description: As an authorized user, I want to get a track's file contents, in order to use it to play the track
Category: Backend server
Type: Must have
Backend Estimation: 5

1.3.4 Epic #4 – Collaboration

Story ID: 19
Description: As a project owner, I want to create a track application, so that people can send submissions to my project
Category: Backend server, frontend client
Type: Must have
Backend Estimation: 3
Frontend Estimation: 3

Story ID: 20
Description: As a project owner, I want to edit a track application, in order to update its info
Category: Backend server, frontend client
Type: Must have
Backend Estimation: 2
Frontend Estimation: 3

Story ID: 21
Description: As a project owner, I want to delete a track application, because I no longer need it
Category: Backend server, frontend client
Type: Must have
Backend Estimation: 1
Frontend Estimation: 3

Story ID: 22
Description: As an authorized user, I want to open a track application, in order to contribute to the project
Category: Backend server, frontend client
Type: Must have
Backend Estimation: 1
Frontend Estimation: 3

Story ID: 23
Description: As an authorized user, I want to search for a project that has open track application, in order to contribute to a project
Category: Backend server, frontend client
Type: Must have
Backend Estimation: 5
Frontend Estimation: 5

Story ID: 24
Description: As an authorized user, I want to apply for a track application, in order to contribute to the project
Category: Backend server, frontend client
Type: Must have
Backend Estimation: 5
Frontend Estimation: 5

Story ID: 25
Description: As a project owner, I want to accept a track submission in a track application, in order to incorporate that submission in the project
Category: Backend server, frontend client
Type: Must have
Backend Estimation: 8
Frontend Estimation: 3

Story ID: 26
Description: As a project owner, I want to reject a track submission in a track application, because I didn't like the outcome
Category: Backend server, frontend client
Type: Must have
Backend Estimation: 8
Frontend Estimation: 3

Story ID: 28
Description: As an authorized user, I want to comment a track in a specific moment, so I can give my input
Category: Backend server, frontend client
Type: Nice to have
Backend Estimation: 8
Frontend Estimation: 13

Story ID: 29
Description: As an authorized user, I want to see any track comments, so I can read other people's feedback
Category: Backend server, frontend client
Type: Nice to have
Backend Estimation: 5
Frontend Estimation: 13

1.3.5 Epic #5 – Music Player

Story ID: 30
Description: As an authorized user, I want to play a project's tracks, in order to listen to the project's music
Category: Frontend client
Type: Must have
Frontend Estimation: 21

Story ID: 31
Description: As an authorized user, I want to play a single track of a project, in order to listen to its music
Category: Frontend client
Type: Must have
Frontend Estimation: 21

Story ID: 32
Description: As a project owner, I want to play a track submission, in order to decide if I accept or reject the submission
Category: Frontend client
Type: Must have
Frontend Estimation: 21

1.3.6 Epic #6 – Server Configuration

Story ID: 33
Description: As a developer, I want to install GitLab on a remote server, so that I can use version control in the project
Category: DevOps
Type: Must have
Estimation: 2

Story ID: 34
Description: As a developer, I want to configure GitLab on a remote server, so that I can use version control in the project
Category: Backend server, frontend client
Type: Must have
Estimation: 3

Story ID: 35
Description: As a developer, I want to configure the React-Redux setup, in order to develop the frontend web client
Category: DevOps
Type: Must have
Estimation: 5

Story ID: 36
Description: As a developer, I want to configure the backend server environments, in order develop the backend API
Category: DevOps
Type: Must have
Estimation: 8

Story ID: 37
Description: As a developer, I want to setup continuous integration in the project, in order to detect problems early
Category: DevOps
Type: Must have
Estimation: 8

Story ID: 38
Description: As a developer, I want to setup domain name and SSL in the server, in order to have a secure online address
Category: DevOps
Type: Must have
Estimation: 3

Masters in Informatics Engineering
Dissertation/Internship
Final Report

Appendix B
Architecture

André Filipe Rocha Macedo
afmacedo@student.dei.uc.pt

Supervisors:
João Monteiro
Hugo Oliveira
September 5th, 2017

FCTUC DEPARTAMENTO
DE ENGENHARIA INFORMÁTICA
FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE DE COIMBRA

Table of Contents

Document Scope

This appendix contains the REST API server's endpoints documentation referenced in chapter 4 of the main document.

Chapter 1 contains a list of all available endpoints.

1. REST API server endpoints

URL prefix: `<hostname>/api/v1`

Possible response codes

200	Everything went OK
201	Resource successfully created
204	No content
401	Unauthorized (User does not have permission)
404	Not Found (Resource was not found)
422	Unprocessable Entity (Unable to process the contained instructions)
500	Something went wrong with the server

1.1 Authentication

1.1.1 Register

Endpoint /sign_up

Method POST

Request

name	User name
email	User email
password	User password

Response

id	User id
email	User email
auth_token	JWT

1.1.2 Login

Endpoint /sign_in

Method POST

Request

email	User email
password	User password

Response

id	User id
email	User email
auth_token	JWT

1.1.3 JWT authentication

All subsequent requests after the user login and/or register must include a JSON Web Token in the header, using the following format:
"Authorization: Bearer <auth_token>"

1.2 Profile

1.2.1 Get Profile

Endpoint /users/<user_id>

Method GET

Response

id	User id
email	User email
name	User name
bio	Short user biography
avatar	User avatar
instruments	User instruments

1.2.2 Update Profile

Endpoint /users/<user_id>

Method PATCH

Request

name	User name
bio	User biography
avatar	User avatar
instrument_ids	Array of instrument id's

Response

id	User id
email	User email
name	User name
bio	Short user biography
avatar	User avatar
instruments	User instruments

1.3 Project

1.3.1 Get Project

Endpoint /projects/<project_id>

Method GET

Response

id	Project id
name	Project name
description	Project description
owner	Project creator
collaborators	Array of users that contributed to the project
tracks	Project tracks
applications	Project track applications

1.3.2 Create Project

Endpoint /projects

Method POST

Request

name	Project name
description	Project biography

Response

id	Project id
name	Project name
description	Project description
owner	Project creator
collaborators	Array of users that contributed to the project

1.3.3 Update Project

Endpoint /projects/<project_id>

Method PATCH

Request

name	Project name
description	Project description

Response

id	Project id
name	Project name
description	Project description
owner	Project creator
collaborators	Array of users that contributed to the project

1.3.4 Delete Project

Endpoint /projects/<project_id>

Method DELETE

1.3.5 Get Project track files

Endpoint /projects/<project_id>/files?track_ids=[<array_of_track_ids>]

Method GET

Response

content	Project file content
encoding	Project files encoding (base 64)
gain	Project files gain
panning	Project files panning

1.4 Track

1.4.1 Get Track

Endpoint /projects/<project_id>/tracks/<track_id>

Method GET

Response

id	Track id
name	Track name
instrument	Track instrument
creator	Track creator
project	Track project

1.4.2 Create Track

Endpoint /projects/<project_id>/tracks

Method POST

Request

name	Track name
instrument_id	Instrument id
track_file	Track file

Response

id	Track id
name	Track name
instrument	Track instrument
creator	Track creator
project	Track project

1.4.3 Update Track

Endpoint /projects/<project_id>/tracks/<track_id>

Method PATCH

Request

name	Track name
instrument_id	Instrument id
track_file	Track file

Response

id	Track id
name	Track name
instrument	Track instrument
creator	Track creator
project	Track project

1.4.4 Delete Track

Endpoint /projects/<project_id>

Method DELETE

1.4.5 Update Track Settings

Endpoint /projects/<project_id>/tracks/<track_id>/settings

Method PATCH

Request

panning	Track panning
gain	Track gain (volume)

Response

id	Track id
name	Track name
instrument	Track instrument
creator	Track creator
project	Track project

1.5 Track Application

1.5.1 Get Track Application

Endpoint /projects/<project_id>/tracks_applications/<track_application_id>

Method GET

Response

id	Track application id
name	Track application name
instrument	Track application instrument
description	Track application description
project	Track application project
submissions	Track application submission list

1.5.2 Create track application

Endpoint /projects/<project_id>/track_applications

Method POST

Request

name	Track application name
instrument_id	Instrument id
description	Track application description

Response

id	Track application id
name	Track application name
instrument	Track application instrument
creator	Track application description
project	Track application project

1.5.3 Update Track Application

Endpoint /projects/<project_id>/track_applications/<track_application_id>

Method PATCH

Request

name	Track application name
instrument_id	Instrument id
description	Track application description

Response

id	Track application id
name	Track application name
instrument	Track application instrument
creator	Track application description
project	Track application project

1.5.4 Delete Track Application

Endpoint /projects/<project_id>/track_applications/<track_application_id>

Method DELETE

1.6 Track Submission

1.6.1 Create Track submission

Endpoint /track_applications/<track_applications_id>/
track_submissions/<track_submissions_id>

Method POST

Request

track_file	Track file

Response

id	Track application id
name	Track application name
instrument	Track application instrument
creator	Track application description
project	Track application project

1.6.2 Delete Track Submission

Endpoint /track_applications/<track_applications_id>/
track_submissions/<track_submissions_id>

Method DELETE

1.6.3 Accept Track Submission

Endpoint /track_submissions/<track_submissions_id>/accept

Method PATCH

1.6.4 Reject Track Submission

Endpoint /track_submissions/<track_submissions_id>/reject

Method PATCH

1.6.5 Get Track Submission File

Endpoint /track_submissions/<track_submission_id>/files

Method GET

Response

content	Track submission file content
encoding	Track submission encoding (base64)
gain	Track submission gain value
panning	Track submission panning value

1.7 History

1.7.1 Get Track History List

Endpoint /projects/<project_id>/tracks/<track _id>/history

Method GET

Response

commits	Track commit list with hash

1.7.2 Get Track File from History List

Endpoint /tracks/<track _id>/history/<history_id>

Method GET

Response

content	Track file content
encoding	Track encoding (base64)

1.8 Search Projects

Endpoint /projects/search?by_name=<name>&by_instrument=<instrument_id>

Method GET

Response

id	Project id
name	Project name
description	Project description
owner	Project creator
collaborators	Array of users that contributed to the project
tracks	Project tracks
applications	Project track applications