Masters in Informatics Engineering
Internship 2016/2017
Final Report

Development of an Android application for recording collaborative music

Pedro Miguel Batista Seabra Simões
pmsimoes@student.dei.uc.pt

Deemaze Software Supervisor:
João Monteiro

DEI Supervisor:
Pedro Martins

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

Abstract

Technology has always acted as the music catalyzer enabling it to reach new milestones and quality prospects. Through technology, music has been able to continuously change the way its consumed and conceived. The appearance of smartphones sparked the cycle to repeating itself.

Global smartphone penetration and Internet easy access triggered a global shift of music consumption towards mobile platforms. Taking full advantage of the high and worldwide usage of these mobile devices, music industry is experiencing an unprecedented growth in consumption, directly affecting the overall state of the industry.

Sooner or later an identical shift will hit the music production industry. This internship attempts to ignite that change through the development of a recording studio where musicians can actively collaborate throughout all the stages of production. The platform takes version control, a common practice in typically found in software development, as an inspiration to concoct a mechanism that enhances teamwork and bring musicians closer together.

Keywords: Recording, Version control, Collaboration, Music Production,

Acknowledgements

The present report represents the culmination of my academic course that had not been possible without the continuous effort and sacrifice of my parents, whom I owe everything I am and have today.

A special thanks to a very special person for all the support and patience throughout this year, independently of the mood and my current situation. For that, I will be forever grateful.

A strong appraise to all my friends for all the motivation and unconditional support.

Also, I would like to praise both my supervisors, João Monteiro and Pedro Martins, for their support and time spent along the entire internship.

Finally, I would like to express my gratitude to Deemaze Software for granting me this opportunity and allowing me to work with such a great team that helped and taught me a lot.

Table of Contents

List of Figures

List of Tables

Glossary

APK	Stands for Android application package file format. It is the format used by the Android operating system for distribution and installation of mobile applications.
Audio Data Compression	Audio data compression is a method for reducing the size of digital audio data, thus enabling a more efficient storage and transmission of the audio data.
Digital Workstation	Digital audio workstation. Professional software used for recording, editing and producing audio files.
MoSCoW	A scale designed to label user stories priority. The letters stand for: Must have, Should have, Could have, Won't have this time
Mono	A sound system with only one channel, regardless of the number of speakers used.
PCM	Is a method used to convert a analog signal into a digital signal so that it can be transmitted through a digital communication network and then converted back into the original analog signal.
Pot	Short for potentiometer. Pots are knobs on a console or other audio component used for a variety of controls such as panning or gain adjustments.
Producer	Person in charge of the creative side of a session, but can also be a technical person. Unlike an engineer, a producer may get involved in the song writing, song selection, and other internal aspects of the music. Often by default, an engineer may take on some roles of a producer.
REST	Stands for Representational State Transfer and represent services capable of proving interoperability between computer system over the Internet.
Stem	Grouping or sub-mixing instruments or sounds. Stems may include grouping drums and bass together, grouping vocals together, or grouping any like sounds into a mono or stereo sub mix.
Stereo	Refers to a sound or system that involves two separate channels. These channels are labelled left and right (L/R). A standard audio CD is stereo, containing 2-tracks, left and right.

1. Introduction

The present document is the outcome of the work done at Deemaze Software during the one-year internship. The internship work was supervised by Pedro Martins, professor at the Department of Informatics of the University of Coimbra, and Engineer João Monteiro, Project Manager at Deemaze Software.

1.1. Historical View

Technology pushed aside some forms of art, on Music, technology acts as a catalyst, providing new opportunities, enabling new heights and quality prospects. Wax cylinder, radio transmission, tapes, CDs, software plugins, all changed the way music is produced, distributed and consumed.

After the end of the First World War, amplification of microphone signals and playback on loudspeakers were introduced. Both these factors drove consumers to buy devices to listen to music at home. The proliferation of electrical recordings in 1920s led the record industry to produce higher quality recordings in order to instigate the attention of a wider range of listeners [1].

By the end of the Second World War, magnetic tape recording appeared and transformed recording studios into musical instruments [2]. Tape manipulation generated three important techniques to the music production: Punch in, Multitrack and Overdubbing. The first technique enabled music editing by cutting and pasting the tape, replacing a short musical passage into a previously recorded track. Multitrack allowed producers to separately record multiple audio sources. This technique unlocked overdubbing, which enabled tracks to be redone without affecting the overall project.

The introduction of digital technologies in the 1980s enhanced the possibilities of creating recordings with limitless sound transformation using the Music Instrument Digital Interface protocol (MIDI). MIDI is a protocol developed to allow electronic instruments and other digital music tools to communicate with each other. MIDI does not play sound, its job is to interpret event messages, such as playing a key or changing volume, performed by the instrument. The appearance of digital solutions drastically reduced the cost of producing a record, since it became possible to produce a record with a very small team of artists using affordable computer-based tools. This paradigm shift was further escalated with the Internet increase penetration and speed, dictating the decline of traditional recording studios and the uprising of independent productions. The decay of the traditional studios and affordability of digital audio equipment gave birth to delocalization, giving artists the opportunity to produce music while being geographically apart, distributing individual tracks and mixes among project collaborators through the various production stages.

1.2. State of the Music Industry

According to the International Federation of the Phonographic Industry (IFPI) 2016 report, recorded music revenues in 2015 totalled 15$ billion, up 3.2% on 2014 and resulted in the first significant growth seen since 1998 [3][4]. Underpinning this breakthrough is the global shift of music consumption to smartphone-based mobile platforms. Digital music has moved rapidly from a fixed line desktop experience to on-the-fly consumption smartphones and tablet devices. The growth of high-speed mobile networks, more powerful smartphones and first-time sales in emerging markets is accelerating smartphone adoption over the traditional PC. Figure 1 is the prime example of this shift, comparing web usage between mobile and desktop devices.

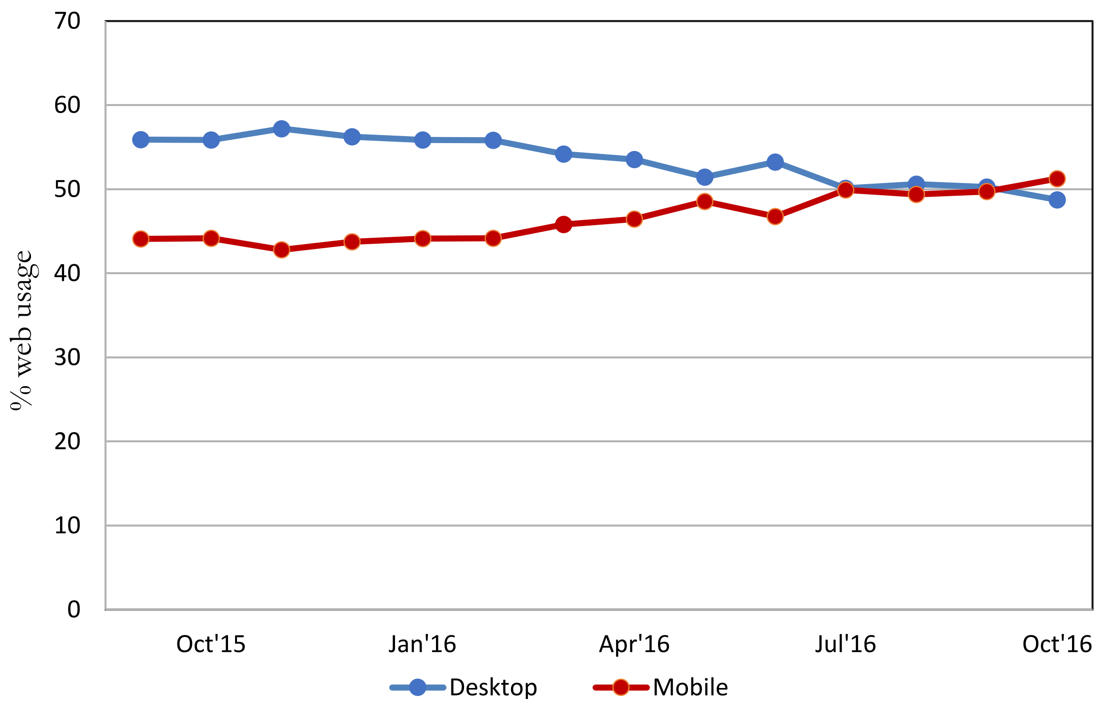


Figure 1 – Mobile and Desktop web usage

Increased competition between the Android and iOS platforms flooded the market with new services that end up widening consumer's choice. Furthermore, with smartphones now being the primary point of internet access, multiple services started focusing on developing a mobile-first offer and increasing scale through ISP partnerships.

1.3. Project

As the previous section stated, music consumption is moving towards mobile. The abundance of quality services expanded the market and boosted music industry revenue. It is only logical for music production to embrace this shift. Therefore, this internship is focused towards the development of an Android application that enables the user to record and escort a track throughout the stages of production. Moreover, to promote collaboration without the need of musicians to be physically together. Our goal is not to create a digital audio workstation, but rather a tool that lowers the barrier of entry for new musicians. This translates into the following key features:

- **Record and play audio:** Allows users to record and later play their recorded tracks;
- **Music Editing/Mixing:** Provide users with tools (metronome, pan pots and volume control) to enhance their recordings;
- **Track Version Control:** Allows users to access all versions of a track since its creation;
- **Collaboration system:** Mechanism where users can integrate new team members to their musical projects.

From this point forward, a strategy is required to establish more specific goals and processes to accomplish a well-developed product. A research of the current music industry and an analysis of the current market paradigm is a priority. This research will contribute to the requirements definition, prioritization and measure of importance.

1.4. Structure of the Report

Beyond the current chapter, the present document is comprised according to the following chapters:

- **State of the Art:** starts by introducing concepts important to the understanding of subsequent chapters. Then, encompasses the analysis of direct and indirect competitors. The last section describes the main traits of the product and compares how the proposed product contrasts with the competition;
- **3. Approach:** describes the approach followed in the internship, introducing the development methodology and all the adjacent artefacts;
- **4. Requirements:** introduces the requirements that compose the proposed product;
- **5. Application flow:** provides a walk-through over the application, explaining major features in the process;
- **6. Tools and technology definition:** presents and compares the existing audio libraries in the Android environment and lists the risks that may affect the project;
- **7. Architecture:** presents the architecture of the proposed features, and provides an overview of the technical decisions made;
- **8. Testing:** presents all the different tests conducted to the developed product;
- **9. Conclusion:** provides an overview of the work developed throughout the internship. Suggests work that can be done in the near future and presents a personal insight about the internship.

2. State of the Art

This chapter is a summary of the complete study of the State of the Art, which consists

- Basic sound concepts in section 2.1;
- Digital sound concepts in section 2.2;
- The process of music production in section 2.3;
- Competitors of the proposed product, whether indirect or direct ones, in section 2.4;
- Product definition, in section 2.5;

2.1. Sound Basic Concepts

Sound is a vibration or series of vibrations that move through the air. Anything that is capable to create vibrations or waves is referred to as the source. The source can be a string, a bell or anything that generates a vibration within our hearing range. As the vibration source moves outward from its normal resting state, it squeezes air molecules into a compressed area away from the sound source. The swept area is imbued with a greater than normal atmospheric pressure, a process called compression (Figure 2). The opposite scenario happens when the source vibration moves inward, an area with a lower than normal atmospheric pressure will be created, this processed is called rarefaction (Figure 2). As inward and outward motions occur, creating areas of higher and lower compression – waveform [5,6].

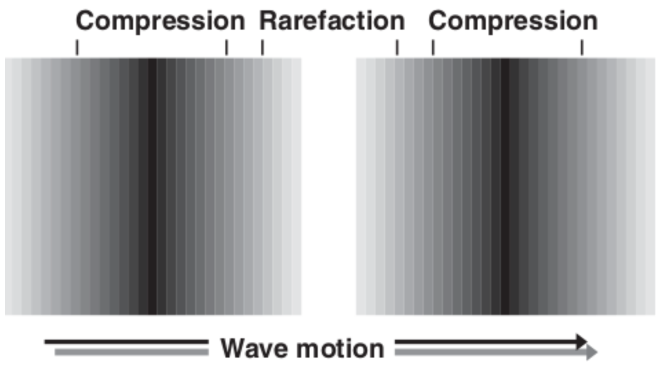


Figure 2 – Compression and rarefaction in a sound wave [5]

A waveform, as can be seen in Figure 3, is the graphic representation of the sound-pressure levels as they move through a medium over time. Each waveform is composed of the following main set of characteristics – Amplitude, Frequency and Wavelength.

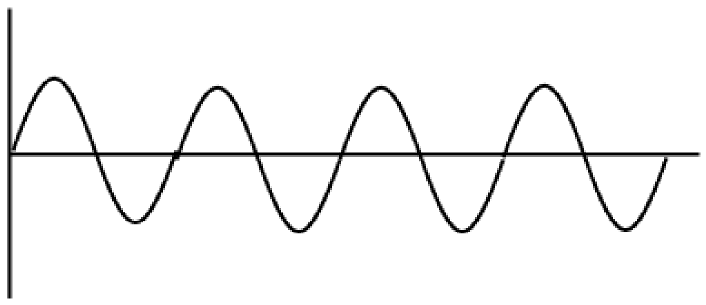


Figure 3 – Sine waveform [5]

Amplitude

Amplitude corresponds to the height of the wave. The greater the distance or displacement from the centerline, the more intense the pressure variation, electrical signal level or physical displacement will be within a medium.

Frequency

Frequency is the rate at which a vibrating mass repeats a round-trip of positive and negative amplitude. This excursion of a wave is known as cycle and the number of cycles that occur within a second are measured in hertz (Hz). The increase or decrease of the rate of vibration by the source directly impacts the number of cycles that happen per second. Frequency is related to the pitch of a sound, high frequencies result in high trebly sounds and slow frequencies translate to low or bass sound.

Wavelength

Wavelength is the length of the sound wave from one peak to the next. The wavelength can be calculated through the division between the speed of the sound and its frequency. Through this formula it is possible to conclude that the lower the frequency the longer the wavelength.

2.2. Digital Audio Concepts

Sampling

In analog audio, the signals are passed, recorded, stored and reproduced as changes in voltage continuously change over time. Digital recording does not work in a continuous manner. Instead, it gathers periodic samples of a changing audio waveform and transforms these sampled signal levels into a binary stream that can be stored for processing or reproduction.

Within a digital audio system, the *sample rate* is defined as the number of samples gather from an analog signal in one second. The sampling process can be compared to a photographer who takes a series of shots of an action sequence. As the number of pictures taken increase, so the accuracy of the captured event [7].

Quantization

Sampling is the first process of the Analog-to-Digital (ADC) conversion. After the analog signal is sampled, the amplitude levels undergo a conversion to transform a continuous amplitude sample into a discrete-time signal [8].

Quantization translates the voltage level of a continuous analog signal into a stream of bits for the purpose of manipulation and storing data in the digital domain. The process maps sample amplitude values into a set of discrete values. Each sample is converted into a codeword consisting of binary bits. The length of the codeword is called bit-depth. A higher bit-depth results in a truer representation of the sound. Resorting to the previous metaphor, a higher bit-depth would translate into a higher resolution of each shot taken.

2.3. Music Production Concepts

This section presents the main stages behind a music production process. Exploring how the entire process is conducted greatly helps the internship product definition. According to Roey Izhaki [6], the process of capturing sound into a recorded medium generally occurs into four major steps – Recording, Overdubbing, Mix-down and Mastering [6,7].

Recording

In the earlier days of music production, producers needed musicians to be gathered in one place so the final recording would make sense, in terms of levels and depth. The idea of combining multiple instruments into an appealing master was later conceived and developed by Ross Snyder, resulting in the first 8-track-recorder. From this, later evolved into a mainstream recording technique – Multitrack recording.

Multitracking is a common recording technique that allows multiple sound resources to be recorded at separate times to create a cohesive whole. The key advantage behind this is isolation. The capability of recording a single instrument to a dedicated track, greatly improves the producer's flexibility since it is possible to vary the level and spacial position without affecting adjacent tracks.

Overdubbing

After recording the fundamental of the musical piece, the process of honing it can begin. This is characterized by the attachment of new musical parts to the current project basis – overdubbing. During this phase, the previously recorded tracks are being monitored while new additions are being laid down.

Mixing Down

After recording all the tracks, producers undergo a procedure where tracks are mixed and bundled – Mixing Down. This mixing resorts to the following processes to change the characteristics of the sound:

- **Relative level:** Tunes the level intensity of each of the produced tracks;
- **Spatial positioning:** Commonly referred to as audio panning. It provides a multi-directional sense to the sound, in other terms, performs a physical placement of a sound within a stereo or surround field;
- **Equalization:** Adjusts frequency bands to produce the desired spectral characteristics. The general goal of Equalization is to make adjustments that allow tracks to inhabit their own frequency areas. This improves the overall tone quality of a recording, allowing the song to be clear and each instrument distinguishable;
- **Dynamics processing:** Defines the ratio between loudest and lowest possible audio level;
- **Effects processing:** Consists in the appliance of signal processing effects like: reverb, delay or pitch-related effects;

Mastering

Mastering corresponds to the step in which the finishing touches are made. It is performed by master engineers, who remove extraneous sounds, arrange the tracks in the most compelling order, create smooth fades and natural pauses, and balance the frequency spectrum of the various tracks so that the album sounds like a coherent piece rather than a collection of unrelated songs [9].

2.4. Competitors

The main objective of this chapter is to analyze the current music industry and the applications that compete against the proposed product. This activity allows us to ascertain the market attractiveness and dynamics, enabling a better view of where the product will fall, both in terms of functionalities and business model. It is of the uttermost importance to conduct a product comparison between the existing music collaboration applications, exposing their strengths and their weaknesses to better identify how our product can penetrate the market.

For each of these analyses, a brief description of the applications is provided, which accommodates some information about their business models, key features and rating in their respective mobile stores. The following sections contain an analysis of products, which, to the best of our knowledge, are regarded as the most prominent and popular applications. The products are presented and analyzed in an alphabetical order.

The competitors of the proposed diverge into the following two types:

- **Indirect competitors**: Applications developed in web environment that possess the main traits of a music collaboration application.
- **Direct competitors:** Applications developed in the mobile environment that possess the main traits of a music collaboration application.

2.4.1. Indirect competitors

Indirect competitors share the same market as the proposed product. Despite not being as important as the direct competitor's, its research could provide valuable insight to better tackle the market.

Blend

Blend, founded on 2013 by Paul Murphy and Alex Kolundzja, is a cloud-based music collaboration platform that integrates with the most popular Digital Audio Workstations. Blend enables the distribution of music through the main known channels (Spotify, Beatport, Apple Music) and features a Marketplace, where artists can buy and sell stems, samples and other music bits [11].

Headquarters: New York, New York, USA
Key Features:

- Projects can be stored in the cloud;
- Music can be distributed to Apple Music, Spotify, Google Play, Beatport,
- Deezer, Amazon MP3, Juno and Traxsource;
- Uses Stripe for payments;

Gobbler

Globber, founded by Jamie and Chris Kantrowitz, is cloud-based platform designed to backup, share and version control music audio files.

Headquarters: Los Angeles, CA, USA
Key Features:

- Workspace as a collaboration tool;
- Project versioning in the cloud;
- Makes money selling plugins in the marketplace;
- Web app for workspace management and desktop app for music edition;

- Supports files from Apple Logic Pro X, Ableton Live 9, and AVID Pro Tools 11;
- Uses google drive;
- Encrypt files in upload.

Kompoz

Kompoz, founded by Raf Fiol in 2017, provides a platform with strong social network features, to approach musicians from all the corners of the world to collaborate with the purpose of creating music [12,13].
<u>Headquarters:</u> Miami, FL, USA
<u>Key Features:</u>

- Supports audio editing software like Pro Tools, GarageBand, Logic Pro X, PreSonus StudioOne, Reaper and others;
- Multitracking;
- Users collaborate to a public project through appliances;
- Upload tracks;
- Users can sell their finished collaborations.

Splice

Splice, founded in October 2013 by Steve Martocci and Matt Aimonetti, is a cloud-based music creation and collaboration platform, which integrates with several Digital Audio Workstations. Splice mainly focus on version control features, enabling online and offline collaboration [14,15].

<u>Headquarters:</u> New York, New York, USA
<u>Key Features:</u>

- Digital audio workstation in the cloud;
- Collaboration platform, where people around the world can contribute to a specific user project;
- Enables music, stems sharing;
- Users can buy individual sample sounds as a model subscription;
- Also sells some of the plugins in partnership with the manufacturer;
- Presumes connection to your DAW (digital workstation) for the music production;
- Supports Ableton, FL Studio, Logic Pro X, GarageBand;
- Version control.

Comparative Analysis

After identifying and describing the main indirect competitors, we present a comparison among them in terms of provided features. This comparison is extremely important, in the sense that it enables us to find their weaknesses and strengths to better identify where our focus will be.

Before starting the comparison, it is important to identify the meaning of the symbols that will appear on the following the tables:

✓ - The application provides the feature

✗ - The application doesn't provide the feature

Table 1 - Indirect competitors feature comparison.

	Splice	SoundTrap	Blend	Gobbler	Kompoz
Collaboration	✓	✓	✓	✓	✓
Version Control	✓	✗	✗	✗	✗
Integration with cloud services	✗	✗	✓	✓	✗
Integration with social media	✗	✓	✗	✗	✗
Multitracking	✓	✓	✓	✓	✓
Upload Audio	✓	✓	✗	✗	✓
Track bouncing	✗	✓	✗	✗	✗
Music Feed	✓	✓	✓	✗	✓
DAW Integration	✓	✗	✓	✗	✗
Sample marketplace	✓	✗	✓	✗	✗
Music marketplace	✗	✗	✓	✗	✗
Music Sharing/Publish	✓	✓	✓	✗	✓
Desktop application	✓	✓	✗	✓	✗
Chat/Message system	✓	✓	✓	✓	✓
Audio recording	✗	✓	✗	✗	✗
Built-in instruments	✗	✓	✗	✗	✗

Trim	✗	✓	✗	✗	✗
Loop	✗	✓	✗	✗	✗
L - R Panning	✗	✗	✗	✗	✗
Track volume	✗	✗	✗	✗	✗
Reverb	✗	✓	✗	✗	✗

2.4.2. Direct competitors

The following applications act within the same marketplace and provide a similar service to the proposed product. Thus, researching their feature set and business model will benefit the proposed product design and definition.

SoundTrap

On April 2012, Gabriel Sjoberg, Bjorn Melinder, Fredrik Posse and Per Emanuelsson founded Soundtrap, headquartered at Stockholm – Sweden [16]. From this day forward, Soundtrap established as a cross-cut digital workstation embedded with a collaboration system. It enables music production through built-in sampled instruments and provides a collaboration mechanism that includes messaging and video chat for communication between collaborators.

Business model:
- Includes a free based model and two subscription models;
- Sells sound samples as an in-app purchase.

Key Features:
- Built-in sampled instruments (keyboards, drums, guitar/bass, synthesizer, strings, brass and woodwinds) that includes a wide range of effects;
- Music edition;
- Multitracking;
- Provides a built-in collaboration system to encourage you to team up with foreign musicians;
- Message and video chat with collaborators

App Store Rating (USA): 3.5/5 stars (21 ratings) [17]
Google Play Store: 4/5 stars (189 ratings) [18]
Play Store downloads: 10k [18]

Speazie

Speazie is a music recording platform available for both Android and iOS, that enables artists to record and share tracks within a built social network.

Business model:

- Freemium application with no in-app purchases.

Key Features:

- Upload audio;
- Overdub;
- Music sharing over social media;
- Music social feed;
- Beats/sound library.

App Store Rating (USA): 3.5/5 stars (29 ratings) [19]
Google Play Store: 4.3/5 stars (79 ratings) [20]
Play Store downloads: 5k

Spire

Spire was released by iZotope back in August 2015 [21] as a recording studio iOS application that enables artists to record, edit, mix and share their recorded tracks.

Main Features:

- 4-track-recorder;
- Metronome;
- L - R panning and volume control per track;
- Sound automatically enhanced with dynamic EQ, compression, stereo imager and a limiter.
- Business model:
- Freemium application with no in-app purchases;
- Track bouncing (0.99€) and 8-track-recorder (1.99€) are in-app purchases.

Apple Store (USA): 4.5/5 stars (83 ratings) [21]

Trackd

Developed by a group of musicians, Trackd is a music collaboration application available for iOS. Packed with a 4 to 8 multitrack recorder and a track bouncing functionality. It also provides a straightforward way to record your sound drafts, promoting collaboration.

Key Features:

- 4 to 8 track studio recording – Multitracking;
- Upload audio;
- Metronome;
- Overdub;
- L - R panning and volume control per track;
- Track bouncing;
- Collaboration through invitation and chat messaging;
- Music social feed;
- Share tracks through social media

Business model:

- Freemium application;
- Track bouncing (0.99€) and 8 track studio (1.99€) are in-app purchases.

Apple Store (USA): 4/5 stars (140 ratings) [22]

2.5. Product Definition

The project premise is to develop a collaboration tool capable of recording and provide basic tools to undergo the track through the previously stated 4 stages of production (recording, overdubbing, mixing and mastering). After the assessment of the strengths and weaknesses of the direct competitors, we had a better understanding of the direction the product should take. Table 2 provides a comparison between the proposed product feature set and the direct competitor's features.

Table 2 - Direct competitors feature comparison.

	Spire	Trackd	Speazie	SoundTrap	**Proposed Product**
Built-in sampled instruments	✗	✗	✗	✓	✗
Audio Recording	✓	✓	✓	✓	✓
Upload audio	✓	✓	✓	✗	✗
Metronome	✓	✓	✗	✓	✓
Overdub	✓	✓	✓	✓	✓
Multitracking	✓	✓	✗	✓	✓
Voice control	✓	✗	✗	✗	✗
L – R panning	✓	✓	✗	✗	✓
Volume control per track	✓	✓	✓	✗	✓
Reverb	✗	✗	✗	✓	✗
Loop	✗	✗	✗	✓	✗
Trims	✗	✗	✗	✓	✗
Vocal sync bar	✗	✗	✓	✗	✗
Version Control	✗	✗	✗	✗	✓
Fork beat/project	✗	✓	✓	✗	✗
Collaboration	✗	✓	✓	✓	✓
Chat/message system	✗	✗	✗	✓	✗
Sample/beat library	✗	✗	✓	✓	✗
Stem/track bouncing	✗	✓	✗	✓	✗
Music sharing	✓	✓	✓	✓	✗
Integration with cloud	✓	✓	✗	✗	✗
Integration with social media	✓	✓	✓	✓	✗

Music feed	✗	✓	✓	✗	✗
Desktop application	✗	✗	✗	✓	✗

Multitrack recorder

As stated before, multitracking is a method of sound recording that allows the separation of multiple sound sources or the capability of recording from different sound sources at separate times to create a cohesive whole. This feature is crucial not only to promote a correct production procedure but as well to explore collaboration. Each track would be recorded individually by a single user and the definitive version of the music would be a combination of all the collective effort.

Metronome

A metronome is a valuable tool that helps musicians keep the tempo of a song. This is another useful feature in terms of collaboration to produce a piece of music. In the sense, that it helps to keep a cohesive tempo throughout the tracks.

Collaboration Mechanism

As stated in the *Sub-chapter 1.3 – Project*, one of the core features of the proposed product revolves around creating a mechanism that instigates users to remotely collaborate in their musical projects. This mechanism revolves around a concept that we call **track application**. For the sake of the example, let's consider a musician which only plays guitar and it is looking to create a new band. The following list describes the subsequent steps through the foreseen application:

1. The musician creates a new project and describes its purpose and objectives;

2. Now remember, this musician only knows how to play guitar. To gather musicians that play other instruments, the project owner starts writing a track application. This track application is like a job opening in which the instrument, music type and play style are described. After its completion, the track application is published – meaning that every user can apply for the opening.

The list above described the perspective of a project owner, the list below describes the perspective of a future applicant:

1. The musician stumbles upon an application of his interest;
2. After reading its description, the applicant can record a sample to compete for the opening. Thus, concluding the application process;
3. If the application is accepted, the user gains access to the project and the possibility to contribute to the project

Music mixing and editing

As stated in the beginning of this chapter, our aim is to develop a platform capable of providing users with tools to undergo through the 4 stages of production, which includes the ability to mix the previously recorded tracks. The envisioned product basic mixture set includes level control and audio panning.

3. Approach

The ultimate goal of software engineering is to produce quality and maintainable software within a reasonable time and cost. This objective lack's structure and process in order to be feasible. This is where the methodology comes in, providing the framework that is used to structure, plan and control the process of developing a system. A wide variety of methodologies have evolved over the years, each with its recognized strengths and weaknesses. Identifying the best suited methodology for the task at hand is the first step towards a successful project.

This chapter starts by introducing the selected methodology and why it suits the internship product. Afterwards, portrays the methodology artefacts and addresses how those artefacts influence the planning and development process.

3.1. Methodology

Scrum is a term that is originated from rugby, where a team must cross the field from side to side to score a point. Typically, this is not achieved by oneself sweep but rather a continuously team effort. Through a series of cyclical increments, the team is able to score a point. Applying this concept in the software development industry defied the traditional sequential model, where each new piece is built upon previously created pieces.

"Scrum is a simple framework for effective team collaboration on complex projects. It provides a small set of rules that create just enough structure for teams to be able to focus their innovation on solving what might otherwise be seen as an insurmountable challenge" [23]. This dynamic nature provides the desired agility to the internship product, where features could hide a wider range of requirements and the technologies that are used are unknown and complex. In contrast to other approaches, Scrum enables its adopter to digest feedback and respond successfully to a changing environment.

3.2. Roles

The following list contains a definition of the roles present in this internship, each one of them with a well-defined purpose [24]:

- **Product owner** - The Product Owner key responsibility is to convey his vision to the Scrum Team in order for them to execute on it and build something tangible. This entity typically should bestow a deep knowledge of the product, its user, the competition and industry trends. Practically speaking, product owners perform their role by creating epics and stories in the product backlog and prioritizing them before the sprint planning meeting.

- **Scrum Master -** The Scrum Master is the entity responsible for assisting the Scrum team to achieve their goals. The responsibility of this member is to schedule meetings, check the team's progress, clarify stakeholders with the situation at hand and, remove or foresee possible risks that could threaten the project.

- **Scrum Team -** A Scrum Team can be characterized as a group of individuals who collectively work towards reaching a specific goal. After the definition of the product backlog, it is their responsibility to translate its items into the sprint backlog. Furthermore, they are left in charge with the estimation and implementation of each story of the previously built sprint backlog.

Supervisor João Monteiro has the Product Owner and Scrum Master roles, performing the duties stated above. Regarding the role of Product Owner, it was decided that the responsibilities behind the creation and management of the user stories should be delegated to the Scrum Team.

Scrum team is mainly composed of myself and André Macedo. André is responsible for the development of a web platform that includes the server-side logic that I constantly communicate with. Besides the stated members, two more actively participate – Paulo Santos and Rafaela Ferro. Paulo acts as the technical mentor and code reviewer of everything that is developed. Rafaela Ferro was mainly in charge of the user interface (UI) and experience (UX) of the application, but actively aided in the process of product design.

3.3. Artefacts

The project-planning phase using the Scrum methodology starts, like in any other, with the requirements definition. However, a requirement in methodology is expressed through a different medium that has a different name and motivation. The following sub-sections introduce this artefact and all his adjacent tools.

3.3.1. User stories

A user story is a tool designed for agile methodologies as a basis for capturing a software feature from an end-user perspective. In contrast with the typical tools, user stories are written in an informal narrative manner enabling a simple and concise description of the functionality of the requirement. Being so, a user story follows a standard structure, which contains information about these 3 elements [25]:

- **Who**: Identifies who wants the feature described in the story;
- **What:** Describes the problem the user story commits to solve;
- **Why**: Clarifies the value the user story brings to its end-user. The completion of this element strengthens the focus on the end-user and validates the user story. Also, provides the Scrum Team with an opportunity to explore other ways of reaching that goal.

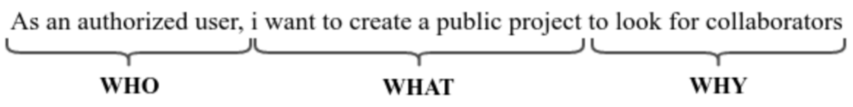

As an authorized user, i want to create a public project to look for collaborators
WHO
WHAT
WHY

Figure 4 – User story template

User stories reveal themselves as a powerful instrument with the distinguish factor of describing the requirement functionality instead of its implementation. Their usage forces difficulty estimation and creation of acceptance criteria. This estimation helps the Scrum Team to choose the user stories to be implemented during a sprint.

Typically, the creation and administration of the user stories are a responsibility of the Product Owner. Yet, for the reasons stated in the previous section, and together with the importance of learning the methodology, the Scrum Team was bestowed of this task.

The definition of all the User Stories that constitute the internship product can be found in *Chapter 4 – Requirements*.

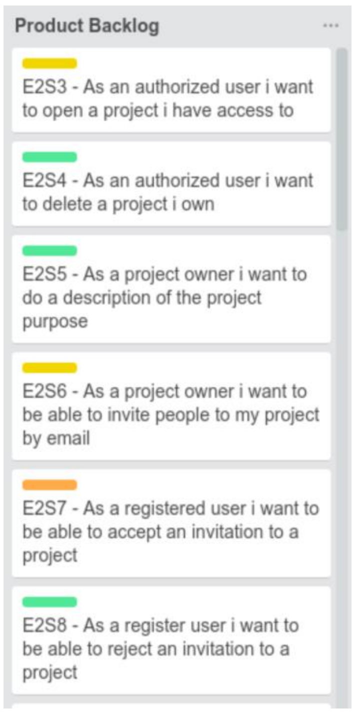
3.3.2. Story points

After defining the user stories, each member of the Scrum Team has the responsibility of estimating their own user stories. Story points emerge as a tool to help measuring the size of a user story. The metrics involved include the complexity of the feature, the effort needed to develop the feature and the risk in developing it [26].

In order to address the degree of difficulty inherent to the user story estimation, the Fibonacci sequence was adopted, which is characterized by the fact that every number after the first two is the sum of the two preceding ones. The sequence ranges from 1 to 13 (1, 2, 3, 5, 8, 13) and each one of these numbers represents the range of choice available to label the complexity of a user story. This scale tremendously helped in the estimation process, granting some standardization to the story point estimation. This method takes into account the relative complexity of each user story and assigns accordingly. A rough estimation can then be made, thus leading to an approximation of how resources the project will consume. This will eventually improve sprint estimation and maintain a realistic forecast throughout the project.

3.3.3. Product backlog

Product Backlog emerges as the prioritized collection of all the user stories that compose the product. As said before, the Product Owner is the entity responsible for this prioritization since it is the member that can better identify the product's needs. Figure 5 below shows a piece of the proposed product backlog.


Product Backlog

E2S3 - As an authorized user i want to open a project i have access to

E2S4 - As an authorized user i want to delete a project i own

E2S5 - As a project owner i want to do a description of the project purpose

E2S6 - As a project owner i want to be able to invite people to my project by email

E2S7 - As a registered user i want to be able to accept an invitation to a project

E2S8 - As a register user i want to be able to reject an invitation to a project

Figure 5 – Product backlog example

User Stories atomic and concise structure grants a dynamic nature to the Product Backlog, enabling its continuous evolution to meet the product needs without the overhead typically found with other techniques.

The complete product backlog is available in *Chapter 4 - Requirements*.

3.2.4. Sprint

After the product backlog is minimally defined, the Scrum Team can start working in the product. This will be done in sprints, which are a set of previously timed stages that allow the team to develop and convey a small portion of the product – deliverable. In this specific case, the defined sprint duration is 1 week, enabling a periodically progress checking and guidance with Supervisor João Monteiro.

The user stories to be performed in a sprint are entries in a sprint backlog. The sprint backlog is the selected set of stories for the current sprint. This selection constitutes the functionalities that Scrum Team proposed to complete, in order to convey the deliverable. A user story is only label as "Done" when is complying with the team Definition of Done. Regarding the proposed product, the Definition of Done is the confirmation of the following evidence:

- Aside from the obvious need of the implementation, each module must possess and pass to all its unit and UI testing;

- Then and only then, the user story implementation is subjected to a code review. This job is conducted by Paulo Santos and aims to evaluate the code structure.

3.2.5. Showcase

Showcase is essentially a sprint meeting. Held on the last day of the sprint, the Scrum Team and the Scrum Master gather in order to discuss the following topics:

- **Review:** The Scrum Team shows what has been done in the current sprint, as well as discuss the sprint challenges and their status.
- **Planning:** The showcase final stage is comprehended by the definition of the user stories that will compose next week's sprint backlog.

With regard to this project, both trainees and supervisor João Monteiro gather every Friday to conduct the Showcase with all previously described stages.

4. Requirements

The prior chapter introduced user stories and how this technique is used at Deemaze Software. In this chapter, the context switched towards the structure and content of the user stories that ultimately describe the product. There are two key tools when building a product backlog:

- **Epics:** Reflect larger bodies of work. Epics represent sections of the application and are composed by multiple user stories. The requirements are divided in 6 Epics – Authentication, Project, Tracks, Mixing, Track application and Version control.

- **User Stories:** High-level definition of a requirement. User Stories are written as an informal narrative, providing a simple and concise description of the requirement. Each User Story follows the MoSCoW prioritization method in order to properly define each user story value;

As stated before, each user story goal is to deliver value to the user. However, the target of each user story is not always the same. The following list exhibits the different roles that currently intervene:

- **Unregistered user:** An anonymous user that is not registered;
- **Registered user:** A user that is registered in the platform but it is not logged;
- **Authorized user:** A registered user that is logged in the platform;
- **Project owner:** An authorized user who is owner of a project;
- **Project member:** An authorized user who is owner or collaborator of a specific project;

The following subsections introduces the epics that comprise the proposed product, their user stories and respective estimation.

4.1. Epic#1 – Authentication

As can be seen in Table 3, the Authentication epic is comprised of 3 user stories.

Table 3 – User stories of the authentication epic

#	Story	Points	Priority
1	As an unregistered user, I want to register in order to create my profile and access the application	5	Must have
2	As a registered user, I want to login in order to get application access	3	Must have
3	As a register user, I want to be able to recover my password in case I forgot it	3	Could have

4.2. Epic#2 – Project

As can be seen in Table 4, the Project epic is comprised of 7 user stories.

Table 4 – User stories of the project epic

#	Story	Points	Priority
1	As an authorized user, I want to create a project in order to start my new music project	5	Must have
2	As an authorized user, I want to access all the projects I own in order to operate over them	8	Must have
3	As an authorized user, I want to be able to see the project bio in order to check the current members and description	5	Could have
4	As an authorized user, I want to be able to see the project current tracks in order to perform operations over them	5	Must have
5	As a project owner, I want to be capable of deleting a project I no longer need	3	Should have
6	As a project owner, I want to create and edit a description of the project in order to better convey the objective the project is aiming for	3	Should have
7	As a project owner, I want to kick user I no longer desire to be part of the project	3	Should have

4.3. Epic#3 – Tracks

As can be seen in Table 5, the Tracks epic is comprised of 8 user stories.

Table 5 – User stories of the tracks epic

#	Story	Points	Priority
1	As a project member, I want to record a track in order to create audio content to my project	13	Must have
2	As a project member, I want to play the track I recorded in order to see the track final result	13	Must have
3	As a project member, I want to be able to start metronome before recording in order to set the tempo of the sound I am about to record	13	Must have
4	As a project member, I want to see the waveform of the project audio	5	Must have
5	As a project member, I want a countdown timer in order to give me time to get ready before the recording	8	Must have
6	As a project member, I want to pick which tracks I would like to hear while recording	13	Must have
7	As a project member, I want to play all tracks of a project so that I can hear the current state of the project	8	Must have

8	As a project member, I want to delete a previously recorded track because I am no longer interested in it	8	Must have

4.4. Epic #4 - Track application

As can be seen in Table 5, the Track application epic is comprised of 7 user stories.

Table 6 – User stories of the track application epic

#	**Story**	**Points**	**Priority**
1	As the project owner, I want to create a track application in order to show my interest in integrating new musicians to my project	5	Must have
2	As an authorized user, I want to be able to search for projects by name or instruments in order to ease the task of finding a project of my interest	8	Must have
3	As an authorized user, I want to record a track in order to apply for a track application	8	Must have
4	As an authorized user, I want to see the status of a track application in order to receive feedback about an application I applied for	5	Must have
5	As a project owner, I want to be able to listen to all the track applications in order to pick the one that better fits my project	13	Must have
6	As a project owner, I want to accept a track application in order to integrate a new musician into my project	8	Must have
7	As a project owner, I want to reject a track application if the applied track does not meet the criteria I am looking for	3	Must have

4.5. Epic #5 - Mixing

As can be seen in Table 5, the Track application epic is comprised of 3 user stories.

Table 7 – User stories of the mixing epic

#	**Story**	**Points**	**Priority**
1	As the project owner, I want to be able to adjust the gain of each one of the project tracks in order to emphasize the desired sound	13	Must have
2	As the project owner, I want a pan control in order to bestow a multi-directional sense to the sound	13	Must have
3	As a project member, I want to trim a undesired piece of a track	13	Won't have this time

4.6. Epic #6 – Version Control

Table 8 – User stories of the version control epic

#	Story	Points	Priority
1	As a project member, I want to play a track and all its milestones in order to hear all the track versions	13	Must have
2	As a project member, I want to travel to a previous milestone of a specific track in order to revert the change of a track	13	Must have

5. Application flow

After laying down the desired features of the product, the next step was to prototype it. This would not only help better envision the product but also served as accompaniment of the user stories, helping establish acceptance criteria for the functional requirements.

This chapter aims to describe the most important flows of the application, matching the order previously found in the *Chapter 4 – Requirements*.

5.1. Project

The flow depicted in Figure 6, focuses on the perspective of the Project Owner and which actions can be made. After the user successful authentication, a screen is prompted with the projects the user owns. From this point forward, the user can press the yellow floating action button to create a new project or navigate to an already created project.

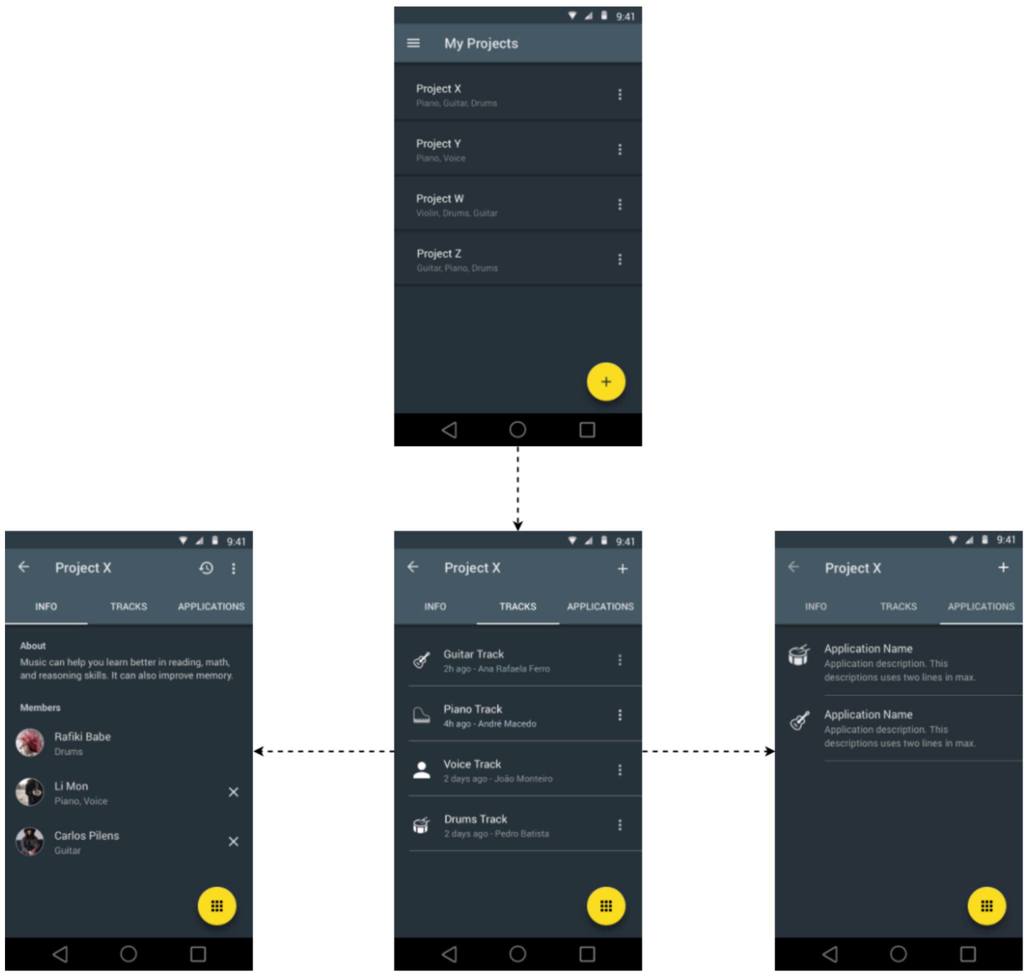


Figure 6 – In-depth navigation of a project

The section shown on the left (Figure 6) starts by exhibiting the project biography, which briefly describes the objective of the project and could potentially display important data to the other musicians, such as the tempo of the song. A team roster is also provided, describing the current collaborators and their respective role in project.

Swiping right leads to the middle screen which displays the current project tracks. This screen serves as a proxy to the following actions:

- start a new recording;
- patch a previously recorded track;
- play a track and all its version;
- delete a previous track.

The last section introduces the medium used to instigate collaboration - Track Applications. These applications are created by the project owner, briefly describing the owners vision, which will be later visible to all users that are not a part of current project roster. This topic is further explored in the flow described in *Section 5.4 – Track application*.

5.2. Record & Metronome

The concept behind this flow was to provide a set of tools that enables the user to properly record. Besides the obvious record mechanism, those set of tools include a metronome and waveforms of the produced sounds.

A metronome is a mechanism frequently used to enforce musicians to follow a specific tempo. This enforcement is done through audible ticks or beats at regular intervals that the user can set in beats per minute. Since the aim is focused towards collaboration, features that enhance the musicians timing and synchronization are a must. Waveform also aids in this honing, since they visually equip musicians to better time their actions.

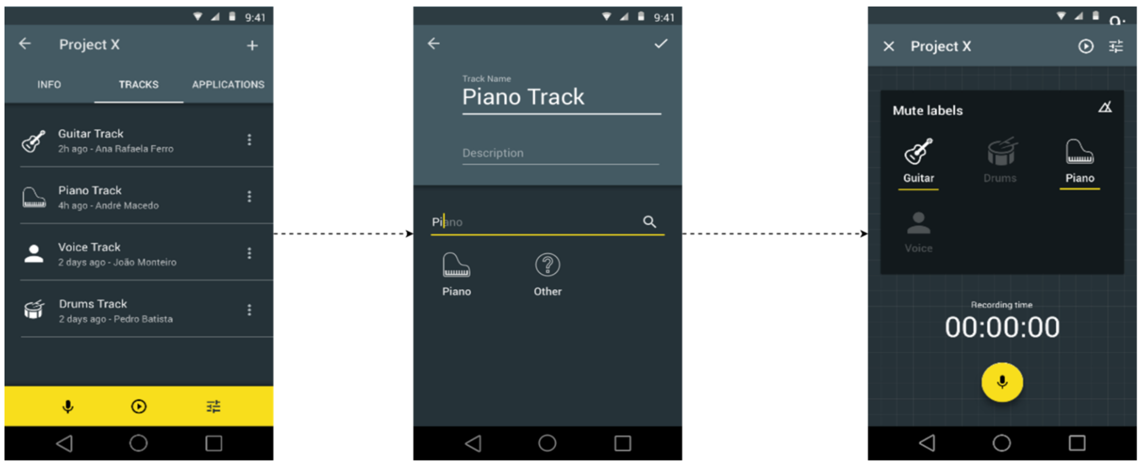


Figure 7 – Name recording and pick track flows

The flow depicted in Figure 7 describes the process behind a track creation. Pressing the bottom left record icon displayed in the first screen, prompts the user to a screen where the naming and instrument picking will take place. This instrument is a mere label to help distinguish tracks within the project. When this preliminary step is completed, the last screen of Figure 7 is presented. From this screen, the user can take the following actions:

- **Select tracks:** In darker rectangle, the user is able to pick which project tracks are going to be played during the recording. The screen displayed above shows 2 selected project tracks.
- **Metronome:** Pressing the metronome icon placed in the right upper corner of the dark blue rectangle prompts the user to a screen where the metronome can be tuned. This is screen is displayed in Figure 8.
- **Recording:** The yellow float action button triggers a countdown timer and starts the recording phase.

As can be seen in the Figure 8, the metronome has multiple settings that can be defined:

- **Beats per minute:** Sets the frequency that the metronome plays;
- **Tap first beat:** Checking *tap first beat* stresses the first beat played by the metronome;
- **Beats per bar:** Switches from stressed to unstressed beats according to the defined value. Stressed and unstressed metronome beats are typically used when the user wants the metronome wants to mimic the song rhythm.

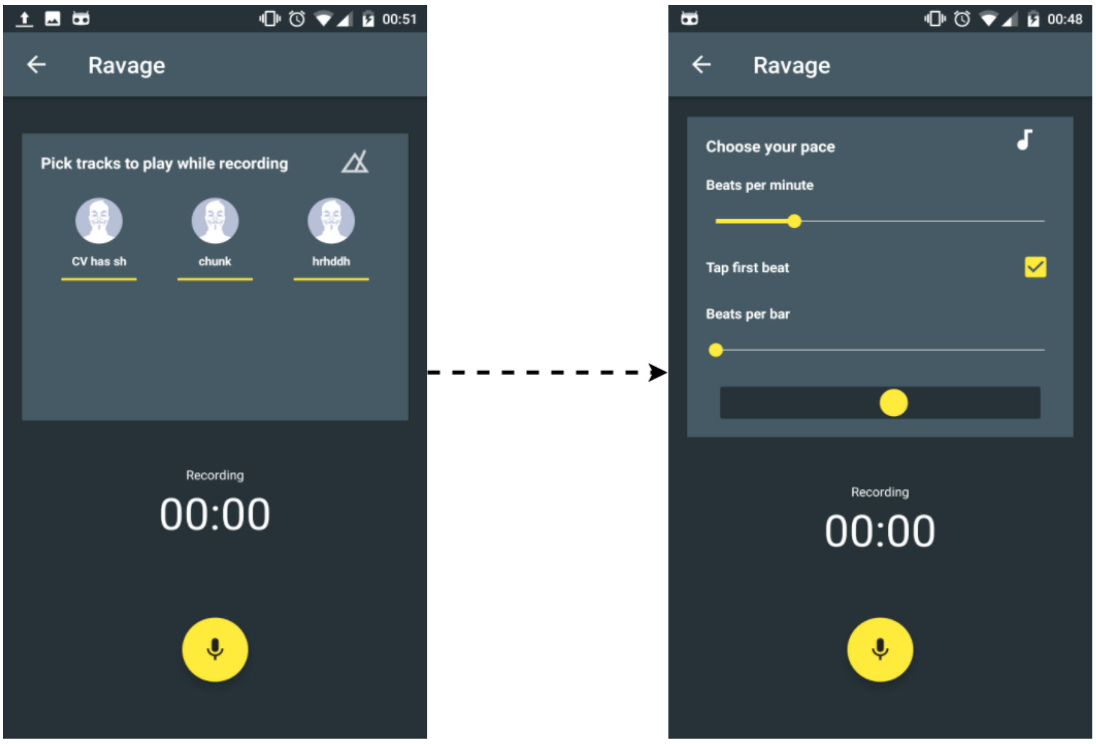


Figure 8 – Metronome screen

Figure 9 shows the flow after the user presses the recording button. Pressing the record float action button pops a countdown timer, in order to let the user prepare for the recording that is about to take place. After the countdown ends, the recorder and metronome animation start. Along with the metronome animation, a waveform of the previously selected project track is rendered. Once stopped, the user can playback the final result of his recording. The playback screen displays 2 waveforms: the upper one displaying the project tracks selected in the beginning of the flow and underneath the waveform of the recorded track. From the playback screen, the user can play, stop and repeat the audio at any given time. While stopped, two side icons appear enabling the user to remove or save the recording.

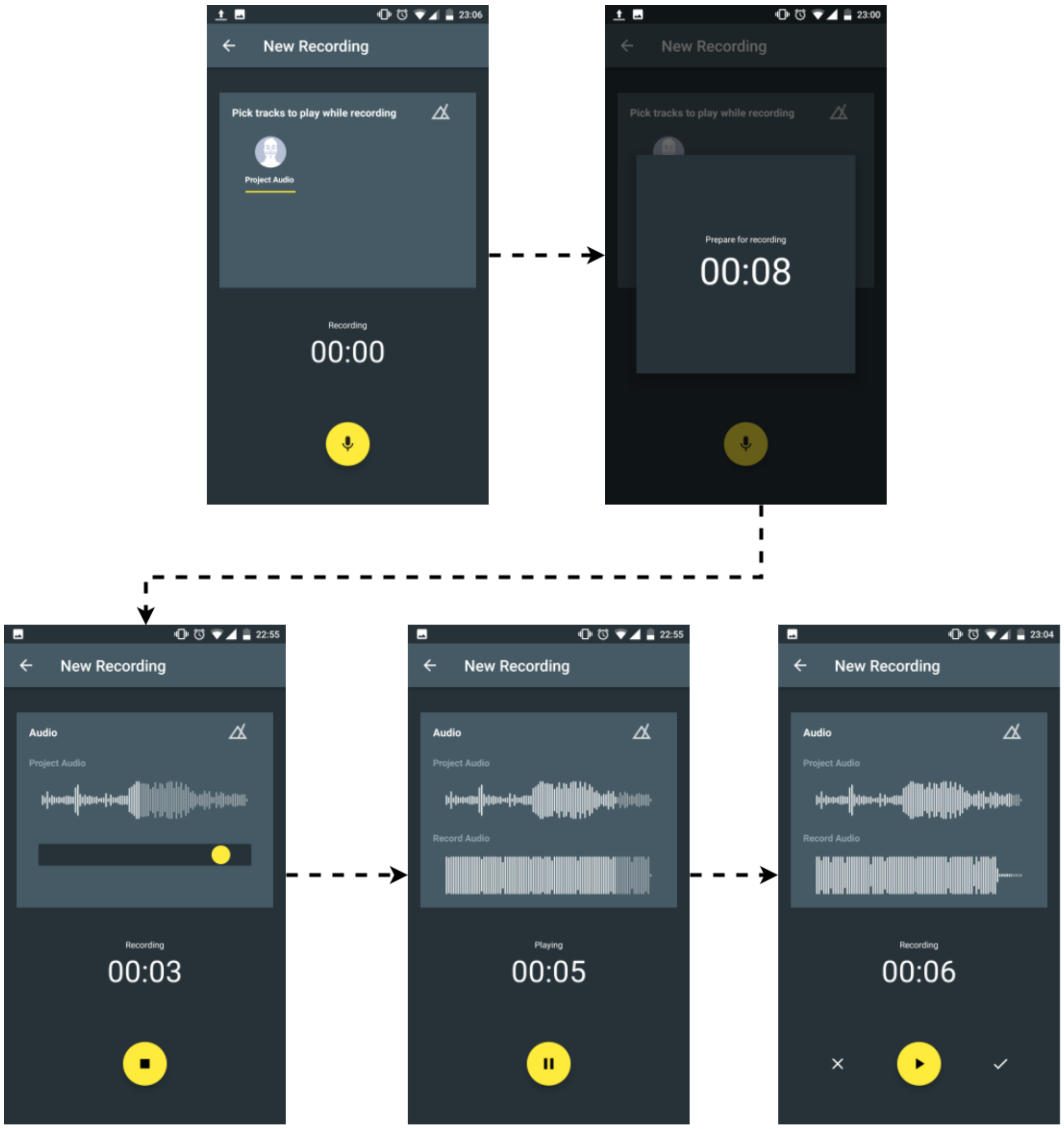


Figure 9 – Record flow

5.3. Project Player

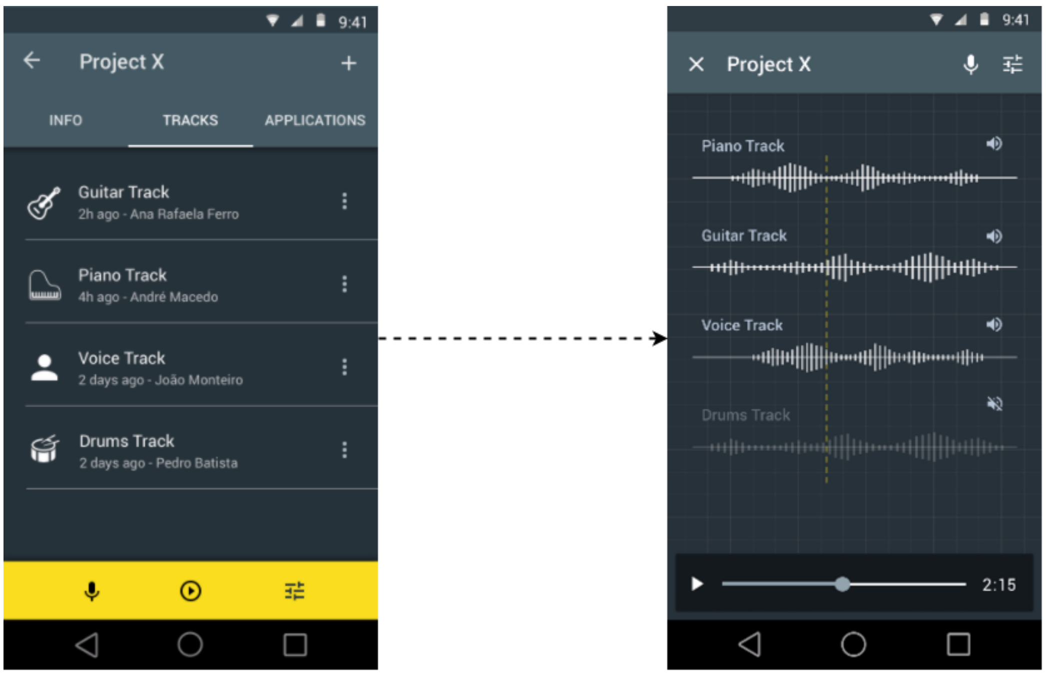
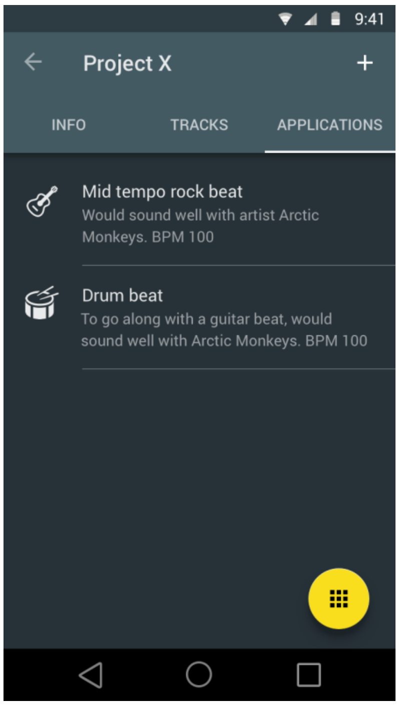


Figure 10 – Player screen

Pressing in the middle button of the bottom menu redirects the user to the project player, where every track of the selected project is fetched. In contrast with the track player displayed in Figure 15, the use case in this flow is to capacitate the user with a player to hear the result of the mix down or simply track the current state of the project.

5.4. Track Application

The aim behind the track applications is to provide the ability to easily tap performers around the globe and spark collaboration between them. The process starts with the project owner interest in integrating new musicians using track application as the medium to convey his intent. This application, illustrated in Figure 11, briefly describes what the owner is looking for in the beat and what will be the contribution of the user to the project.

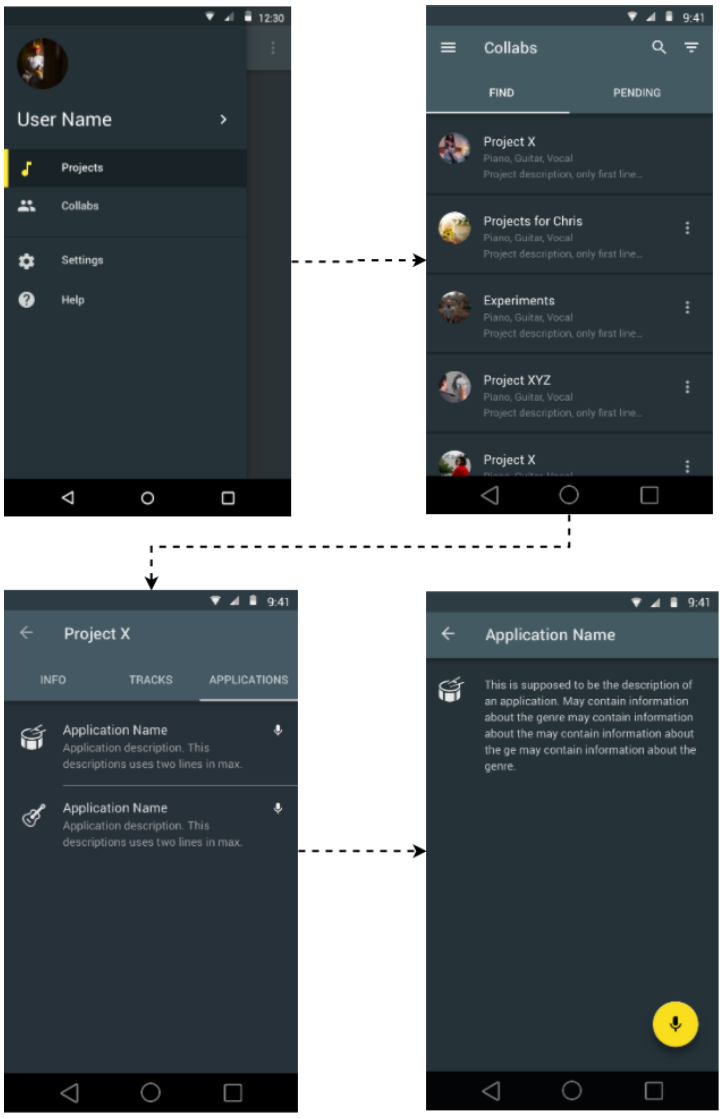


Mid tempo rock beat
Would sound well with artist Arctic Monkeys. BPM 100

Drum beat
To go along with a guitar beat, would sound well with Arctic Monkeys. BPM 100

Figure 11- Track application (Project owner perspective)

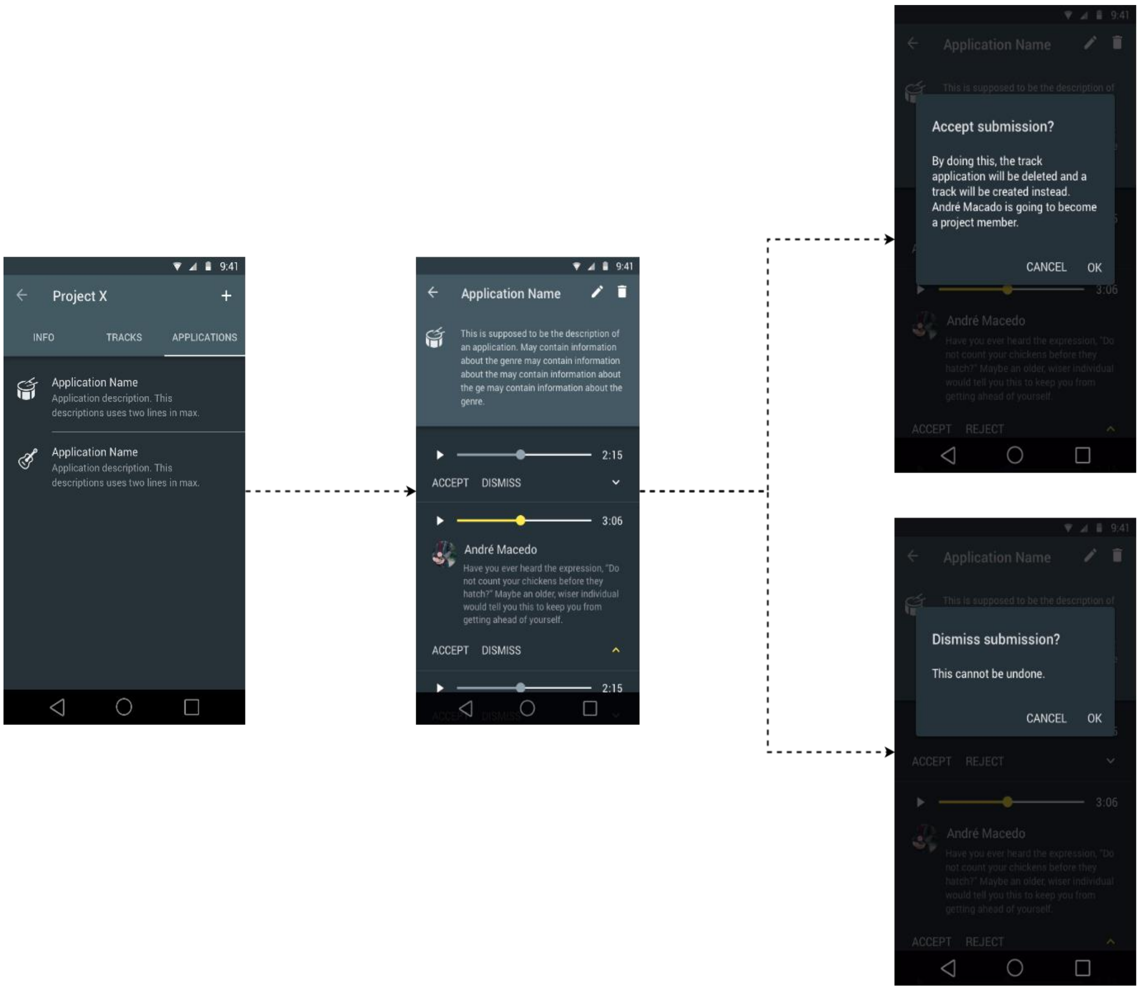
The track application flow from the collaborator perspective starts with the user searching for collaboration, section which could be accessed through the navigation drawer displayed with three horizontal stripes in figure 5 upper screen. After pressing the *Collabs* sections, the user is presented with all the projects that have open track applications. Then, the musician can pick which track application he has more interest in and start recording his application.



User Name
Projects
Collabs
Settings
Help
Collabs
FIND
PENDING
Project X
Projects for Chris
Experiments
Project XYZ
Project X
Piano, Guitar, Vocal
Project description, only first line...
Project X
INFO
TRACKS
APPLICATIONS
Application Name
Application description. This descriptions uses two lines in max.
Application Name
Application description. This descriptions uses two lines in max.
Application Name
This is supposed to be the description of an application. May contain information about the genre may contain information about the may contain information about the ge may contain information about the genre.

Figure 12 – Apply to a track application (Collaborator perspective)

The flow illustrated in Figure 13 displays the project owner hearing every submission made to each of his open track applications. The right screen shows the act of accepting or dismissing a contribution. If the track application is accepted, it closes and the author of the submission is integrated into the team roster. The recently integrated user can only record new tracks or edit the ones of his authorship.



Application description. This descriptions uses two lines in max.

This is supposed to be the description of an application. May contain information about the genre may contain information about the may contain information about the ge may contain information about the genre.

Have you ever heard the expression, "Do not count your chickens before they hatch?" Maybe an older, wiser individual would tell you this to keep you from getting ahead of yourself.

Accept submission?

By doing this, the track application will be deleted and a track will be created instead. André Macado is going to become a project member.

Dismiss submission?

This cannot be undone.

Figure 13 – Track submissions (Product owner perspective)

5.5. Mixing

Figure 14 describes the mixing flow of the application. As previously mentioned, mixing is the last step in a music production process, where the producer applies the final touches to the audio. The aim with this flow was to provide the user with the capability to adjust the relative level and spatial position of the previously record tracks, thus granting the desired audio hierarchy. The vertical equalizer, displayed in the Y-axis, lets the user define the gain of the project current selected track. The pan control, displayed in the X-axis, spreads the audio signal between the right and left channel, giving a multi-directional feel to the audio. The scatter chart complements the latter by providing visual aid to the current audio hierarchy.

In the music production, the mixdown procedure is delegated to the producer, who is in charge of overseeing and managing all the stages of production, including mixing stages. In the application, this role is assigned to the project owner since it is the member who came up with the vision for the project.

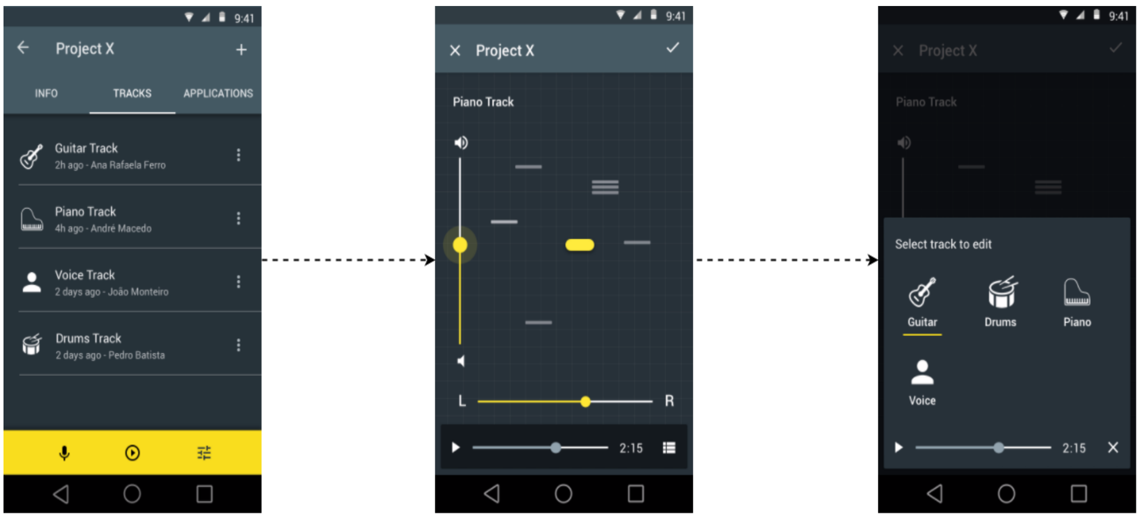


Figure 14 – Mixing flow

5.6. Track Player

Within the track player, users can travel between all the versions of a specific track. As can be seen in Figure 15, each version corresponds to a point in a timeline. We called these entries *history points* and they are added to a timeline when a track undergoes a new mix or recording. Users can browse, listen and activate each history point. In Figure 15, the *Guitar Track* timeline has 3 history points with the *update_new_guitar.wav* as the active version of the track.

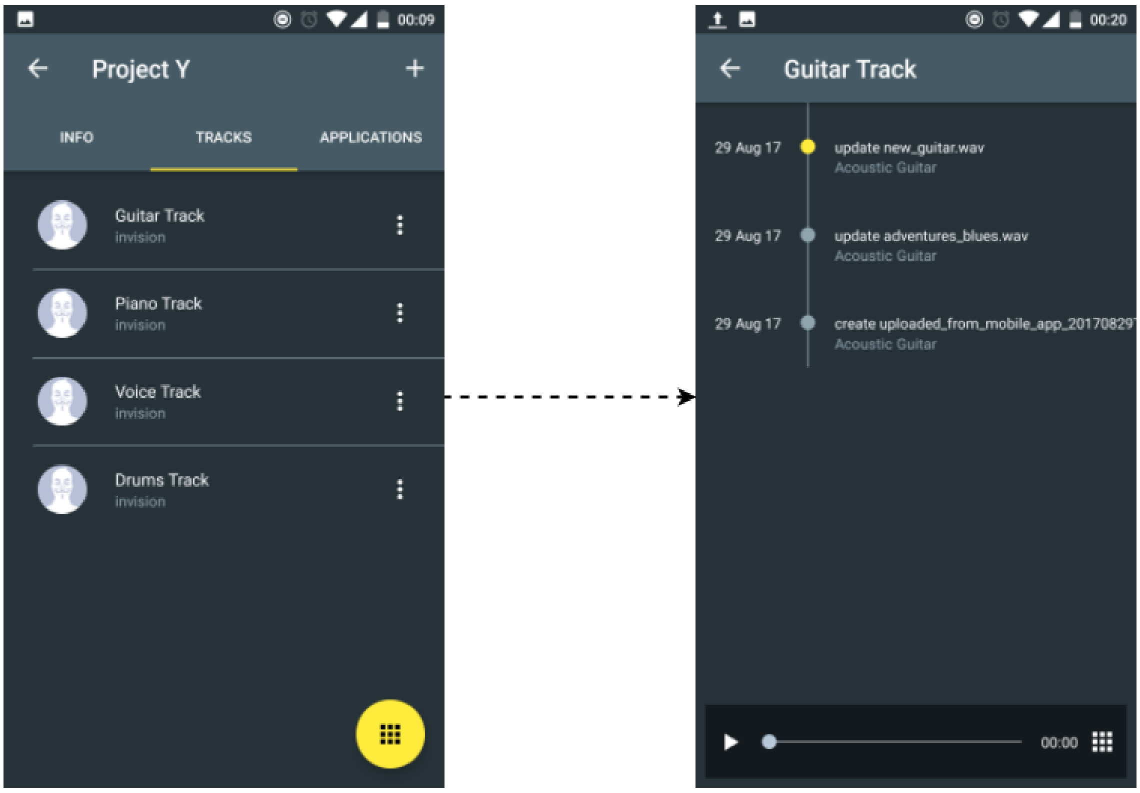


Figure 15 – Track history

6. Tools and technology definition

The first section of this chapter explores and compares the existing android audio libraries. The following sub-chapter presents the risks identified during the internship and the respective mitigation plans. *Section 9.1 – Progress Overview* discusses how the identified risks affected the project and how they were resolved.

6.1. Audio Libraries

This section offers a broad view of the potential libraries that could be adopted to create the media player and audio manipulation logic.

The first two options can be considered as low-level audio libraries because they interact directly with the memory buffer instead of using file URIs, which provides greater control over audio and recording. However, this flexibility comes at cost, which is debated in each respective section.

The last two options, MediaPlayer and Exoplayer, are built on top of the Android low level media libraries and offer a high-level approach to implement a media player.

6.1.1. AudioTrack

The AudioTrack provides a way to manage and play a single audio resource. The audio resource is read and written to an audio sink that AudioTrack controls. Unlike the remaining libraries, AudioTrack can only read PCM data. Since PCM data does not hold audio attributes (sample rate, channels, bit-depth), it is the developer's job to tune it according to the audio that it is about to be written.

AudioTrack operates under two modes – static and streaming. Streaming mode, as the name suggests, is meant for streaming purposes and it is used when playing blocks of audio data that either are [27]:

- too big to fit in memory because of the duration of the sound play;
- too big to fit in memory because of the characteristics of the audio data;
- Received or generated while queued audio is playing.

In the static mode, AudioTrack is in possession of the whole audio data. Which means the audio sink size is the same as the audio file and it is written in only one take.

6.1.2. Soundpool

Soundpool aim is pointed towards low latency audio. Audio is decompressed and decoded into a raw 16-bit PCM mono or stereo stream and then kept in memory for quick access. SoundPool play method is non-blocking thus enabling the desired use case of playing multiples sounds at the same time. Also provides built-in methods like loops, playback rates and left to right volume which could also be used in some of the desired use cases of the application. However, Soundpool is restricted to files up to a maximum of 1MB, which makes SoundPool not fit to record audio with long durations [28].

6.1.3. MediaPlayer

MediaPlayer class can be used to control plackback of audio/video files and streams. It was designed for longer sound files, which is best suited for music or larger files. The files will be loaded from disk each time create method is called, this will save on memory space but introduce a small delay. This matches the product needs, since there is no file size threshold and we are not interested in buffering the music files [29].

6.1.4. Exoplayer

Exoplayer is an open source application level media player adopted by Google in YouTube. Exoplayer is composed by a range of modular components, each one providing a different behaviour to the player, be upon a Network, Buffering, Extraction, Decoding and Rendering. Since its not of our interest to stream audio, we can cross the Network and Buffering modules. Rendering module uses AudioTrack for playing audio and MediaCodec to decode the audio sample. Exoplayer provides an implementation for this type of scenarios called ExtractorSampleSource. In a similar way to MediaPlayer, Exoplayer doesn't have file thresholds or limit of media players. This makes Exoplayer a plausible option for the desired implementation [30].

6.2. Risks

A proactive risk management through the software development lifecycle is an important process that increases the project success probability. Risk management is a series of steps whose objectives are to identify, address and eliminate software risk items before they become either threats to successful software operation or a major source of expensive rework [31]. The procedure included in the [31] risk management is comprehended by the following steps:

1. **Risk Identification:** The step name is self-explanatory. It revolves the identification of the possible that could affect the proposed product;
2. **Risk Analysis:** From the identification, was assessed the loss-magnitude and loss probability associated with each of the identified risks;
3. **Risk Prioritization:** Each risk was labelled due to his type. This allowed the appraisal of the product major source of risk;
4. **Risk Management Planning:** This step accommodated the production of mitigation plans to address each one of the identified risks.

The dynamic nature of the product backlog surfaces the necessity for a periodic risk monitoring. The following list describes the risks identified up to this point:

ID: 1
Type: Infrastructure Failure
Title: Offline Infrastructure
Description: Since the application heavily relies on the server-side - which is stored in a third-party infrastructure - it's not possible to guarantee a precise availability of that infrastructure.
Impact: Very High
Probability: Low
Mitigation Plan: The application will analyse all the responses provided by the HTTP requests in order to assess the availability of those services and produce a correct response. The infrastructure described in the section 5.1 is a solution to increase availability.

ID: 2
Type: Version Control Failure
Title: Version Control Conflicts
Description: The application manages the commits and merges seamlessly, which could trigger version conflicts.
Impact: High
Probability: Medium
Mitigation Plan: The application will analyse all the responses provided by GitLab requests and warn the user about the emerging conflict, forcing the last commit to entry.

ID: 3
Type: Library Dependencies
Title: Modifications on the Libraries
Description: Considering some features have dependencies, there is a risk of those libraries being updated. This could jeopardize the application execution in some devices.
Impact: Medium
Probability: Medium
Mitigation Plan: Rely on a different library to pursue the same goal or update the current library.

ID: 4
Type: Audio Performance
Title: Low quality microphones
Description: Android system is used by various manufacturers and, as such, the quality of the hardware can vary. This could lead to low quality sound recording.
Impact: Depends on the device
Probability: Depends on the device
Mitigation Plan: In this eventuality, the user is advised to use an external microphone in order to improve the audio quality.

ID: 5
Type: Audio Performance
Title: Audio Input Latency
Description: Due to the small size of the microphones, the overall quality of acoustics is poor. To tackle this, signal processing is introduced to improve the overall quality of the sound. This signal processing adds latency [32].
Impact: Depends on the device
Probability: High
Mitigation Plan: Advise the user to use a headset. Since the sound is recorded at a higher quality, this saves the effort of the signal processing - lowering latency.

ID: 6
Type: Audio Performance
Title: Multitracking
Description: MediaPlayer is heavy when compared to the other libraries. Thus, there is a risk of MediaPlayer not fulfil the desired use case of playing multiple sounds simultaneously.
Impact: Very High
Probability: High
Mitigation Plan: Switch to AudioTrack or ExoPlayer.

7. Architecture

This chapter explores the architecture of the product in all its perspectives. The process started with a deep analysis about the purpose and context of this product, as well as the defined requirements.

The following list contains a brief description of this chapter structure. Each item of this list reflects the current perspectives of the product architecture.

- **Deployment View:** This sub-section defines the physical environment in which the system is intended to run;
- **Container view:** Illustrates a high-level technology picture of the internship product;
- **Clean Architecture:** A generic and theoretical approach of the adopted architecture;
- **Android Clean Architecture:** The Android translation of the previous stated architecture identifies the architecture flow and all its components.

7.1. Deployment View

Figure 16 introduce a high-level diagram of the development view. Despite only being responsible for the Android Client module, it is important to have knowledge of the overall infrastructure architecture in order to properly discuss with André.

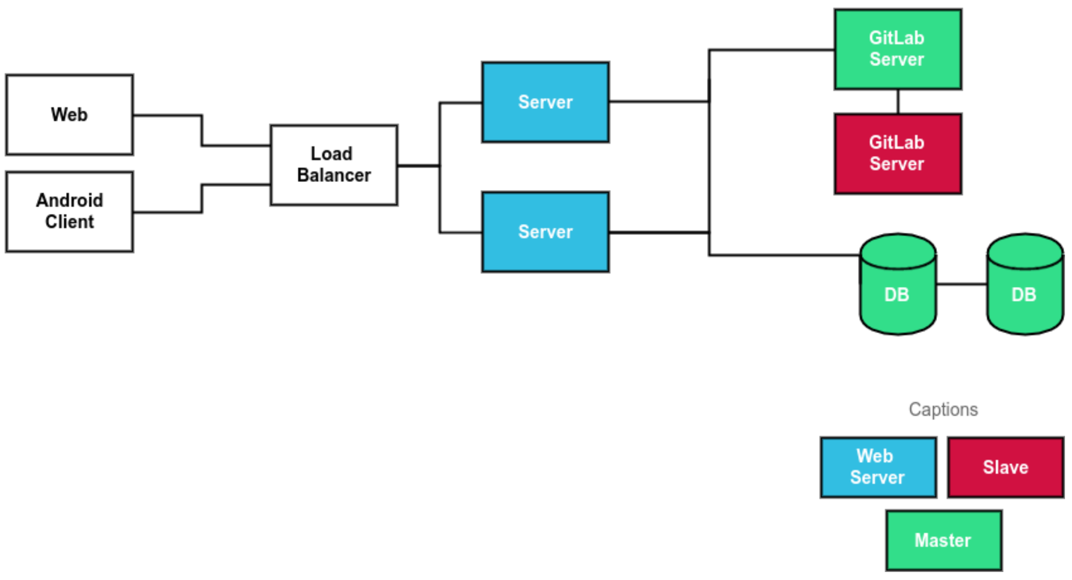


Figure 16 – Deployment view diagram

7.2. Container View

The Figure 17 presents a high-level of the communication of the entire platform (mobile and web).

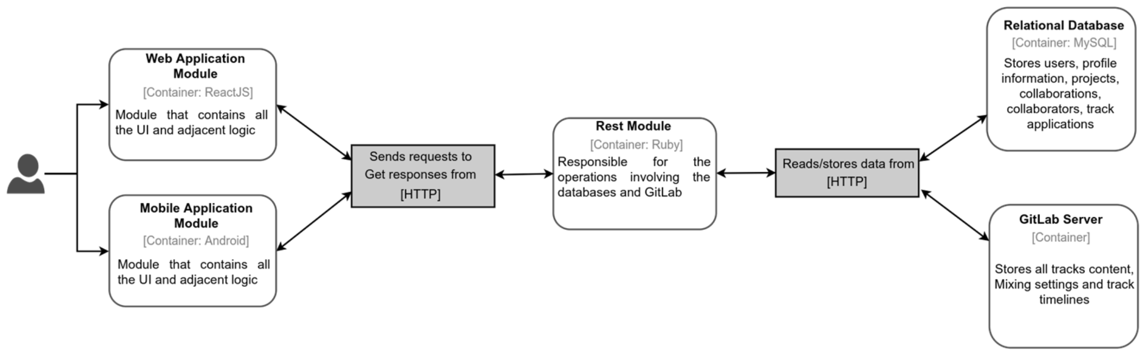

Web Application Module
[Container: ReactJS]
Module that contains all the UI and adjacent logic
Mobile Application Module
[Container: Android]
Module that contains all the UI and adjacent logic
Sends requests to Get responses from [HTTP]
Rest Module
[Container: Ruby]
Responsible for the operations involving the databases and GitLab
Reads/stores data from [HTTP]
Relational Database
[Container: MySQL]
Stores users, profile information, projects, collaborations, collaborators, track applications
GitLab Server
[Container]
Stores all tracks content, Mixing settings and track timelines

Figure 17 – Container view diagram

The Web and mobile application component are responsible for displaying the information to the user. Furthermore, both components are responsible for formatting the data and send to the REST Module capable of concluding the request. Additionally, in the mobile application case, this is where all the logic behind the sound features will be.

The REST Module is only responsible for the connection with the databases and the GitLab Server. The database is responsible for storing: users, projects, applications, track labels. On the other hand, GitLab stores all the project tracks, all their versions and mixing settings

7.3. The Clean Architecture

Before debating the proposed architecture, it is extremely important to clarify why architecture is crucial to develop a quality product. According to Simon Brown [33], architecture is about the decomposition of a product into a collection of components/modules and interactions. To properly do this it is essential to capture the product intent, because this intent is what defines the software genetic composition. This is not a straightforward procedure and it is often approached with the wrong mindset. One of the most common mistakes in this practice is the definition of architecture based on the tools that are being used. This is fundamentally wrong, tools should only be a detail in the system. Otherwise, the tools will overlap the business rules and dictate the application - "The screws in your house don't define its architecture". Robert C. Martin exhibited Clean Architecture, as ground of practices that produce systems that are [34, 35]:

- **Independent of frameworks:** As stated, frameworks should be used as tools rather than having your system constrained by them;
- **Decoupled from any external agency:** Business logic does not need knowledge of any external tools. From this decoupling, it is possible to swap tools with relatively ease;

- **Testable:** This separation of concerns eases business logic testing without UI, Database, Web Server, or any other external dependencies.

To properly achieve a system with the above features, the architecture must take in a account the following key principles [36]:

- **Dependency Rule:** Source code dependencies can only point inwards, which means inner circles do not have knowledge about outer circles. That includes functions, classes, variables or any other entity.

- **Single Responsibility Principle:** Each component or module should be responsible for only a specific feature or functionality.

- **Prefer composition over inheritance:** Wherever possible, use composition over inheritance when reusing functionality because inheritance increases the dependency between parent and child classes, thereby limiting the reuses of child classes.

When designing an application or system, the goal of a software architect is to minimize the complexity by separating the design into different areas of concern. Within each area, the designed components should focus on that specific area and should not mix code from other areas of concern. Clean architecture, as illustrated in figure 16, enforces this separation into the following:

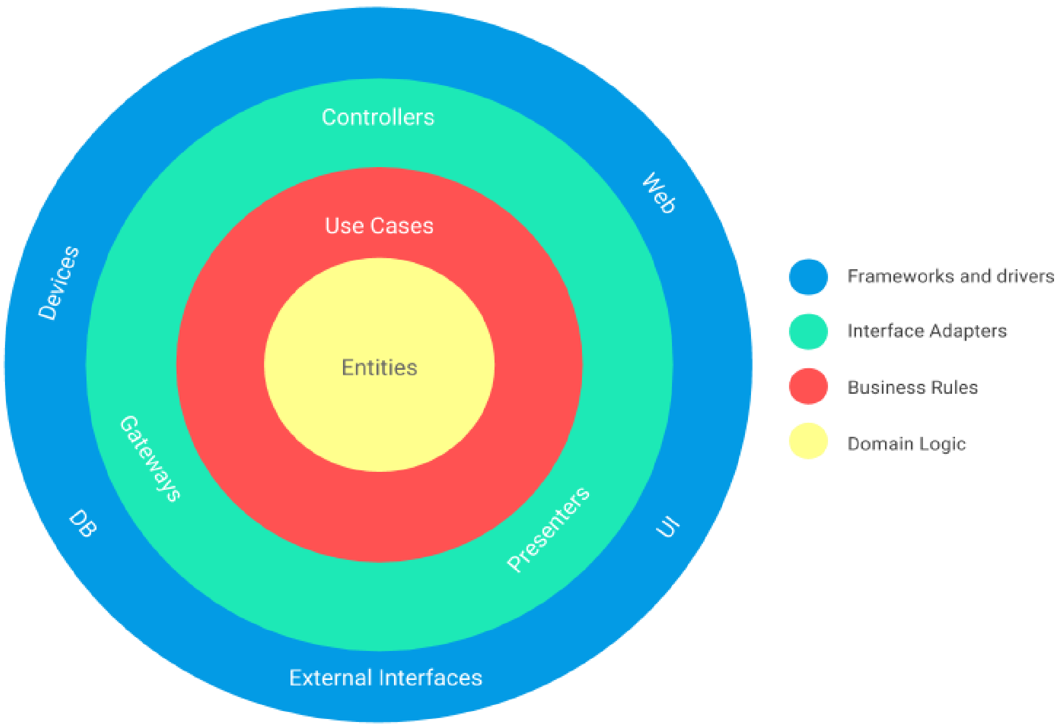


Figure 18 – Clean Architecture [37]

- **Entities:** Define the business objects of the application, which encapsulate the most general and high-level rules of the application;

- **Use cases/Interactors:** Where the application specific business rules are defined. Therefore, it is their responsibility to orchestrate the data flow to and from the entities in order to achieve the goals of the use case;

- **Interface Adapters:** Converts data to the most convenient format for the requesting layer to consume. Adapters create boundaries within the system, isolating business logic from the framework details. Under these circumstances, the system is not entailed to a specific tool;

- **Frameworks and Drivers:** This layer is where all the details lie, from databases to frameworks.

7.4. Android Clean Architecture

Android as a system does not enforce the architectural principles stated in Robert C. Martin CA. It does not enforce decoupling of UI components and the business logic that will populate the component with data. The illustration below is the attempt to enforce Robert CA principles into the android environment [35].

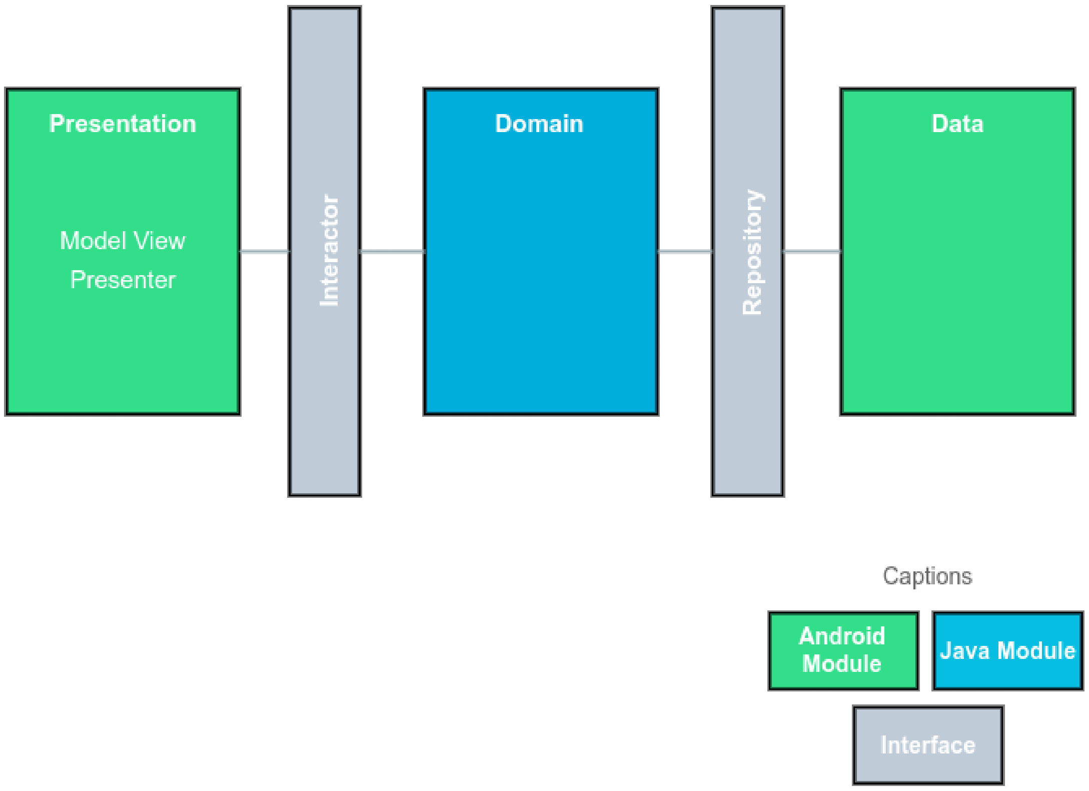


Figure 19 – Android Clean Architecture diagram

Presentation Layer

The presentation layer is where all the UI elements and adjacent logic lie. As can be seen the Figure 20, it adopts a Model View Presenter design pattern.

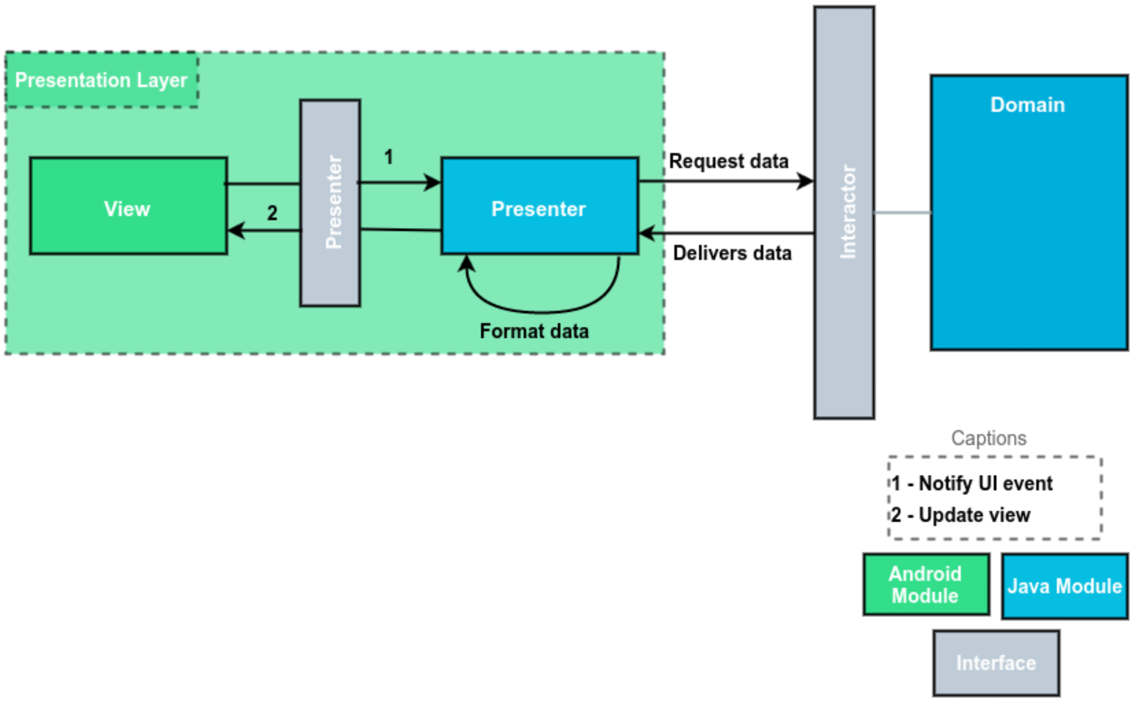


Figure 20 – Presentation layer decomposition

In addition to storing all the UI elements, the view component has the responsibility to alert the presenter when a UI event is triggered. On an opposite side, the presenter signals the view about when to update. In short, the view duties can be narrowed down to:

- Deal with activities, fragments and all UI related elements;
- Scheduling animations/transitions;
- Propagate user input to the Presenter;
- Bind data to the View.

The Presenter component is responsible for receiving input from the UI and the domain layer, basically passing and formatting data for the UI. In a nutshell, the presenter is responsible for the following tasks:

- Orchestrate and execute Interactors;
- Prepare/format data for the View;
- Retrieve the data from the model and return it formatted to the View.

Doing a side by side comparison with Robert Martin's Clean Architecture, this layer comprehends his two outer layers (blue and light blue). This displacement complies with CA's main argument - that the system should not be entangled by a specific tool. Through the usage of the contract, the presenter is unaware of the android framework, thus creating a boundary to the frameworks reach within system while respecting the dependency rule principle.

Domain Layer

The domain layer is where the core business logic lies. Figure 21 describes the modules that comprised the layer.

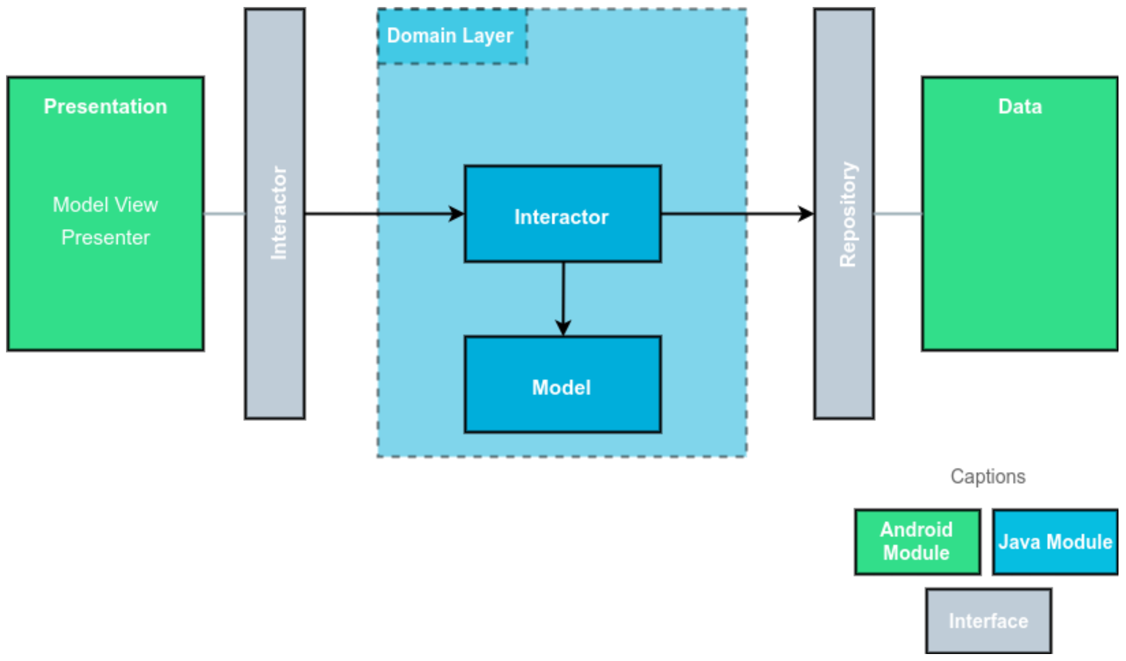


Figure 21 – Domain layer decomposition

Interactor is responsible for defining the business logic of a specific feature and orchestrates the data flow for that feature. Each Interactor communicates events to the upper layers using callbacks. As the figure describes, the Interactor module respects Uncle Bobs Clean Architecture which states that the business logic should be decoupled from any framework dependencies. In this specific case, this translates to a POJO completely decoupled from the android framework. This greatly boosts the architecture extensibility as greatly helps unit testing. Interactors often communicate with models in order to populate data that goes or comes up from the repository layer.

Data Layer

In Robert Martin's Clean Architecture, this layer is represented in the outermost layer because of its framework dependency. The Repository responsibility is to call the desired data source according to the application current conditions. Figure 22 depicts all the modules typically found in this layer.

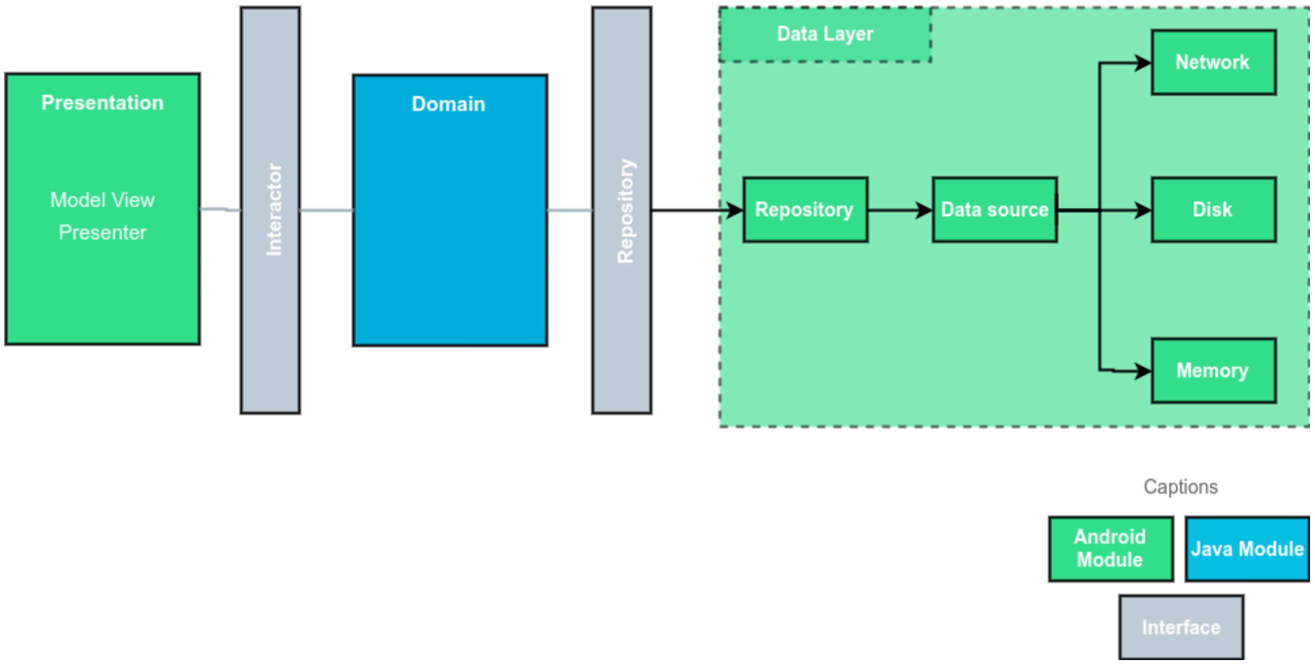


Figure 22 – Data layer decomposition

7.5. Application Architecture

The following sections provide an overview of the internal architecture of the application. The section below was sorted to match the Epics order, stated in the Requirements chapter, and the described screens shown in the Application flow chapter. Although, for the sake of simplicity, only the most relevant diagrams for the internship are described.

Each module has its own clean architecture. The presentation layer, as previously stated, is typically composed by three main elements: an activity, a presenter and a contract. An activity is essentially an android translation of a view. A contract is a schema of the communications that can be maintained between an activity and its presenter. This means contracts cannot be shared, since they represent a unique set of communications within the system.

Domain layer components are responsible for tasks such as fetching, storing data or components that contain the logic of modules such as the recorder, player and metronome and mixing. The prior one's mainly rely on communicating with two distinct data sources – Remote and Local. The remote data source is an exposed endpoint of the API developed by André, which is used to fetch or store information about user profiles, projects, tracks, tracks timeline and mixing settings. The local data source is primarily used to persist data that arises from the authentication process and its needed to later perform requests to the API - authentication token and user id. These modules are typically performed on the background thread as they can be time-consuming and hence could potentially block the main UI thread.

7.5.1. Project Screen

Figure 23 depicts a typical communication with the Web API developed by André. This specific case illustrates the architecture flow in order to get a user music projects.

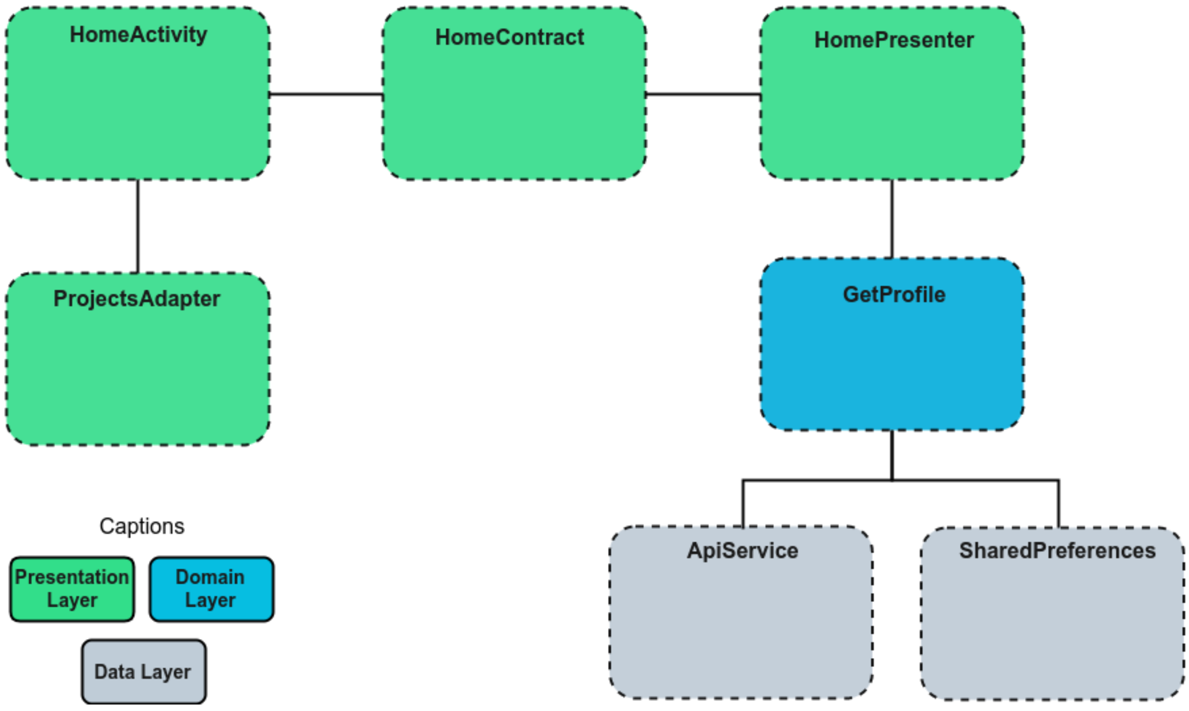


Figure 23 – Landing screen architecture flow

HomeActivity starts the flow by triggering the presenter, which will eventually lead to the API request. When the response reaches activity, the activity spawns an adapter with the received data. A adapter essentially acts as a bridge between a *UI* component and a data source, filling the received data into the *UI* component.

Upon receiving the activity request, the **HomePresenter** unaware of the current logged user calls **SharedPreferences**. This module persists the user information and the authentication token needed to perform a request to any API endpoint, besides authentication and registration. After being in possession of this data, the flow proceeds to the **GetProfile** module, which requests and handles the response of the API endpoint held in API Service module.

The API consumption is done through the usage of a library called Retrofit. As can be seen in the Figure 24, retrofit provides annotation processors that encode details about the parameters and the REST request method.

```
@GET("users/{user_id}") Call<ApiResponse<Profile>> profile(@Path("
      user_id") String userId,
       @Header("Authorization:Bearer") String authToken);
```

Figure 24 – API endpoint example

All the data exchanged between the android client and server uses JSON as a data Format. Retrofit annotation processors are also used on java objects to serialize and deserialize JSON Data. Retrofit parses the API data on a background thread, and then delivers the results back to the UI thread. Thus, facilitating the developer job since UI updates can only be made in the UI thread.

The list of the used API endpoints it is extensively detailed in the *Appendix A – API Endpoints*.

7.5.2. Record & Metronome

Figure 25 presents all the modules that are a part of the recorder and metronome architecture.

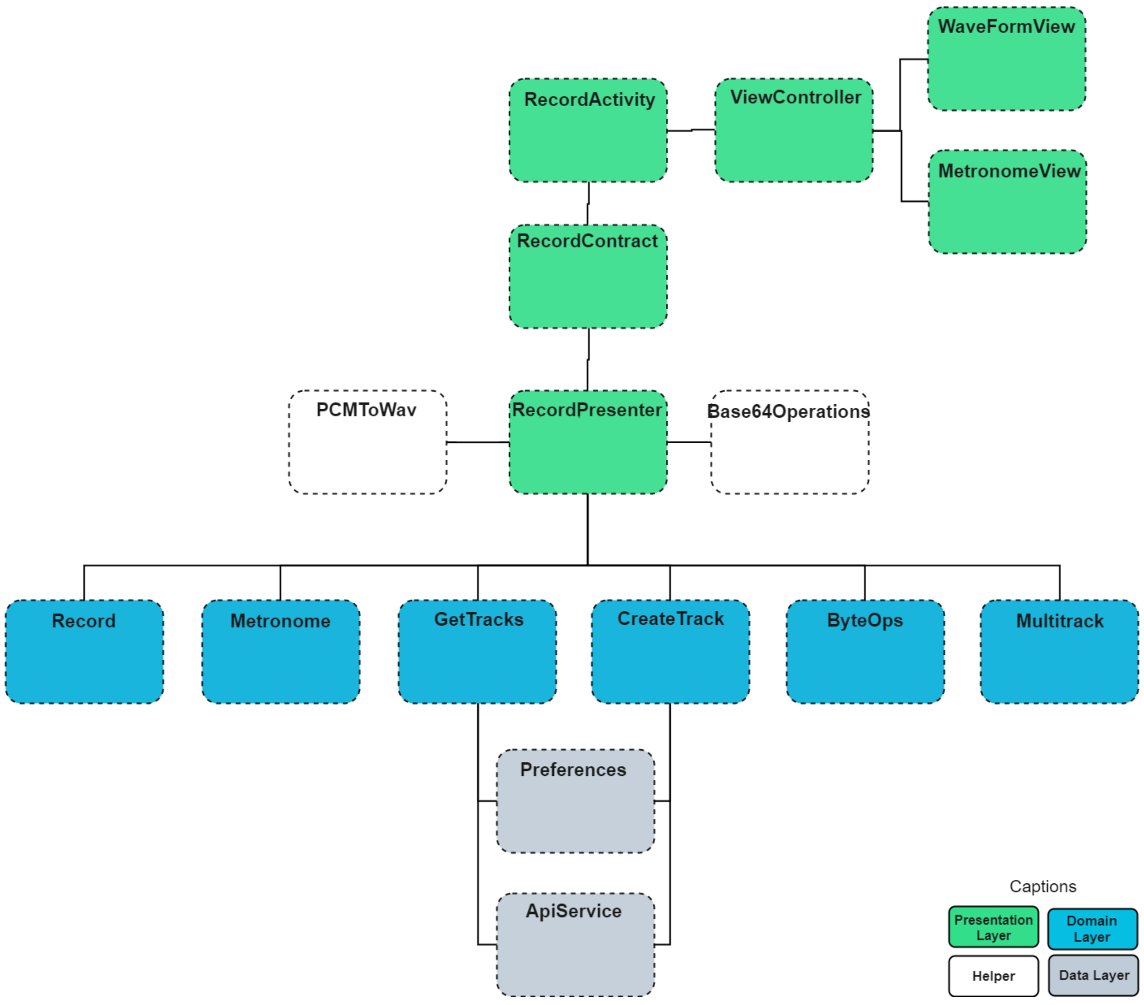


Figure 25 – Record and metronome architecture flow

Since the architecture of both these features is too dense to be tackled all at once, they will be divided into the following subtopics:

- **Track and waveform preparation:** Describes the flow interactions that happen till the user can start recording.

- **Recorder:** Deepens the recorder logic and the flow interactions from the moment the user hits the recording button till the point the recording ready to be played.
- **Metronome:** Portrays how the metronome mechanism works.

Track and waveform preparation

As previously mentioned in *Section 5.2 – Record & Metronome*, the user is able to pick the project tracks that will be played during its recording. Assuming the user already has multiple tracks in his project, these tracks need to be loaded and prepared before the user presses the record button. To tackle this, the recording activity creation triggers the presenter to prepare the project tracks.

At this point, the presenter responsibility is to check if the project tracks are stored in the phone's disk. If they are not stored, **GetTracks** module is instantiated and the endpoint responsible for the track request is triggered. The audio data is then returned in base64 format, which facilitates the data exchange but also ensures its integrity. The response is then redirected to the Base64Operation module, where the audio is decoded and stored in the smartphone's disk.

If the project tracks have been successfully stored, the **MultiTrack** module is instantiated. **Multitrack** is the module where the tracks audio is prepared, played, paused and stopped. Its logic revolves around the usage of AudioTrack library, which writes PCM data into an audio sink for later playback. Since the desired use case involves playing multiples sounds simultaneously, **MultiTrack** creates an AudioTrack instance per picked track and spawns a thread for each of the instances. As soon as the audio sink is full, the track is ready to be played, which triggers a callback sending the audio data back to the UI thread.

Now in possession of the project audio data, the presenter has the conditions to start building the project waveform. The project waveform is the representation of the highest amplitude values of the picked tracks. To assemble these values, the presenter calls the **ByteOperations** module and sends the data of the tracks previously selected by the user. In this case, this module is in charge of receiving all the selected track audio data and, based on that data, create an array with the highest amplitude values. The track audio data is represented in the form of byte array, where each index represents the amplitude of the track at a given moment. The extent to which these values vary can be calculated by using 2^n,where n is the audio bit-depth. Thus, considering the tracks are 16-bit encoded, the returned audio samples have a resolution of 65.536 possible values [38, 39] and since these values are represented in a two complement format, their boundaries fluctuate from -32768 to 32767. Taking this into account, the **ByteOperations** module based off the received audio data creates the highest amplitude array. However, this is not as straightforward as it seems. Imagine a hypothetical case where a user selects 5 tracks to be played while recording, if each one is 2 minutes long, the process of going through all the audio data and store the highest amplitudes values would create a huge overhead. The workaround was to go through 32 indexes at a time, which would be equivalent to collect 1 value every two samples. Despite drastically reducing the processing required, this approach major flaw is in the discarded values, which could prove useful in order to draw a more accurate waveform.

After the highest values are successfully gathered, the data is redirected to the presentation layer, where the waveform view takes the highest amplitude array and spawns the waveform accordingly.

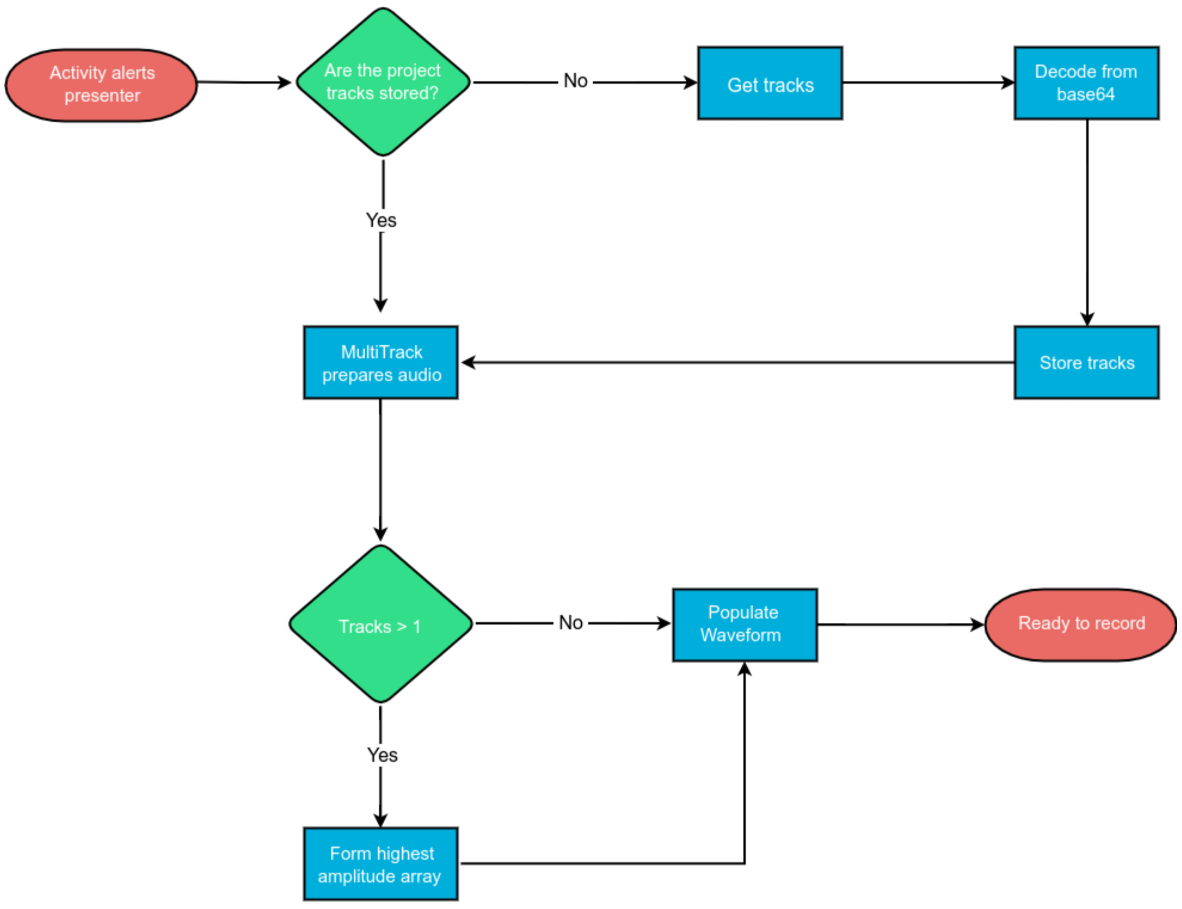

Activity alerts presenter
Are the project tracks stored?
No
Get tracks
Decode from base64
Yes
MultiTrack prepares audio
Store tracks
Tracks > 1
No
Populate Waveform
Ready to record
Yes
Form highest amplitude array

Figure 26 – Flowchart of the track and waveform preparation

Recorder

The aim towards the recording feature was to maximize the audio quality. The audio format plays a huge role in this aspect. There are two kinds of approach that can be made, each one with its pros and cons – Lossy and lossless formats.

Lossy formats, like MP3 and Ogg, resort to lossy compressions to toss out the information deemed as unnecessary to decrease file size. Lossless files, like AIFF and WAV, on the other hand do not toss out any information. Despite some of them resorting to compression, by the time of its decompression the audio is fully restored. Despite maintaining the integrity of the original audio signal, the file size increases substantially. Since the application target market is pointed towards musicians, the natural choice is the lossless format.

WAV was the format picked since it is the most commonly used in the music industry. Knowing this format structure was necessary in order to implement the creation of the WAV file out of the recorded PCM data. Therefore, its structure is mainly composed by three chunks of information:

- The RIFF chunk which identifies the file as a WAV file. Resource Interchange File Format (RIFF) is a generic container format used for storing data in tagged chunks. It is mainly used to stored multimedia such as sound and video [40];
- The format chunk which identifies parameters such as sample rate and bit-depth;
- The data chunk which contains the actual audio data.

Figure 27 deepens the matter through an illustration of the stated chunks and their respective sub-chunks.

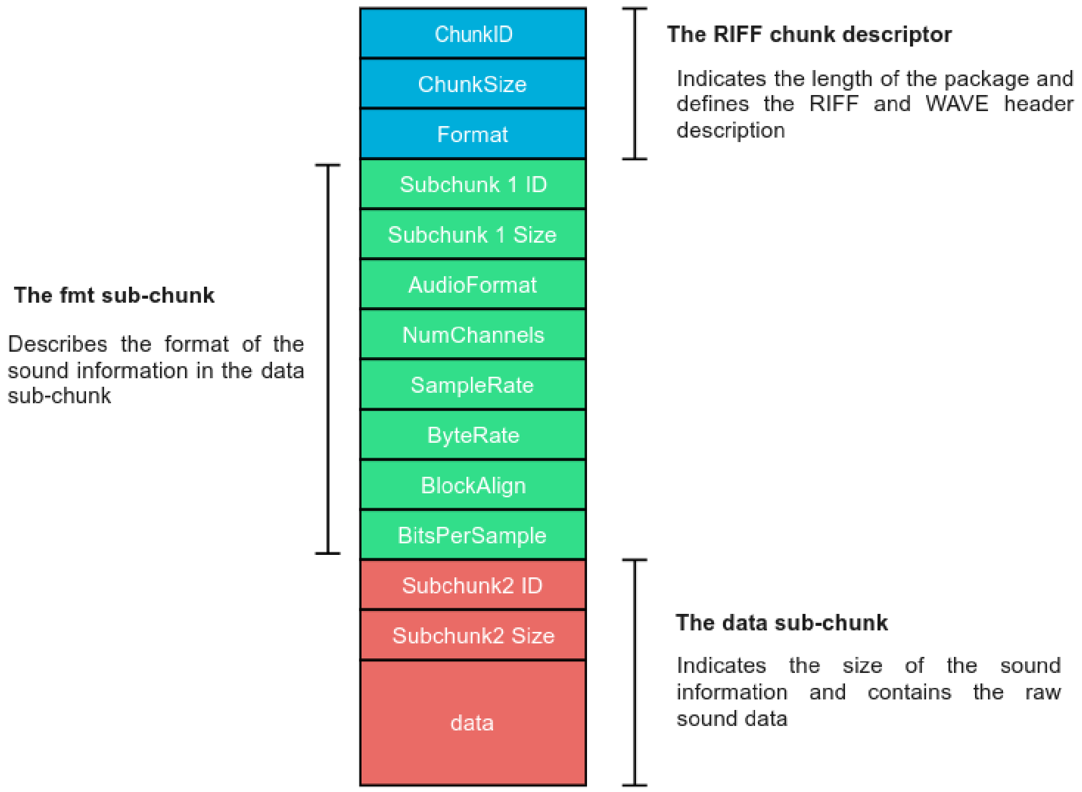

ChunkID
ChunkSize
Format
Subchunk 1 ID
Subchunk 1 Size
AudioFormat
NumChannels
SampleRate
ByteRate
BlockAlign
BitsPerSample
Subchunk2 ID
Subchunk2 Size
data
The RIFF chunk descriptor
Indicates the length of the package and defines the RIFF and WAVE header description
The fmt sub-chunk
Describes the format of the sound information in the data sub-chunk
The data sub-chunk
Indicates the size of the sound information and contains the raw sound data

Figure 27 – WAV file structure

The following list details each one of the sub-chunks illustrated in Figure 27:

- **Chunk Id:** Contains the string "RIFF" in ASCII form.
- **Chunk size:** Defines the size of rest of the chunk.
- **Format:** Contains the string "WAVE" which declares the format of the file.
- **Audio Format**: Format of the audio data. Since PCM data is the desired for the use case, this field is equal to one.

- **Number of Channels:** Indicates the number of channel the audio has, one for mono and two for stereo. In order to do audio panning, each track needs to have two channels.

- **Sample Rate:** Ascertains how many samples of the audio signal are take in one-second period. For example, a sample rate of 48 kHz means that over forty-eight thousand samples are in a one-second period. The implemented recorder sample rate is at 44.1 kHz. Sampling audio at 44.1 kHz grants a good quality allied with the highest compatibility along android devices [41].

- **Bit rate:** The number of bits that is processed per unit of time. The bit rate of PCM audio data can be calculated with the following formula [42]:

  *Bit rate = Sample Rate * Bit Depth * Channels*

- **Bits per sample:** Bit-depth or bits per sample is the resolution of the audio. A higher depth is equivalent to a higher resolution of the audio. Bit depth can be compared to a screen resolution: increasing resolution increases a pixel density. In sound, a higher depth results in more samples per second.

- **Block Align:** Reflects the number of bytes for one sample including all the channels. The value is achieved through the following formula:

  *Block Align = Number of Channels * Bits per sample / 8*

- **Subchunk2 size:** Length of the sound data.

- **Data:** The actual sound data.

After this prologue, the recorder flow explanation can continue. The flowchart illustrated in Figure 28 presents the overall flow of recording that starts with the user pressing the record button and ends when the application is ready to replay the previously recorded track.

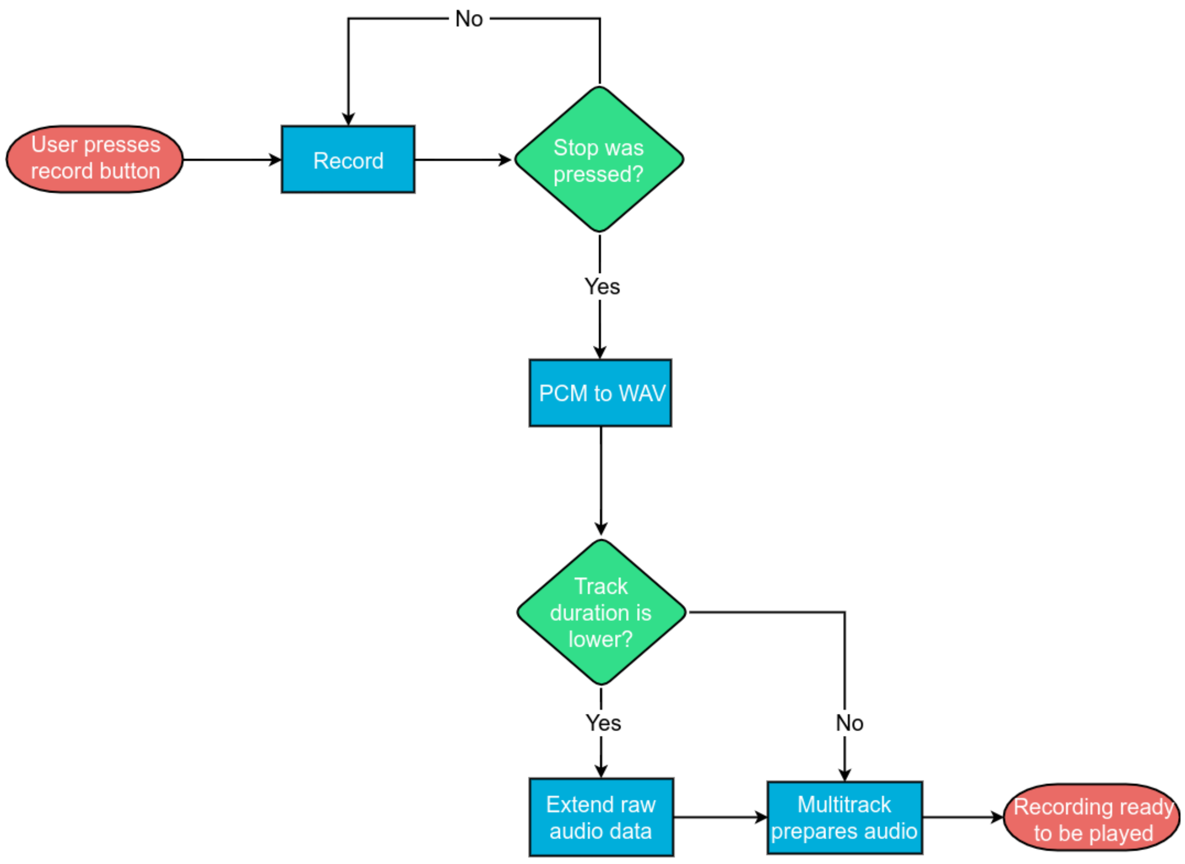


Figure 28 – Record feature flowchart

The recorder logic is held in the **RawRecorder** module, which uses Android AudioRecord [43] to record audio from the audio input hardware. Once the user presses the record button, data starts being pulled from AudioRecord instance and written into a buffer. This process only ceases when the user hits the stop button, which stops and releases the AudioTrack instance and triggers a callback method, sending the recorded data back to the presenter. After arriving the presenter, data is sent to the **PcmToWav** module to be transformed into the structure previously illustrated in Figure 24.

At this point, the recorded audio is ready to be prepared to be later listened by the user. Contrasting with the track preparation and waveform, in this use case there is a need for the recorded audio waveform to be drawn individually. This helps the musician compare the recorded track with the already recorded project audio. However, to properly compare waveforms their duration should be the same. After the audio preparation is complete, a callback to the presenter is triggered which later calls **ByteOps** module in order to assert the recorded track duration. If the audio data duration is lower than the project audio, the recorded audio data is extended with zero amplitude till it matches the project audio duration. Otherwise, the data is sent to the Multitrack module in order to be prepared. This process guarantees all the tracks have same duration, even if half of the duration is silence. Figure 29 displays the outcome of the process.

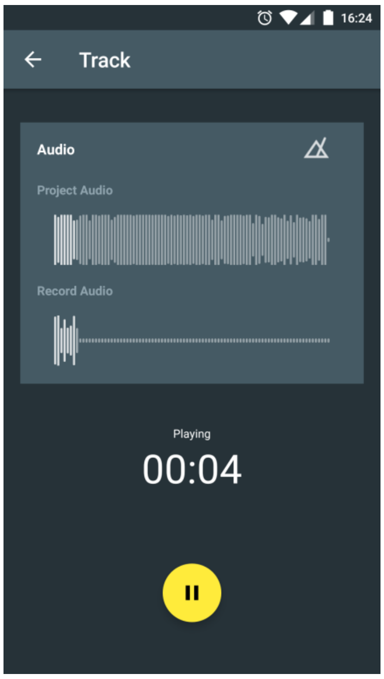


Figure 29 – waveform extension

After this process is complete, the application is ready to play all the tracks at the user's will. When the user is satisfied with the end result of his recording, the conditions are met to send the audio to the server. When the user presses the confirmation button, the audio is encoded to base64 and the API endpoint in charge of the track creation called.

Metronome

As stated before, a Metronome is a mechanism that produces a distinct beat within regular intervals that the user can set in beats per minute (BPM). This audio is played during the recording duration to better help the musician maintaining a sense of timing and tempo.

Despite being two features where the main premise is to reproduce audio output, **Multitrack** and **Metronome** use cases differ. Metronome use case continuously loops a short audio within time periods defined by the user. Instead of aiming for audio quality above all else, the Metronome module main focus is to guarantee all beats are played in the right tempo. Thus, considering the recording screen already contains heavy operations in the whole process of playing and recording tracks, assuring Metronome does not create major overhead is mandatory. Therefore, opting for a lossy format ensures a shorter loading time while maintaining reasonable audio quality. As mentioned in *Chapter 6 – Tools and technology definition*, Android provides two libraries capable of decoding and playing lossy formats – MediaPlayer and SoundPool.

As depicted in Table 3, SoundPool possesses the characteristics needed to satisfy the use case. Looping a short audio file within time periods defined by the user bypasses the problem of the file threshold restriction. In addition, read operations are much fast when compared to MediaPlayer, since the resource is loaded into memory from inside the APK [44].

Table 3 - MediaPlayer and SoundPool comparison

	MediaPlayer	**SoundPool**
File threshold	✗	1MB
Accept lossy formats	✓	✓
Resources are loaded to	Disk	Memory
Use case	Longer files or streams	Short audio bursts

The Metronome flow starts right after the **RecordActivity** creation, which sparks the **RecordPresenter** to instantiate the Metronome module and start loading the audio resource. Before the user starts recording, a view is prompted to tune the Metronome. Within this screen the user is able to define the number of ticks played in a minute (bpm), activate strong beats and specify the frequency they play. The bpm's are then used to calculate the delay between ticks through the formula: *60/beats per minute*. This delay is introduced by a handler, which allows to send and process runnable objects associated with a thread message queue. In this case, each message triggers the SoundPool to play the Metronome sound and schedule the next message to be executed after the previously calculated delay.

7.5.3. Mixing

The figure shown below shows all the intervening modules in the Metronome feature.

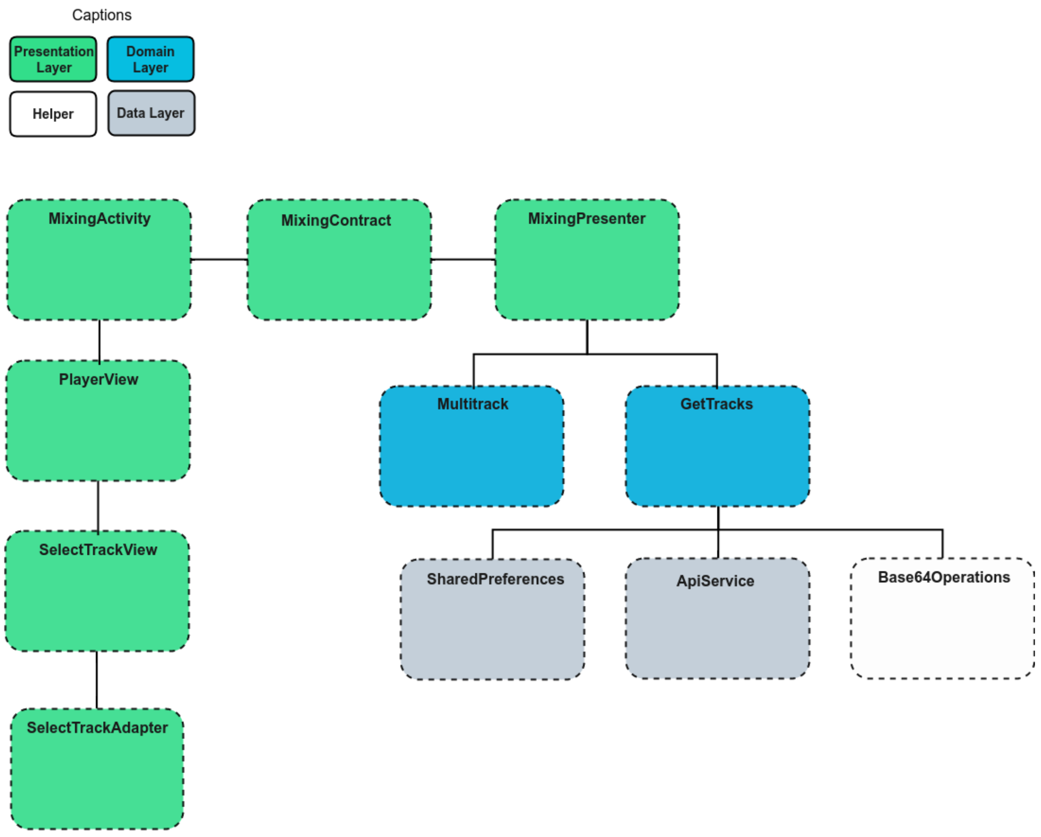


Figure 30 – Mixing architecture flow

All pan pots, sliders that indicate the current audio panning position, range from 0 to 1 in both channels. Sliding the pan pots inverts the amplitude of the channels, thus creating the crossfade effect. When transitioning from one side to the other different scales can be used – Pan laws.

Natural Pan law uses a linear function that ranges from 0 to 1, being the center 0.5 to each channel. However this law is not the most used, since the amplitude is reduced in the center approximately 6.0db. This happens because the way humans perceive loudness is not linear and when the sound is reduced to 50% in one channel and 50% in the other, the perceived sound does not add to 100% in loudness. This law is still applied as an effect since it grants depth to the sound.

Constant Pan law applies a square root to the linear values. This way the center position is placed at 7.07 instead of the latter 0.5, which only creates a difference of -3.0db [45].

As stated in the chapter 5, the mixing flow enables the project owner to change volume and panning of the project tracks. To test the changes to the audio, the application provides a player enabling the user to lively hear what his edition sound like. This means the track preparation previously described is replicated in this flow.

Both panning and gain logic lies in the **MultiTrack** module. When the user changes the current position of the panning or gain, the activity triggers the presenter sending both values. With those values the presenter calculates the gain for each channel. Since we are using Constant Pan law to decrease the dB loss, a volume at 50 % and a panning at 50 % left and 50% right would result in a 0.3535 gain for each channel. The final value is then sent and applied in the **Multitrack** module.

7.5.4. Track Player

This section is the architectural translation of the flow addressed in the *Section 5.6 – Track Player*. Figure 31 displays all the intervene modules in the track player feature.

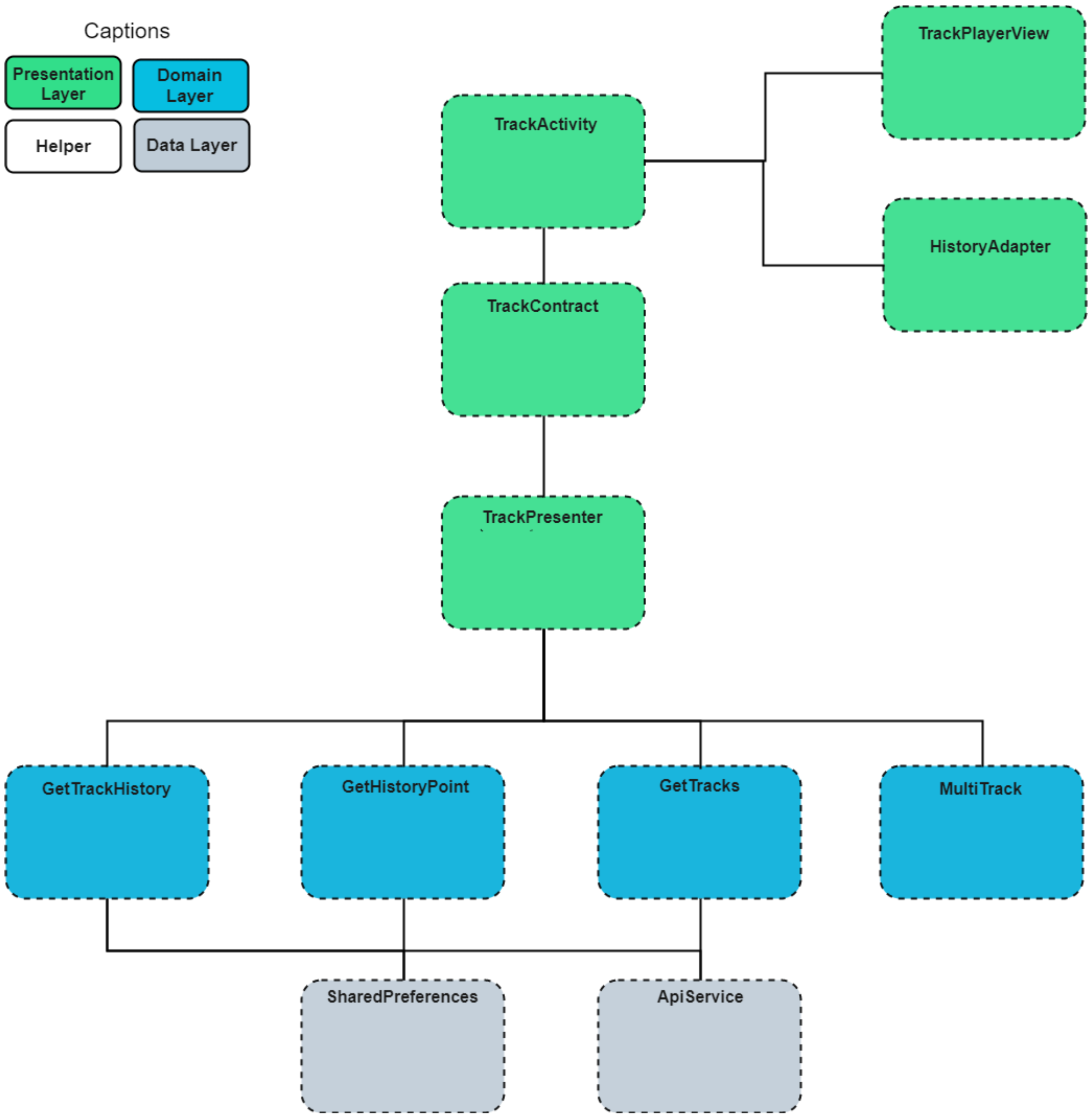
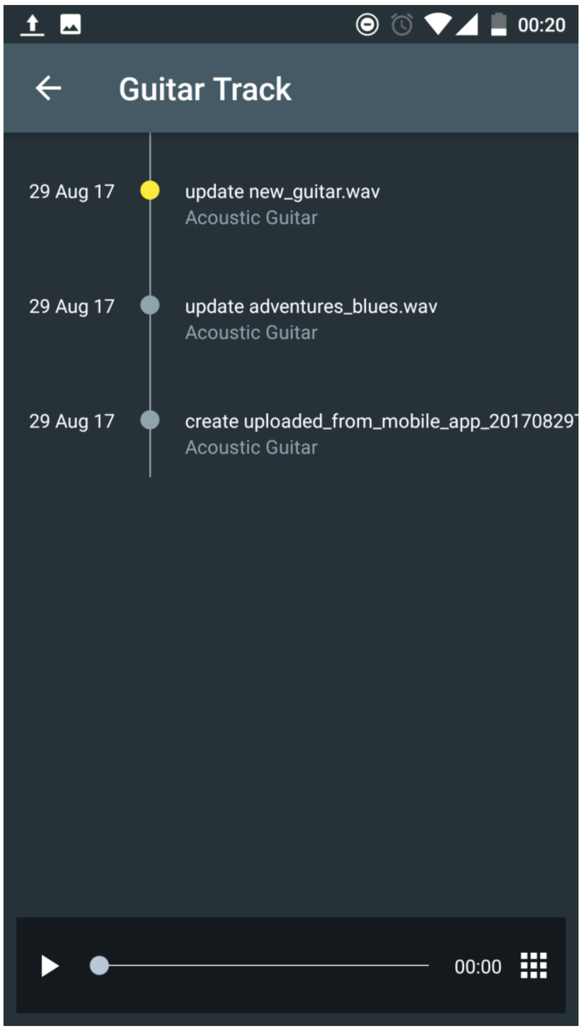


Figure 31 – Track player architecture flow

As stated in Application flow chapter, history points are created when the user rerecords or applies mixing in an already existing track. The illustration displayed in Figure 32 shows a prime example of a track timeline.


00:20

Guitar Track

29 Aug 17 update new_guitar.wav
Acoustic Guitar

29 Aug 17 update adventures_blues.wav
Acoustic Guitar

29 Aug 17 create uploaded_from_mobile_app_20170829
Acoustic Guitar

00:00

Figure 32 – Track History screen

This flow starts like any other. **TrackActivity** triggers the TrackPresenter to instantiate the following domain modules: **GetTrackHistory** and **GetTracks**. **GetTrackHistory** is in charge to communicate with the **APIService** and return all the information about the track timeline minus the audio of each point. Simultaneously, a validation is made to check if the audio of the track is already stored in the phone's disk. If the track is not found, **GetTracks** module is instantiated, the communication with the **APIService** made and the active version of the audio returned. Once the audio data reaches **TrackPresenter**, the cycle of preparation begins with the instantiation of the **MultiTrack** module.

For the sake of the explanation, lets imagine a scenario where the user wants to hear another version of the track. As can be seen in Figure 29, the yellow point displays the active point on the track timeline. By pressing a grey point, **GetHistoryPointData** triggers the **APIService** to query the server for the pressed track version. This operation triggers the previous yellow point to turn grey and selected grey point to turn yellow. However, the pressed track still is not the active version of the timeline. To effectively turn the pressed point into an active one, the user must press the correct icon displayed in the toolbar.

7.5.5. Models

As can be asserted throughout this chapter, it is clearly described and illustrated that the internship application adopts an API first approach. Despite the core features are implemented in the mobile side, all the consumed information is stored in the Server and exposed through the API endpoints. Besides the preferences stated in the previous sections, the application does not store any information. However, to properly consume the REST API, the model illustrated in Figure 33 was built.

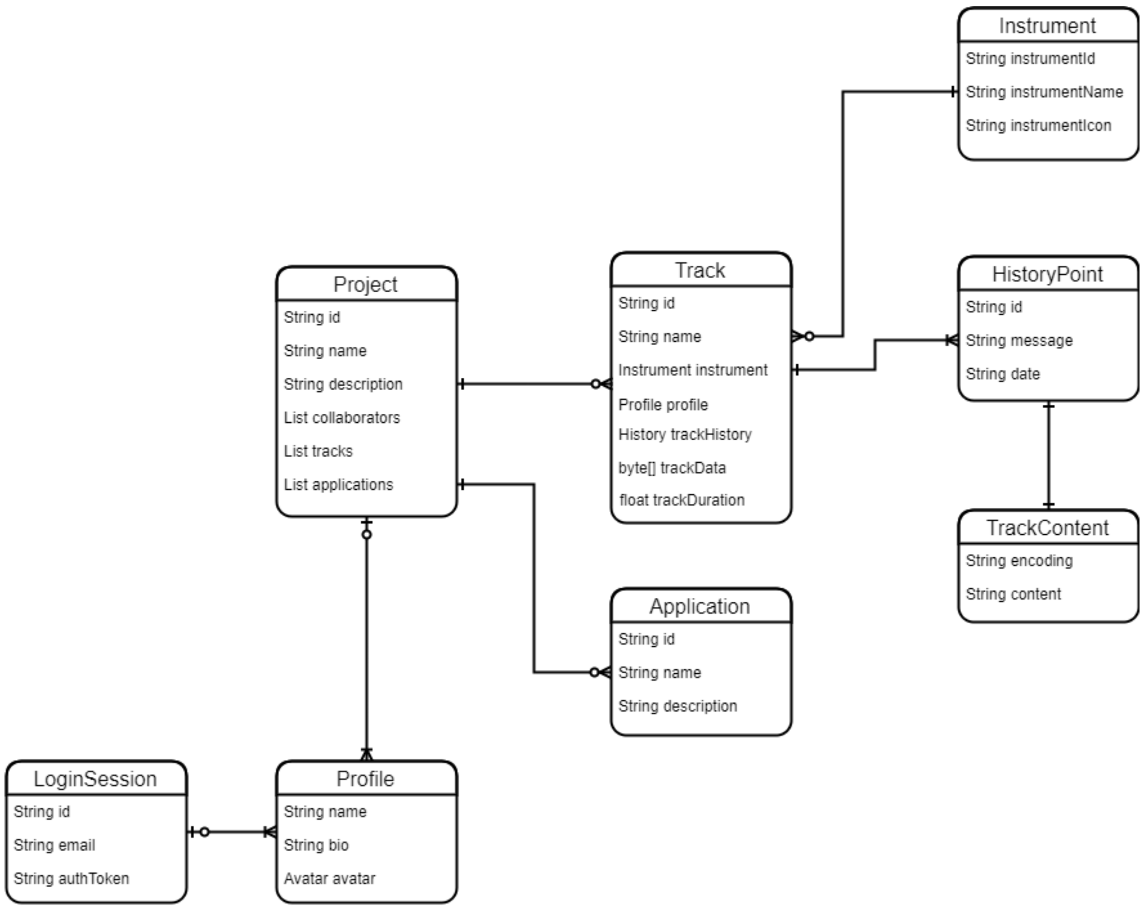


Figure 33 – Entity Relationship diagram of the application

8. Testing

As stated in the approach chapter, the *Definition of Done* (DOD) of a user story includes the development of the feature in question but also states that the developed module must be tested for the later validation. Introducing the test phase right after the development enables the developer to be more efficient in what is about to be tested since the code is still fresh in the developer's mind. Additionally, making sure the tests follow the development progress guarantee newer modules do not break the latter ones, which leads to a good code base.

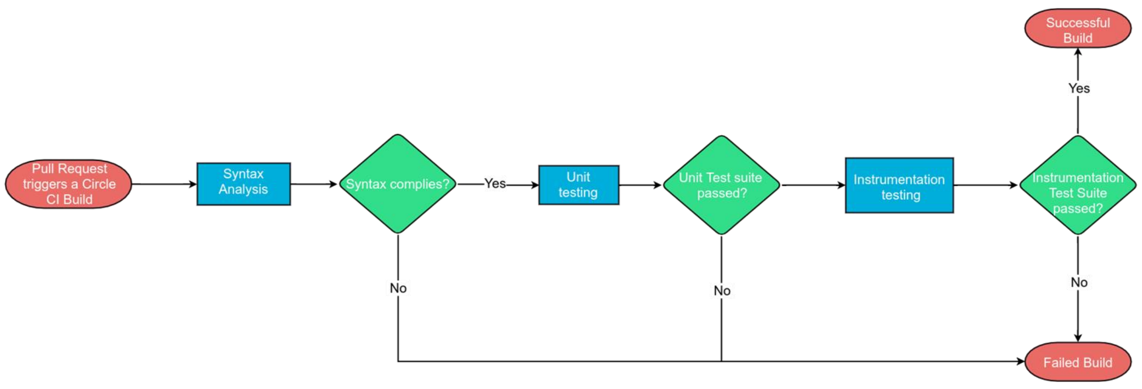


Figure 34 – Circle CI flowchart

8.1. Unit Testing

Unit test as the name suggests is a method to verify the state of a piece of code and its interactions in isolation, or in another words, disconnected from the rest of the system. This methodology is widely used to test the smallest unit of work, reducing as much as possible the amount of dependencies with another resources and then checking a single assumption about the behaviour of that unit of work.

Table 4 contains a full description of the tests performed during the product development:

Table 4 - Performed unit tests

Class	Unit	Result
ByteOperations	Highest amplitude	**Pass**
	Extend byte array	**Pass**
PCMToWav	Raw to Wave	**Pass**
CreateProject	On successful response from the server	**Pass**
	On failure response from the server	**Pass**
CreateTrack	On successful response from the server	**Pass**
	On failure response from the server	**Pass**
GetProfile	On successful response from the server	**Pass**
	On failure response from the server	**Pass**
GetProject	On successful response from the server	**Pass**
	On failure response from the server	**Pass**
Login	On successful response from the server	**Pass**
	On failure response from the server	**Pass**

While the first 3 tests aim to assert a final value, the rest of the unit tests were designed to test the module flow, assuring all the desired code statements are being called and the data received from the server matches the request. The code snippet shown in Figure 35 displays 2-unit tests for the project creation module, the first one verifying a success flow and the latter testing a unsuccessful flow.

```
@Test public void success() {
  String projectName = faker.lorem.characters(5);

  project = new Project(projectName, projectDescription);
  Call<ApiResponse<Project>> call = apiSuccessCall(project);

  when(apiService.createProject(any(Project.class), eq(authToken)
      )).thenReturn(call);

  createProject.create(callback, projectName, projectDescription,
       authToken);

  verifyApiCall(projectName, projectDescription);
  verify(callback).createProjectOnSuccess(projectName);
}

@SuppressWarnings("unchecked") @Test public void error() {
  final ApiError apiError = new ApiError(ApiError.ErrorType.
      VALIDATION);

  Call<ApiResponse<Project>> call = apiErrorCall(apiError);
  when(apiService.createProject(any(Project.class), eq(authToken)
      )).thenReturn(call);
  createProject.create(callback, null, projectDescription,
      authToken);
  verify(callback).createProjectOnFailure(apiError);
}
```

Figure 35 – Unit test sample

The illustration displayed above is a prime example of unit tests that involve communication with the API. To produce this kind of tests it is necessary to send a specific call to the API, and then compare the excepted with the obtained results and test the correct flow is triggered from that point forward.

8.3. UI testing

Automated UI tests are a quick and effective way of ensuring the UI is fully functional. Furthermore, automated tests can substitute the slow process of manual testing and easily ensure that newer iteration of software did not broke older ones. Despite the slow nature of the process, in the Android habitat the case is even worse since the Android environment holds devices from multiple manufactures and currently sustains 5 different Android flavours (versions) that the application targets – *Lollipop, Marshmallow, Nougat* and newer addition *Oreo.*

A total of sixty-nine tests were designed and executed in order to validate the implemented screens. Table 9 presents the conducted tests and their results.

Table 9 – Performed UI tests

Screen	Actions Tested	Result
Login	3	**3/3**
Sign up	7	**7/7**
Landing screen	8	**8/8**
Create Project	2	**2/2**
Create Track	4	**4/4**
Record (includes metronome)	14	**13/14**
Player	7	**7/7**
Track Player	10	**10/10**
Create Track Application	4	**4/4**
Projects sections (Bio, Tracks, Applications)	6	**5/6**
Mixing	4	**4/4**

8.4. Usability Testing

The User Experience is one of the most important attributes of an application nowadays. Despite of the amount of features a system provides, they will mean close to nothing if the system interaction with the user is poor. Therefore, after the development phase, usability tests were conducted in order to receive feedback for the next iteration of the application.

The usability tests were performed to the implemented features. The executed tests were conducted by a total of 10 people. Unfortunately, it was not possible to collect more feedback with the given time, since the application would only make sense to test with people that have some kind of connection with the music practice.

The following tests provided some comments about the usability of each developed feature.

Successfully record a new track

The trial set was challenged to successfully record a new track with the metronome set at 60 BPM. Considering the application has no projects when the user is conducting the test, the minimum number of actions required to perform this test case are detailed in the following list:

1. Press the floating action button;

2. Press "Create";
3. Press the created project;
4. Press add placed in the toolbar icon;
5. Confirm name and instrument of the track;
6. Press metronome icon;
7. Slide "Beats per minute" to 60;
8. Press record float action button;
9. Press stop float action button;
10. Press confirm icon;

The obtained results, illustrated in Figure 36, showed that only 20% of the users were able to complete the test in the minimum number of steps.

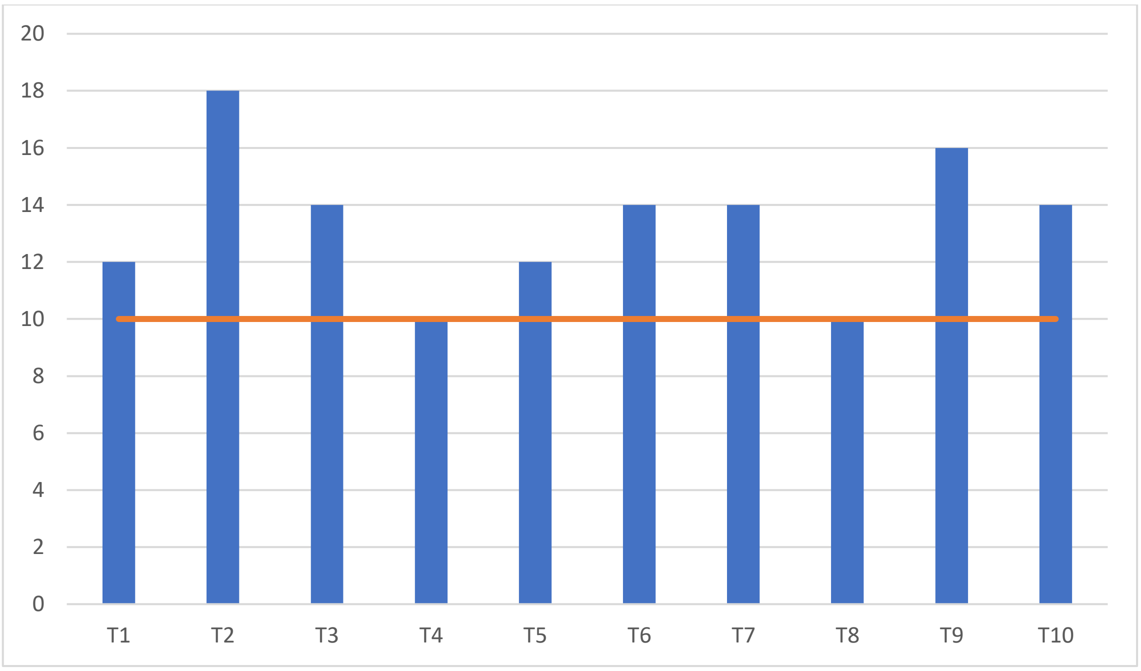


Figure 36 – Successfully record a new track: number of actions done by the trial set in order to execute the test case

Despite the unsatisfactory result, the most important steps of the test were completed quite smoothly except rare exceptions. The following topics explain what happened in order for those users to do more actions than it was supposed to:

- Testers who did 12 actions pressed two times both on step 3 and 11. Since the response takes quite a bit of time, the user is left with no visual clue that already pressed the button;
- Testers who did 14 actions tried to bypass step 5.

Mix a track and play the mixed track

In this test case, users are challenged to mix and play the previously recorded track. Mix in this case means the user is instructed to tune panning to one, which translates into the track only playing in the right earphone. To properly test this feature, earphones must be plugged prior to the beginning of the test, considering most of the devices do not have stereo speakers. In order to complete this test case, a minimum of 5 actions must be taken:

1. Press the previously created project
2. Press bottom floating action button
3. Press the mixing icon
4. Change panning to 1
5. Press play track

These results of this test case, illustrated in Figure 37, were very satisfactory since 60% of the participants executed the test with the minimum amount of actions and the rest are very close to the threshold. The only concern found in this test case was the fact that testers tried to slide the point in the chart instead of the X-axis.

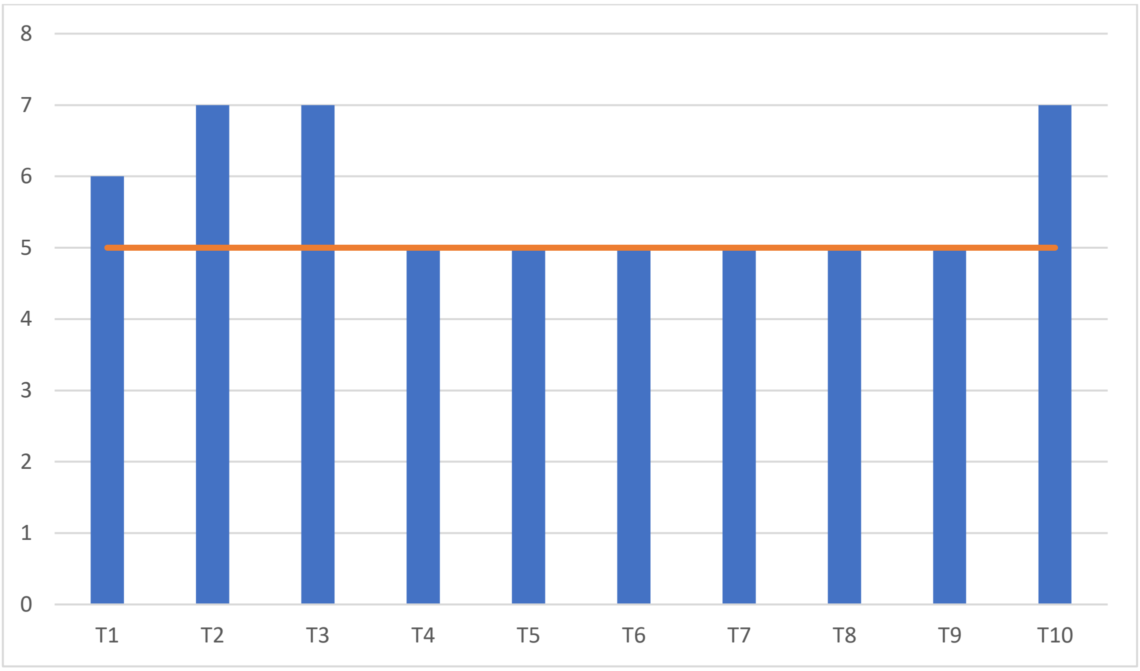


Figure 37 – Mix a track and play the mixed track: number of actions done by the trial set in order to execute the test case

Change between track versions

In order to test this feature, the trial set started in a project with 1 track, this track had two variations. The user objective was to activate the other existing version and hear it. In order to complete this test case, a minimum of x actions must be taken:

1. Press the previously created project
2. Press the track
3. Press change the active point in the timeline
4. Press play

The results, illustrated in Figure 38, show satisfactory results with 60% of the users completing the flow with the minimum amount of actions. However, the rest of the trial set struggled to find where the operation to change track versions could be performed, despite some of them have gone through the correct screen in the first attempt.

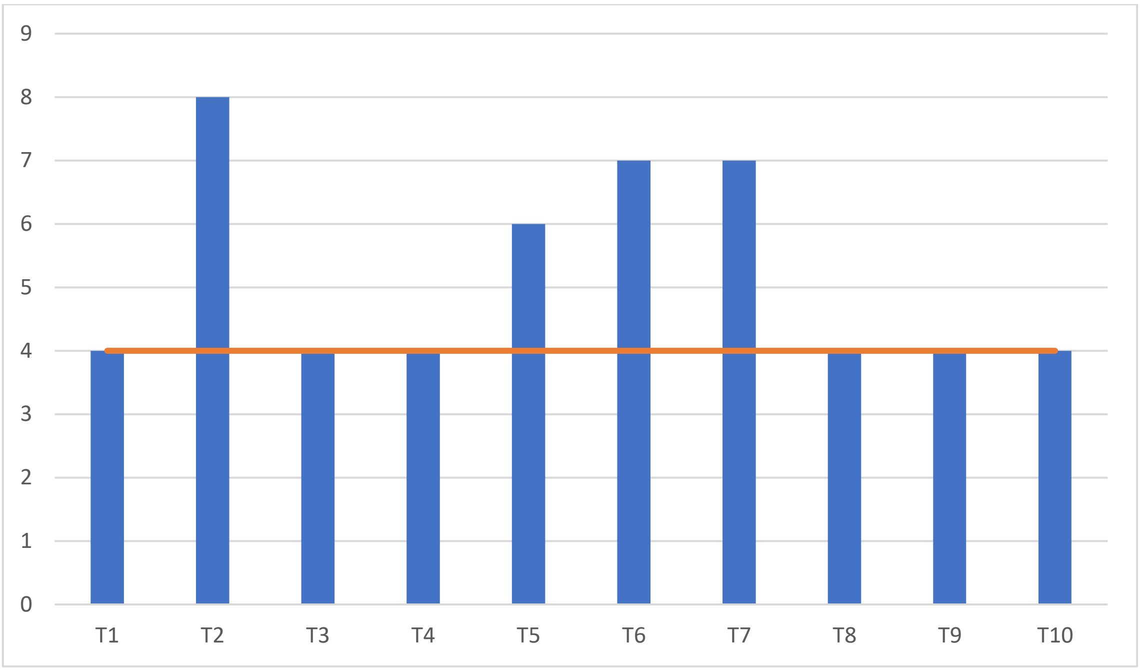


Figure 38 – Change between track versions: number of number of actions done by the trial set in order to execute the test case

9. Conclusion Remarks and Future Steps

This final chapter of the report marks the end of my internship at Deemaze Software. Therefore, there is no better place to make an overview about the work developed during this one-year project.

The first section reviews the project development up until this point, assessing the state of completion of the epics and correspondent user stories. The purpose is to survey my performance during the internship year and assess the overall progression of proposed product.

The second section presents the work that can still be made after the end of the internship, underlining potential new features or improvements in the developed work.

Lastly, the third section presents some personal thoughts about the internship.

9.1. Progress Overview

This section presents the progression state of each of the epics, previously identified in *Chapter 4 – Requirements*. In addition, each epic assessment provides insight about the time consumed in each of the correspondent user stories.

9.1.1. Epic#1 – Authentication

Table 10 shows the final state of the first epic I started to develop – Authentication.

Table 10 – Final status of the authentication epic

Epic#1 – Authentication		
#	**Story**	**Status**
1	As an unregistered user, I want to register in order to create my profile and access the application	✓
2	As a registered user, I want to login in order to get application access	✓
3	As a register user, I want to be able to recover my password in case I forgot it	✗
Total		**2/3**

From this epic, I completed 2 out 3 user stories. Since user story #3 was labelled as a *could have* feature, I transitioned towards more prominent features. Figure 39 shows the elapsed for the two developed completed stories.

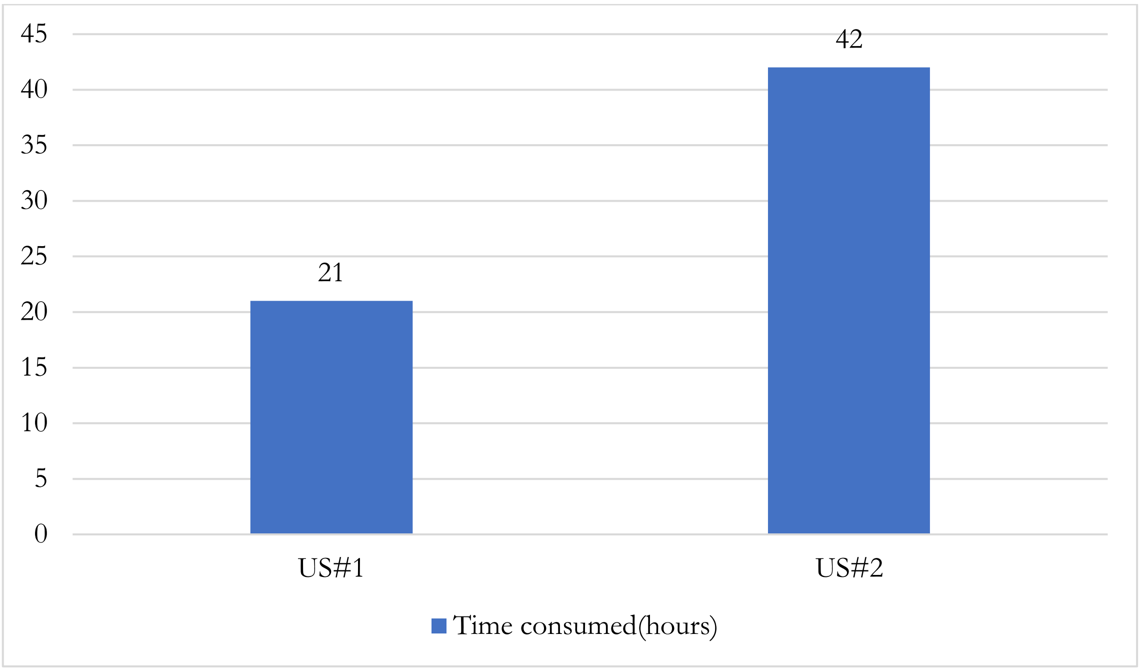


Figure 39 – Time consumed in the authentication epic

The project ramp up was pretty harsh, since the amount of things to learn was overwhelming. Most of the time was dedicated towards to the testing phase. The development of both UI and unit tests was more exhausting and time-consuming than the combined tasks of setting Clean Architecture and developing both user stories.

9.1.2. Epic#2 – Project

The project epic was the next in the development schedule. Table 11 provides insight about the final state of the epic.

Table 11– Final status of the project epic

Epic#2 – Project		
#	**Story**	**Status**
1	As an authorized user, I want to create a project in order to start my new music project	✓
2	As an authorized user, I want to access all the projects I own in order to operate over them	✓
3	As an authorized user, I want to be able to see the project bio in order to check the current members and description	✓
4	As an authorized user, I want to be able to see the project current tracks in order to perform operations over them	✓
5	As a project owner, I want to be capable of deleting a project I no longer need	✓
6	As a project owner, I want to create and edit a description of the project in order to better convey the objective the project is aiming for	✓
7	As a project owner, I want to kick user I no longer desire to be part of the project	✗

Total	6/7

This epic has almost a perfect record with 6 out 7 stories completed. Like in the user story #3 of the authentication epic, user story #7 was bypassed since there were other priorities at the time. Figure 40 displays the time each story consumed.

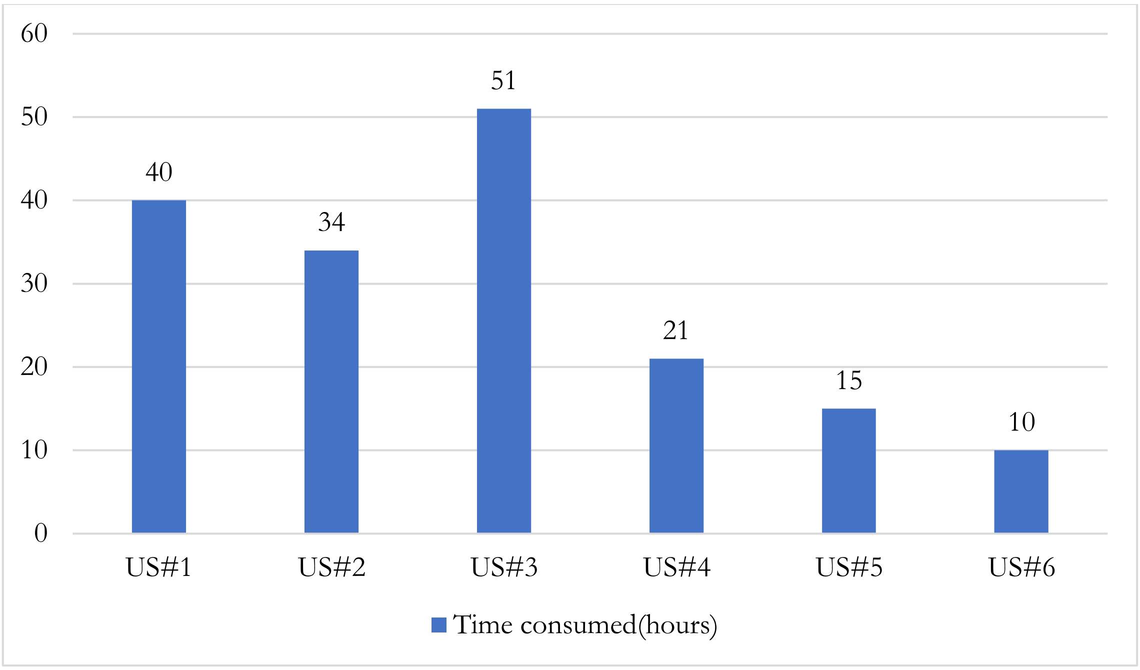


Figure 40 – Time consumed in the project epic

There is nothing major to report within this epic. Despite the sheer amount of work setting the UI and communication with API, the major obstacle at this point was the UI testing.

9.1.3. Epic#3 – Tracks

This epic was definitely the hardest and the one that took more time to develop. A lot of mistakes were made and previously identified risks encountered. Before digging to those matters, Table 12 show the final status of the epic.

Table 12 – Final status of the tracks epic

Epic#3 - Tracks		
#	Story	Status
1	As a project member, I want to record a track in order to create audio content to my project	✓
2	As a project member, I want to play the track I recorded in order to see the track final result	✓
3	As a project member, I want to be able to start metronome before recording in order to set the tempo of the sound I am about to record	✓
4	As a project member, I want to see the waveform of the project audio	✓

5	As a project member, I want a countdown timer in order to give me time to get ready before recording	✓
6	As a project member, I want to pick which tracks I would like to hear while recording	✓
7	As a project member, I want to play all tracks of a project so that I can hear the current state of the project	✓
8	As a project member, I want to delete a previously recorded track because I am no longer interested in it	✓
Total		**8/8**

This epic was of the uttermost importance and its conclusion was mandatory, since the whole application revolves around it. This epic suffered a lot of mutations over the development lifecycle, mainly user stories #1 and #4 and this is reflected in the Figure 41 through the number of hours their development took.

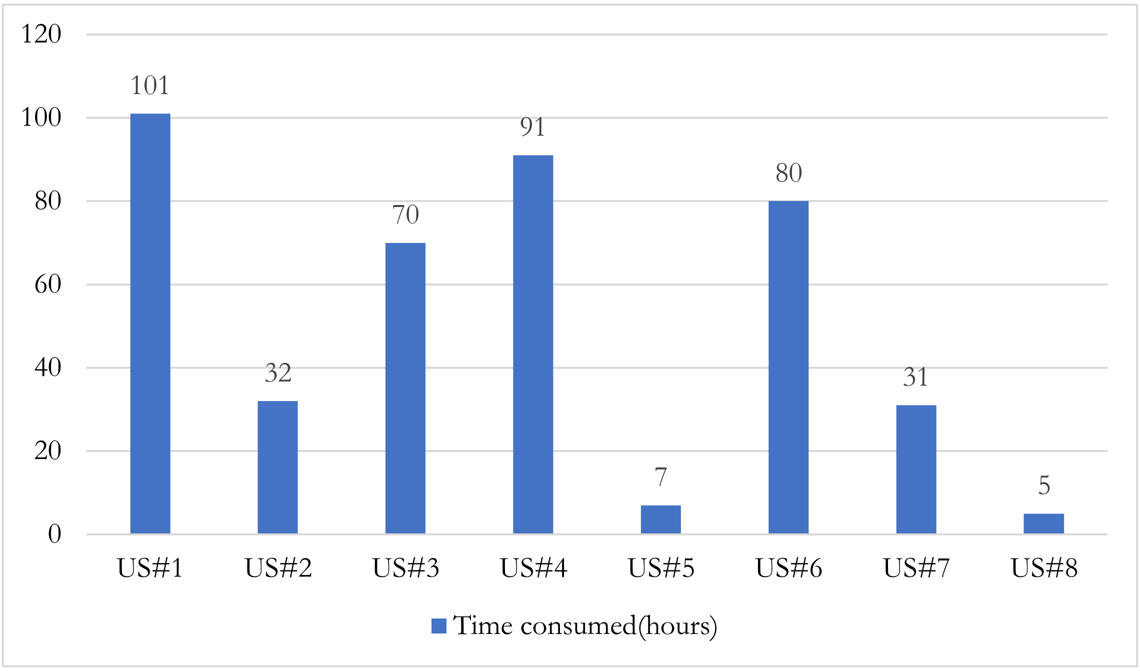


Figure 41 – Time consumed in the tracks epic

The subject introduced above about the user story #1, also involves user story #2, since both these features have a direct correlation. The initial plan was to use MediaPlayer library, since it provides a more straightforward implementation. The usage of this library implied the use of MediaRecorder, which records the audio input and immediately compresses the audio file to the chosen format. Right after development of both these components, I faced the sixth risk identified in the *Sub-Chapter 6.2 – Risk*. The problematic revolved around maintaining multiple instances of MediaPlayer simultaneously. When playing multiple instances simultaneously, audio cluttered and sometimes did not play, culminating ending up a pretty bad overall experience. As anticipated in the mitigation plan, I shifted to the AudioTrack library. However, as stated before, AudioTrack only works with PCM data. So, I had to change both record and play modules to use AudioRecorder and AudioTrack, respectively. Additionally, I had to develop the PCM to WAV transformation module, which created even more overhead. Despite what happened, this change gave more freedom in development of future stories.

User story #4 was also target of 2 iterations over the course of the project. The definitive version was carefully detailed in *Sub-Chapter 7.5.2 – Record & Metronome*. The first iteration had two modules. This module added all amplitude indexes of the selected tracks and then divide by the number of tracks that at that index are different than zero (silence). This approach was completely wrong since it would display the average amplitude instead of the highest. In addition, the overhead of this calculations plus the already existing overhead of fetching the selected tracks was not acceptable.

9.1.4. Epic#4 – Track application

This epic was prioritized as the last in the product backlog. Table 13 exposes the final status of this epic.

Table 13 – Final status of the track application epic

Epic#4 – Track Application		
#	**Story**	**Status**
1	As the project owner, I want to create a track application in order to show my interest in integrating new musicians to my project	✓
2	As an authorized user, I want to be able to search for projects by name or instruments in order to ease the task of finding a project of my interest	✓
3	As an authorized user, I want to record a track in order to apply for a track application	✗
4	As an authorized user, I want to see the status of a track application in order to receive feedback about an application I applied for	✗
5	As a project owner, I want to be able to listen to all the track applications in order to pick the one that better fits my project	✗
6	As a project owner, I want to accept a track application in order to integrate a new musician into my project	✗
7	As a project owner, I want to reject a track application if the applied track does not meet the criteria I am looking for	✗
Total		**2/7**

I was only able to complete 2 out of 7 stories. Despite the fact that more than half of the user stories in this epic were left uncomplete, most of them would be UI modules since all the features that sustain this feature (record, play) were already developed. Figure 42 illustrates the elapsed time on each of the completed stories of this epic.

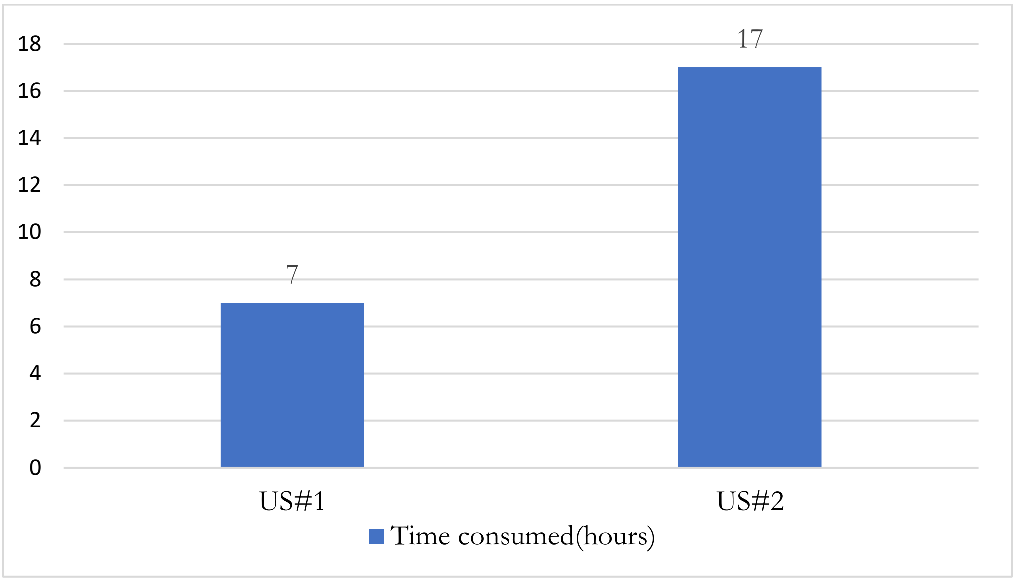


Figure 42 – Time consumed in the track application epic

9.1.5. Epic#2 – Mixing

After the tracks epic, the mixing epic was regarded as the next priority in product backlog. Table 14 shows the final status of the mixing epic.

Table 14 – Final status of the mixing epic

Epic#5 – Mixing		
#	**Story**	**Status**
1	As the project owner, I want to be able to adjust the gain of each one of the project tracks in order to emphasize the desired sound	✓
2	As the project owner, I want a pan control in order to bestow a multi-directional sense to the sound	✓
3	As a project member, I want to trim an undesired piece of a track	✗
Total		**2/3**

From Table 14 is possible to conclude that I completed 2 out 3 user stories, however this is a bit misleading. User story #3 was put on hold since the team at Deemaze think that this feature does not fit the current state of the application.

Regarding the remaining user stories, their development was quite smooth. Since the **Multitrack** module was already ready prepared to accommodate these changes. Figure 43 displays the time spent to develop both stories.

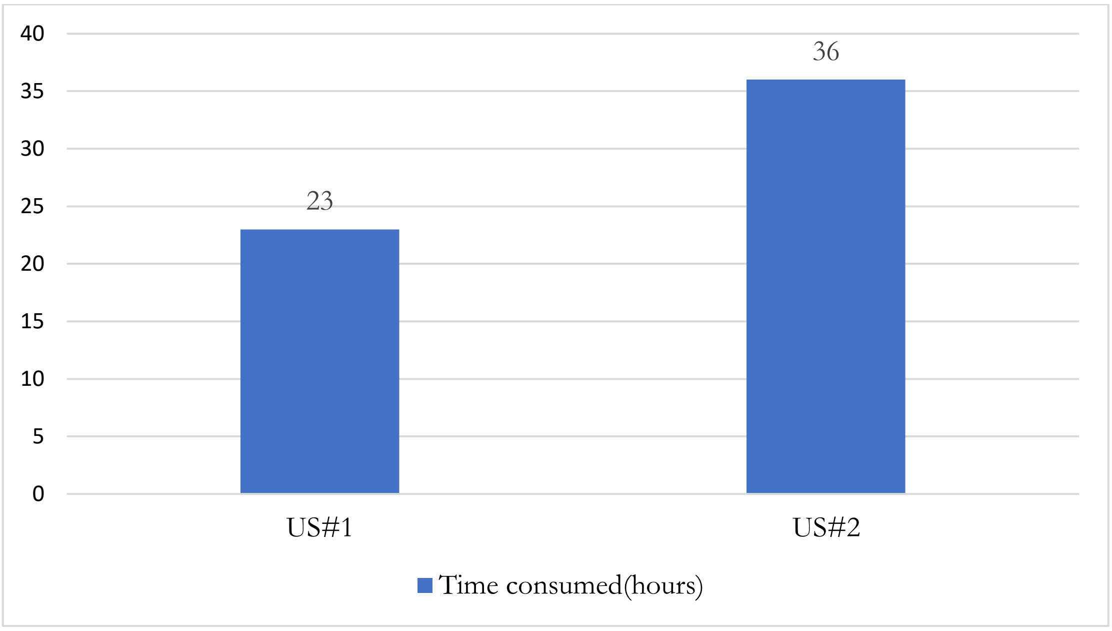


Figure 43 – Time consumed in mixing epic

9.1.2. Epic#6 – Version control

The version control epic, as can be seen in Table 15, was the last that I fully completed.

Table 15 – Final status of version control epic

Epic#6 – Version control		
#	**Story**	**Status**
1	As a project member, I want to play a track and all its milestones in order to hear all the track versions	✓
2	As a project member, I want to travel to a previous milestone of a specific track in order to revert the change of a track	✓
Total		**2/2**

I did not any major obstacles in this epic. Since the **Multitrack** module was already prepared to the track player feature. I had only to develop the UI components and modules responsible for retrieving a track timeline information and the audio file versions. Figure 44 displays the time elapsed in user stories stated above.

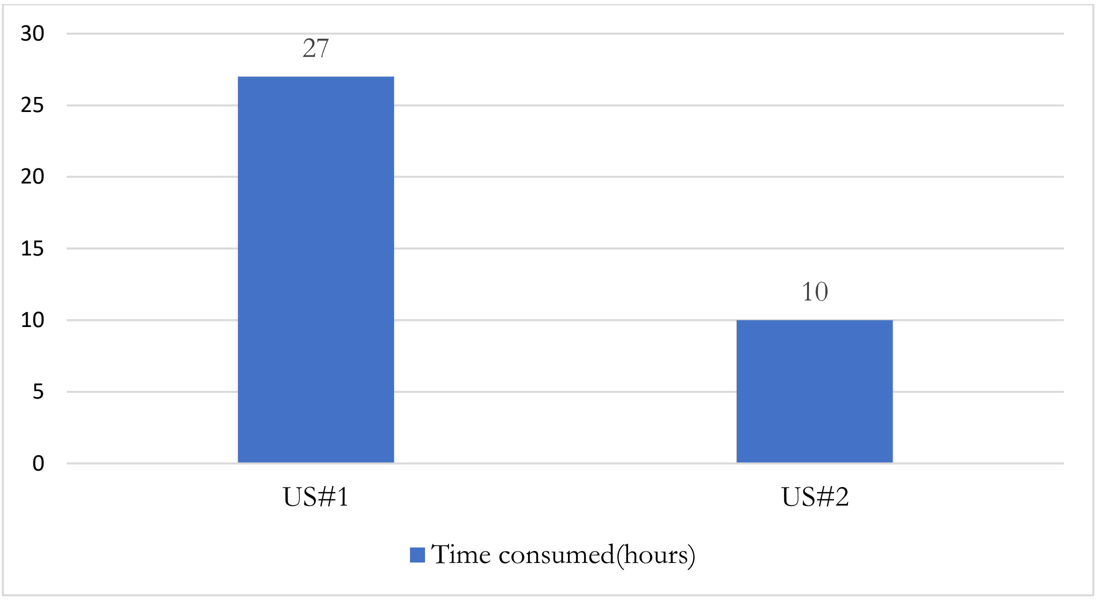


Figure 44 – Time consumed in version control epic

The Table 16 condenses the information provided in Table 12, Table 13, Table 14 and Table 15 providing the amount of completed stories in each of the previously stated epics.

Table 16 – Final overall state of the epics

Epics	**Total amount of story points**	**Stories completed**	**Gathered Points**
Authentication	11	2/3	8
Project	32	6/7	29
Tracks	81	8/8	82
Track Application	50	2/7	13
Mixing	39	2/3	26
Version control	26	2/2	26
Total	**239**	**22/30**	**184**

From Table 16 it is possible to conclude that I finished 22 out of the 30 user stories, approximately resulting in a total of 73% of project completion. However, user stories have different weights and difficulties expressed through story points. Therefore, gauging the state of completion with story points is a more reliable metric. So, throughout the development stage I managed to gather 184 points out of 239, which gives a total of **77% completion**. Although the track applications epic still has more than half of the tasks to complete, most of the logic is already developed. This is reflected in the low amount of points considering the high number of stories the epic holds.

9.2. Future Work

The work developed during the internship is only the first step towards the envisioned product, which means that this project does not end here.

Although it was never part of the requirements initially defined, track upload is one of desired features for the future. This feature was not included initially in order to push the recorder as a core feature of the application, since a lot of the features revolve around the recorder.

Track bouncing is another scheduled feature. This feature was commonly found in the competitors, typically when the user exports a song. For example, Trackd bundles all the audio files into one and then lets the user export the final file to an external tool.

In third place, there are some features that need or may be improved. The audio panning and gain tuning features does not apply alterations to the file. As stated before, panning and gain are values applied to a AudioTrack instance. This is not great solution for the future, since it binds mixing settings to our platform.

9.3. Final Remarks

To conclude this report, it is now time to present some personal thoughts and experiences about the internship.

This one-year internship has been a completely new experience, full of challenges and opportunities to grow. Having the possibility to participate in the product design and directly influence the product direction, reveals that the company listens and values who wants to participate in its growth.

The initial stage plan revolved around an idea, which at the time was not enough to build a product. From this core idea, market analysis and brainstorming were conducted, enabling us to compose a product with features that matter to the market we are targeting. Despite the improvements that the application could use, I think the balance is very positive: I was introduced to the most avant-garde architecture in the android environment – Clean Architecture, had a well-defined workflow through the usage of a Continuous Integration tool which included Unit and Instrumentation testing. Furthermore, since the core features of the application revolved around sound, I got the opportunity to explore all audio related libraries Android has to offer and learn how and when to use them.

I never had an interaction with Android development before the internship, so I knew I had to step out of my comfort zone and a lot of challenges would arise. But in the end, I can say it was a very rewarding experience that I would never take back even if I had the opportunity. It allowed me to grow personally and professionally in a way I could never imagine.

Bibliography

1 Amandine Pras, Catherine Guastavino (2013). *The Impact of the Technological Advances on Recording Studio Practices.* Jounal of the American Society for Information Science and Technology. ResearchGate

2 Virgil Moorefield (2015). *The Producer as Composer.* The MIT Press. ISBN: 978-0-262-51405-7

3 IFPI, IFPI Digital Music Report 2014

4 IFPI, IFPI Digital Music Report 2016

5 Roey Izhaki (2008). *Mixing Audio - Concepts, Practices and Tools.* Elsevier, Ltd.; ISBN: 978-0-240-52068-1. Oxford

6 Tim Dittmar (2012). *Audio Engineering 101 - A Beginner's Guide to Music Production.* Elsevier; ISBN: 978-0-240-81915-0. Oxford.

7 David Miles Huber, Robert E.Runstein (2012) *Modern Recording Techniques* Sixth Edition. Elsevier; ISBN: 0-240-80625-5. Oxford.

8 Tutorialspoint. *Digital Communication* [Internet] . Available at: https://www.tutorialspoint.com/digital_communication/digital_communication_quantization.html

9 Sageaudio. *What is the difference between mixing and mastering* [Internet]. Available at: http://www.sageaudio.com/blog/mastering/what -is-the-difference-between-mixing-and-mastering

10 Blend. *Home* [Internet] Available at: https://blend.io/

12 Kompoz. *Home* [Internet]. Available at: http://www.kompoz.com/music/home

13 Crunchbase. *Kompoz* [Internet]. Available at: https://www.crunchbase.com/organization/kompoz#/entity

14 Crunchbase. *Splice* [Internet]. Available at: https://www.crunchbase.com/organization/splice

15 Splice [Internet]. Available at: https://splice.com/

16 Crunchbase. *Soundtrap* [Internet]. Available at: https://www.crunchbase.com/organization/soundtrap--playwerk-ab#/entity

17 Apple Itunes. *SoundTrap* [Internet]. Available at: https://itunes.apple.com/us/app/soundtrap-make-music-together/id991031323?mt=8

18 Google Play Store. *SoundTrap* [Internet]. Available at: https://play.google.com/store/apps/details?id=com.soundtrap.studioapp

19 Apple Itunes. *Speazie* [Internet]. Available at: https://itunes.apple.com/us/app/speazie-music-recording-collaboration/id983157788?ls=1&mt=8

20 Google Play Store. *Speazie* [Internet]. Available at: https://play.google.com/store/apps/details?id=com.speazie.presentation

21 Apple Itunes. *Spire Record* [Internet]. Available at: https://itunes.apple.com/us/app/spire-recorder/id1013021109?mt=8

22 Apple Itunes. *Trackd recording studio* [Internet]. Available at: https://itunes.apple.com/us/app/trackd-recording-studio-social/id978196692?mt=8

23 Antti Kokko (2013). *Improving requirements management practices in agile software development environment.* HAAGLA-HELIA University of Applied Sciences

24 Hackernoon. *A pratical scrum overview* [Internet]. Available at:_https://hackernoon.com/a-practical-scrum-overview-f46810295e8b#.kdq0o8rec

25 Tirrell Payton (2015). *User Stories In Detail.* AgileLeanHouse

26 Anna Georgsson (2011). *Introducing Story Points and User Stories to Perform Estimations in a Software Development Organization*, pp.5-18

27 Android Documentation. *Audiotrack* [Internet]. Available at: https://developer.android.com/reference/android/media/AudioTrack.html

28 Android Documentation. *SoundPool* [Internet]. Available at: https://developer.android.com/reference/android/media/SoundPool.html

29 Android Documentation. *MediaPlayer official documentation* [Internet]. Available at: https://developer.android.com/reference/android/media/MediaPlayer.html

30 Google Developers. *Adaptative video streaming on Android* [Internet]. Available at: https://www.youtube.com/watch?v=6VjF638VObA&t=196s

31 Boehm, B (1989). *Software Risk Management: Principles and Practices.* Journal IEEE Software Volume 8 Issue 1. IEEE Computer Society Press Los Alamitos, CA, USA.

32 Android NDK. *Audio latency* [Internet]. Available at: https://developer.android.com/ndk/guides/audio/audio-latency.html

33 Simon Brown (2014). *Software Architecture for Developers*. Lean Publishing

34 Robert C.Martin. *The clean architecture* [Internet]. Available at: https://8thlight.com/blog/uncle-bob/2012/08/13/the-clean-architecture.html

35 Fernando Cejas. *Architecting android the clean way* [Internet]. Available at: http://fernandocejas.com/2014/09/03/architecting-android-the-clean-way/

36 Bass, Clements and Kazman (2012). *Software Architecture in Practice*. Addison-Wesley Professional. ISBN: 978-0-321-81573-6

37 Erik Caffrey. *Android clean Architecture* [Internet]. Available at: https://erikcaffrey.github.io/ANDROID-clean-architecture/

38 Android Documentation. *AudioFormat documentation* [Internet]. Available at: https://developer.android.com/reference/android/media/AudioFormat.html

39 Thompson, Dan (2015). *Understanding Audio*. Berklee Press. ISBN: 0-634-00959-1

40 Digital Formats. *RIFF (Resource Interchange File Format)* [Internet]. Available at: https://www.loc.gov/preservation/digital/formats/fdd/fdd000025.shtml

41 Android NDK. *Sampling Audio – Android guidelines* [Internet]. Available at: https://developer.android.com/ndk/guides/audio/sampling-audio.html

42 Gupta, Prakash C (2016). *Data Communications and Computer Network*. PHI Learning

43 Android Developers. *Audio Record* [Internet]. Available at: https://developer.android.com/reference/android/media/AudioRecord.html#read(short [], int, int)

44 Sung Park. *Difference Between SoundPool and MediaPlayer* [Internet]. Available at: https://medium.com/sketchware/difference-between-soundpool-and-mediaplayer-bb79cda8bafc

45 Ian C. *Everything about Pan Law* [Internet]. Available at: https://www.youtube.com/watch?v=vMgCSGI45Cw