Super Looper: Audio looping app with MIDI support

Memoir

Author: Adam Scher Gabín
Director: Xavier Burgués Illa
Software Engineering

Abstract

Music is part of all of our lives. It can make us feel so many different things and be so beautiful and interesting that everyone should try creating or just playing around with music. The objective of this project is to create a looper/drum pad style app that allows the user to create and perform different music using default sounds or their own. It will also support the MIDI protocol, allowing the user to control external MIDI gear through USB. It has been developed for Android since the existing solutions are lacking and not many. Because of this the project has been developed using Android Studio, Kotlin, and Oboe in the most efficient way possible.

Resumen

La música es parte de todas nuestras vidas. Puede hacernos sentir tantas cosas diferentes y ser tan hermosa e interesante que todos deberíamos intentar crear o jugar un poco con la música. El objetivo de este proyecto es crear una aplicación al estilo looper/drum pad que le permita al usuario crear o tocar música diferente usando sonidos predeterminados o propios. También admitirá el protocolo MIDI, lo que permitirá al usuario controlar dispositivos MIDI externos a través de USB. Ha sido desarrollado para Android ya que hay pocas aplicaciones de este estilo y las que hay carecen de varios aspectos. Debido a esto, el proyecto se ha desarrollado utilizando Android Studio, Kotlin y Oboe de la manera más eficiente posible.

Resum

La música forma part de totes les nostres vides. Ens pot fer sentir tantes coses diferents i ser tan bonica i interessant que tothom hauria de provar fer-ne o jugar-hi. L’objectiu d’aquest projecte és crear una aplicació d’estil de looper/drum pad que permeti a l’usuari crear i tocar diferents músiques mitjançant sons predeterminats o propis. També donarà suport al protocol MIDI, cosa que permet a l'usuari controlar dispositius MIDI externs mitjançant USB. S'ha desenvolupat per a Android, ja que falten les solucions existents i poques. Per això, el projecte s'ha desenvolupat utilitzant Android Studio, Kotlin i Oboe utilitzant la forma més eficient possible.

Index

1. Context

1.1 Introduction

This project is a Bachelor Thesis written at the Barcelona School of Informatics[1] in the Polytechnical University of Catalunya. It belongs to the Software Engineering specialty and has been written by Adam Scher Gabin. This project is intended as a musical performance tool to be used by anyone.

I would like to give a bit of context as to my relationship with music and how it has influenced my decision for this project. I started playing piano when I was younger, attending a local classic music school. After some years of that I decided to switch to a modern music school where I learned things like jazz and blues. After a couple years of that I began to get a bit tired of it; jazz and blues are amazing but I personally wanted to begin exploring new ways of making music.

That's how I began getting into music production. As opposed to the more traditional type of band and music where a song is done by different people playing together, musical production involves making and arranging the music on a computer, even if real instruments are used. The advantage of this is that computer software allows the musician to do things slower and with a lot more detail. I've been producing music for the last year after playing a long time and have decided to keep learning in this direction.

1.2 General Concepts

In this section I will present the general concepts and ideas that need to be understood for the rest of the project.

Music can be defined in many different ways and it largely depends on the one who is interpreting its' meaning. For most, music is an expression of emotions, created and presented by the artist for the purpose of enjoyment for themselves and for the listeners. Music is present everywhere around us and influences our lives every day.

An **app** is a program that runs on a smartphone. Phones have become a necessary part of our lives to the point where everyone almost always carries a phone with them. This project is directed towards phones because they're so readily available and relatively inexpensive compared to some instruments.

A **looper** is a program or piece of hardware that (usually) records from an input and then plays back the sound continuously. They're available as pedals for guitarists to use, as a piece of hardware for live performances, as computer software, etc. They can consist of different features, the most important ones being the number of tracks and the ability to overdub (record on top of an existing sound). Loopers are great devices since they can accompany the player while they play allowing them to add or play things on top.

Audio effects are hardware or software devices that manipulate how an audio signal sounds. There is a large number of them available and they are super common in music. Some of the most common audio effects are:

- Reverb, which makes it sound like the instrument is playing inside a space by giving the audio some echo.
- Delay, which takes the signal and repeats it back at an interval.
- Distortion, which makes the audio sound rougher and heavier.

Using the parameters that are available for each audio effect, the musician is able to configure them to change the original sound in many ways.

When effects first came out, they were analog circuits. The sound would go through them and, using electricity, condensers, resistors, etc. it would come out differently. Nowadays, if we want to emulate an audio effect digitally, **digital signal processing** (DSP)[2] is used. DSP allows a programmer to code an audio effect and digitally control the way the sound changes.

When synthesizers and drum machines and similar equipment originally came out musicians had no way to have them communicate with each other. This communication is useful for a number or different reasons, like synchronizing the tempos or playing one instrument from another keyboard. As a solution for this, **MIDI**[44] (Musical Instrument Digital Interface) was introduced in the beginning of the 80s. It basically enables different instruments or musical

devices to communicate with each other, greatly expanding the possibilities. MIDI is still in use, still being built with a lot of synthesizers, keyboards, drum machines, sequencers, etc.

UI and **UX**[3] are two really important factors to have in mind while developing an app. UI stands for user interface and UX stands for user experience. UI refers to the actual appearance of the interface, the look and feel, presentation, design, color choices, etc. UX focuses more towards the overall experience that the service offers, ensuring efficiency and ease-of-use during the process.

Super Looper is intended as a complementary “instrument” to any musician for a number of different occasions. The design is supposed to inspire creativity by making it easy to capture new sounds (or import existing ones) and using audio effects with them. It will also be ready to integrate into any live setup by using usb over MIDI to set a global tempo for the whole set.

1.3 Problem to be solved

Musicians are always looking for new sources of inspiration, a fast way to capture ideas and a tool that is easy to work with. Instruments and musical tools are traditionally in a hardware format, meaning that it’s in a physical device that has to be purchased for exactly that purpose. This means the musician has to pay a usually large amount of money for any new addition to their setup. Super Looper aims to solve this issue by presenting a performance-ready tool for a really low price since most people nowadays already possess a smartphone. It will present a software solution to a typically hardware (and expensive) device, saving the user money, time, and space.

1.4 Stakeholders

1.4.1 Musicians and producers

The main stakeholder that will benefit from this project are musicians and producers around the world. Super Looper is a great complementary addition to any setup, allowing all types of musicians and producers to improve and explore a new way to create sounds.

1.4.2 Audiences

Thanks to the musicians, the next main benefactors from this product would be the musician's audiences. Super Looper is a tool that could be used as part of a live (or recorded) performance. The audience of this performance would benefit from it by being able to listen and enjoy the audio from Super Looper.

1.4.3 Advertisers

In order to generate some revenue, ads would be part of the free version of the app. The advertisers would benefit from displaying their ads on our app since they would receive visits coming from our app. The companies that would be the most interested to display ads would be existing music companies.

1.4.4 The director of the project

The director assigned to me during this project is also interested in seeing this project succeed.

1.4.5 Me as a student, developer and musician

I'm personally involved in this project in many different ways, also making me an important stakeholder. As a student I want to make sure that my project does well; as a developer I'm potentially interested in continuing this project after the completion of my degree and; as a musician I'm looking forward to having this tool available to myself as well.

2. Justification

2.1 Existing Solutions

As is to be expected in today's day where apps tend to already exist for (almost) everything, some apps that are similar to my project are already available on the market. In this section I will talk about some existing solutions similar to my idea[4]. I will only be mentioning software

available on the Google Play Store, since I will be creating an Android app. Some better solutions exist on the Apple store which is also what inspired me to make this app specifically for the Android platform. Further on in this memoir (Section 10) there's a table comparing different apps by their features which goes more in depth than this section. Nevertheless, this initial assessment is enough to warrant the development of Super Looper, seeing that an app with its functionalities and feature set is not available.

The most popular looper app on the Play Store is called LoopStation - Looper[5] with more than 1,000,000 installs. That would lead you to think that it's a good app but in truth it has about 2.5 stars out of 5 with most reviews explaining that the app is unusable. LoopStation also only features quite basic functionalities, with only 4 tracks, only 1 effect (reverb) and no MIDI support. It does provide the ability to record the session and save it or share it but that doesn't make up for the fact that it can't even carry out the basic functionalities of a looper (since it's really hard to sync the loops properly(. The other looper type apps have similarly bad ratings and functionalities.

Quite a few of the other apps[6] are very simple looping apps which only provide one track for the user to simply loop existing audio on their phone. One of these apps is called Loop Player and even though it has good ratings and more than 100,000 downloads, it's not a musician's performance tool. These types of apps allow the user to select an existing audio (and sometimes video as well) loop a section of it in order to practice along or listen to the same thing over and over (for example, maybe to study from an eBook). One of the advantages that these apps usually have is that they allow the user to slow down or speed up the file.

One app present in the store that seems to get it right is called uFXloops Music Studio[7]. It has sample import, microphone recording, MIDI file support, pitch shifting, a sample editor, a sequencer, a simple effects processor, etc. The only issue with it is that the developer stopped supporting it some years ago even though the community is still relatively active and still has questions. Similar to uFXloops, there's Caustic. Caustic is worth mentioning because it's as close as possible to a full feature DAW on a computer, really allowing you to do a huge amount of things. This is way beyond my goal but it's good to keep it as a reference point.

2.2 Justification

After mentioning these existing android apps it's clear that there's no up to date app to work with loops and sampling as well as should be possible on Android. Most of the existing solutions have bad ratings or just don't offer the feature set I want Super Loops to have. Super Loops is worth pursuing because of its flexibility towards different genres, my continued developer support and the lack of similar apps on the Android platform. The apple store has some well received looper/sampler apps which is why I'd like to make mine for Android, to give the opportunity of working with a looper/sampler app on Android without having to pay the Apple premium. The interface will be designed to inspire artists and create sonic possibilities that wouldn't be possible using the existing apps. Android has always had a problem with audio latency which this project will also try to improve on.

3. Scope

3.1 Objectives and sub-objectives

To create a looper/drum pad type app that allows a user to either record his own sounds or import them from their files, easily edit them, and be able to play them back live or with a basic arrangement. Sound effects will be available to add movement and/or change the existing sounds and MIDI will allow the app to sync to external devices.

The most important objective for this project is for it to be useful to the musicians using it. I want this app to inspire creativity and new ideas, allowing for the user to create new concepts or better existing ones at the touch of their smartphones. I hope to make this a well supported tool that is chosen by different artists from around the industry.

- **Processing audio in Android**. This is a clear and essential objective since my whole app will be centered around audio and making it easy for the user to arrange and work with it.
    - Playing and recording audio tracks
    - Looping tracks

    - Volume level per track
    - Panning per track
    - Recording a session
    - Audio effects (per track or global)

- **Working with MIDI**. MIDI is the standard communication protocol between instruments and the addition of MIDI to the app will allow for more flexibility and possibility of use with external instruments.
    - Connecting a MIDI device to an Android phone
    - Sending out MIDI start and stop commands and timing information

3.2 Requirements

In computer science, requirements are divided into functional and non-functional requirements[8]. Functional requirements deal with how the software should perform while non-functional requirements deal with the quality of the system and attributes that aren't as apparent to the user. Here are some requirements that our project will need to meet in order to verify its quality.

3.2.1 Functional Requirements

- The interface is intuitive and usable.
- The user can choose to work with imported audio files or by recording their own voice/instruments directly from the microphone or attached audio device.
- The user should have enough tools (especially audio effects) available to feel like they can make the sound "theirs".
- The user will receive visual confirmation that the action they performed was actually carried out by the app.

3.2.2 Non-Functional Requirements

- Portability: The app will be portable because it’s simply installed on a smartphone (which are portable by nature).
- Reliability: The software should run easily without any bugs or errors. The user should be able to trust it no matter the situation.
- Scalability: The project is to be developed having scalability in mind, allowing for more features and users to be added in the future.
- Performance: The software should run as smoothly as possible, making the most out of the available resources.
- Usability: The app’s interface should be intuitive and usable, allowing the user to utilize the app to the fullest extent as soon as they start using it.
- Audio quality: The app’s audio output should sound good. All the files should be rendered properly and sound as they're supposed to.

4. Methodology

4.1 Kanban

The methodology that I will employ during the duration of the project is the Kanban[9] methodology, which aims at assisting the developer with their task organization and completion through the visual aid.

In Japanese, kanban means billboard or poster, representing the visual focus that this methodology employs. During the development of a product, all the tasks are put up on a kanban board. The board consists of multiple columns, each labeled according to the phase they correspond to in the development of the task (like “To Do”, “Doing”, “Testing”, “Done”, “Discarded”, etc.). As each task is worked on, it’s moved to its corresponding column.

Kanban is considered an agile methodology because it aligns itself with the 12 principles of agile. Even if it’s not iterative, kanban is still considered agile because it's incremental. I chose it because it provides the developer with some flexibility while still keeping the general progress clear. Kanban allows you to easily change the priority or status of a task. It has proved

successful in aiding teams varying in number and deadlines for which I think it's a great methodology for my situation now and possible situations in the future.

4.2 Tools

4.2.1 Trello

Trello will be the tool that I will use for carrying out Kanban[10]. It's regarded as one of the best services to use with kanban and they even have a series of blog posts about getting the most out of Trello for Kanban. In the picture below you can see how a typical Trello board looks. The tasks move from left to right as they are worked on, making their progress clear.

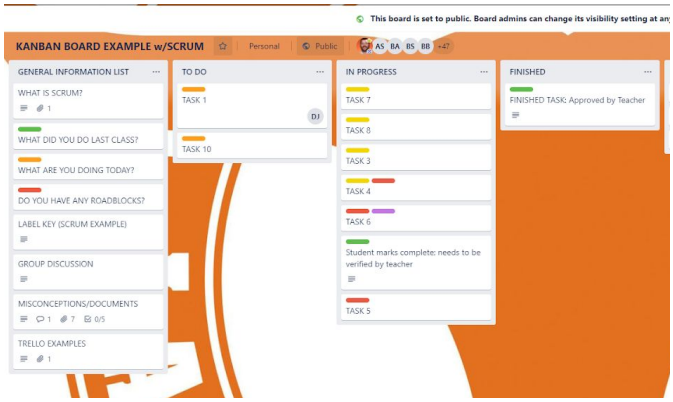

Figure 1: Kanban representation in Trello[11].

4.2.2 Git

Git[12] is a version control technology that allows the user to continually save and manage their work. It's also great for working in teams since different users can work on different branches, to not mix tasks, and then merge them when the different tasks on different branches have been completed.

4.2.3 GitHub

GitHub[13] is a cloud-based hosting service that allows you to manage your Git repositories. It's the most widely used tool for version control and I've been familiarized with it during the duration of my degree.

5. Task Definitions

In this section I will define all the tasks that will be necessary for me to carry out for a successful completion of the project. Each task will be accompanied by a short description about the task and the approximate amount of hours it will take to be completed. This planning is done by myself during the planning portion of the project meaning that I might not be able to predict all of the issues that may come up. This project was started on the 22nd of September and should be finished around the end of January[14].

5.1 Project Management

The objective of this iteration (group of related tasks) is to lay the foundations on which our project will be developed. It consists in creating the base documentation explaining the scope, objectives, etc. All of the following tasks will require a browser (in my case I use Google Chrome[15]) for research and Google Docs[16] for writing in.

- **PM0** - Preparation: Before embarking on a large project like this some time has to be spent brainstorming ideas and researching possible solutions to them. I estimate that 7h were spent during this task.
- **PM1** - Context and scope: This task entails defining the context, objectives, scope, requirements, and methodology used during the rest of the project. The estimated duration is 20h. Requires PM0 to be completed.
- **PM2** - Project planning: This task consists in planning out the development of the project. The tasks have to be defined and the necessary time for each in relationship to each other and the deadlines has to be defined. Time needed is approximately 15h. Requires PM1 to be completed.
- **PM3** - Budget and sustainability: During this task the necessary budget and sustainability reports will be generated. The time needed for this task is approximately 15h. Requires PM2 to be completed.

- **PM4** - Feedback: This task will consist in reading over the input provided by the tutor for the past tasks. I will edit the documents in the way needed to meet the tutor's expectations. This task will require approximately 5h. Requires PM3 to be completed.
- **PM5** - Final document: This task consists in taking all the previous documents and uniting them into one. It should take approximately 3h. Requires PM4 to be completed.
- **PM6** - Memoir documentation: This task consists in continuously writing about my progress in my memoir. I need to work on it consistently so all the work carried out is properly documented. About 50h spread out during the lifecycle of the project will be required. Requires PM3 to be completed.

5.2 Project Development

The tasks in this section will specify how the project will be developed. They are separated into groups based on their context and relationship in the app.

5.2.1 Project Inception

This subsection will deal with finding the right tool and becoming familiar with it in order to be able to work with it during the rest of the project. During these tasks I will decide the IDE (most likely Android Studio[17]) and technologies (most likely Android programming in Java using the Oboe library[18]) to use for the rest of the project.

- **PI1** - Research: This task consists in researching and deciding the programming language and technologies that will be employed during the rest of the project. It needs about 10h to be completed.
- **PI2** - Testing: This task consists in getting familiar with the technology. Making a couple really simple applications to understand the different elements of the libraries and programs used will allow the rest of the project to go by more fluidly. A test program consisting of the basic functions in the Audio programming section below will be developed on each of the considered platforms. About 20h will be needed.
- **PI3** - Design: The functionality and design of this project are closely tied together. A mockup will be designed indicating the possible screens, the flow between them, and the functions that each screen presents. This task needs about 10h.

5.2.2 Audio programming

This subsection will deal with the most basic aspects of getting audio working in my application. During this and the following sections of my project I will use the technologies chosen in the last part.

- **AP1** - Audio play: This task consists in simply making an audio file play at the tap of a button. As this first element of the project it will need about 10h.
- **AP2** - Audio record: This task will deal with the ability to record audio into my app via the selected audio input. This task will need about 10h.
- **AP3** - Audio looping: This task will deal with the ability to seamlessly loop audio recordings. It needs about 10h. Requires AP1 to be finished.
- **AP4** - Audio mixing: This task will deal with the ability to play sound from multiple sources. About 20h will be needed. Requires AP1 to be finished.
- **AP5** - Interface design: Once the functionalities are working the app has to be visually appealing and also perform its functions. The interface will have to be designed having multiple needs and possibilities in mind. About 40h will be needed on this task.

5.2.3 Sequencer programming

This subsection deals with the sequencer programming. A sequencer is a piece of software or hardware that allows you to record what you play as computer information. A sequencer triggers the audio to sound, it doesn't directly record or create the sound.

- **SP1** - BPM solution: Find and implement a way to keep the phones' BPM in sync with the "real world" BPM. This should take about 10 hours.
- **SP2** - Sequencer solution: The objective of this task is to find a software solution for a sequencer which will trigger the pads as how its been recorded. This task will require 40h as a sequencer can become really complicated and have a huge number of different functionalities. Requires SP1 to be completed.
- **SP3** - Sequencer interface: This task will deal with how the sequencer looks and how the controls allow the user to use it easily and effectively. About 20h will be needed.

5.2.4 MIDI programming

This section of tasks will deal with the implementation of MIDI.

- **MP1** - Basic MIDI implementation: This task consists of installing the MIDI library and making sure that the phone is able to recognize an external instrument plugged in by usb. This task needs approximately 5h to be completed.
- **MP2** - BPM information: During this task the sending of the current BPM will be sent to the instrument connected by usb. This task requires 10h to be completed. Requires MP1 to be completed.

5.2.5 Audio effects programming

This section will deal with the programming of the different audio effects my program will have.

- **FXP1** - Audio effect behavior: This task will consist of deciding whether to implement the effects as global effects or as per-track effects. Afterwards this will have to be implemented. Approximately 20h will be needed for this task.
- **FXP2** - Audio effects implementation: After the audio routing for the effects is completed in the previous task, the actual effects have to be implemented. This will take approximately 40h. It's so much time because I've never worked with audio effects and there's many different possibilities. Requires FXP1 to be completed.
- **FXP3** - Audio effects interface: This task focuses on developing an understandable and usable interface to interact with the effects. It would be possible for each effect to have adjustable parameters, for which more time will have to be spent on it. About 20h will be needed to complete this task.

5.2.6 Interface design and general improvements

The tasks in this section will deal with the interface and the general user experience.

- **ID1** - General interface design: This task will consist in making sure the interface is consistent on all the different pages. It will require about 20h.
- **ID2** - General interface functionality: This task will make sure that all the functionalities work as expected. This will require about 30h.

5.2.7 Final tasks

This section will deal with the last tasks that will have to be carried out before the project can officially be called complete:

- **FT1** - Writing documentation: This task will be focused on finishing the memoir. It will consist mainly of finishing the documentation but will also entail editing and formatting. It should take around 10h.
- **FT2** - Thesis presentation: This task will be dedicated to getting prepared for my final thesis defense. It will be focused around creating a demo, powerpoint slides, and practicing. It should take about 40h.

5.3 Task Table

ID	Task	Time Required	Dependencies	Resources
PM	**Project Management**	**115h**		
PM0	Preparation	7h		Google Chrome, Drive, and Docs
PM1	Context and scope	20h	PM0	Google Chrome, Drive, and Docs
PM2	Project planning	15h	PM1	Google Chrome, Drive, and Docs
PM3	Budget and sustainability	15h	PM2	Google Chrome, Drive, and Docs
PM4	Feedback	5h	PM3	Google Chrome, Drive, and Docs
PM5	Final document	3h	PM4	Google Chrome, Drive, and Docs
PM6	Memoir documentation	50h		Google Chrome, Drive, and Docs
PI	**Project Inception**	**40h**		
PI1	Research	10h		Google Chrome
PI2	Practice	20h	(PI1)	Android Studio, Google Chrome
PI3	Design	10h	(PI1)	Mockup Creator

AP	**Audio programming**	**90**		
AP1	Audio play	10		Chosen technologies
AP2	Audio record	10		Chosen technologies
AP3	Audio looping	10	AP1	Chosen technologies
AP4	Audio mixing	20	AP1	Chosen technologies
AP5	Interface design	40		Chosen technologies
SP	**Sequencer programming**	**70**		
SP1	BPM solution	10		Chosen technologies
SP2	Sequencer solution	40	SP1	Chosen technologies
SP3	Sequencer interface	20		Chosen technologies
MP	**MIDI programming**	**15**		
MP1	Basic MIDI implementation	5		Chosen technologies
MP2	BPM information	10	MP1	Chosen technologies
FXP	**Audio effects**	**80**		

	programming			
FXP1	Audio effect behavior	20		Chosen technologies
FXP2	Audio effect implementation	40	FXP1	Chosen technologies
FXP3	Audio effects interface	20		Chosen technologies
ID	**Interface design and general improvements**	**50**		
ID1	General interface design	20		Chosen technologies
ID2	General interface functionality	30		Chosen technologies
FT	**Final tasks**	**50**		
FT1	Finish documentation	10		Google Chrome, Drive, and Docs
FT2	Thesis presentation	40		Google Chrome, Drive, and Docs
	Total	**510h**		

Table 1: Tasks and time required. Of personal creation.

6. Estimates and Gantt

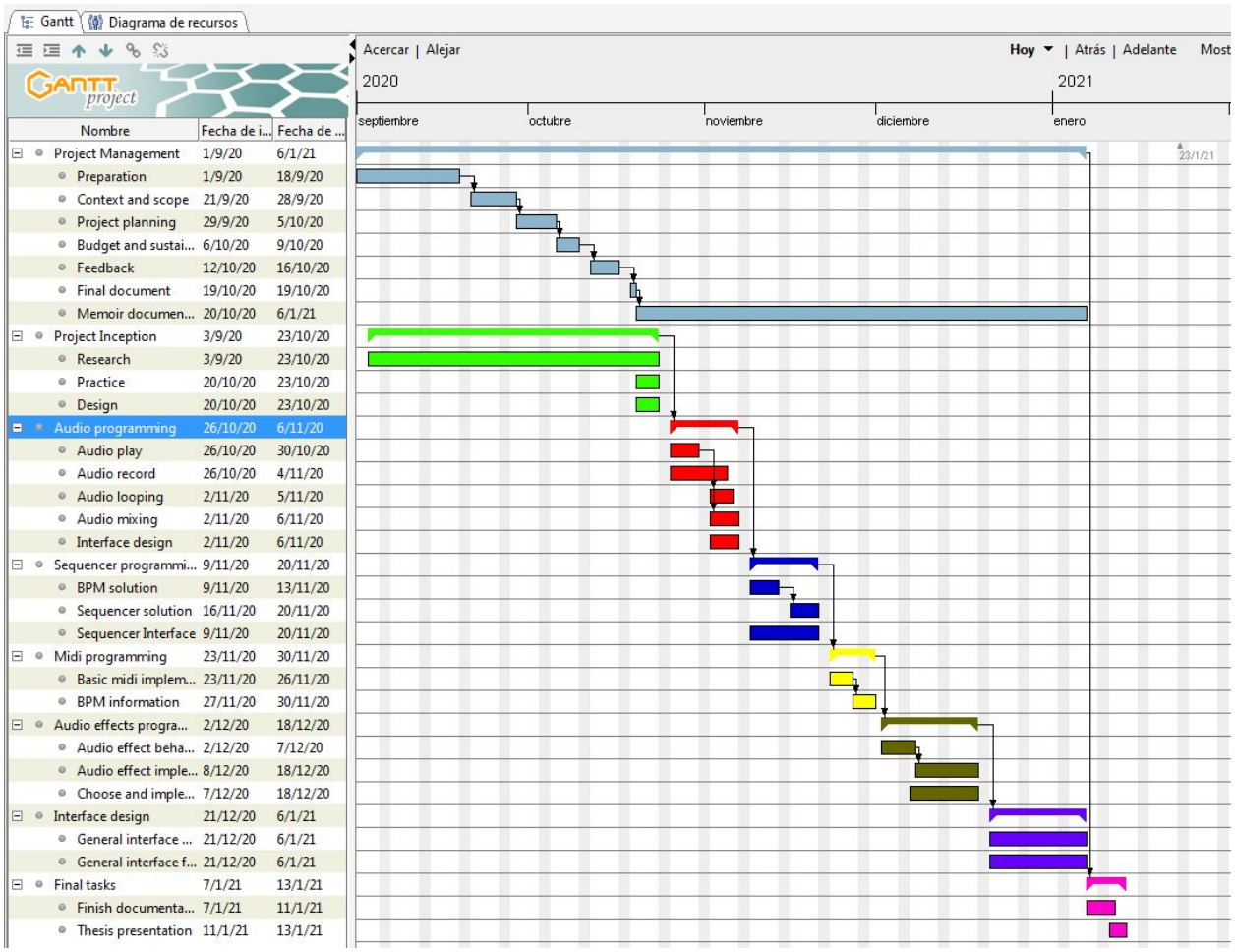

Figure 2: Gantt Diagram created by me using Gantt Project[19]

7. Risk management: alternative plans and obstacles

No matter how much planning and preparation goes into a project there's always the possibility that something goes wrong or just deviates from the plan[20]. This can potentially lead to a defective product, missed deadlines and general dissatisfaction from both the user and developer. In this section I will present some obstacles that may present themselves during the development of the project and the possible solutions and plans to be carried out if that were to happen.

- **Project plan and deadline**: This project being a thesis there's a clear deadline for when I have to turn in the project (and other documentation). This forces me to fully devote myself to this project in order to be able to finish it on time. I will have to make a good plan, stick with it, and avoid possible distractions in order to carry out the development on schedule. If any meaningful deviations to the planning occur during the development of the project, the planning is to be restructured accordingly. If necessary, the scope of the project will be reduced. This would entail removing one (or more) of the app functionalities. Which functionality would be discarded would depend on the current app progress.

- **Unfamiliarity with the technology**: Even though I've worked with music in the past it's only ever been as the user of this technology. I've played different electronic instruments and on different musical apps but I've never built anything audio related in the past. I'm very interested in this area of technology and really excited to learn about it but the lack of existing documentation and information will make this a hard task to carry out. There's also little information available as to which technologies are recommended for a project like this, making it harder for me to choose the right elements to create the project. Sadly there's not a whole lot that can be done about this point apart from the natural learning/researching process. If obstacles with the technology appear I'll have to research how to solve it or, if necessary, discuss it with my tutor.

- **Problems with libraries**: As expected during the execution of my project I will be working with different libraries (most importantly, libraries directed towards sound).

Libraries are a great help but they can always contain bugs that don't allow it to work correctly or not offer enough functions for what is necessary. If this situation arises two different solutions can be carried out. Either I can explore the library implementation and try to fix it or submit a bug report to the library creators and hope that they can solve the problem as soon as possible.

- Meeting with the tutor: In order to advance and overcome certain obstacles, the student should meet semi-regularly with their tutor. Given the current situation with the Covid-19 pandemic, that will be more complicated than during normal years, forcing us to communicate online or over the phone. This makes it harder to contact each other and communicate, possibly slowing down my progress on the app. If any problems arise with the professor about meeting they shall be mentioned and spoken about. In the worst case, a new tutor will be found to direct my project.

- Bugs and implementation issues: No matter how hard we try or how good we are at coding most code never runs on the first try. During my project I will have to deal with a lot of bugs that I'm not familiar with and patiently debug the code until the specific feature works. When this issue is faced there's not many alternatives or options available except debugging. The process of debugging implies first finding out what's wrong, finding where it's caused in the code and fixing it. It might take multiple tries and attempts at different solutions but that's what it takes.

8. Budget

Once the planning for the project development has been completed it's time to calculate and estimate the necessary costs for the development of the application. Different types of costs will be calculated: personnel, general and other costs. Apart from this, a contingency plan will be developed, where potential issues will be taken into account and how to keep the budget on track if they do.

8.1 Personnel Costs

After having laid down all the tasks to be completed in the previous task table and Gantt chart, we have to identify the actors for each task, how much time they will have to spend on each task, and how much it costs to pay each of them. For my project, a project manager (PM), Researcher (R), Programmer (P), Tester (T), Interface Designer (ID) and Technical Writer (TW) are needed. My GEP tutor, my thesis director and I will fulfill the role of project manager while I alone will complete the rest of the roles. In the Table 1 below you can see how much each of these positions are paid on average[21]. These prices don't include taxes, which can be obtained by multiplying the cost of their salaries by 1.35.

Role	**Salary per year**	**Salary per hour**
Project Manager	44800€	21.54€
Researcher	45000€	21.63€
Software Developer	27400€	13.17€
Tester	20300€	9.76€
UI Designer	34500€	16.59€
Technical Writer	38000€	18.27€

Table 2: Salary per role. Of personal creation.

In Table 3 below you can see how the hours have been distributed by person and by task and how the total cost of the personnel salaries has been calculated.

Activity ID	**Task**	**Time Required**	**Personnel Hours**						**Cost**
			Project Manager	Researcher	Software Developer	Tester	UI Designer	Technical Writer	
PM	**Project Management**	**115**							
PM0	Preparation	7	2	5	0	0	0	0	151.25€
PM1	Context and scope	20	20	0	0	0	0	0	430.77€
PM2	Project planning	15	15	0	0	0	0	0	323.08€
PM3	Budget and sustainability	15	15	0	0	0	0	0	323.08€
PM4	Feedback	5	3	0	0	0	0	2	101.15€
PM5	Final document	3	1	0	0	0	0	2	58.08€
PM6	Memoir documentation	50	50	0	0	0	0	50	1990.38€
PI	**Project Inception**	**40**							
PI1	Research	10	0	10	0	0	0	0	216.35€
PI2	Practice	20	0	0	10	10	0	0	229.33€
PI3	Design	10	0	0	0	0	10	0	165.87€

AP	**Audio programming**	**90**							
AP1	Audio play	10	0	2	5	3	0	0	138.41€
AP2	Audio record	10	0	2	6	2	0	0	141.83€
AP3	Audio looping	10	0	2	6	2	0	0	141.83€
AP4	Audio mixing	20	0	2	15	3	0	0	270.14€
AP5	Interface design	40	0	0	0	5	35	0	629.33€
SP	**Sequencer programming**	**70**							
SP1	BPM solution	10	0	2	6	2	0	0	141.83€
SP2	Sequencer solution	40	0	5	32	3	0	0	558.99€
SP3	Sequencer interface	20	0	0	0	2	18	0	318.08€
MP	**MIDI programming**	**15**							
MP1	Basic MIDI implementation	5	0	1	5	0	0	0	87.50€
MP2	BPM information	10	0	1	8	1	0	0	136.78€
FXP	**Audio effects programming**	**80**							
FXP1	Audio effect behavior	20	0	2	15	3	0	0	270.14€

FXP2	Audio effect implementation	40	0	8	30	2	0	0	587.79€
FXP3	Audio effects interface	20	0	0	0	2	18	0	318.08€
ID	**Interface design and general improvements**	**50**							
ID1	General interface design	20	0	0	8	3	9	0	283.94€
ID2	General interface functionality	30	0	0	20	8	2	0	374.71€
FT	**Final tasks**	**90**							
FT1	Finish documentation	10	0	0	0	0	0	10	182.69€
FT2	Thesis presentation	40	0	0	0	0	0	40	730.77€
	Total	**510h**	0						9302.16€
	Taxes		0						35.00%
	Total with taxes								12557.92

Table 3: Task time per person and cost. Of personal creation.

8.2 Generic Costs

Apart from the obvious salaries that personnel need to be paid there's other expenses involved in the development of an app. These aspects will be covered in this section.

8.2.1 Hardware

Cearly, to develop an app, we need different software solutions. Beyond that, though, we need a hardware device that can run that software. During the development of the project I will be using my Xiaomi Air Laptop for programming, research, documentation and testing and the Samsung Galaxy A10 for further testing. I will also use an external monitor to make coding and navigation easier. To calculate the cost for each device we will use the original price, it's total use time and the time that we will dedicate to our project on that device. The formula is:

Project Cost = (hours for project/total hours)*new price

Device	**New price**	**Hours per year**	**Years**	**Total Hours**	**Hours for project**	**Project cost**
Xiaomi Air	800€	1000	3	3000	500	133.33€
External Monitor	150€	500	6	3000	250	12.50€
Samsung Galaxy A10	140€	700	2	1400	300	30.00€
Total cost						**175.83**€

Table 4: Hardware costs. Of personal creation.

8.2.2 Software

Even though I haven't fully tested and decided which software I'll use during the duration of the project I am quite certain that I will use Android Studio[22] and the C++ audio management library Oboe[23]. Both of these are free tools so thankfully my budget doesn't have to worry about paying for software.

8.2.3 Workspace

In my case I will carry out the project while working for home. The rent for my household is 1700€ which divided by the four people living here gives us 425€ a month in apartment rent. Considering that I would live there anyways, only the time spent programming will be counted. The project is supposed to take 510h and 5 months to develop, meaning 102h should be dedicated every month. Considering a month is 30 days and there's 24h per day, there's 30x24=720h per month. The proportion time living per month to time working per month is 102h/720h=0.142. If the rent per month is 425€, the amount of money spent on the rent during the project development will be 0.142*425€=60.5€. Since the project will take 5 months the price of the workspace will be 5x60.5€=302.5€.

8.2.4 Internet

Since the internet will be used during the whole duration of the project it's also necessary to factor it into our budget. A good internet connection with Vodafone can cost around 32eu per month[24]. Dividing this by the 4 people living in the space to get 8eu per person per month for the internet and since the project will take 5 months the total price for the internet will be 8x5=40€. Since I will only use the internet for the project during 3.5h each day: 40x(3.5h/24h)=5.83€.

8.2.5 Electricity

The average electricity cost per month in Barcelona is that of 54.99€[25]. Since we will use electricity for our project during 3.5h each day during approximately 5months the total electricity budget will be 55x(3.5h/24h)=8.20€. Some of that would be used anyways but this number is so small that it wouldn't have a meaningful impact on the budget.

8.3 Potential budget deviations

There are different situations where something unexpected can happen that will change the budget needed to complete the project. In this section we will identify the most possible deviations and plan ahead so we know what to do in case any of these occur.

It’s always safe to assume that things won/t go precisely according to plan. Our actual expenses can exceed the expected costs forcing us to have to resort to other methods to obtain the necessary resources. A way to try and foresee these issues is a contingency. It’s just another budget item that raises the budget by a certain margin (in my case 15%).

8.3.1 Incidentals

During the lifecycle of the project we can already expect certain issues to come up. It’s important to be aware of these and plan for one or two of the following things to go wrong:

- Project plan and deadline: One of the possible risks of these projects is that I have over exceeded the possible reach of my project and don’t have enough time to complete all my objectives. If this occurs, the app won’t have as many features and the budget will be used to redistribute the tasks among the right personnel. The probability of this happening is about 20%. It would require an extension of the budget of 300€.
- Unfamiliarity with technologies: The risk of running into problems when trying to implement technologies that I’m not very familiar with is quite large. The solution would be to invest more time and money into learning these technologies. The probability of this happening is about 25% and would require 200€.
- Problems with existing libraries: The biggest problem that I’m afraid I’ll run into is lack of support for audio on Android. Apple has a really good API [26] which programmers have used for years to create well functioning audio apps for Apple products. On the other hand, Android has little support for audio. During the history of Android different phones have been made different ways and Android has had to adapt. Audio on Android devices can either be managed by OpenSL ES or AAudio, adding to the confusion, but recently Google released Oboe as a framework for both of these. I’m worried that Oboe won’t work as well as I hope it does as it only makes it easier for the programmer to communicate with OpenSL ES or AAudio. The probability of this happening is 15% and it would require a budget extension of 600€.
- Bugs and implementation issues: No matter how much experience any programmer has some bugs and implementation issues are bound to appear. To solve these more time and money will have to be spent. The probability of this happening is 30% and would cost around 250€.

Incidental	Cost	Chance	Budget allocation
Project plan and deadline	300€	20%	60€
Unfamiliarity with technologies	200€	25%	50€
Problems with existing libraries	600€	15%	90€
Bugs and implementation issues	250€	30%	75€
Total incidental cost			**275**€

Table 5: Incidentals. Of personal creation.

8.4 Total costs

Following is the table obtained by adding together all the costs.

Item	Cost	Observation
Total activity costs (AC)	12557.92€	Table
Hardware costs	175.83€	Table
Software costs	0	Free software
Workspace costs	2125€	
Internet costs	5.83€	
Electricity costs	8.2€	
Total generic costs(GC)	2314.86€	
Sum of costs	14872.78€	AC+GC
Contingency	2230.92€	15%
Incidentals	275€	Table
Total budget	**17378.70**€	AC+GC+Contingency+Incidentals

Table 6: Total budget. Of personal creation.

8.5 Management control

Once the initial budget is decided it is important to be able to monitor it in comparison to the real time costs that come up while the project is being developed. Every time a task is completed during the realization of my project the amount of hours that it took will be logged and compared to the budgeted time.

In order to be able to detect and prevent deviations it is important to calculate actual values and compare them to the estimated ones. The following formulas will be used for this:

- Task cost deviation

(estimated cost per hour - real cost per hour) * total hours

- Hardware amortization deviation:

(estimated hours used - actual hours used) * price per hour

- Electricity deviation:

(estimated electricity cost - actual electricity cost) * price per hour

- Incidental cost deviation:

(estimated incidental hours - actual incidental hours) * total incidental hours

9. Existing Solutions

In the initial phase of the project I conducted and documented some basic research on existing solutions to a looper type app. I realized that my idea was a bit vague and found that it was important to compare it to existing solutions to see how to make Super Looper stand out and therefore, successful.

In Table 6, different Android looper and drum pad apps are compared by features. These features are the ones found most frequently throughout these apps (and Super Looper). In order to create this table, I searched “looper” and “drum pad” on the Play store and picked the most popular apps. In Table 7, the same process is carried out for Apple apps. In general, there’s many more apps on the Apple store than on the Android Play Store.

Taking a look at the tables one can see that Super Looper has almost all of the features the other apps have. The only feature Super Looper doesn’t support are sound effects, which originally, were going to be included. The most differentiating feature is, by far, MIDI support. Only two of the other apps support MIDI, Caustic and ufx Loops Music studio. Of these, only Caustic supported external device communication; and it only supports input connections, as opposed to Super Looper.

9.1 Android Solutions

	MIDI	Audio Record	Audio Files	Metronome	Custom tempo	Sequencer	Gain controls	Pan controls	Audio FX	Rating	Packs	Custom Packs
LoopStation [6]	N	Y	Y	Y	Y	N	Y	N	Only reverb	2.2	N	N
Loop Player [5]	N	N	Y	N	N	N	N	N	N	4.5	N	N
ufxloops Music Studio [7]	MIDI Files	Y	Y	Y	Y	Y	Y	N	Reverb, delay	4	Y	Y
Caustic [28]	Y	Y	Y	Y	Y	Y	Y	Y	Y	4.2	-**	-
Groovepad [29]	N	N	N*	N	N	N	N	N	Y	4.8	Y	N
Loopify Beta [30]	N	Y	N	Y	Y	N	Y	N	N	-	N	N
My drum pad [31]	N	Y	Y	Y	Y	N	Y	N	Per pad delay	4.8	Y	Y
Remix Live [32]	N	Y	Y	Y	N	Y	N	N	Y	4	Y	N
Super Looper	Y	N	Y	Y	Y	Y	Y	Y	N	-	Y	Y

Table 6: App comparison. Of personal creation.

Legend: Y = yes, N = no

*Beyond packs, caustic allows the user to create a full song.

**Allows the user to record the session but not to import or export individual sounds/loops

9.2 Apple Solutions[33]

	MIDI	Audio Record	Audio Files	Metronome	Custom tempo	Sequencer	Gain controls	Pan controls	Audio FX	Rating	Packs	Custom Packs
Loopy HD:Looper[34]	Y	Y	Y	Y	Y	N	Y	N	Y	4.7	Y	Y
Yellofier[35]	N	Y	Y	Y	Y	Y	N	N	Y	4.2	Y	Y
Blocs Wave[36]	N	Y	Y	Y	Y	Y	Y	N	N	4.6	Y	Y
Jam Looper[37]	N	Y	N	N	N	N	Y	N	Y	4.4	N	N
Drum Pad Machine[38]	N	N	N	Y	N	Y	N	N	N	4.5	Y	N

Table 7: App comparison. Of personal creation.

Taking a look at the Apple apps one can see that in general the ratings are already better. Also, some of these apps cost money (Looper HD 5.49€, Yellofier 6.99€). It seems that the most successful ones (based on number of downloads and rating) are actually the most simple ones, giving the most functionality without confusing the user with extra features. Also, a couple of the apps (RemixLive and Grooveshark) are available on both platforms, greatly expanding the user base.

10. Research and Theory

For any project that involves programming several things need to be considered. What are the requirements that have to be met? How can all the different functionalities be achieved? What technologies are best to accomplish my objectives?

During the first couple weeks of my project development I researched and tested a couple of different ways to implement the app. One of the most important criteria that has to be taken in mind is low latency [39]. Latency is the time that it takes for an action to produce its result; in this case I want my app to react as fast as possible and make a sound as soon as a pad is pressed.

Low latency is difficult to achieve because of the actual route the audio data takes from the original file to the sound coming out of your headphones or speaker. In Figure 3[40], you can see the typical audio route (which includes different steps, each one adding a little bit of computation time to the process).

Apart from the latency caused by software, the DAC included in any electronic device prepared to play audio inevitably adds some latency to the process. A DAC or ADC (short for Digital to Analog Converter or Analog to Digital Converter) is the component in charge of converting digital data (the audio file) into analog data (the actual sounds the speaker or headphones make) or vice versa. This process will take some time regardless of the device.

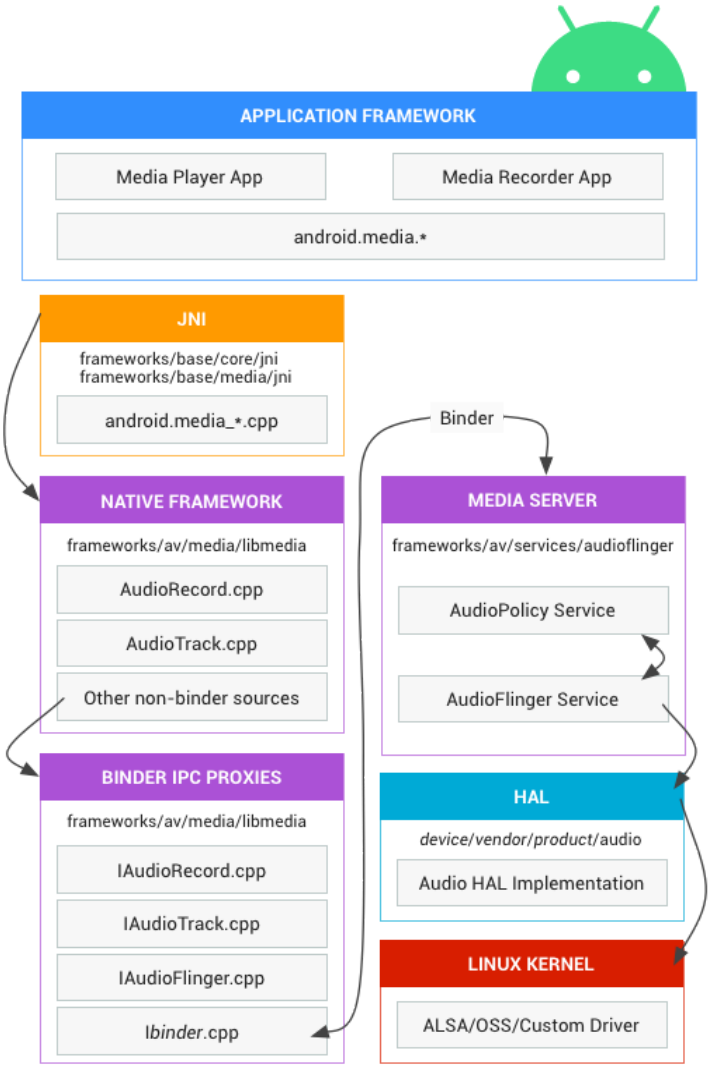

Figure 3: Android audio kernel.[40]

10.1 Audio Management Library

In order to achieve the lowest possible latency it's of the utmost importance that the right library is chosen. Multiple solutions exist for simple audio playing but my app is a lot more complicated since I want to mix multiple audio sources, potentially add audio effects, and have in general much more control than a simple play and stop of one audio file.

10.1.1 Oboe

Reading the Android Developer Guides[41], I found out that historically, two different libraries have been used to manage audio. OpenSL ES was used until AAudio was released in 2017. AAudio is able to achieve much lower latency than OpenSL ES and is the recommended option. At the release of AAudio the developers announced that they would also release a wrapper for these libraries called Oboe. Oboe will choose AAudio whenever available and fall back on OpenSL when it's not.

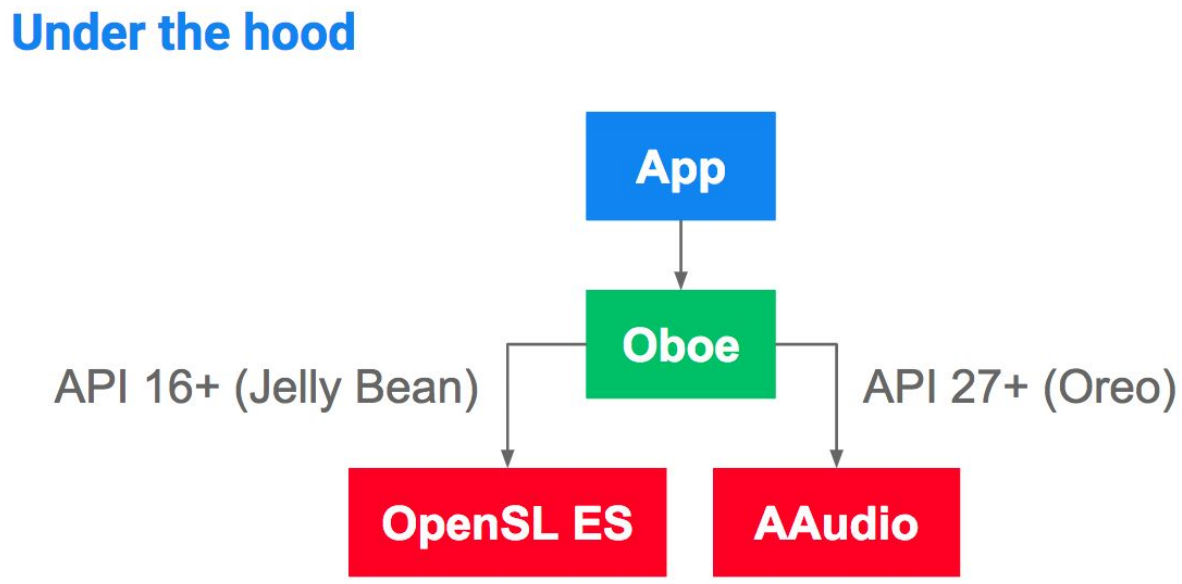

Figure 4: Internal Oboe library selection

Oboe[42] is a C++ library which makes it easy to build high-performance audio apps on Android. Oboe bases itself on the concept of streams to communicate with the exterior, using input streams to read audio and output streams to write audio. It also includes some simple sample programs that show how different things can be accomplished. It has a relatively active community around it allowing the developers to easily find out more information about something specific and receive help from other users.

I ended up choosing Oboe because of its ease of use and recommendation by the Android Developer Community.

10.1.2 Unity

At the beginning of my project Unity was suggested to me because of the fact that video games should have the lowest possible latency since they also need to react to the user input as fast as possible (for example, if the user shoots in a game he should hear the sound as soon as possible). In theory, Unity[43], the most widely used video game development program, could work in my case. In practice, it wouldn't be very efficient for a couple of reasons.

The first reason for Unity not being such a great solution is the audio engine itself. Sure, it can play audio at low latency, but it doesn't provide much control over the audio at all. The only parameter to ensure good audio performance is DSP Buffer Size, which can be set to "Best Latency", "Good Latency", and "Best Performance"[44]. On the other hand, Oboe is a lot more precise than this and gives additional options to achieve the best latency.

Also, the way Unity actually plays sounds is by assigning them to different Audio Sources, which determine an object in the screen from where the sound comes from. That's not necessary in our case. It's also the only way to be able to loop an audio file and once that file is loopable that can't be changed.

The second reason why Unity isn't so great to create my app is the same reason why Unity isn't used to create normal looking apps. Unity, being a game engine, redraws the screen at every frame, unnecessarily using up a large amount of resources and draining the users' battery[45].

10.2 IDE and programming language

Once I chose to make an Android app it was pretty clear the best/easiest way for that was using Android Studio. Android Studio is a platform developed by Google for developers to easily make Android apps. It provides an all in one package with an IDE, device emulator, compiler, debugger, etc. I was a bit unsure about using it because there's quite a bit of controversy surrounding Android Studio and it's capabilities but it's still the best (and one of the only) tools to work on native Android apps.

Once I decided to use Android Studio, started working with it and analyzing the Oboe examples, I realized that Android supports two programming languages: Java and Kotlin. So my next question was, should I use Java or Kotlin?

10.2.1 Java vs. Kotlin[46]

Java has been around for a long time. It's employed in numerous different cases across many technologies and it's famous for it. It's also famous for not being so easy to work with.(maybe some sources?) On the other hand, Kotlin is a relatively new programming language that has been used mainly for developing Android apps and has a growing community. It's considered light weight and implements modern programming techniques.

In my case, I've never worked with Kotlin but have used Java for two or three different projects. But after thinking about it and testing both languages a bit I decided to go with Kotlin. Since this whole project is a learning experience for me, I thought why not learn something new and modern that might be able to give me an edge when working in the future.

10.2.2 C++[47]

Given that the Oboe library is programmed in C++, it will be necessary to utilize it to interact with oboe. Oboe presents multiple different functionalities but they need to be further developed in order to actually be usable; the audio stream needs to be set up and initialized, the effects need to be programmed, audio files have to be decoded, etc. These topics will be covered further in the following sections.

11. Specification

In the specification part of a project, the functionalities and architecture are described. This section is important because it sets the foundation for what the project should be able to do, what criteria has to be met in order to verify the functionalities, and how the project will be structured.

11.1 User Stories

User stories[48] describe simple unit tasks that the user wants to accomplish. They are described in a way that the user can understand easily and are accompanied by acceptance criteria that confirm if that functionality can be properly carried out. The validation of all the criteria will prove that the functional requirements are met.

User Story 1

Title: Basic loop play

Description: As a user I want to be able to start and to stop a loop by tapping on a looping pad.

Acceptance Criteria:

- When the user clicks a looping pad the pad will either start playing, if it was paused, or stop playing, if it was already playing.

User Story 2

Title: Basic one-shot play

Description: As a user I want to be able to trigger a sound by tapping on a one-shot pad.

Acceptance Criteria:

- When the user taps a one-shot pad the pad will play.

User Story 3

Title: Audio mixing

Description: As a user I want to be able to hear all the different pads at once.

Acceptance Criteria:

- All the different pads making sound will be mixed properly and then heard by the user.

User Story 4

Title: Volume Level

Description: As a user I want to be able to individually adjust the volume level of each pad.

Acceptance Criteria:

- After changing a volume level, the user will hear the volume level of the corresponding pad adjusted to their desired setting.

User Story 5

Title: Pan Level

Description: As a user I want to be able to individually change the right/left balance of any pad.

Acceptance Criteria:

- After changing the pan level, the user will hear the pad sound at the stereo location they set it (more to the right or left).

User Story 6

Title: Play/stop

Description: As a user I want to be able to play and stop all of the pads at once.

Acceptance Criteria:

- When stop is pressed, all the pads will stop playing back.
- When play is pressed, the pads that were playing when it was last stopped will continue playing.

User Story 7

Title: Sequence pads

Description: As a user I want to be able to program a one-shot sequence so that I don't have to physically play a pattern.

Acceptance Criteria:

- The playing sequence has to match the programmed sequence.
- The user hears the sequence mixed in with the rest of the pads.

User Story 8

Title: Record pad sequence

Description: As a user I want to be able to play a one-shot pad sequence and have it recorded to the sequencer.

Acceptance Criteria:

- The played sequence will be correctly recorded by the sequencer and played back.

User Story 9

Title: Clear sequences

Description: As a user I want to be able to clear all of the existing sequences at the click of a button.

Acceptance Criteria:

- After tapping the button, all the sequences will be emptied and no audio will be played back from the one-stop pads.

User Story 10

Title: Change pack

Description: As a user I want to be able to change to a different sound pack.

Acceptance Criteria:

- All the available sound packs are displayed as choices.
- After choosing a different sound pack, the corresponding sounds will be assigned to the pads.

User Story 11

Title: Create pack

Description: As a user I want to be able to create my own pack using sounds found on my device.

Acceptance Criteria:

- After selecting the sound files, tempo, and pack name, the created pack will be added to the available packs list.

User Story 12

Title: Midi connection

Description: As a user I want to be able to connect external MIDI gear to my phone and have it be recognized by Super Looper

Acceptance Criteria:

- After plugging in the device, Super Looper successfully connects to the external device and displays a message.

User Story 13

Title: Midi messages

Description: As a user I want my connected MIDI device to receive a play message followed by timing messages when play is tapped and a stop message when stop is tapped.

Acceptance Criteria:

- The connected MIDI device will play at the correct tempo when play is tapped and will stop playing when stop is tapped.

User Story 14

Title: Metronome toggle

Description: As a user I want to be able to turn on and off the metronome sound.

Acceptance Criteria:

- After toggling the metronome button, the user will either hear the metronome or not.

User Story 15

Title: Adjust metronome tempo

Description: As a user, I want to be able to adjust the metronome tempo.

Acceptance Criteria:

- The metronome ticks reflect the set tempo.

User Story 16

Title: Screen Navigation

Description: As a user I want to be able to easily navigate between the screens.

Acceptance Criteria:

- Screen navigation is clear and state is maintained throughout the app.

11.2 Architecture

The architecture of my app is divided into two sections: the android layer programmed in Kotlin, which takes care of the views, state, activities and fragments and the native layer, programmed in C++, which takes care of the audio playback.

The Android layer of the app has two main components, MainActivity and PadPlayer. MainActivity is the activity that handles the displayed screen and the calls between fragments and PadPlayer. A fragment is a screen (or part of one) and has a corresponding layout file and a class file. The layout file describes the physical appearance of each fragment, like where the buttons are located, which sliders are used, their sizes and colors, etc. The class file contains the different variables, functions and methods that are to be used by that fragment. Pad Player is my custom Kotlin class responsible for audio management. It's connected to the metronome,

which keeps track of time, and to the NDK layer, which physically plays back sound. This is where the calls to the NDK are performed. Midi Constants is a file containing different MIDI message values.

On the other hand we have the NDK layer. NDK stands for Native Development Kit and it allows developers to develop an app (or parts of one) using c++, which Oboe is coded in. For this part of the app I used certain functions and classes provided by Oboe to facilitate playing audio files.

The singleton pattern was used for PadPlayer and metronome since it's only necessary for 1 instance of each of those to be present during the execution of the program.

In the following diagram you can see the main structure of my app. The triangles represent the different fragments, the squares activities, the rounded rectangles regular classes, the hexagon a C++ file and the circle the Oboe library.

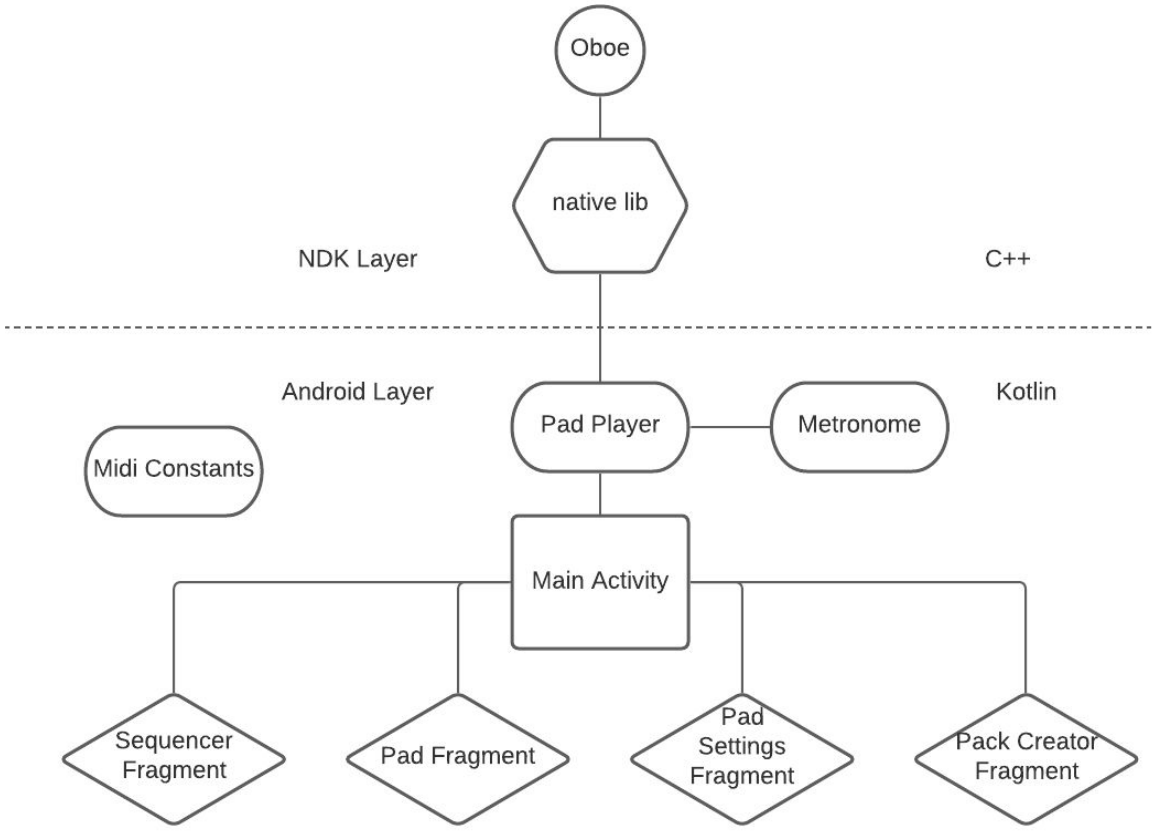

Figure 5: App architecture. Of personal creation

12. Implementation

In this section the implementation of the different features will be reasoned and explained.

12.1 Audio implementation

One of the key elements of Super Looper is audio. The app requires low latency audio transmission and the ability to work with audio in different ways. In the following sections different aspects of the audio implementation will be explored and explained.

12.1.1 Audio playback

The most important thing is that different audio tracks play (and play at a low latency, reacting as quickly as possible to the user's touch). For reasons previously explained in this paper, I chose Google's audio library Oboe to manage the audio aspect.

The actual audio processing and interaction with the android kernel is carried out by Oboe, which is programmed in C++. This requires my program to be split into two and to manage the app logic and design in Kotlin and audio processing in C++.These two parts of the same app operate using Android NDK, a toolset that allows parts of an app to be coded natively (in C or C++). The NDK needs two different files, one on the Kotlin side (called PadPlayer) and on the C++ side (called native-lib by default) . The C++ file contains functions that can be called from Kotlin. Following you can see an example of how these functions are defined and called.

- In PadPlayer:
    - Definition:

```
external fun triggerDownNative(drumIndex: Int)
```

    - Call:

```
triggerDownNative(padIndex)
```

- In native-lib:

```
extern "C"
JNIEXPORT void JNICALL
Java_com_example_superlooperk_PadPlayer_triggerDownNative(JNIEnv       *env,
jobject thiz, jint drumIndex) {
    ...
}
```

Oboe works using the concept of audio streams. A stream can be one of two types: input and output stream. As the names imply, an input stream would allow for external sounds to be inputted, aka recorded, into the phone while an output stream would allow the phone to output sound. Before using a stream, it has to be initialized with the settings that work best for the situation. In my case, I want the lowest possible latency and that is achieved by making an audio stream exclusive and setting it to low latency. The exclusive setting will make it so that only the Super Looper app has access to the output stream. These settings are set in the following way:

```
builder.setPerformanceMode(PerformanceMode::LowLatency);
builder.setSharingMode(SharingMode::Exclusive);
```

12.1.2 Audio Files

Playing an audio file is more complicated than playing programmed audio. An audio file is actually audio information that has been encoded into a file using a specific format. Various different formats are available and they're all a bit different. Regardless of how the audio is originally encoded, it has to be fed to the corresponding oboe audio stream in PCM format. Pulse-code modulation format is the uncompressed digital representation of analog audio. DACs (digital to analog converter), tend to require data to be fed to them in PCM format.

Converting audio files from whatever format they’re in can be a complicated process. For my project, I considered two different options for audio decoding. FFMPEG, a popular audio and video compression and decompression library, or the audio utility library included in the Google Oboe library. Originally I wanted to use FFMPEG since it’s been around for a long time and used in different applications and is considered able and reliable. After trying for a couple of

days to incorporate it into my project I found it was a lot more complicated that it was supposed to be, mainly due to issues with getting FFMPEG to compile on Windows.

Realizing that it would not be worth the effort, I used the already available conversion classes that come with the Oboe utilities. This had a much more straightforward implementation but it came with it's disadvantages. The utility that comes with oboe was only able to decode mono wav files. This presents the user with the restriction of only being able to use mono wav files in the app. The files that the PCM data is created from have to be mono but the actual playback can be stereo (more towards the right or left).

The actual audio files are located in two different places. The files in the default packs are part of the app itself and are all found in the assets folder.These assets are managed by a class called AssetManager which facilitates the interaction between the files in the assets folder and the app. The files that are not part of the default packs and added when an external pack is created stay where they are on the phone. Their locations are saved by the app and then accessed externally using a class called ContentProvider.

12.1.3 Audio looping

One of the features of Super Looper is the ability to make certain pads loop. The implementation of this section is fairly straight forward. When the file playing for each pad reaches its end, it starts over from the beginning. For this section I had to go into the playback classes provided by Oboe and add the code needed to accomplish this.

```
if (mCurFrameIndex >= numSampleFrames) {
   if (mIsLooping) mCurFrameIndex = 0;
   else mIsPlaying = false;
}
```

This code snippet is located in the function responsible for rendering the audio. mCurFrameIndex is the index which indicates which audio frame is being rendered and numSampleFrames indicates how many frames the audio stream has. An audio frame is the smallest possible unit of a sound (a sound stream is composed of an array of audio frames).

Originally, when mCurrFrameIndex would surpass numSampleFrames, mIsPlaying would be set to false to indicate that it's not to play that audio stream any more. The way I modified the function, when the external if is entered, it's checked if that audio stream is looping. If it is, it resets the frame iterator to the beginning of the audio stream so that the file will start playing from the beginning. If it's not looping, then it stops playing. It took me a long time of going through all the different oboe classes to understand what was what and how to change things to achieve this looping functionality.

One of the possible issues that can arise with this implementation is that it doesn't stay in sync with the metronome bpm. In order to stay in perfect sync the file would have to be EXACTLY the correct length when set to its BPM. If it's a bit longer or shorter, every time the file finishes playing and starts over it will be more off tempo. This effect would be noticed the more it loops since each "iteration" of the loop will make the pad tempo be off by a bit more. Further on in the project I found a way to solve this issue. It's documented under the 13.4 Sequencer section.

12.1.4 Audio mixing

When more than one audio file is playing at once, it requires that it's output is mixed with the other playing files. For this, I used the SimpleMultiPlayer.h file included in the sample utilities. I used it as opposed to making my own because I had never programmed audio processing before this and it provided me with everything that I needed. I also wanted to use this existing utility because it allowed me to individually set the gain (volume) and pan (left/right audio balance), giving more control to the user.

The basic concept of audio mixing is the following: there's an array of audio streams, each stream corresponding to one pad (looping or one-shot). Each stream has its own corresponding volume and pan level. Everytime the DAC is ready for more audio information, it calls a function (called onAudioReady). Inside onAudioReady the different streams are all added together, keeping in mind the volume and pan levels, and then sent to the DAC so that it can convert the audio data into sound.

12.2 Packs implementation

Since the inception of this app I wanted Super Looper to adapt as much as possible to the user's needs. This includes allowing the user to be able to use his own sounds. There's a lot of similar apps to Super Looper but not many of them allow for the user to use his own sound files. This feature would allow a user to be able to export his song from a DAW on their computer and be able to perform it using Super Looper.

Originally I wasn't sure how to implement this feature; I thought that the user could maybe just change the audio files assigned to each pad for the duration of the session. I then realized that the user would lose his file choices if they weren't saved to the device somehow. Basing myself off of other apps, I found that most of them employed a type of "pack" concept that let the user choose the sounds assigned to the pads.

I found that this system made a lot of sense but it's missing the customization aspect that I wanted my app to include, since the packs tended to be precompiled. I realized then that the solution was simple: just allow the user to create their own packs.

When the user wants to create their own packs, they tap on the "Create new pack" button on the settings fragment. This will bring them to a screen with 8 pads, a metronome slider and a text box. They can tap on each of the pads to select the sound file assigned to it, set the slider to the desired tempo and choose the pack name by writing it in the text box. After they create the pack the user will be able to simply choose it from the pack dropdown menu.

The pack lifecycle is relatively complex and consists of different parts; first, I will cover the file picking part of pack creation. When the user clicks on a pad to assign the file to, an android file picking event is created and then executed. The result of the chosen file is received through a callback where the file URI is stored in a local array. This continues for each pad. After the user is satisfied with their chosen audio files and has filled out the loop lengths, pack name and tempo, the app will send this new pack information to the PadPlayer class where it will be saved to internal storage as text and added as a pack option.

When the app starts up, it looks for the text file where the user created packs information is stored. It reads the corresponding information and then loads it into the app. According to the

Android documentation[49] there's four ways for an app to store data to the phone past it's lifecycle:

- App-specific storage: Stores files only intended for your app.
- Shared storage: Stores files that can also be accessed by other apps.
- Preferences: Stores private data in key-value pairs.
- Databases: Stores structured data into a database.

I decided to go with App-specific storage since it is the most adequate for my situation. It allows me to save my data in whatever format I choose and it only needs to be accessible to my app. Saving to preferences was limiting because of the key-value pairs, which would make saving data in array form more complicated than necessary and saving to a database would be overly complex and complicated since it would require SQL.

12.3 BPM implementation

Originally I struggled quite a bit with how to implement the concept of BPM in my app. BPM, or beats per minute, are used to identify the tempo that a song is at. First I have to find a way to convert BPM to something the device will understand. After doing some research[50], I found that the best way to implement BPM is by creating a new thread that sleeps between each "beat" and then sends an metronome event to its parent class.

My app assumes that all files used will be in 4/4 tempo since it's the most common in western music. That means 1 in every 4 beats is the downbeat, which is when I want the metronome sound to be triggered. That's why there's a boolean sent as an argument to the parent, indicating if the current tick is a downbeat.

Originally the metronome (the device/class that keeps tempo) would only "tick" every beat, which was enough for the app to indicate downbeats. Following is the original implementation:

```
Thread.sleep((1000*(60.0/bpm)).toLong())
```

When I implemented the sequencer, I realized that if the metronome only ticked on the downbeats I would only be able to trigger sounds on those downbeats. Since I consider that all the uploaded files will be in 4/4, that would mean we have 1/4 note resolution. This doesn't allow for very intricate sequences though so I extended the metronome to tick on each 1/16 note, therefore being able to trigger different sounds on each 1/16 note. Following you can see the implementation.

```
Thread.sleep((1000*(60.0/(bpm*intervalPrecision))).toLong())
```

I decided to make the intervalPrecision a variable so that it could be changed easily in the future if need be.

When I got to the MIDI section of my project I realized that I needed to extend the metronome some more. If I want to be able to send the play command to external gear the app has to be able to send out clock information to the receiving device 24 times per quarter note. Because of this, the metronome thread will have to tick 24 times per quarter note and send a message to the MIDI device. I implemented this in the following way:

```
Thread.sleep((1000 * (60.0 / (PadPlayer.bpm*24 ))).toLong())
```

12.4 Sequencer implementation

Originally I struggled with implementing the sequencer because I couldn't find much information on how they tend to be implemented and configured. Java has its own sequencer class[51] but it didn't quite fit my needs because it's tied to a MIDI device and uses MIDI files, which is not what I was searching for.

In the end I decided to implement my sequencer as a matrix. In the sequencer matrix, each column represents a 1/16 note and each row represents a different pad. The matrix is composed of booleans, true indicating that the corresponding pad has to be triggered at that tick and false indicating that it doesn't have to be triggered. Below is a code snippet of how the sequencer is implemented.

```
    fun seqBeat(isDownbeat: Boolean, isUpbeat: Boolean) {
        if (isDownbeat) downBeatMetronome()
        if (isUpbeat) upBeatMetronome()
        if (playingOn) {
            for (i in seq.indices) {
                if (seq[i][currentBeat]) triggerDownNative(i + 4)
            }
        }
        ++currentBeat
        if (currentBeat == 16) currentBeat = 0
    }
```

This function is called every sixteenth beat and it receives two parameters which give extra information about the current beat. First, If it's a downbeat, as indicated by isDownbeat, the function downBeatMetronome() is called. In this function, the metronome downbeat sound is triggered (along with some other things). If it's an upbeat, as indicated by isUpbeat, the upBeatMetronome() function is called which triggers the metronome upbeat sound. After either of the "beat" functions are called, an iterator goes through the sequencer matrix at the current beat. If a true is found, the corresponding pad will be triggered on that beat.

```
   private fun downBeatMetronome() {
       //tick
       if (playingOn) {
           if (metronomeOn) triggerDownNative(METPADDOWN)
           while (!playLoopQueue.isEmpty()) {
               triggerDownNative(playLoopQueue.poll())
           }
           for(playingPadIndex in playingLoops) {
               if (loopingPadsBarsPlayed[playingPadIndex] ==
loopingPadsBarLength[packChoice][playingPadIndex]) {
                   triggerDownNative(playingPadIndex)
                   loopingPadsBarsPlayed[playingPadIndex]=0
               }
               loopingPadsBarsPlayed[playingPadIndex]++
           }
       }
   }
```

Above is the downBeatMetronome() function. Here is how the issue commented in the audio looping section (13.1.3) has been solved. First, it's checked that audio is playing; then, if the metronome is on. If it is, the metronome sound is triggered. After that, the queue of playing

loops is polled and all the loops that it contains are triggered. Then, for each pad in playingLoops, it's checked if it has played for its length and if so, the sound is triggered and the loops played set back to 0. A counter then increases the amount of times the loop has played.

In order to make the app as usable as possible I decided to allow the user to record a sequence in two different ways. If the record option is on, the pads the user triggers will be recorded to the sequencer. If the record option isn't on, the user can play any of the pads without recording it to the sequence. From the sequencer screen, the user can either create his sequences visually or modify the sequence(s) that they have previously recorded.

12.5 MIDI Implementation

The MIDI protocol[52] has been around since the 80s and has barely changed since then; MIDI 1.0 is still being used now! I thought it would be a nice and uncommon feature to have my app support external devices directly through the device's USB port, especially since I personally have a couple of electronic instruments that accept MIDI messages.

I implemented MIDI support using MIDI libraries for Android provided by Google[53]. I was hoping the integration process would be fairly straight forward but various complications arose. Most of the articles I found about MIDI implementation were in Java and when I tried to follow them in Kotlin it wouldn't work.

The first step in MIDI communication is obviously connecting to the device. In order to be able to connect to an external MIDI device, the phone running Super Looper has to support MIDI. This can be checked using the package manager in the following way: If the phone supports the MIDI protocol, it will continue connecting. To do so, a list containing all the connected MIDI devices (usually just one) is requested from the MIDI manager. If the list is not empty, the MIDI device manager will open the device and one of it's input ports. It doesn't really matter which MIDI channel is used because the MIDI real time messages are channel agnostic. Once the input port is opened it's saved as a private variable so it can be used in the future.

Upon plugging in a device it would be nice to notify the user that the device has been connected successfully. The best way to do this would be using a toast message, which is a message in a

small black text box that appears for a set length of time on the screen. I tried to implement it but no matter which way it was done, the message wouldn't display. I'm pretty sure it's because the MIDI connection is asynchronous, meaning the program keeps running after plugging in the device but before it's connected. Somehow, this didn't allow me to create any toasts.

Once two devices are connected through MIDI, there's a large range of different commands that can be sent[54]. The MIDI Constants file has a list of the byte codification for each command. I decided to add this file to my project in case any other MIDI messages were to be sent from Super Looper in the future. The ones that are necessary for Super Looper are start, stop and a timing message. The start and stop commands are simple byte messages that are accompanied by a timestamp. As explained in the BPM section, the timing message, or ticks, have to be sent 24 times per quarter note. Below you can see the code for the MIDI stop message and for the timing message.

```
 private fun stopMidi() {
        var stopBuffer = ByteArray(1)
        stopBuffer[0]=MidiConstants.STATUS_STOP
        val count = 1
        val now = System.nanoTime()
        mInputPort?.send(stopBuffer, 0, count, now)
    }
```

```
var clockBuffer = ByteArray(1)
clockBuffer[0]=MidiConstants.STATUS_TIMING_CLOCK
val count = 1
while (metOn) {
   if (PadPlayer.mInputPort!=null) {
       val now = System.nanoTime()
       PadPlayer.mInputPort?.send(clockBuffer, 0, count, now)
   }
```

I carried out the MIDI implementation using my Novation Circuit[55] as the external MIDI device connected to my phone by USB with a USB-A to USB-C adapter. To test that my phone was able to communicate with the Circuit, I used Android's MIDI Keyboard app, a simple app that allows you to connect your phone and use it with any external MIDI device to play notes on.

12.6 Front-end implementation

The visual appearance of an app is really important since it contributes to the user experience. If the app doesn't give the user a straight forward or enjoyable experience the user is not going to like it and that's obviously not what any of the stakeholders want. The appearance of an app is supposed to make the app easier to be used and understood. In this section I will explain how I achieved this.

12.6.1 Pad Screen

This is the main app screen. It shows the 8 different pads, 4 of them looping and 4 one-shot.They are grouped into two so it's easier to tell between the two types of pads. The looping pads have a small icon to show playing state: stopped or playing. At the bottom the start and stop controls are found, along with the metronome and sequencer record buttons. These elements are located on this screen because they are strongly tied to the pads and pad state. The metronome, record, stop and play buttons all change color depending on their state. An image of this screen is found below.

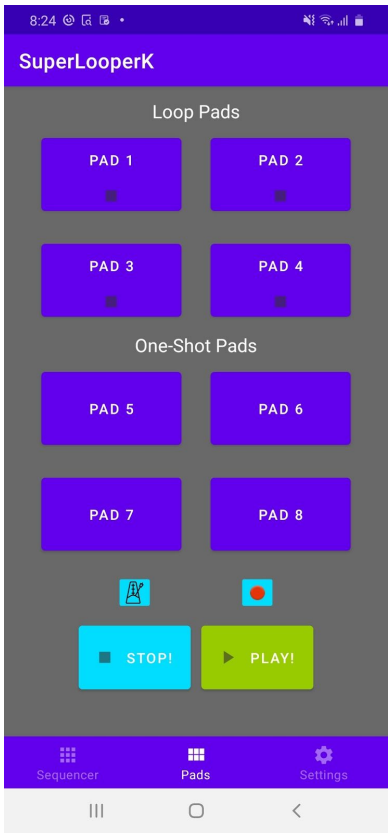

Figure 6: Pad Screen. Of personal creation.

12.6.2 Pad Settings Screen

From the pad settings screen all of the volume and pan levels can be adjusted. For these two functions, sliders are used. Towards the bottom of the screen, the user can see the bpm and adjust it using the slider. Underneath that, a dropdown list is used to select the desired pack.

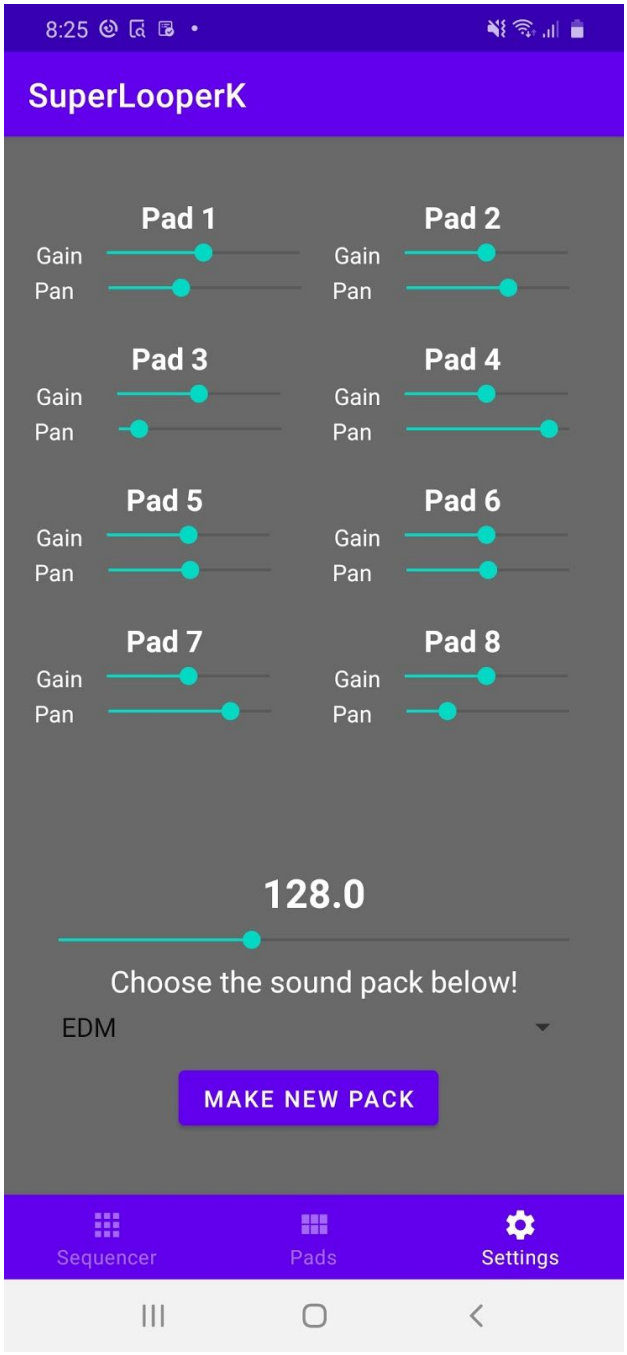

Figure 7: Pad settings screen. Of personal creation.

12.6.3 Pack Creation Screen

In this screen the user is able to pick his desired files for the new pack and choose the pack tempo and name. Buttons are used for file picking and input boxes for the bpm and the name. The appearance is similar to the pack settings screen to keep the look and feel consistent.

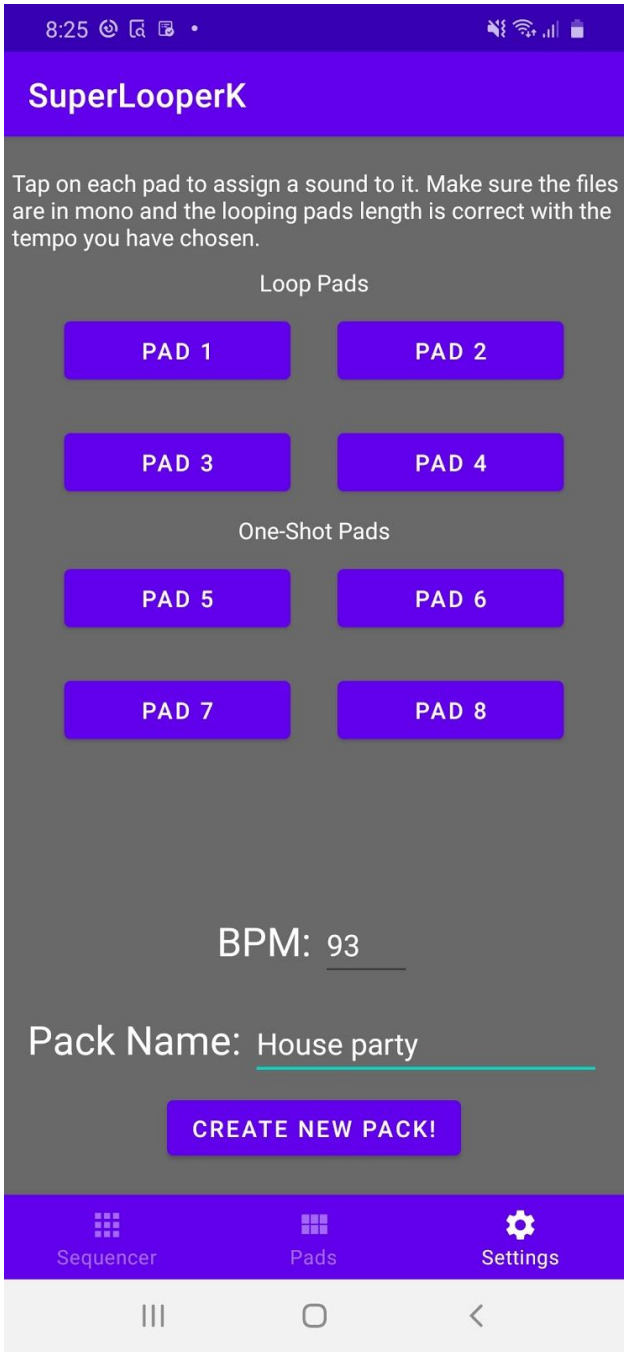

Figure 8: Pack creation screen. Of personal creation.

12.6.4 Sequencer Screen

This screen is used to manually select (or deselect) where the one-shot pads will be triggered. The toggle button was used because of its ability to discreetly let the user know about the pad state. The clear all button is at the bottom of the screen for easy access.

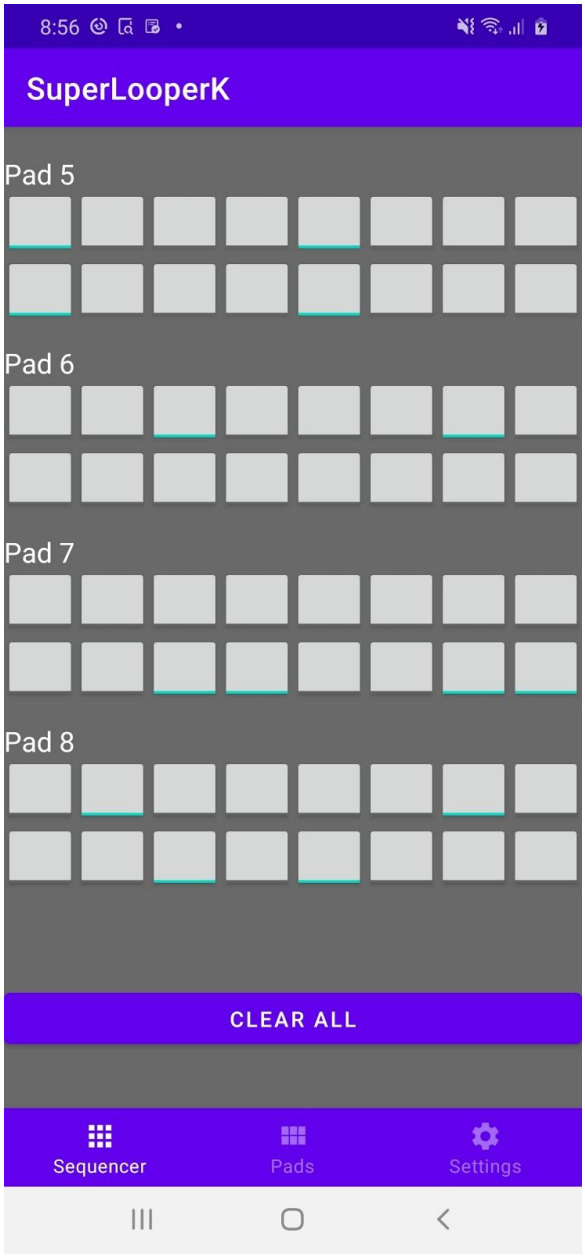

Figure 9: Sequencer screen. Of personal creation.

13. User Feedback

During the development of this project I, Adam Scher, have been the client/user at the same time as the programmer. Discussing this with my project director, we decided that it would be helpful to have external opinions and feedback on different aspects of my app. User feedback[56] is important because it provides the developer with information about the project that they might have not realized on their own.

In order to carry out user feedback I decided to do online video meetings (due to the pandemic). I asked the users to use the app for about 5 minutes, during which they were able to ask me questions and observations. After that, I conducted a short survey[57] with 8 people: 4 musicians (Manu, Josep, Agata and Heidi) and 4 non-musicians (Sonka, Mike, Aida and Linzann).

From a scale from 1 to 10:

1. How intuitive was it to use Super Looper?
2. How reliable did Super Looper feel?
3. How was the audio quality?
4. How much did you enjoy using Super Looper?
5. How likely are you to recommend SuperLooper to a friend?
6. What would you review Super Looper (out of 10)?
7. How complete did you consider Super Looper?

	Q1	Q2	Q3	Q4	Q5	Q6	Q7
Manuel	7	9	8	8	7	8	8
Josep	6	8	7	8	8	7	9
Agata	8	9	7	8	7	8	8
Heidi	6	8	7	8	7	7	8
Sonka	8	9	7	7	6	7	9
Mike	7	10	10	10	10	10	8
Aida	7	8	8	8	9	7	9

Linzann	6	8	7	9	9	8	8

Looking at the feedback given some conclusions can be extracted. The users were happy with the experience in general, seeing that it got high points in the enjoyment question (Q4) and also in Q5 and Q6. The users were also content with the feature set, as indicated by Q7. None of the users could take advantage of the MIDI capabilities since they don't have any MIDI gear. The first three questions, which are related to the requirements, also get high points from most of the users. One of the things that could be improved is the appearance of the app, since most users made a comment about that. No other large observations were provided by the users. Most of them explained that they would know how to use Super Looper better if theft had more experience using similar apps.

14. Testing and Validation

14.1 Tests

Testing, in one way or another, is part of any software project; it involves making sure that the implemented functionality works properly and meets all the requirements. Visual testing[59] is a type of testing where the user/developer tests the functionality by carrying out the corresponding task. During the realization of said task the errors and observations are noted so that they can be fixed afterwards.

I chose this approach to testing because it's direct and simple but also provides almost all of the information necessary. Many other testing methods function by comparing the actual output and comparing it to the desired output. This would be hard to do with Super Looper since the "output" is in the form of sensory data, like hearing the music and seeing the pads on the screen instead of a defined numerical output which can be more easily defined and compared.

14.2 Validation

It's important to make sure, during and at the end of development of a project, that the project meets the set requirements and objectives. These requirements help describe the needs of the user that are to be solved by the project. In the case of Super Looper, user stories (section 12.1) were used to be able to define and validate the requirements (section 3.2).

There's two different types of requirements: non-functional and functional. In order to validate that all the functional requirements have been met, I conducted a manual validation where I manually checked that all the user stories were successful.

Functional Requirement Validation

For each user story, different tests will be done to confirm that the story is successful (or uncesseful). In order to document them, each story will include the:

- Description: Describes the user story and situation.
- Result: The result of the validation.
- Observations: Observations about the current test.

User Story 1
Title: Basic loop play
Description: As a user I want to be able to start and to stop a loop by tapping on a looping pad.
Result: Successful
Observations: If the pad was playing, then it stops. If it was stopped, then it starts playing and if the app wasn't in play mode it would switch into play mode.

User Story 2
Title: Basic one-shot play
Description: As a user I want to be able to trigger a sound by tapping on a one-shot pad.
Result: Successful
Observations: When the pad is tapped, it plays the sound assigned to it. If record mode is on, the step will be recorded into the sequencer.

User Story 3
Title: Audio mixing

Description: As a user I want to be able to hear all the different pads at once.
Result: Successful
Observations: Whenever more than one pad is playing, the pads audio is played correctly while respecting the corresponding pan and volume levels.

User Story 4
Title: Volume Level
Description: As a user I want to be able to individually adjust the volume level of each pad.
Result: Successful
Observations: When the volume level of a pad is adjusted the volume level audibly changes for that pad.

User Story 5
Title: Pan Level
Description: As a user I want to be able to individually change the right/left balance of any pad.
Result: Successful
Observations: After adjusting the pan level of a pad that pad is successfully heard more to the right or to the left with the assigned pan levels.

User Story 6
Title: Play/stop
Description: As a user I want to be able to play and stop all of the pads at once.
Result: Successful
Observations: When stop is pressed, all the pads stop playing and no more audio is heard form the app. When play is pressed, the pads that were playing before are all started and if there's any content is the sequencer it's played.

User Story 7
Title: Sequence pads
Description: As a user I want to be able to program a one-shot sequence so that I don't have to physically play a pattern.
Result: Successful
Observations: After the sequence has been programmed it's correctly heard as audio output.

User Story 8

Title: Record pad sequence

Description: As a user I want to be able to play a one-shot pad sequence and have it recorded to the sequencer.

Result: Successful

Observations: Whatever the user played on the one-shot pads is successfully added to the sequencer in the correct order.

User Story 9

Title: Clear sequences

Description: As a user I want to be able to clear all of the existing sequences at the click of a button.

Result: Successful

Observations: After the user has clicked the clear all button in the sequencer page the sequences are all erased for all the pads.

User Story 10

Title: Change pack

Description: As a user I want to be able to change to a different sound pack.

Result: Successful

Observations: When the user changes sound pack, the playback stops, the metronome stops, and all of the pads are assigned to their new pack sounds.

User Story 11

Title: Create pack

Description: As a user I want to be able to create my own pack using sounds found on my device.

Result: Successful

Observations: The new pack is saved with its corresponding name and bpm and is now selectable in the dropdown pack list in the settings page.

User Story 12

Title: Midi connection

Description: As a user I want to be able to connect external MIDI gear to my phone and have it be recognized by Super Looper

Result: Successful

Observations:Regardless if the device is plugged in before or after Super Looper is started, it will be recognized and connected to correctly.

User Story 13

Title: Midi messages

Description: As a user I want my connected MIDI device to receive a play message when play is tapped and a stop message when stop is tapped.

Result: Successful

Observations: The MIDI device responds to the start, stop and timing messages.

User Story 14

Title: Metronome toggle

Description: As a user I want to be able to turn on and off the metronome sound.

Result: Successful

Observations: The metronome sound is heard if the toggle is on and not heard when the toggle is off.

User Story 15

Title: Adjust metronome tempo

Description: As a user, I want to be able to adjust the metronome tempo.

Result: Successful

Observations: The metronome sound reflects the set tempo.

User Story 16

Title: Screen Navigation

Description: As a user I want to be able to easily navigate between the screens.

Acceptance Criteria:

Result: Successful

Observations: The sure is successfully led to the selected screen when clicking on the corresponding navigation icons.

Non-functional Requirement Validation

In order to validate the non-functional requirements a combination of manual validation and user feedback were used. The first three questions in the user feedback section are directly related to the reliability, usability and audio quality. The rest of the non-functional requirements were validated using manual validation.

- Portability: Success. The app runs on a smartphone so it's portable.
- Reliability: Success. The app doesn't crash and can be relied on. Supported by user feedback.
- Scalability: Success. The app has been programmed allowing for scaling possibilities in the future.
- Performance: Success. The software runs smoothly without any hiccups or errors. The audio stream isn't interrupted at any moment.
- Usability: Success. The interface is intuitive and usable as confirmed by the user feedback.
- Audio quality: Success. The audio files remain in their original quality when played back and user feedback supports this.

15. Future Improvements

As content as I am with my app, there's always room for improvement. In this section I will touch on different features that could be added or improved on.

- Audio file formats: Right now Super Looper can only read mono .wav files. This limitation is caused by using Oboe's file translation. It would provide the user with much more flexibility if Super Looper accepted stereo files in different formats (MP3, FLAC, etc.).

- Audio effects: This feature was part of my original desired feature list. Having an extra way to transform one's sound would allow them to create even more things in different

ways. There's at least one app, FXLab[59], which uses oboe to create different audio effects which would be a great place to start if this feature were to be implemented.

- Global record: Another feature that I was originally hoping to make available in Super Looper is a global session record. This would allow the user to record whatever they make in the app so that they could export it in the future. This would be a great feature to have since it would allow the user to bring their creations beyond Super Looper and use them in whatever way they want.

- MIDI implementation: One of the existing features that I'm really proud of with Super Looper is MIDI (because of its wide use and multiple applications). The way Super Looper uses MIDI is just one of the possible things it's capable of doing. In the future, MIDI functionalities could be expanded to include:

    - MIDI Slave mode: Currently, the way MIDI is implemented, the device running Super Looper will act as a host for the external MIDI device. This means that Super Looper can only send information. It could be useful to also make Super Looper receive information, like to set it's tempo externally, play the pads externally, etc. It would also play a part in implementing the other following MIDI functionalities.

    - MIDI CC: MIDI control changes are messages that can be sent/received between different MIDI devices. CC messages can be used to represent turning a knob, sliding a slider, checking/unchecking a button, etc. CC messages could be used to control Super Looper from an external device.

    - MIDI Sequencing: Since my app already implements a sequencer, it could be used to also sequence external devices (that might not have a built-in sequencer).

- Pro accounts: Right now the app has no realistic way to generate revenue. There's two main ways that an app can generate money: ads and pro accounts. If I wanted to monetize Super Looper I would introduce pro accounts which would enable session

records if purchased. Midi connectivity could also be a possible pro feature. If ads were also added to Super Looper, purchasing the pro version would disable them.

- Pack extension: The concept of packs works well in the app: allowing the user to choose which audio file he wants to use on each pad. That works well but the user has to put the (mono) audio files on the device and then choose them from inside the app. The addition of an online pack marketplace or something similar would extend the pack functionality by a lot more. It could also allow the developer to release new packs without requiring an app update.

- Sample editor: A sample editor would be a screen that can be accessed which shows the waveshape of the sound file. It would allow the user to choose the start and end point of the sample and could include extra parameters such as EQ, filter, etc. This would give the user more control of each sample.

- Time signature: Right now the metronome and sequencer are both restricted to music in 4/4, the typical time measure used in modern western music. The metronome functionality could be expanded to allow other time signatures, such as 3/4 or 7/8.

- Sequence lengths: The sequences are found in the sequencer screen. Currently, each one-shot pad has 16 different available places where you can place a note or trigger. This is because the sequence length of the non-looping pads is restricted to 1 bar (since there's 16 sixteenth notes in one bar in 4/4). Because of this, the same sequence is repeated every bar and the repetition can become a bit too much for the listener. A way to add variety would be to make the sequences longer (maybe 2 or 4 bars instead, 32 and 64 triggers, respectively).

16. Deviations

Throughout the development of my project, there have been multiple unforeseen obstacles. No matter how much previous research is done or how much matters are considered, things may not always turn out as one wants them to. In this section I will talk about the deviations in planning and costs.

16.1 Planning

Originally I knew that I wanted a quite versatile app which the user would be able to use in different situations and for different reasons. I needed to find a way to make both the app versatile and usable for people with different needs, requirements, and experience levels. That's how I decided to implement “packs” and their customization into my app. Initially, the sound files were hardcoded and the user wasn’t able to change them. I realized that this made Super Looper very limited in multiple ways so I decided for the concept of packs. In order to have enough time to be able to implement packs I didn’t get to implement audio effects. The budget and time originally dedicated to audio effects was dedicated to packs.

16.2 Costs

In general, since most of the tasks were completed within their time constraints or managed to fit the set constraints, no deviations to cost occurred.

17. Sustainability Report

Nowadays almost all modern companies produce a sustainability report to go along with most of their projects. The aim of a sustainability report is to provide the reader with the impact on climate change, human rights, transparency of life, etc. that the product/project might have. Every invention or addition to the world tends to have unforeseen side effects, these being able to affect society in different ways, mainly: socially, economically and environmentally.

One of the main reasons why I have chosen to develop this app for Android (given the existing issues and lack of support with audio programming) is to make the app reachable to people that can’t, or decide not to, buy an iPhone. Android is open source meaning a lot of phones run Android and they can be a lot cheaper than Apple products. I want to build this app using open source software and introduce it to a market where people don’t have to pay more than is necessary to make and interact with music.

17.1 Environmental dimension

I have estimated that the environmental impact of undertaking the project will be as small as possible given my current resources. I personally recycle as much as possible and try to reduce my general environmental impact by using public transport or a bicycle when possible. I'm aware that the electricity that I use at home is generally non recyclable but I tend to be conscious of my electrical spending and don't do things like leaving the lights on when I leave a room. I'm also pescatarian, therefore doing my part in what I believe to be an environmentally unfriendly way of growing and consuming meat (and fish, but I'm working on that one step at a time). Specifically, towards this project, I am reusing an old monitor and an used phone from one of my family members. I will also do the meetings with my director online given the Covid situation therefore saving myself the environmental impact of travel.

Currently there are not many live looping solutions out for Android, the standard operating system on cheap phones (the alternative being iOS which runs exclusively on iPhones costing more than the average Android device). Hopefully this can prevent a user that needs to use a mobile looping app but doesn't own an iPhone to not have to buy one just for the operating system for which many audio apps have already been designed. A mobile app will also prevent the musician from having to buy a purpose-built hardware alternative like a Boss RC-505, costing around 450€.

The environmental impact of the app after it's release will be close to 0, since almost everyone already has a smartphone. I've also decided to create this app for android because it's an open source project and because of that, it is included as the operating system for most of the phones on the market. If something, it could have a positive environmental impact, since some users could use their phone instead of buying a dedicated device. There's very little risk of anything occurring that could make the environmental impact worse. The minimum amount of resources was used during the development of the project.

17.2 Economic Dimension

My will to make this project was born out of necessity, of me thinking about what I could add to my music setup at a minimum cost and maximum functionality. I feel like my project has the lowest cost possible especially since I'm reusing old devices and not consuming any new resources.

Since my project doesn't aim to have a large economic impact I don't think it has much importance in the economical dimension. It's a tool to support musicians with playing and creating music, the most economical benefit would be the money made from the songs in the app.

The costs of undertaking the project were only the necessary ones. The cost of the project during its useful life would only be the potential costs of the developers if the project was to be continued. The addition of pro accounts would hopefully make the project costs worth it. Situations that are detrimental to the project's viability are not expected, unless the user were to use it in live performances and they were cancelled (for example due to a pandemic).

17.3 Social Dimension

Personally I feel that this app is a large stepping stone in my career. During most of the degree they teach us about multiple aspects of computer engineering but we never really got into sound or audio processing. For me, this is an opportunity of bringing my two worlds together, music and computers. I have proposed to myself to, in the future, find a job at an electronic music company. I think this project is a great way for me to learn some knowledge about audio, dsp and MIDI that will hopefully be of use for in the future, be it to get a job or just for personal use/enjoyment.

With this project I hope to introduce a cheap and easy solution to an audio looper. In today's markets most audio apps are only available for Apple devices (which are more expensive) so I think that my app will help expose people that have Androids to a new aspect of music and interaction with it that they may not have experienced before. Hopefully this project will provide the users with enjoyment, knowledge and creativity. This could be translated into the general population enjoying the music created by the app user or enjoying themselves more at a live

show thanks to the artist using my app. I feel like there is a need for an app like this on Android since the existing ones aren't very good and many people will be able to benefit from my app.

Undertaking the project has led me to appreciate music and the process of creating it even more. Hopefully everyone could benefit from using Super Looper but the people who are most likely to use it are probably existing musicians that are looking to expand their ways to create music or connect different MIDI devices. The project successfully solves the problem that was proposed initially and it's not expected to put anyone in a weak or adverse position.

18. Technical Competencies Justification

CES1.2: Donar solució a problemes d'integració en funció de les estratègies, dels estàndards i de les tecnologies disponibles. [Bastant]
Most of this project consisted in combining different technologies, strategies and standards. There is little information available on in depth audio programming and I had also never worked with Kotlin or Oboe. Lots of obstacles were encountered while trying to get different parts of the program to work together and I feel like I have figured out the best solution in most of the cases. Because of this, I can say that the indicated level of competence has been achieved.

CES1.3: Identificar, avaluar i gestionar els riscos potencials associats a la construcció de software que es poguessin presentar. [Una mica]
At the beginning of the project, Super Looper was just an idea. In order to find out if it was possible to develop the project, some time was spent researching and evaluating different ways and methods to achieve its completion, apart from the existing solutions on the market.. Because of this, I can say that the indicated level of competence has been achieved.

CES1.8: Desenvolupar, mantenir i avaluar sistemes de control i de temps real. [Una mica]
Super Looper has to be able to keep the tempo of the music, the rhythm and also has to be able to sync with external gear and communicate with it as fast and responsively as possible. The

app is also reactive and responses to user input quickly. Because of this, I can say that the indicated level of competence has been achieved.

CES2.1: Definir i gestionar els requisits d'un sistema software. [Bastant]
As with any project, it's important to set and meet requirements; but even more with Super Looper, due to its specific area of audio programming. The requirements were defined initially and then validated using user stories and acceptance criteria. Because of this, I can say that the indicated level of competence has been achieved.

CES3.1: Desenvolupar serveis i aplicacions multimèdia. [En profunditat]
Given that the main focus of my app is multimedia (specifically, audio), I think this competence has been fully achieved. Super Looper is able to transmit audio at original quality and low latency. It uses icons and different shapes to transmit the message in the clearest way possible.

19. Conclusions

In conclusion, I'm satisfied with the outcome of my project. All the objectives have been attained and the final product is functional. When I started, I had never worked with Android Studio, audio or MIDI in the past. Now I consider myself slightly more knowledgeable and experienced in these areas. I feel like I have successfully completed my objectives of creating a low-latency audio android app with MIDI support that can be used in a number of different musical/sound applications.

A secondary objective of this project was learning how the employed technologies work and how they're used and programmed. For me this was the most important part of this project: learning. As a hobby musician and software student I wanted to find a way to bring these two worlds together and I couldn't be happier with how it's come out. I feel confident that I will be able to apply this knowledge in the future, either as a hobby or as a part of my career.

Admittedly, I don't think my app is perfect. Apart from the possible future improvements (previously stated) that could be carried out to make the app better, I think that the app tries to serve too many purposes. At the beginning, I wasn't 100% sure of exactly what and how I

wanted my app to do. I wrote down my idea and objectives and hoped that I would figure out how to solve different problems along the way. Looking at my app now, I wish I would've been more specific at the beginning. In my opinion, Super Looper tries to be too many things: it has a metronome, sequencer, audio file playback, MIDI, etc. This isn't a case of these features not being implemented properly, but of these features not working the best way together. Regardless, the app works well and the experience of developing it has been very enriching.

20. Glossary

Music note: A “note” in the music world, is a visual representation for the length and the pitch of the sound to be produced (regardless of the instrument). The pitch is usually described by letters (C, D, E, F…) or names (Do, Re, Mi, Fa….) and their corresponding location on sheet music. This concept is important to understand other concepts explained further on in this glossary. Figure 10 shows how note lengths are expressed.

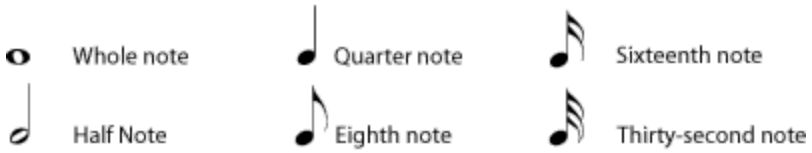

Figure 10: Source: libretexts.org[60]

Written on actual sheet music, these notes would look the following way:

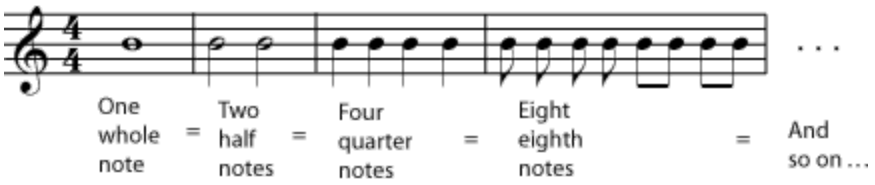

Figure 11: Source: libretexts.org[60]

Measure or bar: The section between the two vertical lines (| ….. |) in Figure 11.

Time Signature: The two numbers found at the beginning of Figure 11 (in which it’s 4/4). It indicates how the beats are distributed. The bottom number is the “beat unit”, the note length used to “count” all the beats. The top number is how many of the “beat units” are in one bar (or measure). For example: in 4/4, there’s four quarter notes (¼ notes) for each measure, in 3/4 there would be three quarter notes in one measure and so on.

BPM: Beats per minute. This term is used to indicate the tempo of music.

Downbeat: The first beat in a bar.

Upbeat: The rest of beats in a bar.

Sequence: A set of notes/silences that have been set in a certain order.

Sequence length: How long a sequence is, usually expressed in bars or measures.

Sequencer: A device capable of recording and playing back sequences. Figure 12 shows a professional sequencer called Cirklon.

Figure 12: Cirklon sequencer[61]

MIDI: A technical standard that describes a communications protocol, digital interface, and electrical connectors that connect a wide variety of electronic musical instruments, computers, and related audio devices for playing, editing and recording music.

Audio effects: A type of audio processing that takes the original sound and modifies it. There's a lot of different audio effects available. Following is a list of the most common/popular ones:

- Reverb: Determines the “size” of the room where the sound is played. Used to give songs a bit more of a realistic feel and/or to push instruments to the back.
- Delay: Determines how many times the sound is played back after the first time. Parameters can be adjusted to determine the interval at which the sound is played back,

with how much volume the repetitions are played and other options. This is used to add a bit of variation and fullness to a sound.

- Chorus: Slightly detunes the original sound and then pans it around the stereo field. Useful to make sounds sound a bit thicker and spread out.
- Distortion: Works by forcing the wound waves past their maximum and then cutting the tops and bottoms off. Can make a sound sound cleaner or dirtier.
- EQ: Adds or removes gain on certain frequencies. This is a really important audio effect that is heavily used in the mixing stages of production. It helps to make sounds not clash and fit better.
- Compression: Compression and EQ could deserve essays of their own! Compression reduces the dynamic range. It takes quiet parts of a sound and makes them louder and louder parts and makes them more quiet. Userful to make a sound sound a bit tighter but has tons of different uses. Also mainly used in the mixing stage of production.
- Filters: Filters are used to cut out unwanted frequencies from a sound. Similar to EQ but used dynamically to make sounds more interesting or to just cut off frequencies that the user doesn't want to come through.

21. References

[1] Facultat d'Informàtica de Barcelona |. (n.d.). Retrieved September 28, 2020, from https://www.fib.upc.edu/ca

[2] Strauss, W. (2000). Digital signal processing. In IEEE Signal Processing Magazine (Vol. 17, Issue 2). https://doi.org/10.1109/79.826412

[3] Lamprecht, E. (2019, December 19). The Difference Between UX And UI Design - A Layman's Guide (Updated for 2020). https://careerfoundry.com/en/blog/ux-design/the-difference-between-ux-and-ui-design-a-laymans-guide/

[4] Maxie, J. (n.d.). The Best Loop Pedal Apps in 2020 | Campfire Musician. Retrieved September 28, 2020, from https://campfiremusician.com/loop-pedal-apps/

[5] Toth, A. (n.d.). Loop Player - A B Audio Repeat Player - Apps on Google Play. Retrieved September 28, 2020, from https://play.google.com/store/apps/details?id=com.toth.loopplayer

[6] Zuidsoft. (n.d.). LoopStation - Looper - Apps on Google Play. Retrieved September 28, 2020, from https://play.google.com/store/apps/details?id=com.zuidsoft.loopstation2

[7] ufxmedia. (n.d.). uFXloops Music Studio - Apps on Google Play. Retrieved September 28, 2020, from https://play.google.com/store/apps/details?id=de.ufxmedia.ufxloops&hl=en

[8] Alsaleh, S., & Haron, H. (2016). The Most Important Functional and Non-Functional Requirements of Knowledge Sharing System at Public Academic Institutions: A Case Study. Lecture Notes on Software Engineering, 4(2), 157–161. https://doi.org/10.7763/lnse.2016.v4.242

[9] Radigan, D. (n.d.). Kanban - A brief introduction | Atlassian. Retrieved September 28, 2020, from https://www.atlassian.com/agile/kanban

[10] Siderova, S. (2019, October 25). The Kanban Way: How To Visualize Progress And Data In Trello. The Kanban Way: How To Visualize Progress And Data In Trello. https://blog.trello.com/kanban-data-nave

[11] Phillips, K. (n.d.). KANBAN BOARD EXAMPLE w/SCRUM | Trello. Retrieved September 28, 2020, from https://trello.com/b/TBUZ7pE5/kanban-board-example-w-scrum

[12] Martin, A. C. (2018). An introduction to Git and GitHub. 1–25.

[13] GitHub. (n.d.). Retrieved September 28, 2020, from https://github.com

[14]Treball de Fi de Grau | Facultat d'Informàtica de Barcelona. (n.d.). Retrieved October 5, 2020, from https://www.fib.upc.edu/ca/estudis/graus/grau-en-enginyeria-informatica/treball-de-fi-de-grau

[15]Google Chrome - Download the Fast, Secure Browser from Google. (n.d.). Retrieved October 5, 2020, from https://www.google.com/chrome/

[16]Google Docs. (n.d.). Retrieved October 5, 2020, from https://docs.google.com/document/u/0/

[17]Download Android Studio and SDK tools | Android Developers. (n.d.). Retrieved October 5, 2020, from https://developer.android.com/studio

[18]GitHub - google/oboe: Oboe is a C++ library that makes it easy to build high-performance audio apps on Android. (n.d.). Retrieved October 5, 2020, from https://github.com/google/oboe

[19]GanttProject - Free Project Management Application. (n.d.). Retrieved October 5, 2020, from https://www.ganttproject.biz/

[20]Siriwardhana, S. (n.d.). Why People Miss Deadlines and How to Avoid Missing Them - Creately Blog. Retrieved October 5, 2020, from https://creately.com/blog/culture/why-people-miss-deadlines/

[21] Technical Writer Salary in Spain | PayScale. (n.d.). Retrieved October 10, 2020, from https://www.payscale.com/research/ES/Job=Technical_Writer/Salary/7d396eb3/Barcelona

[22] High-performance audio | Android NDK | Android Developers. (n.d.). Retrieved October 10, 2020, from https://developer.android.com/ndk/guides/audio

[23] GitHub - google/oboe: Oboe is a C++ library that makes it easy to build high-performance audio apps on Android. (n.d.). Retrieved October 10, 2020, from https://github.com/google/oboe

[24] Fibra óptica: Tarifas de fibra óptica | Vodafone particulares. (n.d.). Retrieved October 10, 2020, from https://www.vodafone.es/c/particulares/es/productos-y-servicios/fibra-optica-adsl/

[25] Denys, D. (n.d.). Vivienda: Coste de gastos e Internet - ShBarcelona. Retrieved October 10, 2020, from https://www.shbarcelona.es/blog/es/vivienda-coste-de-gastos-e-internet/

[26] Audio - Apple Developer. (n.d.). Retrieved October 10, 2020, from https://developer.apple.com/audio/

[27] Android 10 ms problem: Explaining the delay of Android audio path | by Techtronix services | Medium. (n.d.). Retrieved October 11, 2020, from https://medium.com/@techtronix.tms/android-10-ms-problem-explaining-the-delay-of-android-audio-path-9daaecdbd78d

[28] Caustic 3 - Apps on Google Play. (n.d.). Retrieved December 31, 2020, from https://play.google.com/store/apps/details?id=com.singlecellsoftware.caustic

[29]Groovepad - Music & Beat Maker - Apps on Google Play. (n.d.). Retrieved December 31, 2020, from https://play.google.com/store/apps/details?id=com.easybrain.make.music

[30]Loopify Beta - Apps on Google Play. (n.d.). Retrieved December 31, 2020, from https://play.google.com/store/apps/details?id=com.zuidsoft.looper

[31]My drum pad: Custom drum pad & beat looper DrumPad - Apps on Google Play. (n.d.). Retrieved December 31, 2020, from https://play.google.com/store/apps/details?id=com.voxbox.drumpad

[32]Remixlive - Make Music & Beats - Apps on Google Play. (n.d.). Retrieved December 31, 2020, from https://play.google.com/store/apps/details?id=com.mixvibes.remixlive

[33]My Top 5 Sampler and Looper Apps for iOS 2011 - 2019 | haQ attaQ - YouTube. (n.d.). Retrieved January 12, 2021, from https://www.youtube.com/watch?v=wYnf4nwrrEk

[34]Loopy HD: Looper en App Store. (n.d.). Retrieved January 17, 2021, from https://apps.apple.com/es/app/loopy-hd-looper/id467923185

[35]Yellofier on the App Store. (n.d.). Retrieved January 17, 2021, from https://apps.apple.com/us/app/yellofier/id622611915

[36]Blocs Wave: Record Music Live on the App Store. (n.d.). Retrieved January 17, 2021, from https://apps.apple.com/us/app/blocs-wave-record-music-live/id1085697317

[37]Jam Looper on the App Store. (n.d.). Retrieved January 17, 2021, from https://apps.apple.com/us/app/jam-looper/id1061465697

[38]Drum Pad Machine - Beat Maker on the App Store. (n.d.). Retrieved January 17, 2021, from https://apps.apple.com/us/app/drum-pad-machine-beat-maker/id1057968965

[39]Your app and low-latency audio output | by Júlio Zynger | Medium. (n.d.). Retrieved December 31, 2020, from https://medium.com/@juliozynger/your-app-and-low-latency-audio-output-d21d7b672305

[40]Audio | Android Open Source Project. (n.d.). Retrieved December 31, 2020, from https://source.android.com/devices/audio

[41]Audio latency | Android NDK | Android Developers. (n.d.). Retrieved December 31, 2020, from https://developer.android.com/ndk/guides/audio/audio-latency

[42]Oboe, a Library for Low Latency Audio Apps on Android. (n.d.). Retrieved December 31, 2020, from https://www.infoq.com/news/2018/11/android-oboe/

[43]Unity Real-Time Development Platform | 3D, 2D VR & AR Engine. (n.d.). Retrieved December 31, 2020, from https://unity.com/

[44]Unity - Manual: Audio Manager. (n.d.). Retrieved December 31, 2020, from https://docs.unity3d.com/2018.2/Documentation/Manual/class-AudioManager.html

[45]Unity - Manual: Unity User Manual (2019.4 LTS). (n.d.). Retrieved December 31, 2020, from https://docs.unity3d.com/Manual/index.html

[46]Kotlin vs Java Comparison - Which One is Better? (2020). (n.d.). Retrieved December 31, 2020, from https://www.xenonstack.com/blog/kotlin-andriod/

[47]C++ documentation — DevDocs. (n.d.). Retrieved December 31, 2020, from https://devdocs.io/cpp/

[48]How to Write User Stories and Why They are Crucial for App & Software Success - Droids On Roids. (n.d.). Retrieved January 17, 2021, from https://www.thedroidsonroids.com/blog/how-to-write-user-stories-why-they-are-crucial-for-successful-app-development

[49]Data and file storage overview | Android Developers. (n.d.). Retrieved December 31, 2020, from https://developer.android.com/training/data-storage

[50]Metronome - Rosetta Code. (n.d.). Retrieved December 31, 2020, from https://rosettacode.org/wiki/Metronome

[51]Sequencer (Java Platform SE 7). (n.d.). Retrieved December 31, 2020, from https://docs.oracle.com/javase/7/docs/api/javax/sound/MIDI/Sequencer.html

[52]MIDI.Org Home. (n.d.). Retrieved December 31, 2020, from https://www.MIDI.org/

[53]MIDI | Android Open Source Project. (n.d.). Retrieved December 31, 2020, from https://source.android.com/devices/audio/MIDI

[54]MIDI Messages. (n.d.). Retrieved January 13, 2021, from https://users.cs.cf.ac.uk/Dave.Marshall/Multimedia/node158.html

[55]Circuit | Novation. (n.d.). Retrieved December 31, 2020, from https://novationmusic.com/en/circuit/circuit

[56]A Guide to User Feedback - How to collect and use it. (n.d.). Retrieved January 14, 2021, from https://userguiding.com/blog/user-feedback/

[57]15 useful user feedback questions for online surveys - UXM. (n.d.). Retrieved January 14, 2021, from http://www.uxforthemasses.com/online-survey-questions/

[58]What is Visual Testing? - Automated Visual Testing | Applitools. (n.d.). Retrieved January 11, 2021, from https://applitools.com/blog/visual-testing/

[59]oboe/apps/fxlab at master · google/oboe. (n.d.). Retrieved December 31, 2020, from https://github.com/google/oboe/tree/master/apps/fxlab

[60]1.2.1: 2.2.1-Duration- Note Lengths in Written Music - Humanities LibreTexts. (n.d.). Retrieved January 17, 2021, from https://human.libretexts.org/Bookshelves/Music/Book%3A_Understanding_Basic_Music_Theory_(OpenSTAX)/01%3A_Notation/1.02%3A_Time/1.2.01%3A_2.2.1-Duration-_Note_Lengths_in_Written_Music

[61]Sequentix Cirklon V2 Hardware Sequencer Announced With Faster CPU... (n.d.). Retrieved January 14, 2021, from https://www.synthanatomy.com/2019/05/sequentix-cirklon-v2-hardware-sequencer-announced-faster-cpu-colour-display-more.html