Facultade de Informática
UNIVERSIDADE DA CORUÑA

TRABALLO FIN DE GRAO
GRAO EN ENXEÑARÍA INFORMÁTICA
MENCIÓN EN SISTEMAS DE INFORMACIÓN

FUGATRA
The missing note, from concert to misstep

Estudante: Hilda Romero Velo
Dirección: Óscar Fresnedo Arias
José Pablo González Coma

A Coruña, xuño de 2021.

To Music, I will never leave you behind.

Acknowledgements

First, I would like to thank my family. Thanks to my parents for always supporting me and dedicating themselves to me. Their guidance and encouragement has been vital for me to grow in all aspects of my life.

I would also like to make a special mention to my piano teacher Nacho, who has been passing on his knowledge and passion for music to me since I was 7 years old, without which it would have been impossible to undertake this work.

Finally, I would also like to thank my TFG directors for supporting my idea and giving me the opportunity to carry it out.

Abstract

The project aims to create a web application to help pianists identify the mistakes they make during the performance of a piece of music. It is targeted at the initial phase of study, when it is crucial to correct errors in order to avoid acquiring them, as they are more difficult to fix afterwards. Users can create their own account, upload their own sheet music and connect their keyboard to the computer so that the application can support them during the learning process.

Resumo

O proxecto pretende crear unha aplicación web para axudar aos pianistas a identificar os erros que cometen durante a execución dunha peza musical. Está dirixido á fase inicial do estudo, cando é crucial corrixir os erros para evitar adquirilos, xa que despois son máis difíciles de rectificar. O usuario poderá crear a súa propia conta, subir as súas partituras y conectar o seu teclado ao ordenador para que a aplicación o apoie durante a aprendizaxe.

Keywords:

- Fugatra
- Fugateca
- Piano
- Music
- Learn
- Note
- Error
- Mistake
- Missing
- Web application
- MIDI

Palabras chave:

- Fugatra
- Fugateca
- Piano
- Música
- Aprender
- Nota
- Erro
- Fallo
- Perdida
- Aplicación web
- MIDI

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Music, it is becoming more and more easily accessible. There are countless applications that allow us to listen to it. There are also applications that have facilitated the composition of new pieces of music. Producing new music has never been so accessible. What about learning to play it? There are certainly more and more opportunities to do so, and an increasing number of people understand its benefits.

Learning to play a musical instrument does not only provide musical knowledge. For instance, playing the piano leads to better coordination and independence. With any instrument one learns to listen and to play in group. Concentration and controlling nervousness are also necessary when playing a piece of music.

However, it is hard to learn to play an instrument. A fundamental part and common to all of them is technique, but then each instrument has its own particularities. There are musical instruments for which getting the tuning right is a constant concern. This is not a problem for piano, but which are the difficulties of learning to play it?

The piano is a polyphonic instrument, which means that many notes can be played at the same time. Hence, the player will have to read several notes together in the score. In addition, the notes played in the right hand are usually read in a different clef from those played in the left hand. What does this mean? Let's make an analogy.

Imagine that what the pianist reads for the right hand is a Spanish text and what he reads for the left hand is an English text. Both texts deal with the same subject but with particular characteristics. The pianist must understand everything, the basis and the individual features. The students will read these texts as many times as they practice. What happens if they have misunderstood some of the particularities of one of the texts, skipped some of them or confused English with Spanish? Most likely they will not realise the mistake they are making and once learned it will be much more difficult to fix.

In short, and returning to music, the difficulty for the pianist is in finding the misplayed notes as he/she is reading many notes at the same time and sonorously the error may not be obvious to him/her.

What if there was an app that could help pianists detect the mistakes they make? That is what this project seeks to develop. In the next two sections, the origin of this idea is clarified and its objectives are explored in more detail.

1.1 Origin

The motivation arises from the difficulty that pianists sometimes have in identifying the mistakes they make, either because they are minimal and go unnoticed or because they have already been acquired from committing them repeatedly so often. The aim is to make it easier for the user to follow his or her progress. In addition, this improves the motivation of the student, as finding and correcting the most subtle technical errors is, usually, the most arid part of the study of a piece of music. The possibility of visualising his/her mistakes with regard to what is to be expected from the musical sheet provides an unprecedented learning control. Therefore, a more accurate execution gives him/her more confidence to continue working on the composition.

Nowadays, there is a strong move towards online classes. One of the biggest challenges of this migration lies in the difficulty for the teacher to reach the student because, in order to identify the student's problems, he/she needs a direct feedback from them, observation is no longer enough. In arts education, however, this gap between teacher and pupil increases dramatically. In fact, for the study of a musical instrument, this link is essential because the teacher must detect the student's mistakes immediately so that he/she can learn correctly.

As will be seen in section 1.3, there are no applications on the market that focus on this gap. The development of this project could lead to establishing the online connection between the learner and the teacher because, the application will detect the mistakes the learner makes, the teacher could see them and understand where the difficulties are for the student. Depending on the teacher's evaluation of the mistakes, he/she can identify whether the student needs an explanation of a theoretical musical concept or needs some extra study on technique, for example. Therefore, another potential use is for educational purposes in music schools by facilitating online classes.

1.2 Purpose and Objectives

The aim is to develop a web application with which users will be able to create their own account and upload their scores. Then, after having connected a keyboard to the computer, they can record their performance at the tempo of their choice. Once they have finished, they will be shown the mistakes they have made. These mistakes are then recorded in a history that is available for the user to follow his/her evolution.

The first steps of the study of a music score are critical. Thus, one of the objectives of the application is to help at this tricky moment prior to consolidating passages that may contain errors. Therefore, just the rhythm and the pitch of the notes will be checked. However, the use of pedal, interpretation, dynamics, technique or repetitions will not be reviewed.

The decision of creating a web application to offer these functionalities is targeted at relieving the user of the computational burden and minimising the use of his/her computer's resources. Moreover, with this solution, the customer can access the application from any device. Accordingly, the goal is also to maximise availability and usability, that is, make it easier for the user to access the application by avoiding any type of complex configuration.

It is important to highlight that, no matter how much assistance the application can provide to the student, it is not a substitute for the teacher. The software is intended to be complementary and supportive. The work done by the teacher is not replaceable. There are many concepts and subjects that only a person, not a machine, can convey, such as technique, interpretation, or advice, among others.

1.2.1 Particular Objectives

In light of the purpose of the application, the objectives could be summarised as follows:

- User management so that each user has individualised access to its content.
- The clients will have their own library of scores, so they must be able to manage it.
- Recording of user performance, error detection and error display.
- Design of a simple and functional interface, allowing users to focus on their work and avoiding, as far as possible, technical complications.

1.3 Similar Existing Solutions

In the following, I will present some existing applications that can be similar to the program developed in this project. All of them aim to teach or help to learn to play the piano but each one offers certain functionalities.

- Moon Piano [1]
  It offers both a desktop application and an online service. It allows you to upload your own scores (with a limit of 10 scores if working on the web). Displays the score and plays the sound for the user to rehearse but does not show the mistakes the performer has made, it only shows some statistics at the end.

- Pianu [2]
  Web application that offers a collection of sheet music but does not allow users to upload their own. During the rehearsal it shows the score and indicates the keys to be played. Like Moon Piano, it does not represent errors in the score and only offers some final statistics.

- Playground Sessions [3]
  One of the most complete web applications. Although it does not allow users to upload their own scores, it has a large library. It displays the score and represent the mistakes made by the performer. Green marks represent the correctly played notes whereas red marks are used when the original note is wrongly executed. It also draws the incorrect note played by the user. However, even though it provides some statistics, it does not keep a history of user errors. The concept of the application is a video game in which you advance in levels and unlock extra content as you learn.

- Flowkey [4]
  The strength of this web application lies in all the configuration options that it offers (speed adjustment, selection of passages, repetition of fragments...). It shows the score and a video of its execution. It allows the player to advance through the score as he/she hits the right notes. It does not allow users to upload their own scores.

- Skoove [5]
  Web application that displays the score and points out mistakes to the user. It offers a playback of the sound of the score for learning by listening. Although this may seem enough at first, in order to learn music, not only from a pianistic point of view, it is essential to associate sounds with their graphic representation. Also, it does not allow users to upload their own sheet music.

There are many similar applications on the market that offer the same functionalities as the above mentioned [6]. In this list I have highlighted the most complete and the most famous ones [7].

As it has been checked, none of the applications seen above offers what this project offers. This project allows the users to have their own library of scores, so it is up to them to choose what they want to study. Moreover, the student will not be distracted during the execution of the piece because errors will be displayed upon completion. Another important detail is that the mistakes will be shown written in a score. This helps to relate the musical representation of the error to the sound. Finally, the learners will be able to review their progress thanks to the application's error history.

To sum up, the application of this project aims to serve as a tool for the user, making it easier for them to focus on what is important to them, to detect the mistakes they make and to study with confidence.

1.4 The Fugatra Brand

This section is intended to present the key to the project by explaining its name, slogan, logo and associated terminology.

- **Name**
  The name of the application is Fugatra. It arises from the union of *Fuga*, which means escape in Spanish, and the Greek suffix *-atra*, which means "the one who takes care". Therefore, its literal meaning is "the one who takes care of the escape". In context, it refers to the application taking care of the "lost notes".

- **Slogan**
  "The missing note, from concert to misstep". It refers to the "lost notes", those that are not well played and therefore do not exist, they are missing. The last part, "from concert to misstep", means that the program gets the errors from the concert given by the user.

- **Logo**
  The figure 1.1 shows the logo of the application. The letters "UT" refer to the old name for the note C (Do) [8] and it represents the "missing note". The symbol surrounding the letters is actually a bass clef, but it also resembles a question mark. Combined, it comes to represent the idea of: "Where is the missing note?".

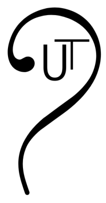


Figure 1.1: Fugatra logo

- **Terminology**
  - Fugalogo: The Greek suffix "-logo" means "specialist" or "knower", so Fugalogo is the "leak specialist" or "the one who knows about leaks". It is used to refer to the user who uses the application. In the end, he/she is the one who "commits the leaks" and with the application he "knows" about them.
  - Fugateca: The Greek suffix "-teca" means "place where something is kept" or also "collection". Fugateca is the place where the user stores his scores.
  - Fugáfono: The Greek suffix "-fono" means "sound". Fugáfono is the place where the user "produces sound" and where those "sound leaks" are detected. This means, the place where the user studies, where he/she plays the score and where the errors, the "leaks', are communicated to him/her.

1.5 Motivation For The Report Structure

The flow of this memory is based on the experience lived by the student who writes it in the development of the project.

In the introduction, the identified problem has been presented, the idea for the solution has been introduced and an investigation of similar existing systems has been made.

After having the reason, the idea and the certainty that what is going to be developed is novel, the first thing the student does is to wonder: How to approach the project? (methodology), What are the steps to follow? (planning), Is it feasible? (risk analysis). This is explained in the Methodology and Planning chapter (2).

The next stage is clearly defining the project and getting a more concrete visualisation of what it is going to be. This is covered in the Analysis chapter (3) with requirements, prototypes and the use case diagram.

Once the scope of the project is known and defined, the structure of the program (Design chapter, Architecture section 4.1) and the required data (Design chapter, Data Modelling section 4.2) need to be determined.

Knowing all of the above, and having a clear idea of what is to be developed, the technologies to be used are chosen. For example, if the risk analysis determines that a desktop application should be developed, the technologies to be used are not the same as those needed to develop a web application.

Finally, before explaining how the application has been developed, the chapter of Theoretical Fundaments (6) provides the theoretical knowledge necessary to understand it.

Therefore, the structure of the report reflects the student's experience and aims to be as clear as possible to facilitate the reader's comprehension.

Chapter 2

Methodology and Planning

In order to develop a project, it is essential to conduct a proper planning of it. In this chapter, the methodology selected to develop this project, its planning and its cost estimation will be discussed in depth.

2.1 Methodology

An iterative and incremental methodology has been chosen. This allows the product life cycle to be divided into temporary blocks in order to satisfactorily complete each phase of the project, taking the customer's requirements as a reference for the limitation of these blocks. At the end of each phase, before proceeding to the next one, the necessary tests are carried out to check that what has been implemented works according to the requirements.

In an incremental development, with each phase one more piece of the system is obtained. More specifically, it aims to gradually add functionality for the user, which are known as user stories. Therefore, this allows the project requirements to be not fully defined at the beginning.

Under iterative development, in each cycle (iteration) the product is reviewed and improved. This does not involve adding functionality to the product as with incremental development, but it does include review and improvement.

Therefore, an iterative and incremental development is one in which, with each delivery, it adds completely new features (incremental) but each increment also includes improvements on features that already existed (iterative). This simplifies the error detection and their early resolution.

Hence, with this methodology, the project can be simplified by applying the principle of "divide and conquer". This approach facilitates planning and estimation by breaking down development into a larger amount of smaller tasks so that they are easier to manage.

Furthermore, this methodology adapts well to the project to be developed because its requirements are not robust and can fluctuate during development as the client considers its evolution. Moreover, as it is not a completely defined project, it is easier to make modifications or fix errors, and the nature of this methodology allows them to be detected and consequently solved as soon as possible.

2.2 Planning

In this section, the available resources to undertake the project are presented, a planning is made considering them and a cost estimate is provided. All tools and technologies chosen during these sections will be explained in chapter 5.

2.2.1 Resources

A resource is any element that is necessary to complete a task or, in this case, a project. Knowing and planning resource management is essential to complete a project as effectively and efficiently as possible.

Human Resources

For human resources management, it is essential to recognise the strengths of each team member in order to be able to correctly assign roles within the project. Although in this project there are very few human resources available, the profiles that would need to be covered and how they would be distributed will be simulated.

- Project Manager
  This is the person in charge of establishing the guidelines for a project and overseeing that its progress is in accordance with the plan. Experience is a key factor in this position. For this project, the two degree thesis directors will assume this role.

- Analyst
  It is responsible for studying the system to be built. He/she performs a research of the client's needs and translates them into requirements that describe the software to be developed. The most important document which he/she has to produce is the Software Requirements Specification (SRS). For this project, the student will be in charge of interpreting this role.

- Designer
  This is the one who designs the system to be developed according to the requirements gathered by the analyst. The student will undertake this role.

- Developer
  The person who implements the system according to the designs generated by the designer. He/she must also be responsible for performing the relevant unit tests that guarantee the correct development of the project. This role will be performed by the student.

- Tester
  It is responsible for conducting the remaining tests to ensure that the project successfully satisfies the requirements. In this project, user acceptance testing will be undertaken to obtain the feedback of a real user. The user in charge will be the student's piano teacher.

Material Resources

- Software Resources: As detailed in chapter 5 of this report.

- Hardware Resources:

  – Personal Laptop:
    Model: MSI GF63 Thin 10SCXR-042XES
    CPU: Intel(R) Core(TM) i7-10750H
    Hard disk: 1 TB NVMe PCIe Gen3x4 SSD
    RAM: 16GB DDR4
    OS: Microsoft Windows 10 Pro
    Graphic card: Nvidia GeForce GTX 1650 MAX Q

  – Keyboard with MIDI support: CASIO PRIVIA PX-560 DIGITAL PIANO

  – USB Type A to Type B cable: Used to connect the keyboard to the computer to capture the MIDI signal.

  – Samsung Galaxy S7 Edge Mobile

  – Samsung Galaxy S20 Mobile

  – Sony Xperia Z1 Tablet

2.2.2 Project Planning

1. **Study of initial product requirements**

Although this project did not follow the Scrum methodology, one of its techniques was borrowed by taking advantage of the fact that both, this methodology and the iterative and incremental methodology, follow the Agile Manifesto [9]. This is the Product

Backlog, a list of all the initial requirements of the product to be developed.

In order to plan the project, different user stories will be distinguished and specified in the simplified Product Backlog shown in table 2.1. The reading of a user story is: "As «role», I want «functionality» to get «outcome»". In our case we have one actor, the user, in a single role, as administrator of his/her own account. Therefore, the Product Backlog only specifies the functionality and the result that the user expects to obtain as the administrator of his/her account. In addition, a column is added with an estimate in hours of the cost that each feature may require to be developed.

Table 2.1: Product Backlog

Nº	**Functionality**	**Outcome**	**Estimate (H)**
1	Create an account	Account	27
2	Log in	Obtain authorisation to access their data	16
3	Edit account information	Updated account information	8
4	Log out	Authorisation to access their data revoked	4
5	Recover password	New password has been updated	24
6	Upload score	Saved score	20
7	Edit score information	Updated score information	8
8	Delete score	Score and related information have been deleted	8
9	View list of scores sorted by title, composer, genre or most recently uploaded	Scores displayed ordered by title, composer, genre or most recently uploaded	20
10	Create folder	Folder created	10
11	Delete folder	Folder and related information have been deleted	4
12	Edit folder information	Updated folder information	6
13	Add score to folder	Score has been added to the folder	14
14	Delete score from folder	Score has been deleted from the folder	8
15	View score	Score is displayed	14

16	Zoom control in score visualisation	Score is expanded or reduced	2
17	Turn on metronome	Metronome started playing	3
18	Turn off metronome	Metronome stopped playing	3
19	Choose number of input bars	Number of input bars has been set	3
20	Modify metronome tempo	New metronome tempo is set	3
21	Record score performance: Backend	Committed errors are detected	50
	Record score performance: Frontend	Committed errors are visualised	30
22	Browse error history	Error history is displayed	20
23	View errors from an old execution	Old execution errors are displayed	16

2. **Feasibility and risk analysis**

   Before planning a project, it is very important to conduct a feasibility study. A project's feasibility is its ability to be completed successfully by delivering the expected results. Different perspectives such as technical, economic or legal must be analysed. For this project, it will only be studied from a technical point of view, because, for example, there is no budget limit, although a cost estimate will be provided in step 2.2.3.

   - Project scope: It has been delimited by the Product Backlog list shown in table 2.1.
   - Requirements definition: Specified in chapter 3.
   - Determination of the approach: The adopted approach will be to rely on a development in iterations to ensure progressive delivery of functionalities to the user, with continuous improvement according to the feedback collected. Hence, the chosen methodology was the iterative and incremental one (section 2.1).
   - Situation analysis: Existing technologies in the domain have already been presented in 1.3 and it has been determined that the product to be developed is new to the market. A detailed explanation of the challenges faced, the solutions taken and the risks involved together with a management of them is provided in a special section below..

Detailed Situation Analysis

When the idea of undertaking this project arose, after having defined the scope for it, the critical points that it could have were studied. Three of the most noteworthy technical risks will be presented below.

- **Collect the user's MIDI input**
  One of the most important points because it could determine the type of application to be developed. If there was no way to gather the MIDI signal from the user's keyboard in a web application, a desktop application would have had to be developed. After some research and tests implementing an HTML and a JavaScript file to collect the MIDI input from the keyboard and print the emitted data, the Web MIDI API was chosen as a solution to this challenge.

- **Obtain score information**
  In order to be able to compare the user's playing with the original score, it is necessary to be able to obtain the information that is in the sheet music. The research focused on whether there were any standards for representing music. Although there are several notations, such as abc notation [10], the most widespread and complete format is MusicXML [11]. In fact, MusicXML has been developed by MakeMusic [12], the world leader in music technology. Among its products we find Finale [13], the world's best-selling music notation software that allows the music community to create, edit, print and publish sheet music.

- **Display the score to the user**
  Another key factor was to be able to provide the user with both the original score and the score with the mistakes he or she had made. One of the safe options was to use the LilyPond software [14] to obtain an image in SVG or PNG format or a PDF file and then display it to the user. However, after an exhaustive search, OSMD [15], the missing link between MusicXML and VexFlow, was found. VexFlow [16] is an open source online music notation rendering API that is written entirely in JavaScript and runs directly in the browser (it supports HTML5 Canvas and SVG). Thanks to OSMD [15] it was possible to work completely with the scores in MusicXML format and then render them easily for the user.

After the analysis, it is determined that all the necessary elements are provided to undertake the development and that the required workload is adequate for a final degree project.

3. **Project phases planning**

The project has been planned in three main phases, each with a package of associated functionalities that have a strong common link:

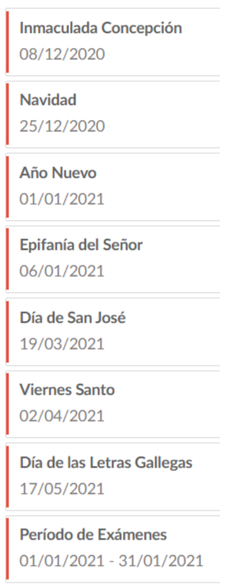
I Authentication:
Account management functionalities (creation, log-in, profile modification and log-out).

II Fugateca:
Score library management functionalities (uploading of scores, edition of score information, creation and deletion of folders, association of scores to a folder, etc.).

III Fugáfono:
Study-related functionalities (view score, configure metronome, record performance, display errors, browse error history...).

Additionally, two more phases are distinguished, an initial phase where the project is defined and the necessary preliminary studies are undertaken, and a final phase for user testing. No distinction will be made between backend and frontend development in planning. Besides, at the end of each iteration, a review meeting with the project managers is scheduled to evaluate the overall progress of the system.

For project planning, the holidays in figure 2.1 have been taken into account and the month of January has been disabled because the developer was not available. Also, 4 hours of labour time have been established due to the student's situation.



Inmaculada Concepción
08/12/2020

Navidad
25/12/2020

Año Nuevo
01/01/2021

Epifanía del Señor
06/01/2021

Día de San José
19/03/2021

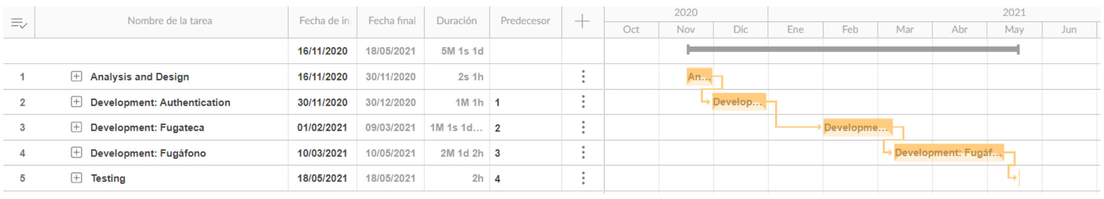
Viernes Santo
02/04/2021

Día de las Letras Gallegas
17/05/2021

Período de Exámenes
01/01/2021 - 31/01/2021

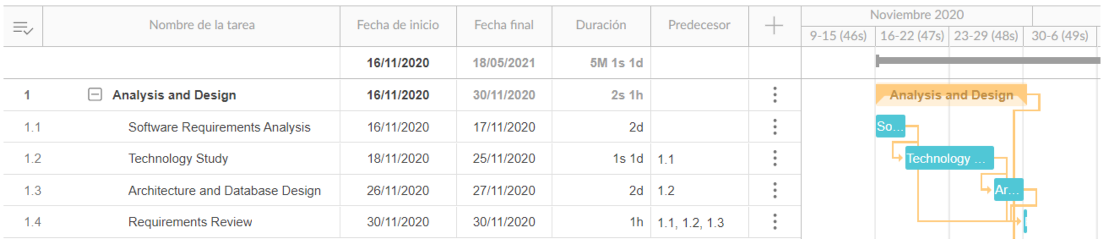
Figure 2.1: Non-Working Days

- **Gantt Diagram**
  The Gantt diagrams in the following figures show the initial planning of the project. It should be noted that, as there was only one developer, the planning of activities was sequential, although many could have been performed in parallel.



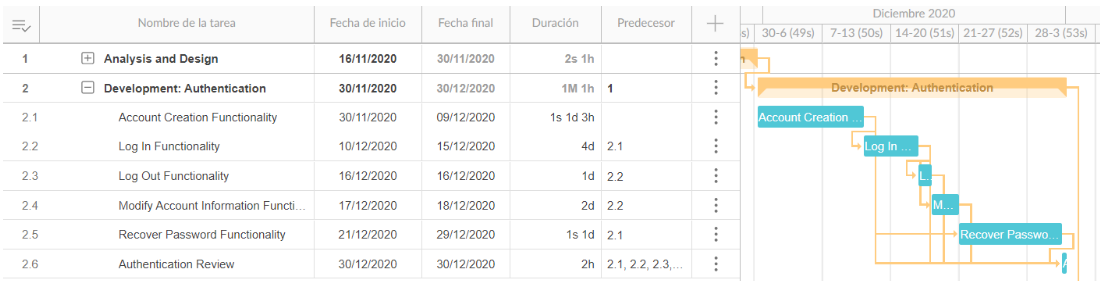
	Nombre de la tarea	Fecha de ini	Fecha final	Duración	Predecesor
		16/11/2020	18/05/2021	5M 1s 1d	
1	Analysis and Design	16/11/2020	30/11/2020	2s 1h	
2	Development: Authentication	30/11/2020	30/12/2020	1M 1h	1
3	Development: Fugateca	01/02/2021	09/03/2021	1M 1s 1d…	2
4	Development: Fugáfono	10/03/2021	10/05/2021	2M 1d 2h	3
5	Testing	18/05/2021	18/05/2021	2h	4

Figure 2.2: Main Phases Gantt Diagram



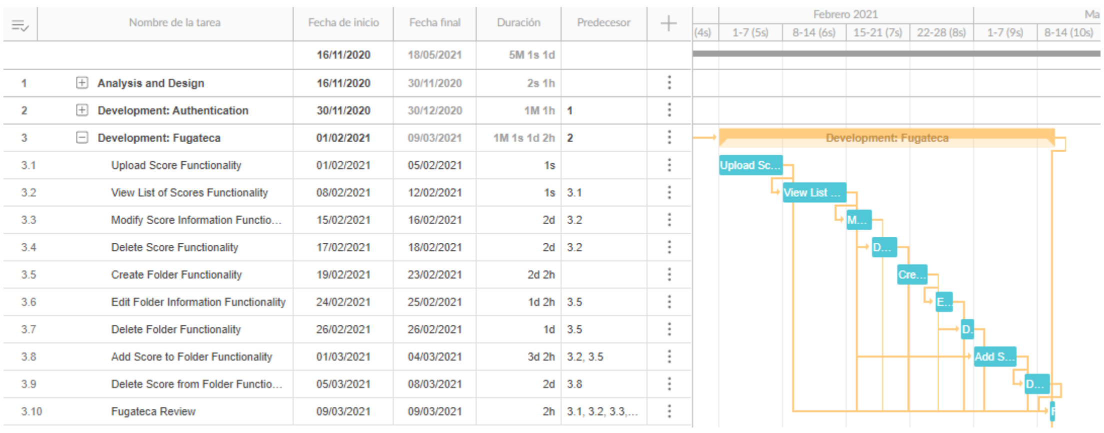
	Nombre de la tarea	Fecha de inicio	Fecha final	Duración	Predecesor
		16/11/2020	18/05/2021	5M 1s 1d	
1	Analysis and Design	16/11/2020	30/11/2020	2s 1h	
1.1	Software Requirements Analysis	16/11/2020	17/11/2020	2d	
1.2	Technology Study	18/11/2020	25/11/2020	1s 1d	1.1
1.3	Architecture and Database Design	26/11/2020	27/11/2020	2d	1.2
1.4	Requirements Review	30/11/2020	30/11/2020	1h	1.1, 1.2, 1.3

Figure 2.3: Analysis and Design Gantt Diagram



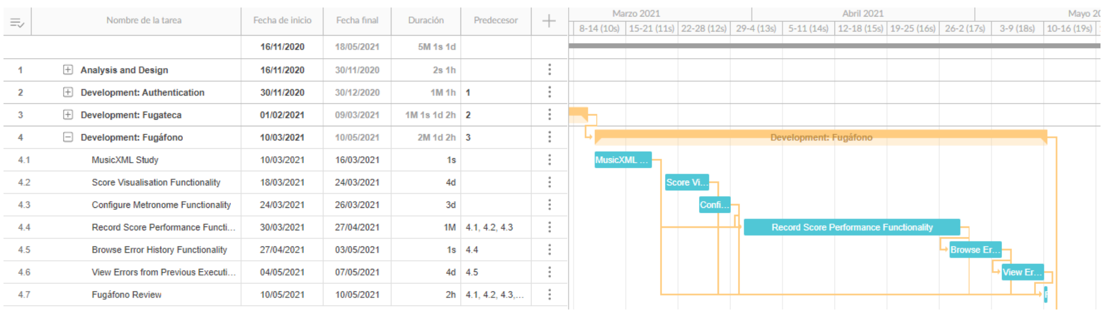
	Nombre de la tarea	Fecha de inicio	Fecha final	Duración	Predecesor
1	Analysis and Design	16/11/2020	30/11/2020	2s 1h	
2	Development: Authentication	30/11/2020	30/12/2020	1M 1h	1
2.1	Account Creation Functionality	30/11/2020	09/12/2020	1s 1d 3h	
2.2	Log In Functionality	10/12/2020	15/12/2020	4d	2.1
2.3	Log Out Functionality	16/12/2020	16/12/2020	1d	2.2
2.4	Modify Account Information Functi…	17/12/2020	18/12/2020	2d	2.2
2.5	Recover Password Functionality	21/12/2020	29/12/2020	1s 1d	2.1
2.6	Authentication Review	30/12/2020	30/12/2020	2h	2.1, 2.2, 2.3,…

Figure 2.4: Authentication Gantt Diagram



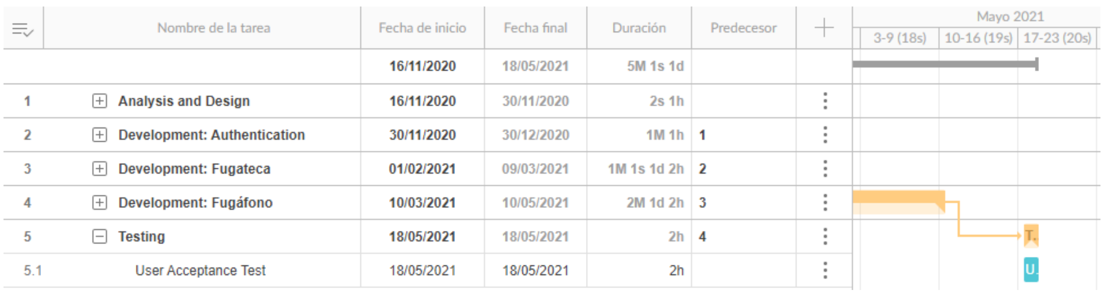
	Nombre de la tarea	Fecha de inicio	Fecha final	Duración	Predecesor
		16/11/2020	18/05/2021	5M 1s 1d	
1	⊞ Analysis and Design	16/11/2020	30/11/2020	2s 1h	
2	⊞ Development: Authentication	30/11/2020	30/12/2020	1M 1h	1
3	⊟ Development: Fugateca	01/02/2021	09/03/2021	1M 1s 1d 2h	2
3.1	Upload Score Functionality	01/02/2021	05/02/2021	1s	
3.2	View List of Scores Functionality	08/02/2021	12/02/2021	1s	3.1
3.3	Modify Score Information Functio...	15/02/2021	16/02/2021	2d	3.2
3.4	Delete Score Functionality	17/02/2021	18/02/2021	2d	3.2
3.5	Create Folder Functionality	19/02/2021	23/02/2021	2d 2h	
3.6	Edit Folder Information Functionality	24/02/2021	25/02/2021	1d 2h	3.5
3.7	Delete Folder Functionality	26/02/2021	26/02/2021	1d	3.5
3.8	Add Score to Folder Functionality	01/03/2021	04/03/2021	3d 2h	3.2, 3.5
3.9	Delete Score from Folder Functio...	05/03/2021	08/03/2021	2d	3.8
3.10	Fugateca Review	09/03/2021	09/03/2021	2h	3.1, 3.2, 3.3,...

Figure 2.5: Fugateca Gantt Diagram



	Nombre de la tarea	Fecha de inicio	Fecha final	Duración	Predecesor
		16/11/2020	18/05/2021	5M 1s 1d	
1	⊞ Analysis and Design	16/11/2020	30/11/2020	2s 1h	
2	⊞ Development: Authentication	30/11/2020	30/12/2020	1M 1h	1
3	⊞ Development: Fugateca	01/02/2021	09/03/2021	1M 1s 1d 2h	2
4	⊟ Development: Fugáfono	10/03/2021	10/05/2021	2M 1d 2h	3
4.1	MusicXML Study	10/03/2021	16/03/2021	1s	
4.2	Score Visualisation Functionality	18/03/2021	24/03/2021	4d	
4.3	Configure Metronome Functionality	24/03/2021	26/03/2021	3d	
4.4	Record Score Performance Functi...	30/03/2021	27/04/2021	1M	4.1, 4.2, 4.3
4.5	Browse Error History Functionality	27/04/2021	03/05/2021	1s	4.4
4.6	View Errors from Previous Executi...	04/05/2021	07/05/2021	4d	4.5
4.7	Fugáfono Review	10/05/2021	10/05/2021	2h	4.1, 4.2, 4.3,...

Figure 2.6: Fugáfono Gantt Diagram



	Nombre de la tarea	Fecha de inicio	Fecha final	Duración	Predecesor
		16/11/2020	18/05/2021	5M 1s 1d	
1	⊞ Analysis and Design	16/11/2020	30/11/2020	2s 1h	
2	⊞ Development: Authentication	30/11/2020	30/12/2020	1M 1h	1
3	⊞ Development: Fugateca	01/02/2021	09/03/2021	1M 1s 1d 2h	2
4	⊞ Development: Fugáfono	10/03/2021	10/05/2021	2M 1d 2h	3
5	⊟ Testing	18/05/2021	18/05/2021	2h	4
5.1	User Acceptance Test	18/05/2021	18/05/2021	2h	

Figure 2.7: Testing Gantt Diagram

2.2.3 Cost Estimate

The project cost estimate will be made according to the resources involved:

- Human Resources
  For human resources costing, the salaries of reference [17] as summarised in table 2.2 are taken as a baseline. The tester profile will not be taken into account for the estimates as his/her conducted activity is a minor user acceptance test.

  Table 2.2: Human Resources Salary

Profile	Hourly Salary	Annual Salary
Project Manager	20.50€	40,000€
Analyst	14.30€	28,000€
Designer	12.80€	25,000€
Developer	9.70€	19,000€

  Taking into account the planning of the project and assuming that the project managers participate in activities 1.4, 2.6, 3.10 and 4.7; the analyst participates in activities 1.1, 1.4 and 5.1; the designer participates in activities 1.2, 1.3 and 1.4; and the programmer participates in all activities in phases 2, 3 and 4, the total estimated working hours according to the profile is as specified in the table 2.3.

  Table 2.3: Total Estimated Working Hours by Profile

Profile	Working Hours
Project Manager	7h
Analyst	11h
Designer	33h
Developer	347h

  Considering that there are two project managers and that the tasks they perform are not effort-driven, the total estimated cost for human resources is 4,232.60€.

- Software Resources:
  None of the required software resources represent an additional cost for the project. Total cost of software resources: 0€.

- Hardware Resources
  Both the cable and the keyboard are personal and no additional cost will be charged for

them since their use in the project is minimal compared to their normal wear and tear. On the other hand, the personal computer depreciation will be imputed. Assuming the original cost of the computer is 1,200€ and considering a useful lifetime of 4 years, the value of the computer consumed in 1 year is 300€. If the duration of the project is approximately 5 months, the imputable depreciation of the computer is over 125€.

A final summary of the estimated costs for the project is shown in table 2.4.

Table 2.4: Estimated Costs

Resource	**Cost**
Human	4,232.60€
Software	0€
Hardware	125€
Total	**4,357.60€**

2.2.4 Project Monitoring

The evolution of the project has followed the planning with minor deviations in the duration of some activities, which have balanced each other out. The only significant deviation was that of activity 4.4 “Record Score Performance Functionality” which, due to its great complexity, delayed the project by 4 days. However, the date initially set for undertaking the user acceptance test could be maintained. Table 2.5 details the forecast and the final result for the project parameters.

Table 2.5: Forecast vs. Follow-up

	Forecast	**Follow-up**
Start Date	11 Nov 2020	11 Nov 2020
End date	10 May 2020	14 May 2020
Duration	158 days	162 days
Work	405 hours	421 hours
Cost	4,357.60€	4,512.80€

Chapter 3

Analysis

This chapter explains both the functional and non-functional requirements of the application. In order to help obtain them, prototypes of the screens that could be shown to the user have been developed, and they are attached in Appendix A. Finally, the use cases that shape and define the project are presented.

3.1 Functional Requirements

A functional requirement is a description of the service to be provided by the software. In table 3.1 the requirements identified for the application are specified according to their input, the behaviour of the system, the expected output and the corresponding user story from those specified in the Product Backlog 2.1.

- **Functional Requirements**

Table 3.1: Functional Requirements

User Story	**Input**	**System Behaviour**	**Output**
Create an account (1)	The user's name, surname, email address and password.	Performs the relevant checks (correct data, email is not already registered...) and, if everything is correct, it registers the user in the database. Otherwise, an appropriate error will be issued.	If everything is correct, the information is registered and therefore there is an account created. Otherwise, the output will be the corresponding error.

Log in (2)	The user's email address and password	If the email is registered and the password is correct, the user is authorised. Otherwise, an appropriate error will be issued.	If everything is correct, authorisation to access data is obtained. Otherwise, the output will be the corresponding error.
Edit account information (3)	Authorisation and new user information (name and/or surname).	Perform relevant checks and, if everything is correct, update the user's information in the database. Otherwise, an appropriate error will be issued.	If everything is correct, new user information has been registered. Otherwise, the output will be the corresponding error.
Log out (4)	Authorisation	Perform relevant checks and, if everything is correct, revoke authorisation. Otherwise, an appropriate error will be issued.	If everything is correct, authorisation has been revoked. Otherwise, the output will be the corresponding error.
Recover password (5)	The user's email address	If the email is registered, send an email to the user to update his/her password and, when this happens, if everything is correct, the new password is updated. Otherwise, an appropriate error will be issued.	If everything is correct, the new password has been updated. Otherwise, the output will be the corresponding error.
Upload score (6)	Authorisation, score title, composer, genre, tempo and file in MusicXML or compressed MusicXML (mxl) format.	Perform relevant checks and, if everything is correct, it registers the score in the database. Otherwise, an appropriate error will be issued.	If everything is correct, the score has been registered. Otherwise, the output will be the corresponding error.

Edit score information (7)	Authorisation, score to be updated and new score information (title, composer, genre and/or tempo).	Perform relevant checks and, if everything is correct, update the score's information in the database. Otherwise, an appropriate error will be issued.	If everything is correct, new score information has been updated. Otherwise, the output will be the corresponding error.
Delete score (8)	Authorisation and score to be deleted.	Perform relevant checks and, if everything is correct, delete score (related files and information). Otherwise, an appropriate error will be issued.	If everything is correct, score information and related files have been deleted. Otherwise, the output will be the corresponding error.
View sorted list of scores (9)	Authorisation and sorting order.	Perform relevant checks and, if everything is correct, gather user's scores sorted by title, composer, genre or most recently uploaded. Otherwise, an appropriate error will be issued.	If everything is correct, a list of scores sorted by title, composer, genre or most recently uploaded is obtained. Otherwise, the output will be the corresponding error.
Create folder (10)	Authorisation and folder name.	Perform relevant checks and, if everything is correct, it creates the folder in the database. Otherwise, an appropriate error will be issued.	If everything is correct, the folder has been registered. Otherwise, the output will be the corresponding error.
Delete folder (11)	Authorisation and folder to be deleted.	Perform relevant checks and, if everything is correct, delete folder (but not the related scores). Otherwise, an appropriate error will be issued.	If everything is correct, the folder and its links to scores have been deleted. Otherwise, the output will be the corresponding error.

Edit folder information (12)	Authorisation, folder to be updated and new folder name.	Perform relevant checks and, if everything is correct, update folder name. Otherwise, an appropriate error will be issued.	If everything is correct, folder name has been updated. Otherwise, the output will be the corresponding error.
Add score to folder (13)	Authorisation, score to be added and the respective folder.	Perform relevant checks and, if everything is correct, attach score to the folder. Otherwise, an appropriate error will be issued.	If everything is correct, score has been attached to the folder. Otherwise, the output will be the corresponding error.
Delete score from folder (14)	Authorisation, score to be deleted and the respective folder.	Perform relevant checks and, if everything is correct, detach score from folder. Otherwise, an appropriate error will be issued.	If everything is correct, score has been detached from folder. Otherwise, the output will be the corresponding error.
View score (15)	Authorisation and score to be displayed.	Perform relevant checks and, if everything is correct, gather the score file and the corresponding information to display it. Otherwise, an appropriate error will be issued.	If everything is correct, the score has been displayed. Otherwise, the output will be the corresponding error.
Zoom control (16)	New score size smaller than the current one.	Zoom in score visualisation if possible.	If it has been possible, score visualisation has been zoomed in.
Zoom control (16)	New score size bigger than the current one.	Zoom out score visualisation if possible.	If it has been possible, score visualisation has been zoomed out.
Turn on metronome (17)	Tempo.	Turn on metronome with the respective tempo.	Metronome started playing with the specified tempo.

Turn off metronome (18)	-	Turn off metronome.	Metronome stopped playing.
Choose number of input bars (19)	Number of input bars.	If the new number is greater than one, the number of input bars is updated.	If possible, number of input bars has been set to the new number.
Modify metronome tempo (20)	New tempo	If the new tempo fits within the range, the new tempo is set.	If possible, new metronome tempo has been set.
Record score performance (21)	Authorisation, user's midi signal information and score to be compared to.	Perform relevant checks and, if everything is correct, gather the score file, compare user's midi signal and original score information, save user's errors in a copy of the original score and display them to the user. Otherwise, an appropriate error will be issued.	If everything is correct, the score of commited errors has been displayed. Otherwise, the output will be the corresponding error.
Browse error history (22)	Authorisation and score to search history for.	Perform relevant checks and, if everything is correct, gather error history information for that score and display it to the user. Otherwise, an appropriate error will be issued.	If everything is correct, a history of errors for the given score has been displayed. Otherwise, the output will be the corresponding error.
View errors from an old execution (23)	Authorisation and a selection from the error history for a given score.	Perform relevant checks and, if everything is correct, gather the corresponding error score file and the related information to display it. Otherwise, an appropriate error will be issued.	If everything is correct, the error score file and the corresponding information have been displayed. Otherwise, the output will be the corresponding error.

3.2 Non-Functional Requirements

Non-functional requirements represent general characteristics and constraints of the system being developed. They are also known as quality attributes. For this project, the following non-functional requirements will be pursued:

- Performance efficiency: The amount of used resources will be minimised. Above all, the aim is to relieve the user of the burden on its resources, therefore, the most costly tasks will be performed on the server side.

- Compatibility: Beyond having opted for a REST style architecture as will be described in section 5.1, the aim is to provide maximum compatibility (e.g. different devices such as mobiles or laptops with different screen sizes and resolutions, different browsers, ...).

- Usability: An ever-present point in this project is the pursuit of user-friendliness. Therefore, all the development will be focused on simplifying the tasks as much as possible for the user so that the learning curve is fast. The interfaces will seek to be as intuitive as possible and the user will have feedback from the application at all times, plus, errors will be communicated appropriately. Another objective of the application is to be web responsive so that the user can operate from any device.

- Security: Certainly, as it is a system that works with the user's personal data, security is an ever-present factor. Any interaction with user data will require prior authentication, which will always be checked. In addition, data such as passwords will be encrypted before being recorded in the database.

- Maintainability: The system shall be divided into modules in such a way that the impact of changing a component is minimal. A simple example of this is the use of the DTO pattern, as well as the use of interfaces, such as the one that defines the contract necessary for the interaction with the file system. At the moment, the chosen filesystem for storing the score files is local, however, it would be easy to switch to a cloud solution if an implementation of this defined interface is developed.

3.3 Use Cases

Figure 3.1 shows the coarse-grained use case diagram for this project. There will not be a detailed description of each one, specifying preconditions, flow, post-conditions, etc. We will simply clarify some general preconditions:

- With the exception of the "Log in", "Create account" and "Recover password" use cases, all of them have as precondition that the actor must be authenticated in the system, i.e. he/she must have obtained the output of having executed the "Log in" use case.
- Likewise, to execute the use cases "Log in" and "Recover password", it is necessary to have obtained the output of having executed "Create account" use case.
- Additionally, it should be clarified that in order to execute the use case "Record score performance" the user must have his/her keyboard connected.

In section 7.1, when the screens of the application are explained, the flow of the use cases will also be clarified.

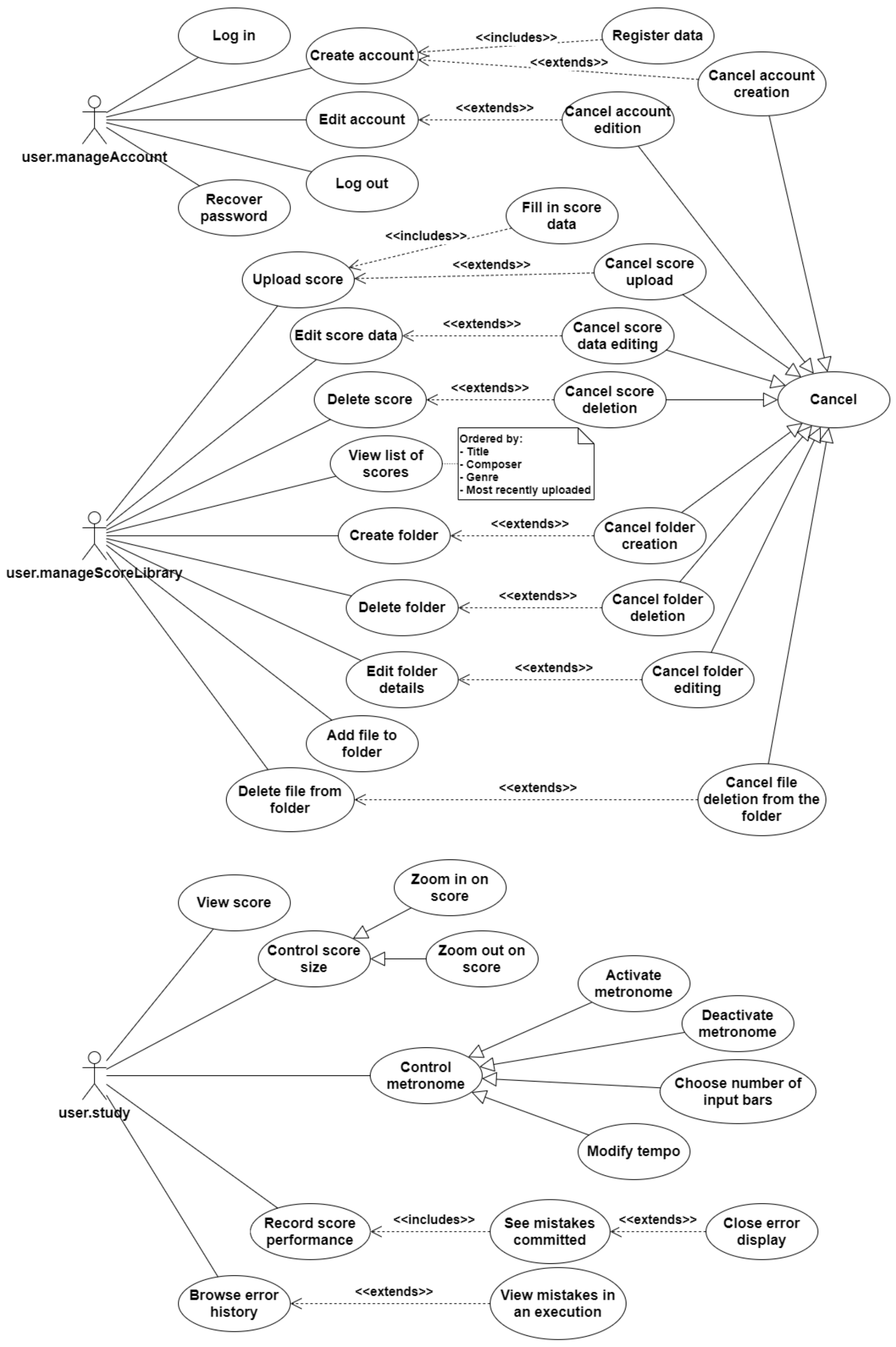


Figure 3.1: Use Case Diagram

Chapter 4

Design

THIS chapter explains the architecture and data model designed for the application.

4.1 Architecture

For the design of the architecture, the MVC (Model-View-Controller) pattern has been applied as it provides a standard web development framework for creating scalable and extensible projects. Its goal is to separate the data and business logic of an application from its representation and the module in charge of handling events and communications. In order to achieve this aim, it divides an application into three main logical components:

- Model: It is the representation of the information with which the system works. It is in charge of managing all the accesses to that information, also controlling the access privileges described in the application specifications (business logic). It sends to the "view" the piece of information that is requested at each moment. Requests for access or manipulation of information reach the "model" through the "controller".

- View: It presents the "model" (information and business logic) in a format suitable for interaction (e.g. with a user interface), and therefore it requires from that "model" the information to be represented as output.

- Controller: It responds to events and invokes requests to the "model" when a demand is made on the information. It acts as an interface between the "model" and "view" components to process all business logic and incoming requests, manipulate the data using the "model" component and interact with the "view" to render the final result.

A basic diagram of what would be the architecture of the application of this project is shown in figure 4.1. The "Backend" component stores the model and the controller while the "Web Application" component would provide the view.

In order to communicate between the backend and the frontend, a REST style architecture has been chosen so that both parts can be independent from each other, thus clearly separating responsibilities. The standard communication protocol used by keyboards is MIDI, so it will be necessary to have a frontend handler that, with the help of the Web MIDI API, will manage the messaging. Moreover, PostgreSQL is selected as the DataBase Management System (DBMS). During the design it is also decided that five controllers will be needed at the backend to maintain modularity, security and efficiency.

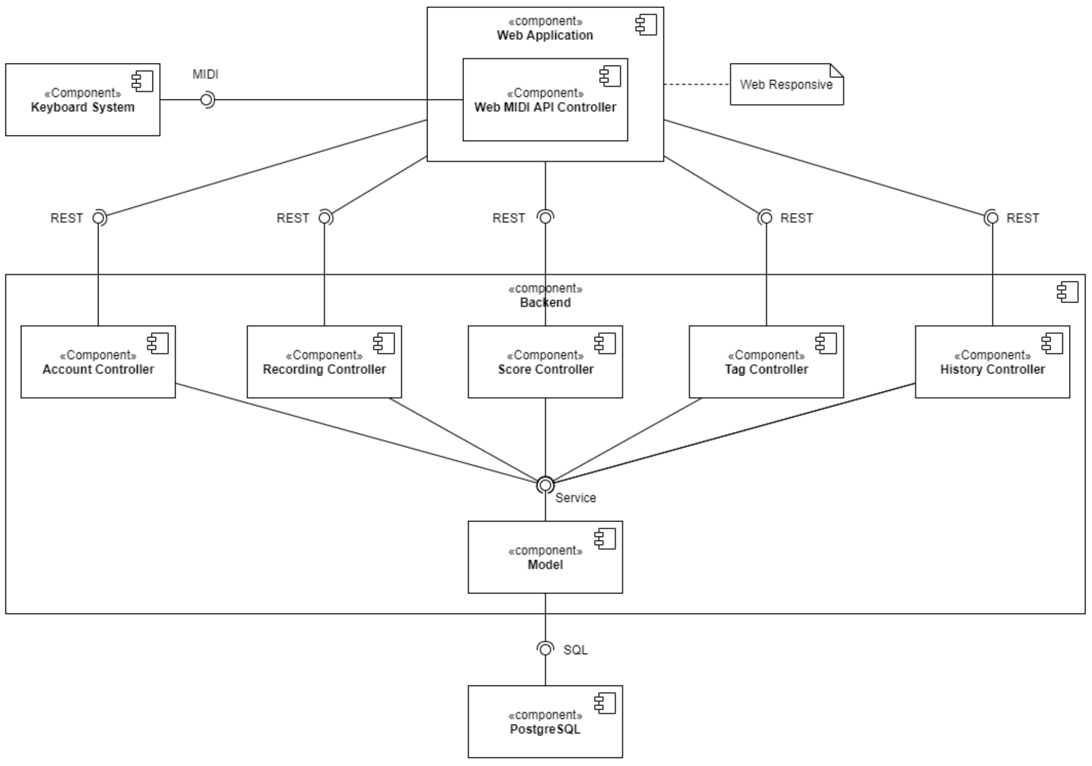

«component» Web Application
«Component» Web MIDI API Controller
Web Responsive
«Component» Keyboard System
MIDI
REST
REST
REST
REST
REST
«component» Backend
«Component» Account Controller
«Component» Recording Controller
«Component» Score Controller
«Component» Tag Controller
«Component» History Controller
Service
«component» Model
SQL
«component» PostgreSQL

Figure 4.1: Architecture Diagram

4.2 Data Modelling

Before starting to explain the data modelling, it is convenient to clarify that among the requirements of the application, some of them allow the user to work with folders. In order to represent this situation in the database, it has been chosen to use the concept of “tags”, so that a tag represents a folder and it can establish relationships with the scores, simulating the action of “saving a score in a folder”. Besides, only one level of folders will be allowed, thus discarding nested folders.

4.2.1 Entity Relationship Modelling

Figure 4.2 provides an entity-relationship diagram to explain the data the application works with.

First, four entities are established: “Fugalogo” will store all the information related to the users’ accounts, “Score” will save the scores data, “Tag” will be the representation for the folders and the “Recording History” will handle the users’ performances. The names chosen for the attributes are self-explanatory, but several details should be clarified. The “reset_token” and “expiration_reset_token” attributes in the “Fugalogo” entity are required for password recovery. The “music_xml_path” of “Score” stores the location where the original score file is kept. The ”music_xml_path” of “Recording History” holds the path to the corrected file of the user’s performance. Moreover, it is important to note that surrogate identifiers are used to achieve a simpler and more efficient implementation. The only uniqueness constraint defined is for the user’s email.

Afterwards, the required relationships between the entities are established so that reality can be portrayed. A Fugalogo can have several scores, but a score can only belong to one Fugalogo. A recording in the “Recording History” can only arise from one score, however, a score can have several recordings. In addition, the Fugalogo can have several tags that only belong to him/her. Furthermore, the same tag can be associated with several scores and a single score can be attached to multiple tags.

Once the entities and relationships are established it is also desirable to specify how the entity-relationship diagram would be designed if no surrogate identifiers had been established. The primary key of the “Fugalogo” entity would be its email. The “Score” and “Tag” entities would be weakly dependent on “Fugalogo”. “Score” would have a numeric discriminant and “Tag” could use the “name” attribute (as a discriminant) since duplicates are not allowed within tags belonging to the same user. Finally, the “Recording History” entity would be weakly dependent on “Score” and would also need a discriminant.

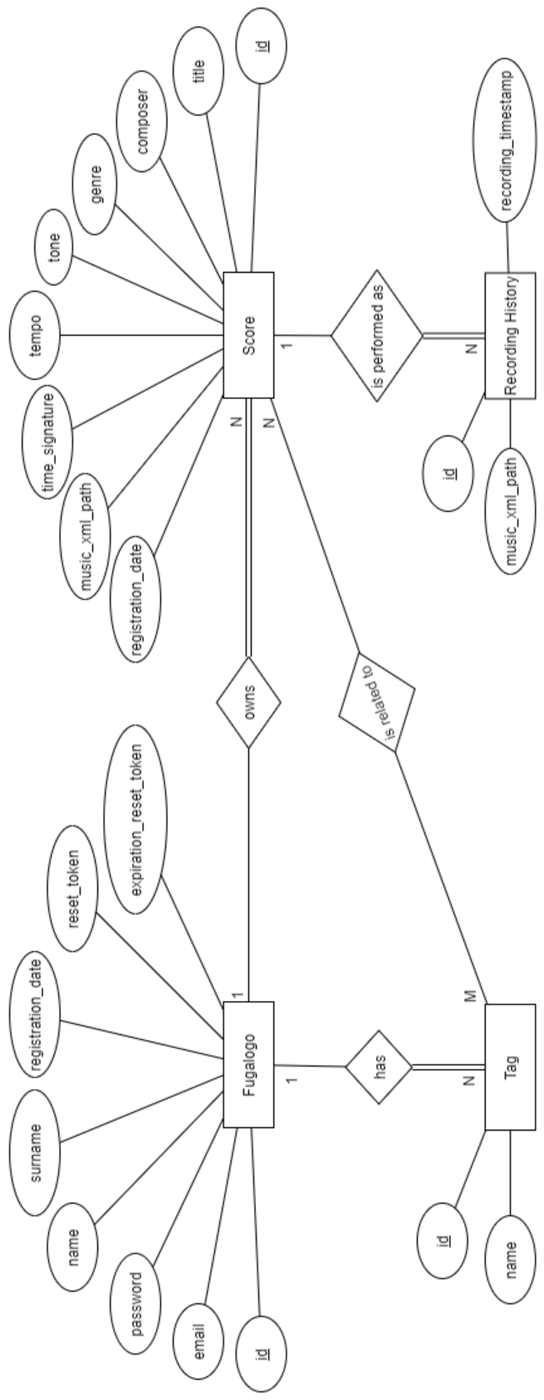


Figure 4.2: Entity Relationship Diagram

4.2.2 ODL Schema

It is decided that the language used for the backend development will be Java, an object-oriented programming language. For this reason, table 4.1, with the definition of an ODL (Object Definition Language) schema, is added. This table shows the established mapping between the objects and the model defined for the database. Likewise, only the relationships that were necessary from the object-oriented side have been added, which have been chosen taking into account the navigation, the flow, of the program.

Table 4.1: ODL Schema

Entity	**Schema**

Fugalogo	class Fugalogo {
	    attribute Long id;
	    attribute String email;
	    attribute String password;
	    attribute String name;
	    attribute String surname;
	    attribute Timestamp registrationDate;
	    attribute String resetToken;
	    attribute Timestamp expirationResetToken;
	}

Score	class Score {
	    attribute Long id;
	    attribute String title;
	    attribute String composer;
	    attribute enum Genre {Blues, Classical, Country...} genre;
	    attribute String tone;
	    attribute String tempo;
	    attribute String timeSignature;
	    attribute String musicXmlPath;
	    attribute Timestamp registrationDate;
	
	    relationship Fugalogo ownerFugalogo;
	    relationship set<Tag> tags
	            inverse Tag::scores;
	}

Tag	class Tag {
	    attribute Long id;
	    attribute String name;
	
	    relationship Fugalogo ownerFugalogo;
	    relationship set<Score> scores
	            inverse Score::tags;
	}

Recording History	Recording History {
	    attribute Long id;
	    attribute String musicXmlPath;
	    attribute Timestamp recordingTimestamp;
	
	    relationship Score originalScore;
	}

Chapter 5

Technological Fundaments

This chapter describes the different technologies chosen during the analysis and design phases that have been used to develop the project.

5.1 Structure

Already in the previous chapter, the architecture diagram of the application was presented in Figure 4.1. Since this chapter details the technologies used in each part of the architecture, below are summarised the rationale for each section:

- **Backend**
  It is the data access layer and the one that holds the business logic. It is responsible for responding to requests made by the frontend, processing the received data and returning appropriate responses. It is the only one that can access the stored data, for instance, in a database, which provides persistence for the data.

- **Frontend**
  This is the presentation layer. It matches with the web application in the diagram. It is the part of the software that interacts with users. It is responsible for collecting input data from the user and adjusting it to the specifications required by the backend so that it can be processed. Finally, it returns the response received from the backend in a way that is understandable to the user.

- **REST**
  The connection between the frontend and the backend is a type of interface. REST (Representational State Transfer) is a style of software architecture based on the HTTP protocol that serves to establish such a relationship. The information is exchanged via HTTP in one of the following formats: JSON (JavaScript Object Notation), HTML, XLT, Python, PHP or plain text. Some of the principles it follows are:

- Statelessness: State is stored and maintained on the client and not on the server. That is, requests must provide all the necessary information to be performed on a server that does not maintain state, so it does not keep context between calls for the same client.
- Uniform interface between elements: so that information is transferred in a standardised way. To this end: the requested resources must be identifiable and independent of the representations sent to the client; the client must be able to manipulate the resources through the representation it receives, as it contains sufficient information to allow this; the self-describing messages sent to the client must contain the necessary information to describe how it should process it; among others.

5.2 Backend

To develop the backend of this project, Java has been selected as the programming language and PostgreSQL as the DataBase Management System.

5.2.1 Java

It is an object-oriented and class-based programming language. When Java is compiled, it is not compiled into platform specific machine, rather into platform-independent byte code. So, Java is cross-platform, capable of running on most operating systems and devices, with a single code base [18]. This is achieved thanks to a virtual machine that exists on each system that is capable of running Java and bridging the programming language and the device.

Furthermore, Java is an interpreted language whose byte code is translated on the fly to native machine instructions and is not stored anywhere which makes the development process more rapid and analytical. Moreover, Java attempts to eliminate error-prone situations by placing the main emphasis on compile-time and run-time error checking. Apart from that, with Java's secure feature, it enables to develop virus-free and tamper-free systems. Authentication techniques are based on public-key encryption. In addition, with Java's multithreaded feature, it is possible to write programs that can perform many tasks simultaneously.

It is not only the above features that have made it the programming language of choice for backend development. The wide library of packages it offers and the programmer's previous experience with this language have been decisive for its selection.

- **Spring**
  Spring is an open source application development framework and inversion control container for the Java platform which helps to solve many technical related difficulties [19]. A key element of Spring is application-level infrastructure support: Spring focuses

on the hidden, common and more standard part of applications so that the developer can focus on application-level business logic, without unnecessary ties to specific deployment environments. The modules of this framework used in this project are described below:

- Spring Web: This module is responsible for integrating web applications with the Spring framework. It uses servlet listeners and a web-oriented application context. It also provides a web-oriented integration feature and a multi-part file upload functionality.
- Spring Security: Module to provide authentication, authorization, SSO and other security features for web applications.
- Spring Data JPA: Spring Data JPA facilitates the implementation of JPA-based repositories. Implementing a data access layer involves writing too much repetitive code to execute simple queries, as well as to perform paging and auditing. Spring Data JPA aims to significantly improve the implementation of data access layers by reducing the effort to the amount that is actually needed. The developer writes the interfaces to their repository, including custom lookup methods, and Spring will provide the implementation automatically.

• **Hibernate**
Hibernate is an Object-Relational Mapping (ORM) tool for the Java platform that facilitates the mapping of attributes between a traditional relational database and the object model of an application, by means of declarative files (XML) or annotations in the entity beans that allow these relationships to be established [20]. It streamlines the relationship between the application and the SQL database, in a way that optimises the workflow and avoids repetitive code.

It is worth noting that unlike Spring, Hibernate works largely as a bridge to connect two different data models or to generate one from code, while Spring behaves mostly as a large library full of tools for object-oriented programming.

• **Maven**
It is a software tool for managing and building Java projects. It uses a Project Object Model (POM) to describe the software project to be built, its dependencies on other modules and external components, and the order in which the elements are built. It comes with predefined objectives to perform certain clearly defined tasks, such as code compilation and packaging. [21]

Thus, Maven allows the developer to automate the handling of creating the original folder format, performing the assortment and testing, and packaging and deploying the final output. It reduces the considerable number of steps in the basic process and makes it a one-step process to make a build.

5.2.2 DBMS

A DataBase Management System (DBMS) is a set of non-visible programs that administer and manage the information contained in a database [22]. DBMSs manage all access to the database as their purpose is to serve as an interface between the database and the applications, allowing information to be stored, data to be modified and knowledge assets to be accessed.

In turn, the DBMS can be understood as a collection of interrelated data, structured and organised within the ecosystem formed by that set of programs that access them and facilitate their management.

It should be noted that access to data is independent of the programs that manage them. Therefore, some of its characteristics are: independence, minimum redundancy, consistency of information (concurrency control), abstraction of information from its physical storage, as well as secure access and the adoption of the necessary measures to guarantee data integrity.

- **PostgreSQL**
  It is an open source object-oriented relational DataBase Management System [23]. It contains several advanced data types and robust feature sets that increase the extensibility, reliability and data integrity of the software. An interesting feature of PostgreSQL is the Multi-Version Concurrency Control (MVCC). It adds an image of the database state to each transaction. This allows for eventually consistent transactions, offering great performance advantages.

  The main reason for its selection as a DBMS is the developer's previous experience working with it.

5.3 Frontend

For frontend development, in this case where a web application is being developed, it is necessary to work with three technologies [24]:

- HTML: It refers to the markup language for the creation of web pages. It is a standard that serves as a reference for software that connects to the development of web pages in its different versions. It establishes a basic structure and code for defining the content of a web page using tags. This language is highly adaptable, logically structured and easy to interpret by both humans and machines.

- CSS: It is a graphic design language for defining and creating the presentation of a structured document written in a markup language. It is widely used to establish the visual layout of web documents and user interfaces written in HTML.

- JavaScript: It is an interpreted, object-oriented, prototype-based, imperative, weakly typed, dynamic programming language. JavaScript (JS) is the only programming language that works natively in browsers (without the need for compilation). It is therefore used as a complement to HTML and CSS to create dynamic websites.

5.3.1 Vue

It is a progressive JavaScript framework for creating user interfaces that makes it easy to work with HTML, CSS and JS [25]. It is modularised in different separate libraries that allow adding functionality as needed. The most important libraries used in the project are described in the following:

- **Vue Router**
  It is the official Vue client-side routing library that provides the necessary tools to map the components of an application to different browser URL paths. [26]

- **Vuetify**
  It is a complete UI library whose goal is to provide developers with the tools they need to build rich and engaging user experiences. Moreover, Vuetify takes a mobile first approach to design, that is, it focuses on the design for mobile phones and then approaches it to other devices. [27]

- **Axios**
  It is a promise-based HTTP client for JavaScript that allows sending requests to REST endpoints and performing CRUD operations in a very simple way. [28]

- **Vuelidate**
  It offers a simple, lightweight model-based validation for Vue. Validation rules are added to a validation object where a given component is defined. This enables easy checking of form entries, for instance. [29]

- **Vue-i18n**
  It is an internationalisation plugin. Thanks to it, the developed application in this project is available in Spanish and in English. Besides, it will be very easy to translate it into more languages due to the use of this complement. [30]

It is worth mentioning that Node.js is used as the JavaScript runtime environment and that its package management system, NPM, is used to install all these libraries.

5.3.2 MIDI

MIDI (Musical Instrument Digital Interface) [31] is a protocol that allows electronic musical instruments, computers and other devices to communicate with each other. This protocol includes an interface, a language in which MIDI data is transmitted and the connections necessary to communicate between hardware elements.

It should be clarified that MIDI does not transmit audio signals but a set of instructions containing the necessary data. These sets are called events. Some of the instructions that these events store are: Key ON and OFF, when the key is pressed and when it is released; pitch, which note is played; or the speed, velocity and force with which the key is pressed.

It is also important to clarify that, for this project, a USB Type A to Type B cable is used to establish the MIDI connection between the keyboard and the computer. Nowadays, most of the keyboards on the market allow this type of connection and it is the easiest one to establish for the user. Therefore, this project report will not go into the details of a connection established with a 5-pin DIN cable or the sequencers that would be necessary to use it.

- **Web MIDI API**
  The Web MIDI API [32] uses this protocol and allows to have a MIDI instrument enabled, such as a keyboard, connect it to the computer and get the information sent from the keyboard to the browser.

  Therefore, this specification that defines an API supporting the MIDI protocol is essential in the development of this project. It should be recalled that one of the key points is to collect the MIDI information that the user transmits from the keyboard through the browser in order to send it to the backend.

  One of the main shortcomings of the Web MIDI API is cross-browser compatibility. It is not supported in: Internet Explorer, Edge 12 to 18, Firefox, Chrome 4 to 42, Safari, Opera 10 to 29, Safari on iOS, Opera Mini, Android Browser 2.1 to 4.4.4, Opera Mobile 12 and 12.1, Firefox for Android, QQ Browser and KaiOS Browser. However, according to calculations provided by the website caniuse.com [33], it would be usable by 74.51% of global users.

  Given this weakness, solutions have been sought to provide compatibility with more browsers. The most robust is the JZZ.js library, which "enables Web MIDI API in Node.js and those browsers that don't support it" [34]. It has not been used during the implementation of the project but it will be registered as a future enhancement.

5.3.3 OSMD and VexFlow

As specified in the risk analysis (section 2.2), the selected format for working with the scores is MusicXML. The users will upload their score in this format, but they must be able to visualise it as they would if they had the score printed (that is, they cannot be shown the XML code of the score).

Luckily, there is an open source online music notation rendering API that is written entirely in JavaScript and runs directly in the browser called VexFlow [16]. What VexFlow does is, it takes all the information it needs to render a score in JavaScript and creates an output in either HTML5 Canvas or SVG of what the score would look like to display it to the user in the browser.

Now, the JavaScript representation of music that VexFlow understands is not what a MusicXML file looks like. This is where Open Sheet Music Display (OSMD) [15] plays a key role. They present themselves as "The first Open Source JavaScript engine for MusicXML using VexFlow." and that is because, what this library offers is the link between MusicXML and VexFlow. Therefore, thanks to OSMD, the score in MusicXML format is converted to the format that VexFlow needs to be able to represent the music sheet in the browser in a way that users are able to understand.

5.4 Working Tools

The working tools used to develop the project will be described below.

5.4.1 Project Development Tools

- **GitLab**
  It is the selected version control web service. It is based on the Git version control software [35]. Although the project has been developed individually and therefore, it has not been possible to exploit the collaborative potential offered by this tool, its use was essential in order to develop a comfortable workflow and maintain a project version history.

- **Visual Studio Code**
  It is the source code editor selected to develop the project. In addition to all the functionalities that it offers, such as integration with the version control software Git, countless help packages during programming, customisation of the theme or keyboard shortcuts, etc., its lightness compared to all the power offered has been decisive in its choice.

5.4.2 Extra Tools

- **Draw.io**: It is a free diagram creation and editing tool with models for different types of diagrams such as UML diagrams. It is available to work online, although it also has a desktop version, and diagrams can be saved in Google Drive, OneDrive or locally. It also allows diagrams to be exported in different formats such as PDF, PNG or JPG. This will be the tool used for creating the necessary diagrams that are shown throughout the report (for example, the architecture diagram from figure 4.1 or the Entity Relationship diagram from figure 4.2).

- **LaTeX**: It is a text composition system, oriented to the creation of written documents with a high typographic quality. LaTeX facilitates the creation of documents in a consistent way by separating content from presentation. It is the tool used for the preparation of this report.

Chapter 6

Theoretical Fundaments

This chapter aims at explaining the fundamental theoretical concepts that are required to understand the development of the application.

6.1 Musical Concepts

Firstly, it is necessary to clarify some music theory notions in order to understand the challenges and solutions that arise in the application. As the focus will be on detecting rhythm and pitch errors, the first two parts of this section will be devoted to defining some of the terms needed to comprehend how corrections are made. Additionally, the last part will explain how a piano score is organised to be able to understand how it is described by MusicXML.

6.1.1 Rhythm

A series of concepts related to rhythm are defined below. Each explained concept builds on its predecessor to reach a satisfactory definition.

- **Pulse**
  It is the basic unit used to measure time in a musical piece. This is the beat of the piece. It is measured in the same way as the heartbeat: beats per minute (BPM). For example, 60 BPM means that there are 60 pulses per minute.

- **Metronome**
  A device that emits a sound with each pulse. It signals the pulse to the player so that he/she knows the speed to perform the music piece.

- **Tempo**
  It is a quantitative or qualitative description for the speed of the pulse. It can be indicated by a number or by Italian words that cover a range. For example, *Andante* ranges from approximately 76 to 108 BPM, but there is no standard that sets the limits.

- **Patterns**
  When making music, patterns are made with these pulses by accentuating the first of every two, the first of every three or any other combination. Example: **1**, 2 or **1**, 2, 3.

- **Measure**
  It is each block of a repeating pattern. This is the way of organising the music. Example: **[** 1, 2, 3 **][** 1, 2, 3 **]**.

- **Time Signature**
  It describes the repeating pattern. It is composed of two numbers, the first one indicates how many pulses there are in each measure and the second one shows how the pulse is represented (which musical figure depicts it).
  Figure 6.1 shows the mapping between the rhythmic figures and the number that represents them. Therefore, if the time signature is 3/4, it means that in each measure there are 3 beats and that the quarter note is the figure that represents each pulse. So, the quarter note will be the reference for the other figures. Following the 6.1 tree, if a quarter note is worth one pulse (let the pulse last 2 seconds), an eighth note is worth half a pulse (it would last 1 second).

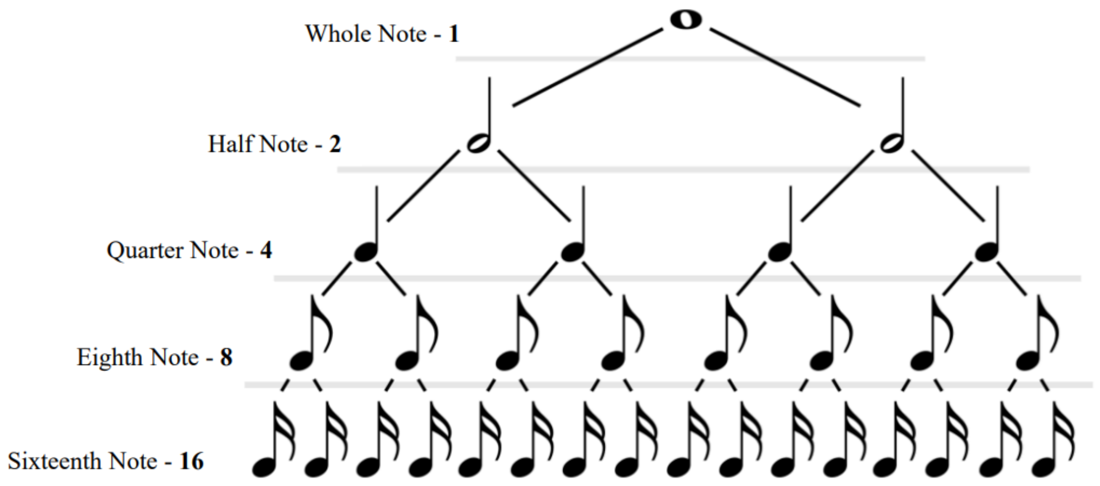


Figure 6.1: Figures Tree

  Notice that, for instance, a quarter note is made up of two eighth notes, that is, an eighth note is half the duration of a quarter note.

- **Tie**
  Ties join two notes of the same pitch (frequency) that are placed consecutively. Their durations are added together so that the player finally plays only one sound with a

duration equal to the sum of the durations of the two tied notes. For instance, if a quarter note lasts 2 seconds and there are two quarter notes tied together, the musician will only play one note of duration 4 seconds.

Concepts review

Figure 6.2 shows the concepts described above represented in a score. Starting with the time signature, it says that it is a 4/4, therefore, there will be 4 pulses in a measure and the representation of the pulse will be a quarter note. Each measure is marked by vertical bars. In the first measure the figures that appear are all whole note figures. Looking at the figure 6.1, 1 whole note is equivalent to 4 quarter notes, so there are four quarter notes in this measure (as indicated by the time signature).

The tempo is 150 BPM, therefore, as the pulse is represented by a quarter note, this means that 150 quarter notes enter in 1 minute. Thus, a quarter note lasts $60 \div 150 = 0.4$ seconds. Since one measure has 4 quarter notes, one measure lasts $0.4 * 4 = 1.6$ seconds.

Finally, all the arcs represent ties. Considering that what the tie does is to add up the duration of the figures it joins, for the first three notes of the first five lines, the total duration will be $1.6 + 1.6 = 3.2$ seconds (two measures).

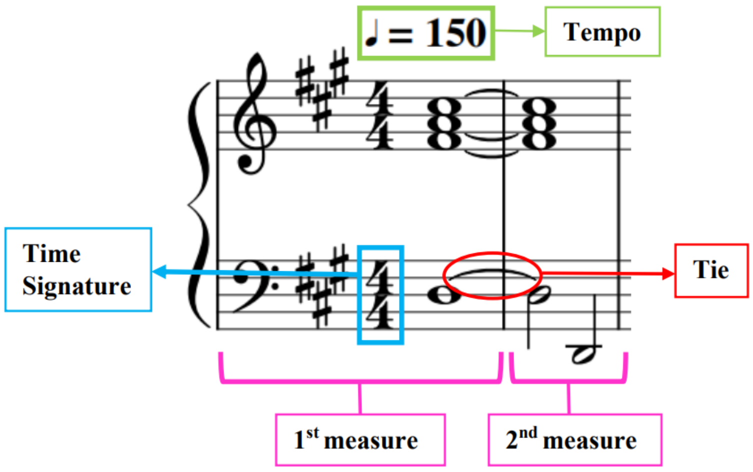


Figure 6.2: Rhythm Concepts

6.1.2 Notes

This section explains concepts related to notes. A note is a representation of the pitch and relative duration of a sound. In this section the focus will be on pitch, which from a physical point of view is known as frequency.

Scales

In occidental music, the 12-tone equal temperament is usually used [36]. What this tuning system does is:

1. Takes a frequency (f_0).
2. Doubles that frequency ($f_1 = 2 \cdot f_0$).
3. Refers to the distance between f_0 and f_1 as octave.
4. Divides that interval into 12 equal parts called semitones.

Thus, the semitone is the distance between a note and its consecutive (or predecessor).

Hence, occidental music works with 12 notes. However, not all notes are usually used to compose a song. Instead, it works with what is known as a scale, which is just a selection of notes. This selection is not random as it is important to take into account the distance between the notes you choose (how many semitones there are between one note and another in the scale).

The most familiar and widely used scale is called the major scale. What are the distances between notes in a major scale? In a major scale there are 7 notes and the distance between one note and the next in semitones is: 2-2-1-2-2-2-1. Thus, starting from the note Do and counting the distances in a major scale, the 7 well-known musical notes are obtained: Do, Re, Mi, Fa, Sol, La, Si.

Which are the other 5 missing notes? They have no proper name to identify them. They need surnames in order to be identified. The surnames can be: #, indicating that 1 semitone is to be added; or *b*, indicating that 1 semitone is to be subtracted.

Where are these 5 notes? Knowing the names of the selected notes and that they form a major scale, the distances can be checked to find out:

Do - 2 - Re - 2 - Mi - 1 - Fa - 2 - Sol - 2 - La - 2 - Si - 1 - Do.

Therefore, where there is a distance of 2 semitones, a note with a surname is missing. Now, these missing notes can be named in two ways: with the preceding note and a # or with the following note and a *b*. Therefore, the whole list of the 12 notes would be:

Do, Do#/Re*b*, Re, Re#/Mi*b*, Mi, Fa, Fa#/Sol*b*, Sol, Sol#/La*b*, La, La#/Si*b*, Si.

These 12 notes are like an infinite string because they are always repeated in the same order. When they start repeating, the frequency between the first 12 notes and the next 12 notes is twice the frequency of the corresponding one for the former repetition. This is called an octave. A piano has 7 full octaves and a few more notes. In fact, the major scale can be formed from any note. All it takes is to choose a starting note, count the appropriate distances and obtain the 7 notes that compose it. To label this scale, it is only necessary to take the name of the starting note and make it clear that it is major. For example:

- Do major: Do, Re, Mi, Fa, Sol, La, Si.
- Re major: Re, Mi, Fa#, Sol, La, Si, Do#.

Another essential scale in occidental music is the minor scale. The distance between their notes are: 2-1-2-2-1-2-2. They can also be built from any starting note and the naming is the same as for major scales but with the word minor. For example:

- La minor: La, Si, Do, Re, Mi, Fa, Sol.
- Re minor: Re, Mi, Fa, Sol, La, Si*b*, Do.

An important detail shown in the examples is that the Do major scale and the La minor scale have the same notes although in a different order. What happens is that each major scale has a relative minor with which it shares the same notes but in a different order. To find the relative minor of a major scale, it is enough to take the note from which it starts and then subtract 3 semitones from it. For example: Do major, if 3 semitones are subtracted from the note Do, the note La is obtained, therefore, the relative minor is La minor.

Tonalities

Every piece of tonal music rotates around a main musical scale (or more if it modulates). Depending on the name of the principal scale, it is said that a composition is in that key. Thus, a composition where the used notes move around the scale of Do major, is said to be in Do major (referring to its tonality).

However, if the notes for a major scale and its relative minor are the same, how does one distinguish whether the key of the piece is Do major and not La minor? It all depends on the use given to the notes, because within each key they have a different hierarchy. So it depends on how the composer has worked with the notes. Getting to know this requires a much deeper analysis that is beyond the scope of this work .

Why is it important to know the tonality? Because it indicates how many accidentals (surnames) will be in the piece (# or *b*). This allows the performer to anticipate to some extent what he or she will find in the score.

Circle of Fifths

Starting from the note Do, counting 7 ascending semitones, the note Sol is obtained (this distance is called a perfect fifth). Counting another 7 semitones, the note Re is reached, and so on, until the note Do is achieved again. In this way, the 12 musical notes can be arranged in such a manner that they are all within 7 semitones of each other. This ordering of the musical notes is called the circle of fifths.

The circle of fifths is a very powerful tool in music. It is a kind of map that helps to understand music in a simpler way. A representation of this circle is shown in figure 6.3 below.

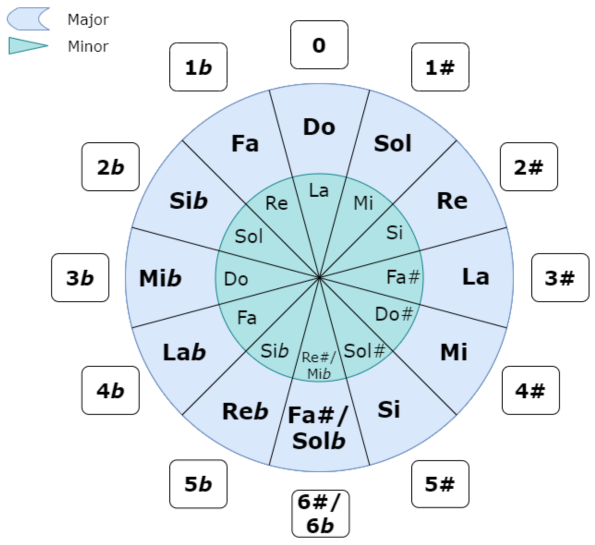


Figure 6.3: Circle of Fifths

Knowing that the tonality of Do major has the Do major scale as its axis, and that this scale is made up of the notes Do-Re-Mi-Fa-Sol-La-Si, it can be deduced that the tonality of Do major has no accidentals (# or *b*). Thus, the notes to be played in the song will probably not have surnames.

In the circle of fifths, starting from Do, which, as explained in the previous paragraph, has no accidentals: if moving clockwise, for each new note a sharp (#) is added (e.g. Sol major has one # and Re major has two); if moving counterclockwise a flat (*b*) is added (e.g. Fa major has one *b* and Si*b* major has two).

To sum up, by being aware of the tonality of the score, the musician can already intuit what he/she is going to encounter in it.

6.1.3 Piano Score Structure

All the above concepts are used to describe a musical composition. The figures representing the duration (figure 6.1), when placed on the staff (five parallel lines) also indicate the pitch according to their position. Therefore, the figures (duration) on a staff (pitch) become notes (duration + pitch).

On the other hand, if the notes are drawn one on top of the other, vertically, it means that they are played at the same time (it is a chord). If they are shown next to each other, it indicates that they are played on different pulses.

However, the piano is played with two hands. To indicate what is played with one or the other hand, two staves are used. The top one is for the right hand and the bottom one for the left hand. Even if the hands are separated, the notes that fall on the same vertical are played at the same time.

As explained in section 6.1.1, music is organised in measures. Graphically, these are represented by a vertical line through the staves which is called a bar. Figure 6.4 attempts to clarify all these concepts.

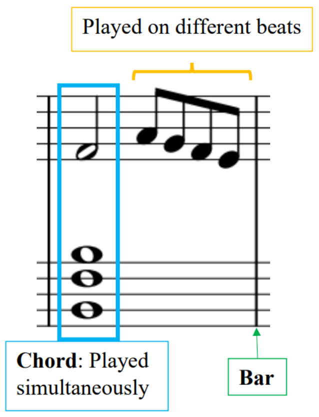


Figure 6.4: Measure of a Piano Score.

6.2 MusicXML Notation

This notation is the XML standard for representing music. It has a compressed format using a zip-based XML format with the extension *.mxl* [37]. During this section it is explained in a question and answer format how MusicXML depicts the score information.

1. **How does MusicXML represent the duration of notes?**

   What MusicXML does is to establish a reference system at the beginning of each song for itself, which can only be translated into seconds thanks to the time signature and tempo. With the tag *<divisions>* it indicates the value of a quarter note in the MusicXML file (for instance, *<divisions>24</divisions>* says that a quarter note lasts 24). Then, it indicates the duration of each note according to this reference with the *<duration>* tag (for example, an eighth note would have *<duration>12<duration>* because, as can be seen in figure 6.1, an eighth note lasts half the length of a quarter note). The *<divisions>* index may change throughout the piece, such that all notes after the shift will measure their duration by this reference.

Example

Given a time signature of 4/4 and a tempo of 150BPM, it is known that a quarter note lasts $60 \div 150 = 0.4$ seconds. The MusicXML indicates that *<divisions>24</divisions>* and there is a note with *<duration>12</duration>*. How long does the note last in seconds?

Since *<divisions>* is the representation of a quarter note and a quarter note lasts 0.4 seconds, 24 in MusicXML is 0.4 seconds. So, 12 in MusicXML will be $12 * 0.4 \div 24 = 0.2$ seconds. Therefore the duration of the note is 0.2 seconds.

However, the *<duration>* tag is not enough to represent the note. MusicXML sets the *<type>* tag (with values like: whole, half, quarter, eighth...) to indicate the necessary figure. The reason for this is that a rhythmic figure can have a dot attached to the side which adds half its duration to the actual duration. Therefore, if the quarter note is worth 24, a quarter note with a dot will have a duration of $24 + 12 = 36$.

2. **How does MusicXML organise the piano score?**

In the previous section it was explained that the music in a piano score was organised in blocks, with the notes for the right hand on one staff and for the left hand on another. However, in a MusicXML file not all notes can be written at the same time, the representation of notes is sequential.

What MusicXML does to put each note in its place is to set an imaginary cursor at the beginning of the score. It then starts typing the right hand notes (which it marks as *<staff>1</staff>* to denote that they belong to the first staff) in the order in which they appear. At the end of the bar, it places a *<backup>* label with a number indicating the duration that the cursor must move back to the beginning of the measure and starts reading the left hand (notes that it marks as *<staff>2</staff>* to denote that they belong to the second staff). When all the notes of the left hand have been written sequentially, it proceeds to describe the next measure in the same way as this one. All these notes, both right hand and left hand, belong to the same measure, and therefore MusicXML groups them together in the *<measure>* tag.

It is worth noting that, for chords (several notes in the same pulse), MusicXML prevents the cursor from moving by placing the </chord> tag on the notes involved (not on the first note of the chord it describes, but on the notes that follow it).

3. **How does MusicXML represent the pitch of notes?**

Inside the <note> tag there is also the *<pitch>* tag that defines the note with three subtags:

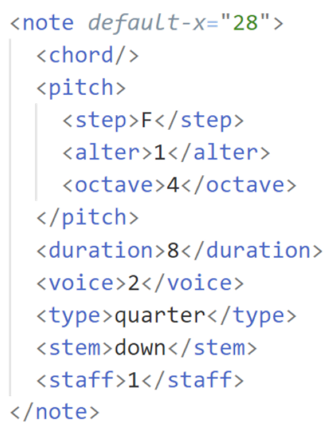
- *<step>*: Indicates the name of the note (not the surname) but with the American notation system. Its values can be A, B, C, D, E, F, G for La, Si, Do, Re Mi, Fa, Sol, respectively.
- *<alter>*: Indicates the surname of the note. If it is # (adds a semitone), it is indicated by a +1. If it is *b* (subtracts a semitone), it is indicated by a -1. If no surname is required, it can be 0 or the label can be omitted.
- *<octave>*: In the previous section, it was explained that the 12 notes were repeated in the same order only with the frequency doubled, that each of these 12-note blocks was called an octave and that a piano had 7 octaves. To find out to which frequency range a note belongs, the acoustic index is used [38]. It numbers the octaves in order from the lowest to the highest. MusicXML uses this system, specifically, the scientific acoustic index [39].

Let us take an example. The frequency 440 Hz is an La$_4$ in the physicists' system and MusicXML represents it as:

<step>A</step>
<octave>4</octave>

<alter> being 0, it would normally not appear.

Figure 6.5 shows a fragment of a MusicXML file containing some of the described tags.



```
<note default-x="28">
  <chord/>
  <pitch>
    <step>F</step>
    <alter>1</alter>
    <octave>4</octave>
  </pitch>
  <duration>8</duration>
  <voice>2</voice>
  <type>quarter</type>
  <stem>down</stem>
  <staff>1</staff>
</note>
```

Figure 6.5: MusicXML *<note>* tag.

4. How does MusicXML represent silences?

The rests indicate when not to play. They have no pitch but they do have duration. Instead of the *<pitch>* tag used for notes, silences use the *<rest/>* tag. Of course, all tags concerning duration (*<duration>*) and representation (*<type>*, *<staff>*...) are also used for rests because they have duration and must be represented on one staff or the other.

5. Other notations

- ***<key>***: This label appears at the beginning of the file to indicate the tonality. It has one or two tags inside:
  - *<fifths>*: A value of 0 indicates that there are no accidentals. If it is greater than 0, it indicates that the tone has sharps (#) and how many sharps it has. If it is less than 0, it indicates that the tone has flats (*b*) and how many flats it has.
  - *<mode>*: It is optional. It specifies whether the tonality is major or minor.
- ***<time>***: Label to indicate the time signature. Inside it has two tags:
  - *<beats>*: It indicates the number of beats per measure.
  - *<beat-type>*: It specifies the figure that represents the pulse.
- ***<tie>***: Used to represent ties. If its *type* attribute is "start" it means that the tie starts on that note. The next note will have the same tag with *type="stop"*, which indicates the end of the tie. The pitch of both notes must be the same.

Chapter 7

The Application

DURING this chapter a journey through the screens will take place in the order of action followed by the user. On each screen, the different events that may occur will be explained in depth to understand how they unfold. In section 7.1, an explanation of the whole application is given, and in section 7.2 a deeper look at the heart of the application, error detection, is provided.

7.1 Web Application

The explanation is arranged in four main blocks grouping sets of related screens. The first group deals with authentication and everything related to the user account itself. The second group discusses the score library and the scores per se. The third group covers the "study area" and the last group addresses the error history review.

1. **Authentication and user account**

   - **Sign Up**
     The first thing the user must do when accessing the application for the first time is to create an account. To do this, they will access the "Sign Up" screen. There, users must fill in a form containing the following fields: name, surname, email, password and password confirmation (see figure 7.1). Once the user completes the action, the frontend of the application performs the basic checks that the data entered are valid (for example, that the password and password confirmation match). If they are not valid, it alerts the user and if they are valid, it proceeds to send a POST with a JSON of the data to the */api/account/users* endpoint in the backend.

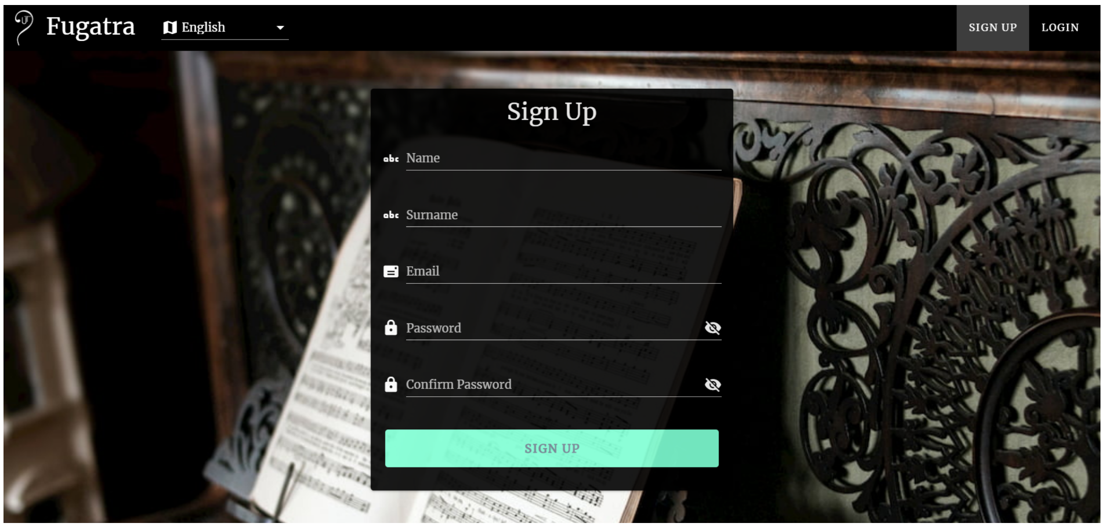


Figure 7.1: Sign Up Screen

Once the request arrives at the backend, if the JSON is well formed (otherwise an "invalid argument" exception is thrown) the user service is called. The user service checks with the user's repository that there is not already an account created for that user (that the same email is not registered). If there is already a user with that email, it throws a "user already exists" exception, otherwise it creates a new *User* entity object that persists in the database through the repository.

In response, the frontend may receive an exception, in which case the user is alerted, or a confirmation that the account has been successfully created, in that case the user is notified and redirected to the "Sign In" screen.

- **Sign In**
  To access their account, users must fill in a form with their email address and password and click on the "Login" button. The frontend will check that the form is correctly completed (for example, that the email structure is appropriate) and will proceed to send a POST with the data in JSON format to the */api/account/session* endpoint of the backend.

  Once the backend receives the request, after verifying that the user exists and that the password is correct, it generates an authentication token that the frontend will be informed of in the response it receives together with the data it needs from the user (id, name, registration date...).

  If the frontend receives the response with the authentication token, it proceeds to redirect the user to its Fugateca, otherwise it alerts the user of the error.

- **Sign Out**
  Once the user has logged in, he/she can always log out by clicking on the "Sign

Out" button, which will discard the authentication token and release all resources that may have been used during the session.

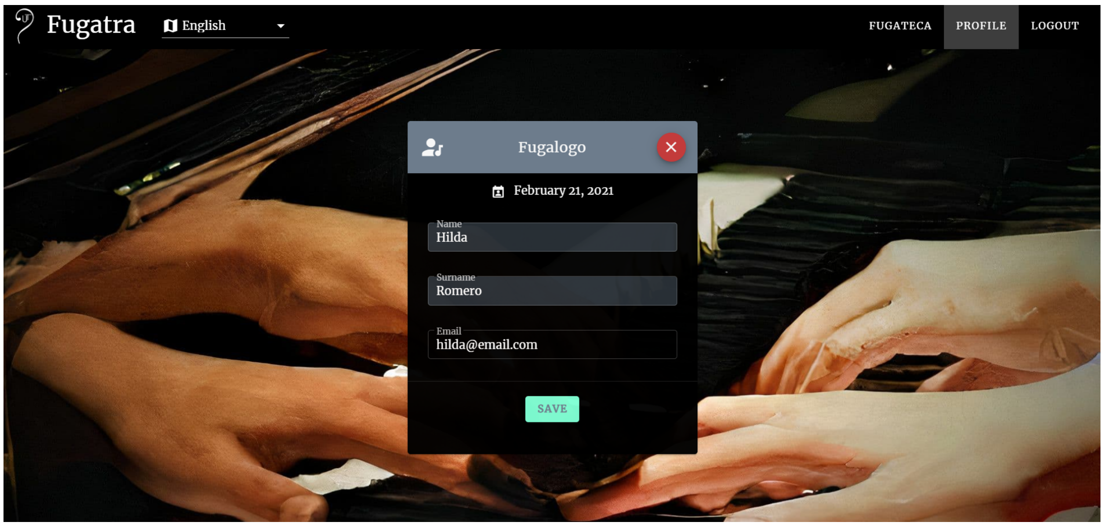
- **User Profile**

  When the user accesses his/her profile in the application, the frontend makes a GET request to the backend endpoint */api/account/users/{id}*, substituting the *{id}* with the user's real id number and also sending the authentication token.

  The backend, if the authentication token is valid, asks the service to return the user data. The service calls the repository with the user id to retrieve the user information. If a user with such an id does not exist it returns a "user not found" exception. If the user of the requested data is not the same as the user corresponding to the authentication token, it returns a "unauthorised user" exception. If everything is correct, it returns the user data.

  The frontend, on receiving the response with the user's data, shows the user his/her name, surname, e-mail address and registration date.

  The application also allows the user to edit their name and surname. Figure 7.2 shows the "User Profile" screen in edit mode.



February 21, 2021

Name: Hilda

Surname: Romero

Email: hilda@email.com

Figure 7.2: User Profile Screen Edit Mode

If the user updates their data, the frontend sends a PUT to the backend endpoint */api/account/users/{id}* with the user id, the authentication token and the new data.

The backend checks the authentication and calls the service to update the data making it persistent thanks to the repository. As before, "user note found" or "unauthorised user" exceptions can occur.

The frontend receives the new data, updates the view and notifies the user of the success of his/her action, or, if it receives an exception, alerts the user of the error.

- **Reset Password**
  Returning to the "Sign In" screen, it may be the case that the user does not remember his/her password. To solve this, the frontend offers a "Forgot password?" button. When clicking on it, the user will be asked to enter his/her e-mail address. Afterwards, the frontend makes a request to the backend endpoint */api/account/-forgot* with the user's email and the URL to which the user should be redirected to reset the password.

  The backend calls the service, which checks with the repository that the user's email is registered. It then generates a token to reset the password, which expires in 30 minutes, and stores the token and the expiry timestamp in the database. Afterwards, it creates a message with the reset URL indicated by the frontend and the generated reset token added as a parameter to this URL to send it to the user's email address. If the email has been sent correctly, the frontend notifies the user of the successful delivery. If an exception occurs, it alerts the user of the error.

  Ideally, the user should check their email address and, having received the password reset email, follow the link in the URL which will take them to a screen where they will be asked to fill in a form with the new password and confirmation of it. Once the user clicks the "Reset" button, the frontend will take the reset token from the URL and send it together with the new password to the backend */api/account/reset* endpoint. The backend will call the service that checks if the token exists and has not expired. If so, it sets the new password and persists it encrypted in the database while discarding the reset token. If everything was successful, the frontend notifies the user of the update and redirects the user to the "Sign In" screen, otherwise, it alerts the user of the occurred error.

This same dynamic in which the frontend collects all the necessary information to make a request to a backend endpoint and where the backend responds to it by making all the necessary checks before carrying out an action is maintained during any exchange between frontend and backend performed in the application. Therefore, from this point on in the report, only the actions that the user can perform in the application will be specified and only some sections that may be of interest will be detailed.

2. **Fugateca**
   In the Fugateca there are two sub-screens between which the user can toggle through a switch, a paginated one with the scores and one paged with folders.

Scores

This screen shows the scores that the user has in his/her library (see figure 7.3) . The user can choose to display the scores sorted by name, by composer, by genre or by most recent upload date. Pagination has been implemented such that the frontend tells the backend what page number it wants and how many scores are at each of the pages. Each time the user loads a new page or changes the sorting, the frontend must request the music scores from the backend.

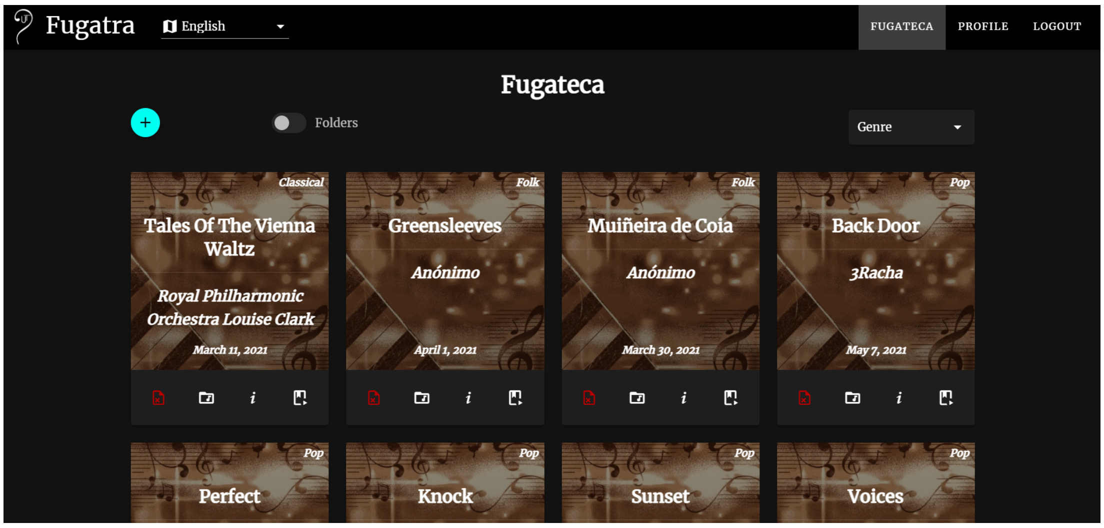


Figure 7.3: Fugateca Screen - Scores

With each score the user can perform several actions. One of them is to delete it. When the user presses the button to do so, the frontend launches a dialog warning of the action and asking for confirmation. When the backend receives this request, beyond all the verifications it performs, it also takes into account that all the error history associated with the score must be deleted as this is the way the database is designed.

Another action that the user can perform on a score is to associate it to a folder. The following section will explain how the folder system is configured. It is only necessary to clarify that from this screen the user can link a score to one or more folders, but not unlink it (although the user is shown in which folders it is already registered).

In addition, for each score the user can view its information (title, composer, genre, tempo, time signature, tonality and upload date). From this same information display panel, the user can only edit the title, composer, genre and tempo of the score since tonality and time signature are inherent to the score itself.

Finally, also from a score you can access the Fugáfono of that score, that is, the study area of the score.

- **Score uploading**
  Within the scores screen is the action to upload a new score. When clicked, a form is displayed where the user must specify the title and composer of the piece, choose the genre it belongs to (or select the "unknown" option from the drop-down menu), set the tempo and upload the score file in MusicXML format (.musicxml) or in its compressed format (.mxl). When the "Save" button is clicked, the frontend sends all the necessary information to the backend. Beyond the usual or expected verifications that it performs, three details concerning the tempo, the time signature and the tonality of the score are worth to be examined in more detail.

  Regarding tempo, in the chapter on "Theoretical Fundaments" (Chapter 6), it was explained that it could be represented by a number or by a Italian word. Therefore, pretending to establish the tempo by analyzing the MusicXML file provided by the user is risky. Moreover, it seems important that the student, perhaps also on the recommendation of his/her teacher, can select the tempo he/she expects to reach while playing the piece. Furthermore, the concept of tempo is something that a student is quickly familiar with when first introduced to the music world. Therefore, it was decided that it would be the user who would indicate the tempo for the score in the form.

  Now, as mentioned before, among the information shown to the user about a score are the time signature and the tonality. Where do they come from?

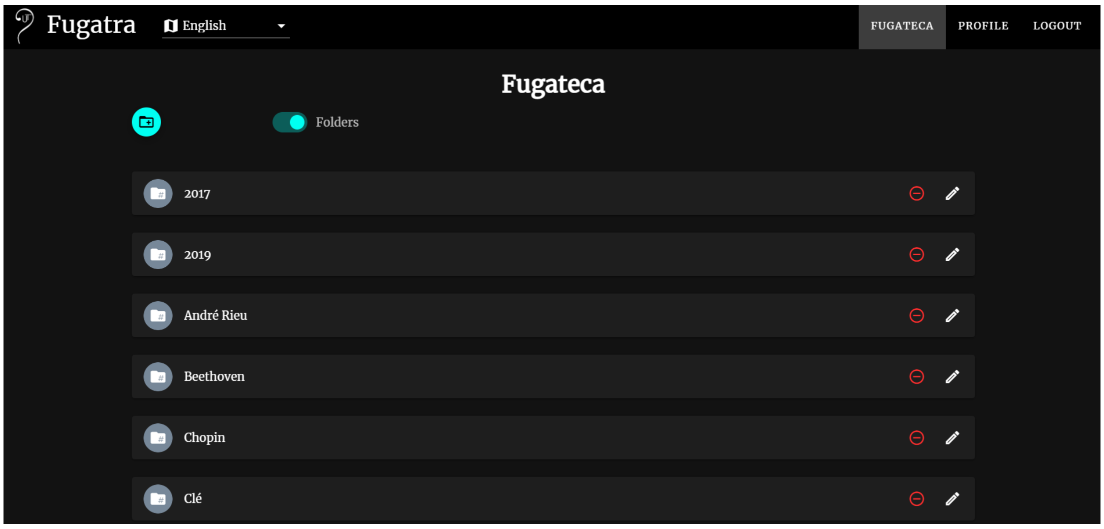
  As stated in section 6.2 of the previous chapter, MusicXML sets the *<time>* tag from which the number of beats per measure (*<beats>*) and the type of figure representing the pulse (*<beat-type>*) are obtained. Therefore, the backend parses the file provided by the user to obtain this information and store it as the time signature.

  However, MusicXML does not offer the tonality of the composition directly. What MusicXML provides is, as previously explained in section 6.2, the number of accidentals and their type (*<fifths>*) and, optionally the mode (*<mode>*).

  What the backend does to obtain the tonality from the information provided by the file is to check the circle of fifths. First, it determines whether it is an unaltered/sharp (#) tone (the number given by *<fifths>* will be greater than or equal to 0) or a flat (*b*) tone (the number given by *<fifths>* will be less than 0). By doing so, the backend knows whether it should traverse the circle of fifths clockwise or counterclockwise. It is important to remember that, in the circle of fifths, by knowing the number of accidentals, the tonality can be known. Finally, if the *<mode>* tag exists, the backend sets the tonality in major mode or minor mode according to its value. If this tag does not exist, the application stores both possible tonalities to show them to the user, who can then determine which is the real one.

Folders

When the switch is activated, the folders screen appears and all the folders created by the user are listed in alphabetical order (see figure 7.4). This screen also has pagination, which works exactly the same as with the scores.



2017
2019
André Rieu
Beethoven
Chopin
Clé

Figure 7.4: Fugateca Screen - Folders

It is important to highlight that only one level of folders can be created, that is, nested folders are not allowed. The purpose of the folders is to give the user the opportunity to organise the scores in such a way that it is easy and quick to access them. So, this view is not intended to create a representation of a complete file system.

Having clarified the above, one of the actions to perform in this screen is to create a new folder. For this, the user must enter the name with which he/she wants to register it (that must be unique among the folders belonging to him/her). The backend works with the folders using the concept of labels, seeking always the maximum efficiency. So, when a user "saves" a score in a folder, what the backend does is to associate a tag to it.

For each folder it is possible to modify its name, to delete it and to access its contents. When it is removed, the scores it contains will not be erased (and this is stated to the user), they will simply cease to be associated with that folder. When accessing its contents, the linked scores can be viewed as they would be displayed in the scores screen with all related actions available (including the ability to select the sorting criteria). The only exceptions are the action of uploading a new score, which is not available here, and the action of saving the score in another folder, which has been replaced by unlinking the score from the folder itself.

3. **Fugáfono**
The Fugáfono can only be accessed from a score, because it is the study area dedicated to that score. In fact, when the frontend creates the Fugáfono view, if it cannot get the MusicXML file that it needs to display the music sheet, it notifies the user of the error and redirects him/her to the Fugateca. So, when accessing the Fugáfono, the first thing the frontend does is to ask the backend for the MusicXML file corresponding to the score the user wants to practice. Then, thanks to the OSMD library explained in section 5.3, it renders the score so that the user can see it as the conventional score he/she knows.

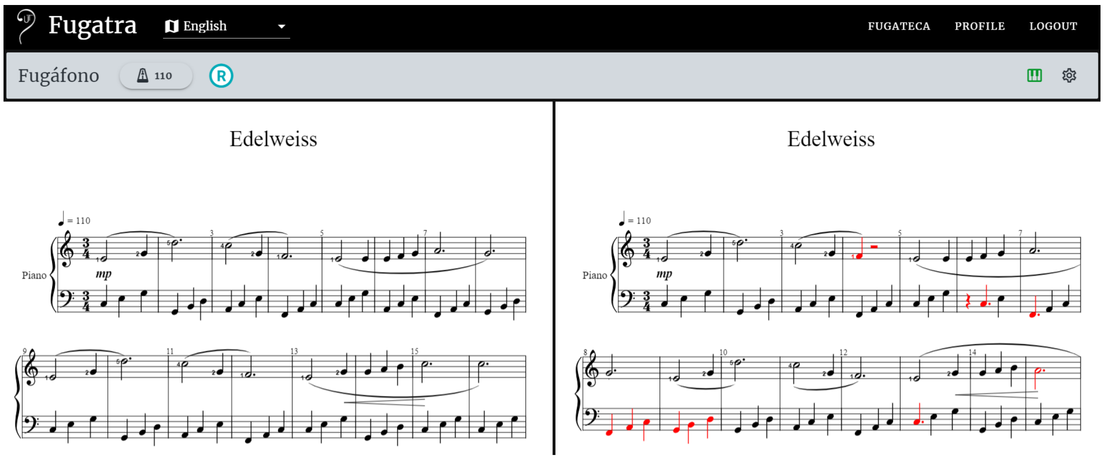


Figure 7.5: Fugáfono Screen

Beyond this, a top bar offers several actions that the user can perform on this screen. One of them is to zoom in and out the score. When the user performs this option, the frontend re-renders the score at the new size. One of the other buttons allows triggering a dialogue in which the user can set the metronome before starting to record his/her performance. The initial tempo established for the metronome is the one saved when the score was uploaded, however, it can be modified here. The user can also listen to the metronome to make sure that the tempo is correctly adjusted.

Moreover, they can choose whether they want the metronome to be active during the whole recording or only during the input bars. Another choice is how many input bars are preferred.

To simulate the beats of a metronome, a very short audio file with a click track recorded is played every set time. In order to know how often the audio has to be played in milliseconds, taking into account that the tempo is set in BPM, it is calculated as

$$t = \frac{60 * 1000}{\text{beats\_min}}, \tag{7.1}$$

where the variable beats_min is the tempo in BPM. The parameter t hence indicates the duration of a pulse.

The number of input bars indicates how many measures of the running metronome the student wants to hear before starting to record the rehearsal. They are important both for the student to feel the tempo at which the piece should be played, and for the program to calculate from where it should start correcting (because, for example, it may happen that the piece starts with a silence, and it must be checked that it is played correctly).

Another important action is to attach the keyboard to the computer. To establish a proper connection after plugging the keyboard into the computer, the user must press a button on this screen. Afterwards, the frontend, thanks to the Web MIDI API (section 5.3), will link to the device. If more than one MIDI device is connected or if no device is found, the user is alerted. If the connection was successful, he/she is also notified. As a bonus, if the keyboard gets disconnected at any time after the link has been established, the frontend detects this and alerts the user. Also, if he/she was shooting at that moment, the recording is stopped. Indeed, the main action that can be performed on this screen is to record a performance of the song in order to obtain the committed mistakes. This is the core of the project and will be explained in depth in the next section (7.2).

Finally, from this view it is also possible to access the score's error history screen.

4. **Error history**

The error history of a score can only be accessed from the score's Fugáfono. On this screen, a list of all the user's recordings is displayed, sorted and identified by timestamp, from the most recent to the oldest. Again, this list is paginated in the same way as the score and folder lists are.

One of the options offered in this window is to filter the list by a range of dates, making it easier for the user to search for a specific recording. The filtering is done in the backend in a request against the database because performing it in the frontend could be very computationally demanding given the amount of recordings there may be of a score.

Afterwards, when accessing an item in the list, a window with the original score and the corrected record of that performance is displayed (the frontend requests both files from the backend). For this purpose, the OSMD library is used again. The zoom in and zoom out functionalities are also available, and affect both displays when activated (the original score and the recording).

Finally, it should be noted that DTOs have been used throughout the development of the application, and that exceptions have been created and managed by a common handler, among other useful best practices that seek maximum security and efficiency for the application. Moreover, throughout the application the user is given the opportunity to change the language. At the moment, only English and Spanish are available, but it would be easy to add more languages since the translation is centralised and, by adjusting a few parameters and providing the JSON file with the relation between the phrases in English and in the new language, would be enough.

7.2 Core: Detect Errors

This section describes the core functionality and the main purpose of the application, the detection and correction of errors. It is accessible from the Fugáfono screen and only involves a call from the frontend to the backend.

The explanation is divided into five parts. The first part narrates the flow from the collection of the user's MIDI in the frontend to the sending of the data to the backend. The second part discusses the reception of the information in the backend and the arrangements needed before starting the actual correction. The third one explains how MIDI and MusicXML are linked in order to be able to compare the user input and the original score. The fourth one talks about the actual correction. The last one comments on how the results are displayed to the user.

7.2.1 Frontend: From user to backend

The following explains how the frontend collects the MIDI input from the user and how it is transmitted to the backend in the order in which the events occur.

1. **The input bars. When recording starts.**
   As soon as the user presses the record button, the first thing the frontend does is calculate how many metronome beats to give before starting to record what the student is

playing. Besides, this way it also indicates the tempo and gives the entry to the student so that he/she can start playing correctly.

To calculate how many beats it should give, it takes the number of input bars indicated in the metronome settings and the time signature of the score. It is important to remember that the first number of the time signature specifies how many beats there are in a measure. Therefore, all it takes to find the total number of beats before starting to record is to multiply the number of input bars by the number of beats per measure.

Once this is computed, the frontend plays the metronome that calculated number of times and starts recording the user's MIDI input. Previously the user must have connected his/her keyboard and established the connection as explained in the previous section, otherwise an error will be displayed because there is no keyboard to take the information from. It should also be noted that, if the metronome configuration has been specified to keep it active during the whole song, the metronome does not stop after the input bars have been given.

The timestamp at which the recording starts, after the input bars, is recorded in a variable to be sent later to the backend.

2. **Data capture.**
From the MIDI events that arrive at the frontend, the Key ON and OFF instructions are captured, that is, when a key is pressed and released. The type of event, whether it is ON or OFF, the frequency of the note and the timestamp of the event are stored as objects with attributes containing that information in an array according to the order in which they occur.

However, MIDI does not provide exactly the frequency of the note but some numbers that represent them. Let n be the number that MIDI gives to a note, thanks to the use of an appropriate formula [40], specifically $2^{\frac{n-69}{12}} * 440$, the frontend transforms these MIDI numbers into frequencies to store the pitch of the notes in this way.

3. **Delivery to the backend.**
To send the recording information to the backend, the frontend makes a POST to the endpoint */api/recordings/{id}*, where the *{id}* is replaced by the id of the score being rehearsed. In this message, the body carries the following information: the timestamp of the start of the recording, the array with the user's MIDI information as mentioned in point 2, and the tempo in BPM that the metronome was set to when it was recorded.

7.2.2 Backend: Reception and preparation

The backend controller receives the request from the frontend and calls the corresponding service. Let us assume that after performing all the necessary checks (that the score exists, that the user owns the score...), no exception is raised. Afterwards, the first thing the service does is to make a copy of the original score file, which will be used to store the corrections. After that, it saves the information about the recording in the database (when it was made, which is the original score, the path to the file containing the correction, ...).

The last part of the preparation stage seeks to get rid of the key "on" and key "off" labels in the items of the received array with the user MIDI data by unifying the pressing and releasing of a key into one object that stores the pitch, the note start timestamp and the note end timestamp. This will reduce the array of data about what the user played by a half because, for each note there would no longer require two objects.

In order to achieve this, it is important to realise that the same note cannot be played again on a piano if it has not been previously released. Based on this, a loop is implemented on the array of objects with the type of action ("on" or "off"), the frequency of the note and the timestamp at which the event occurs to create a new array with simplified information. In it, if the note is of type "on", it is recorded in the new array with its frequency, the timestamp of the event as the start timestamp of the note and with the end timestamp of the note set to null. If the note is of type "off" it means that previously there was an "on" event, therefore, it searches in the new array which note with the same frequency has the note end timestamp to null, because that means that this was its release event and the end timestamp must be recorded. If for some reason a key was pressed before recording started, the backend will receive an "off" event without a corresponding "on" event. In this case, a note is registered with the frequency and end timestamp of the "off" event, and the timestamp of the start of recording as start timestamp.

Finally, a basic calculation is also required. It is necessary to find out the duration of a measure in milliseconds. For this, the frontend sends as information the tempo in bpm. First, the backend determines how long a pulse lasts in milliseconds according to equation 7.1 by using the tempo information that sets the value of the variable *beats_min*. Then, having the time signature, its first number is used to know how many pulses fit into a measure. Therefore, by multiplying the duration in milliseconds of a pulse by the number of pulses in a measure, the duration in milliseconds of the measure is obtained.

7.2.3 The connection between MIDI and MusicXML

Taking into account that the score file is in MusicXML and that what is obtained from the user, after processing it, is an array of notes with the frequency, start timestamp and end timestamp of each one, it is necessary to establish a relationship between both representations in order to be able to compare them.

For this purpose, five functions and a common auxiliary one for some of them are implemented. They will be the cornerstone for correction because they will be the bridge among the representations to enable comparison.

There are two elements to be compared for a note, its duration and its pitch. Let us see how each is represented in the two formats:

- **Duration**
  - User data: The note's start timestamp and end timestamp are given. The difference between them gives the duration of the note in milliseconds.
  - MusicXML: From the *<divisions>* tag the value for a quarter note is known. With this reference, the *<duration>* tag sets the duration of the notes.

  Let us refer to the duration in the user format as *time* and the duration in the MusicXML format as *duration*. This gives rise to two functions, one for going from *time* to *duration* (*fromTimeToDuration*) and one for going from *duration* to *time* (*fromDurationToTime*). Both rely on the auxiliary function *calculateQuarterTime*, which calculates the duration in milliseconds of a quarter note (this will allow to calculate the duration in milliseconds of the other figures). Moreover, let *division* be the value given by the *<divisions>* tag.

- **Pitch**
  - User data: The frequency of the note in hertz is given.
  - MusicXML: From the *<step>*, *<alter>* and *<octave>* tags, the note name, the note's surname if it has one, and the octave are obtained.

  Let us refer to the user representation with the variable *hz* and the MusicXML representation as *pitch*. This gives rise to two functions, one to go from *pitch* to *hz* (*fromPitchToHz*) and one to go from *hz* to *pitch* (*fromHzToPitch*).

The other unmentioned function returns the type of the note given its *duration* in order to correctly complete the *<type>* tag in MusicXML, but it will not be explained. The other four functions and the auxiliary function are described in detail below.

1. ***calculateQuarterTime***
   In order to calculate the *time* of a quarter note, it is necessary to calculate how many quarter notes fit into a measure. To do this, it is necessary to work with the time signature. First of all by finding the ratio between the rhythmic figure that represents the pulse in the measure and a quarter note. That is, divide the second number of the time

signature (the rhythmic figure) by 4 (which is the representation of a quarter note, see section 6.1).

Then, it is enough to divide the first number of the time signature (the number of pulses in the measure) by this proportion. This gives the number of quarter notes per measure, and it is only necessary to divide the duration of a measure in milliseconds calculated in the preparation phase by this number to obtain the *time* of a quarter note.

Example:
Let us calculate the *time* of a quarter note (let it be *quarterTime*) knowing that the time signature is 6/8 and that the length of the measure in milliseconds is 3,000.

$\mathit{proportionToQuarter} = 8 \div 4 = 2$
$\mathit{quartersPerMeasure} = 6 \div 2 = 3$
$\mathit{quarterTime} = 3,000 \div 3 = 1,000$

Solution: *quarterTime* = 1,000 ms.

2. ***fromTimeToDuration***
The first thing this method does is to call the auxiliary function to obtain the *time* of a quarter note. Then, to calculate the *duration* for a given *time* (let it be *noteTime*), what it does is to multiply it by *division* and divide it by the *time* of a quarter note (let it be *quarterTime*), so that the calculation looks like this

$$\mathit{noteDuration} = \frac{\mathit{noteTime} * \mathit{division}}{\mathit{quarterTime}}. \tag{7.2}$$

Note that the quotient between *noteTime* and *quarterTime* defines the relationship between the duration of the current note and the quarte note.

This result is rounded to units because the *<duration>* tag in MusicXML works with integers and the time a student holds the note can fluctuate while still being correct. Therefore, without rounding, the playing accuracy would be too strict.

Example:
Let us calculate the *duration* of 500 milliseconds (let it be *noteTime*), knowing that the time signature is 6/8, that the length of the measure in milliseconds is 3,000 and that *division* is 24.

By calling the function *calculateQuarterTime*, we obtain that $\mathit{quarterTime} = 1,000$ ms.
$\mathit{noteDuration} = \mathit{noteTime} * \mathit{division} \div \mathit{quarterTime} = 500 * 24 \div 1,000 = 12.$

Solution: $\mathit{noteDuration} = 12$, which defines the duration of the considered note in the MusicXML language.

3. ***fromDurationToTime***

   Similarly, the first thing this method does is to call the auxiliary function to obtain the *time* of a quarter note. Then, to calculate the *time* for a note with a given *duration* (let it be *noteDuration*), what it does is to multiply it by the *time* of a quarter note and divide it by *division*, so that the calculation looks like

   $$noteTime = \frac{noteDuration * quarterTime}{division}. \tag{7.3}$$

   Note that the quotient between *quarterTime* and *division* sets the time correspondence in milliseconds for one unit of duration in MusicXML.

   Example:

   Let us calculate the *time* of a note with *duration* 48, knowing that the time signature is 6/8, that the length of the measure in milliseconds is 3,000 and that *division* is 24.

   By calling the function *calculateQuarterTime*, we obtain that $quarterTime = 1,000$ ms. $noteTime = noteDuration * quarterTime \div division = 48 * 1,000 \div 24 = 2,000.$

   Solution: $noteTime = 2,000$ ms.

4. ***fromPitchToHz***

   To convert the MusicXML *pitch* into the corresponding frequency, what this function does is to store the relation between the name of the notes in American notation ("A", "B", "C"...) and their corresponding key number in octave 4 on the keyboard. This way, when getting the <*step*> from MusicXML, it looks for the number equivalent to that note name (let it be *stepNumber*). Then, if there is a value for <*alter*> ("+1" or "-1") it takes it and adds it to the previous number. It then applies the <*octave*> value (let it be *noteOctave*), such that $stepNumber = stepNumber + 12 * (noteOctave - 4)$. Finally, once the real keyboard key number of the note is determined, by applying the convertion function $2^{\frac{stepNumber-49}{12}} * 440$, the frequency for the note is obtained [41].

   Example:

   Let's calculate the frequency (let it be *hz*) of a note with *step* A, *octave* 3 (let it be *noteOctave*) and no *alter*. Knowing that the corresponding registered keyboard key number for name A is 49 (let it be *initialStepNumber*), we have:

   $$stepNumber = initialStepNumber + 12 * (noteOctave - 4) = 49 + 12 * (3 - 4) = 37.$$

   $hz = 2^{\frac{stepNumber-49}{12}} * 440 = 2^{\frac{37-49}{12}} * 440 = 220.$

   Solution: The frequency of this note is $hz = 220$ Hz.

5. ***fromHzToPitch***
This function needs to know the value of the *<fifths>* tag (let it be *scoreFifths*) used in the previous section to determine the tonality of the piece. If the value of this tag was greater than 0 it indicated that the tonality had sharps (#), if it was less than zero it meant that the tonality had flats (*b*), and if it was 0 it showed that there were no accidentals. Like the previous function, it also needs to store the mapping between the keyboard key number and the note name for octave 4.

Let hz be the frequency to transform, by applying the inverse of the conversion formula, i.e., $12 * \log_2(\frac{hz}{440}) + 49$, the corresponding keyboard key number is determined (let it be *stepNumber*).

To convert this number into the three parameters needed by MusicXML, the procedure is as follows: calculate the octave, determine the *<alter>* and find out the *<step>*.

To determine the octave (let it be *noteOctave*), it is assumed that it is in octave 4 and it is checked whether the *stepNumber* falls within the limits of the known keyboard key numbers in the equivalences. If the number is larger, it adds one octave to *noteOctave* and subtracts 12 from the *stepNumber* until it fits within the limits. If the number is smaller, it subtracts one octave from *noteOctave* and adds 12 to the *stepNumber* until it fits within the limits. At the end of this process, the value for the *<octave>* tag and the *stepNumber* within the known values are obtained.

Afterwards, the function tries to find a number that matches the *stepNumber* in the equivalence of the keyboard key number to the note name. If it finds it, it records the name in *<step>* and discards the *<alter>*. If it finds no equivalent, it verifies in *scoreFifths* if it is a tonality of sharps (#), in which case it sets *<alter>* to "+1" and subtracts one from the *stepNumber* to now find the corresponding note name in the equivalence mapping, and record it in *<step>*. Otherwise, it is a tonality of flats (*b*), it records *<alter>* as "-1" and adds one to the *stepNumber* to then find the equivalence of the name, and store it in *<step>*. Such calculations are due to the fact that the same frequency can be named differently (according to the note's surname) and it is the tonality that elucidates this.

Example:
Let us calculate the *<step>* (*noteStep*), *<alter>* (*noteAlter*) and *<octave>* (*noteOctave*) values for frequency 880 Hz (let it be f). Knowing that the corresponding registered keyboard key number for name A is 49 and assuming an initial *noteOctave* $= 4$, we have:
stepNumber $= 12 * \log_2(\frac{f}{440}) + 4 = 12 * \log_2(\frac{880}{440}) + 4 = 61$.

The value 61 exceeds 51 which is the highest number known in the equivalences, therefore:

$noteOctave = noteOctave + 1 = 4 + 1 = 5$
$stepNumber = stepNumber - 12 = 61 - 12 = 49$
The value for the *stepNumber* is in the equivalences so the *noteAlter* is discarded and the *noteStep* can be determined.

Solution: <*step*> value is "A" and <*octave*> is 5. The tag <*alter*> would be 0 but usually this tag would be skipped in this case.

7.2.4 Backend: The correction

The ultimate goal is to get a representation of what the user has played with the errors marked in red. Therefore, each note and silence in the MusicXML file must be checked and, if the user has executed them wrongly, their representation must be replaced by the representation of what the user has played in red.

The user's interpretation is stored in an array that had been built in the reception and preparation phase prior to the correction step. During the explanation this array will be referred to as *fixedMidiArray*. Once one of the notes played by the user is represented (either because it was not necessary to correct it or because it replaced another note/silence and was marked in red) it is removed from the array. Therefore, at the end of the process, the array will be empty.

Since the score in a MusicXML file is organised by measures, the correction will also be done by measures. Throughout the whole process, the copy of the original file to store the committed errors is used.

The score file is parsed and the first thing that is done is to collect all the <*measure*> nodes, obtaining a list of measures. Then, an imaginary cursor that will go through this list is initialised at the start timestamp of the recording.

For each measure, its elements are traversed four times because different types of errors will be corrected each round. In order to refer to these four laps around the measure items, let us use the terms first, second, third and fourth loop. What is to be done in each of them is specified below.

The pitch checks will be performed with the frequency because the same frequency can give rise to different representations in MusicXML depending on certain parameters and therefore verifying with the frequency is more reliable. However, the duration check is performed in MusicXML notation (<*duration*>), since in this case the inaccuracy is in the time the student holds the note, which can slightly fluctuate and still be correct.

Throughout the loops, the functions mentioned in the previous section will be used countless times, so they will only be quoted at the beginning to show the tendency.

1. **First loop**
   This loop detects the notes that are well played and removes them from the *fixedMidiArray*. Moreover, the notes that are not played correctly are marked in blue. The blue colour indicates that the note is wrong but has not yet been replaced by the user's representation. Besides, it represents those wrong notes that match in pitch but fail in duration by marking them in red and then removes them from the *fixedMidiArray*.

   How does it do this?

   Thanks to the cursor, the function knows between which timestamps it must search for the notes played by the user in the *fixedMidiArray* that match the note to be evaluated in MusicXML. For each element (note or silence) of MusicXML, the corresponding cursor is stored as an attribute to facilitate this search process within the *fixedMidiArray* in the other loops.

   On the other hand, not all MusicXML elements are notes, there may also be silences or <backup> tags that indicate the beginning of the description of the notes on the left hand (so the cursor must go back to the beginning of the measure). The function behaves differently depending on the type of MusicXML element encountered:

   - For a silence, the cursor is simply moved by the time in milliseconds corresponding to the duration of the silence (using the *fromDurationToTime* function).
   - If it is a note, it searches in the *fixedMidiArray* if there are notes of the same pitch that fit somehow between the cursor time and the cursor + time of the original duration of the note. The functions *fromDurationToTime*, *fromTimeToDuration* and *fromPitchToHz* are used here.
     - If there is no note played that meets the requirements, the note is marked in blue because it means that it is wrong, as there is no well-played note that fits, and that this loop will not represent the user error since it only corrects errors where the pitch is the same (and the search in the array has not found any).
     - If one matching element is found, the duration is checked to ensure that it equals the duration of the original note. If it does not, the original note duration is replaced by the user's duration. Also, adjustments are made to make the representation as standard as possible (e.g. it may be necessary to add silences). Naturally, this is a representation of a user error and, as such, is marked in red.

    - If there are several matching notes played, it is already known to be wrong because several notes take the place of one. All these notes played by the user are represented as required (it may be necessary to add silences) and in red colour.

  - Within each measure there will be a *<backup>* label indicating the beginning of the notes in the left hand. Then, the function places the cursor at the beginning of the measure (thanks to the basic function *fromDurationToTime*, which allows the duration of the measure to be subtracted from the cursor), because the notes in the left hand have been played at the same time as the notes in the right hand.

   Naturally, there are many other factors to take into account when correcting as, for example, the case of ties or the case of chords (for which the cursor must be adjusted).

2. **Second loop**
   For the second loop it is no longer necessary to maintain a cursor because this data has been recorded in all the necessary elements in the first loop.

   In this case, notes with the same duration but different pitches are corrected. It will only be necessary to check those notes in blue because they are the ones marked by the first loop as erroneous but not yet representing the user error. So, from *fixedMidiArray* the selected notes are those that fit in duration with the MusicXML note and that fit in time within the stored cursor (from the first loop) and that cursor + time of the original duration of the note.

   If there are selected notes, the one closest in pitch to the original note is chosen for substitution. There is only one exception, when checking the right hand, if the pitch of the note is too low (beyond a fixed threshold), it is not replaced because that note may be substituting a similar case in the left hand that will be evaluated later because of the MusicXML layout.

   If the note has not been modified in pitch, it is replaced by a silence of equivalent duration. This means that the user has not played that note, he/she has missed it, he/she has been silent during that moment. In both cases, modified or replaced by silence, the color attribute is set to red.

3. **Third loop**
   All the above verifications have discarded different types of errors, so that it is now possible to check the silences. Previously it would have been impossible to check them because it may be the case that in the right hand there is a silence and in the left hand there is not. Therefore, as the MusicXML file is designed to go through staff one first, if there is a note played by the user that fits the silence time, it could be due to the left

hand and be correct, with the silence in the right hand also being correct. With the two loops above, only this kind of error can now occur.

In this loop, for each silence it is checked if there is any played note that fits in time. If so, the silence is replaced by the representation of the note(s). Similar to what happened in the second loop, it may be the case that there is a silence in both hands, and, as the right hand has to be checked first, perhaps the played note substitutes a silence that will come later in the second staff. Thus, again, a threshold is set so that, if the note is lower, it does not replace the silence and awaits for the left hand correction.

4. **Fourth loop**

   For both the second and third loops, by letting the note elapse in the right hand verification to determine whether it fits better in the left hand, if the latter is not the case, there are notes played by the user that have not yet been represented. Those are the target of this loop.

   Unlike the previous ones, this loop looks for the best fit for the remaining notes in *fixedMidiArray*. For each note in this array, it searches in the score for the time in which the note matches thanks to the cursor attribute set in the first loop, then it is represented (perhaps replacing a silence, or as a chord).

Example

The following is an example of how the loops would correct a user performance. Figure 7.6 shows the original measure as it is stored in the uploaded score file. Each measure element to be referred to during the text is marked as "O" (original) + the number of appearance in the MusicXML file. Meanwhile, figure 7.7 illustrates the final correction, and thus what the user has played. Here the elements of the measure are named with the letter "U" to identify that they are the user's version. For each loop, it is explained how it operates for each "P" (position) and then a picture of the representation after finishing the iteration is shown.

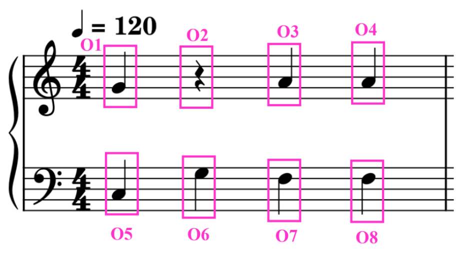


Figure 7.6: Original Version of the Measure

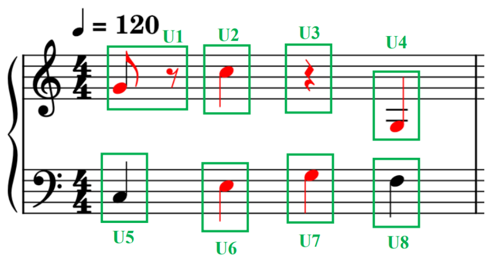


Figure 7.7: User Version of the Measure

1. **First loop**

    - P1: The loop searches in the *fixedMidiArray* for a note of the same pitch as O1 and finds U1, but it is of shorter duration than the original. So, it represents the incorrect duration with the corresponding figure and completes it with a silence.
    - P2: The first loop does not check silences, so it is skipped.
    - P3: When searching in the *fixedMidiArray*, it does not find a note of the same pitch, so it marks it in blue (it knows it is wrong but does not know how to correct it).
    - P4 & P6 & P7: It is the same as for P3.
    - P5: When searching the array, it finds U5 which has the same pitch and duration as O5, so it is correct and does not have to be modified.
    - P8: When searching the array, it finds U8 which has the same pitch and duration as O8, so it is correct and does not have to be modified.

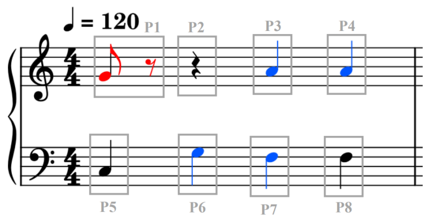


Figure 7.8: First Loop Version of the Measure

2. **Second loop**: This loop only checks the notes that the first loop marks in blue.

    - P3: The loop searches for a note of the same duration (same figure) but different pitch that can replace O3. It finds U7 but U7 is lower than the set threshold so it may be replacing another note played by the left hand that is to be evaluated later (O7). Therefore, it discards U7 and replaces O3 with a silence.
    - P4: The same happens as in P3, it finds U4, but it is lower than the threshold, so it is replaced by a silence.
    - P6: In this case, the loop finds U6 which fits within the limits. O6 is replaced by U6.
    - P7: In P3, U7 was skipped because it was lower than the threshold and it could be replacing a note in the left hand. Now, the loop finds U7 to replace O7.

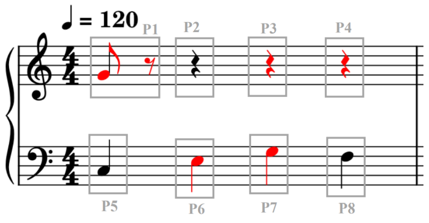


Figure 7.9: Second Loop Version of the Measure

3. **Third loop**: This loop only checks silences.

   - P2: The loop searches for a note in the array that fits the time of O2 and finds U2. U2 then replaces O2.
   - P3: When searching for a note that fits, none is found because U7 (the only one that would fit in time) has already been placed in the previous loop. Therefore, it keeps the silence.
   - P4: When searching the array, it finds U4, however this note is lower than the threshold so it is skipped because it may be replacing a silence in the left hand (case P8 was a silence).

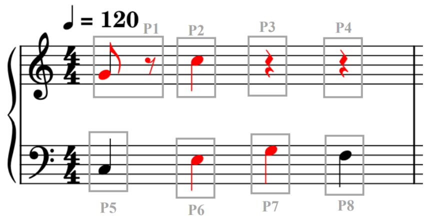


Figure 7.10: Third Loop Version of the Measure

4. **Fourth loop**

   This loop takes the remaining notes of the *fixedMidiArray* (which have not yet been placed) and finds their place in the measure. It takes U4, which had not been placed in either the second or third loop because it did not fit within the threshold. Now, this loop replaces O4 by U4, thus obtaining the final representation of the user's interpretation shown in figure 7.7.

7.2.5 Frontend: Back to the user

Once the file is corrected, it is sent as a response to the frontend, which renders it next to the original score representation. In this way, the student visualises both the original score and the corrected score with the committed errors in red at the same time. If he/she wants to start a new recording, the display of the file with the corrections is discarded and only the original score is shown again so that he/she can focus on it.

Chapter 8

Performed Testing

Beyond the unit tests carried out and the rigorous evaluations to check the correct operation of the system for both frontend and backend, individually and jointly, a user acceptance test has been conducted, which will be highlighted below.

Nevertheless, it is worth noting firstly that, although the computational load is apparently very high, the correction results are obtained quickly, each loop is lighter than the previous one and the intention is to optimise the process. Obviously, the longer the score, the longer it takes to correct it. If it is a two-sheet score, for example, the correction time is around one second. However, even if the correction time is longer, within reasonable limits, the user should understand it, because the perceived feeling of the correction time is conditioned by the time taken to play the score (the longer it takes to play the score, the longer the correction will take).

8.1 User Acceptance Testing

The aim of this test was to check that the interface is intuitive for a first-time user. It was intended to verify that the functionalities are easy to find and to exploit. Furthermore, it was essential to get the opinion of a real user, to know what improvements they need and to find out what features they are missing.

The user selected to take the test was the piano teacher of the student writing the report. During the test, at no time did the student intervene, who, as a mere spectator, witnessed the user's progress.

The "Sign Up", ”Sign In” and password recovery screens have been extremely natural and familiar to the user. The Fugateca screen has been a novelty and, although the functionalities were anticipated, the user has gone through a strong process of discovering them. In this screen, the student has made the decision to add a tooltip to some of the buttons, which, although easily interpretable, can take advantage of it. The functionality of being able to group scores into folders has been very appreciated by the user, who sees it as a great organisational advantage.

More specifically, the ability to classify scores into genres or the fact that the application displays information about the tonality of a score have surprised the user. He has even highlighted how useful it is for new pianists to know the tonality of the piece, because they are also reminded of the importance of this concept.

On the Fugáfono screen, thanks to the button that must be pressed to establish the connection with the keyboard, the user has understood that this is what he should do. The metronome setting is something he has discovered because of his curiosity to be able to change the speed at which he plays the piece. Once inside the configuration dialogue, all the concepts were clear and quickly understood.

The process of recording the performance of a score and observing the errors has caught the user's attention, at times surprised at the accuracy of the corrections. However, it is true that he has shown his desire for the program to take into account certain details such as repetitions in a score, although they were not within the scope of this work.

Finally, he also wanted to test the functioning of the website on tablet and mobile. He was pleased with the rendering for these devices and highlighted the usefulness and easiness that this implies, for example, for uploading a score that is already stored on these devices. Another special mention he made is the convenience of taking into account the rendering for tablet, which is usually forced to deal with the format for mobile or the format for computer.

From the observation, the student has been able to detect the difficulties that the user may have. Subsequently, as a developer, the appropriate improvements have been made in order to solve them.

Chapter 9

Conclusions

Upon completion of the work, a fully functional application that can be useful for users has been obtained. The project offers the essential functionalities to be able to operate comfortably and meets the objectives that had been set. The core of the application, error detection, is at a level of correctness that meets the expectations.

Direct improvements to the current application would be to add the functionality for the user to change their password from their profile (currently they can only change it with the password recovery feature) and to add a search filter for scores and folders. They are not technically complex and could enhance the user experience.

Throughout the project, concepts learnt during the entire degree have been applied, both from a computer science point of view and from a mathematical perspective. In fact, one of the lessons that, in the student's opinion, is particularly relevant is to remember that computer engineering is in continuous movement, in continuous development and, therefore, it is necessary to hold on to the basics in order to be able to move forward. Following this idea, it is known that the same concept has multiple representations, all of which are correct, and that it is possible to move from one to another because the nature remains the same. With this in mind, before trying to join MIDI and MusicXML, it was known that it was possible.

Moreover, the philosophy and way of thinking that is acquired during the studies allows problems of different scales to be handled, delimiting and sectioning them, with clear procedures. It was the first time that the student had dealt with a project of this size, and all the phases involved in its execution were covered. This has helped to connect all the areas seen during the career, which now find their reason within a general vision of the picture.

During the degree, students learn to work with various standards and technologies that make it possible to have the necessary tools to face this work. For example, the analysis diagrams, design diagrams, programming languages and some of the frameworks used to develop this project had been presented during the Information Systems specialisation in several subjects. Knowing how to collect the necessary data, process it, work with it, design the appropriate database and implement it is also thanks to the knowledge acquired in the specialisation.

Beyond the previous knowledge that the student had to undertake the project, during its development new topics have been learnt. For example, the developer has worked with new libraries and technologies like OSMD, Web MIDI API or MIDI. But, especially, the student has learned about the MusicXML standard, and has understood all the potential it can have, opening up an interesting line of research for the future.

Afterwards, the ideas for the evolution of the project are presented, distinguishing three tracks: one for improvement, one to add new functionalities, and the last one that seeks to explore other areas with new objectives that could make the project grow.

- Improvements
  These are improvements from a musical point of view. It has already been mentioned that in the user acceptance test, the user showed interest in repetitions being taken into account in the correction, so it would be interesting to add them. Also, thanks to another of the parameters offered by MIDI, it would be possible to check the accents (which notes the student presses harder) because they can sometimes be the origin of errors in the duration of notes.

  In order to broaden the musical range, some styles of modern music use a type of interpretation of the figuration that varies the duration of the represented notes. This is the so-called swung rhythm [42], and it would be interesting for the program to consider it in the corrections.

  Finally, it would be useful to give the student the opportunity to study a specific passage, that is, to be able to select the measures he/she wants to study and have the application correct only those measures, avoiding the user having to play the whole piece.

  Apart from the above, from a technical point of view, another improvement objective is to increase the application's compatibility with more browsers as mentioned in section 5.3 (thanks to the JZZ.js library).

- New features
  Among the ideas for extending the functionalities of the application are to offer the users to listen to the original score and their version of it. Statistics could also be provided to the students regarding the mistakes they make overall (for instance, if there are more rhythm errors than pitch errors).

  Another feature that users would certainly appreciate is the ability to upload sheet music in PDF format. This would require the use of an optical recognition software for music that allows the PDF to be converted into MusicXML notation. It would therefore be necessary to conduct a market analysis in this area in order to verify its existence. If such software does not exist or if it is not reliable, it would be interesting to initiate a project to develop it, because it could be a key element in the relationship between music and computing.

- Growth
  With a view to the growth of this project, it has been kept in mind from the outset that, after this preliminary phase, in which the foundations were established, the most promising objective is to be able to predict where the user is going to fail. To achieve so, it would be necessary to use artificial intelligence in such a way that it simulates the student's learning. Thus, if, for example, a new rhythmic sequence that the student does not yet know appears, it would be a candidate for error. Indeed, with sufficient data, it might be possible to calculate how long it will take a student to learn a given score.

  Finally, another attractive feature that the application could evolve towards is the creation of a "stage mode" section. This would display the score to the user and detect what he/she is playing in order to be able to turn the music sheet at the right moment. This would allow the users to use their device to read the score in a very comfortable way that would allow them to fully focus on the instrument and the music itself.

Appendix A

Prototypes

The Justinmind [43] tool was used to develop the prototypes that helped to define the requirements. Images of the prototype for the different views in the application are attached below.

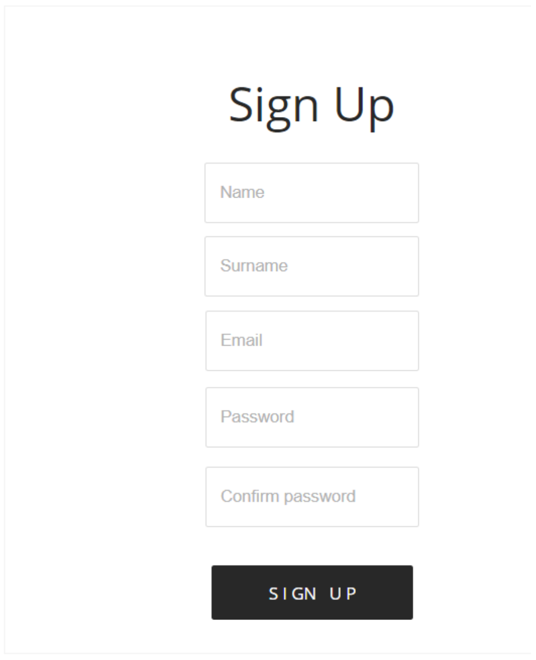


Figure A.1: Sign Up screen

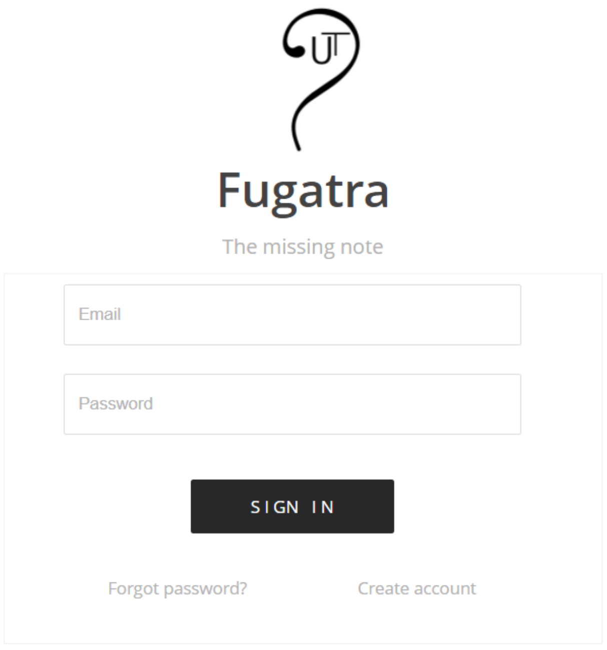


Figure A.2: Sign In screen

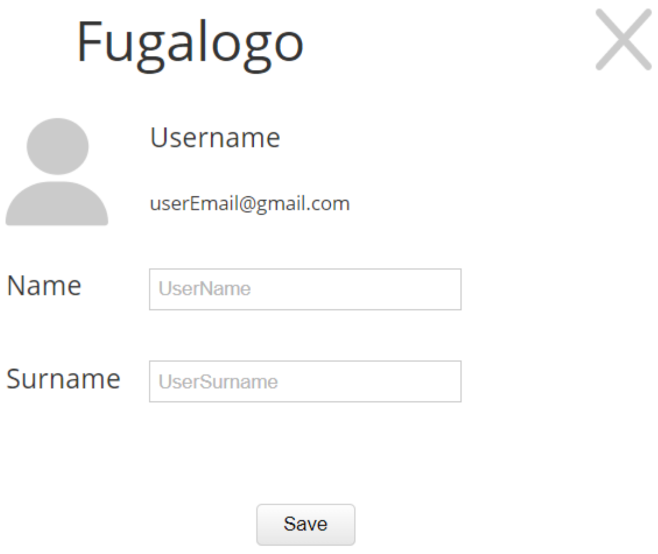


Figure A.3: Fugalogo screen

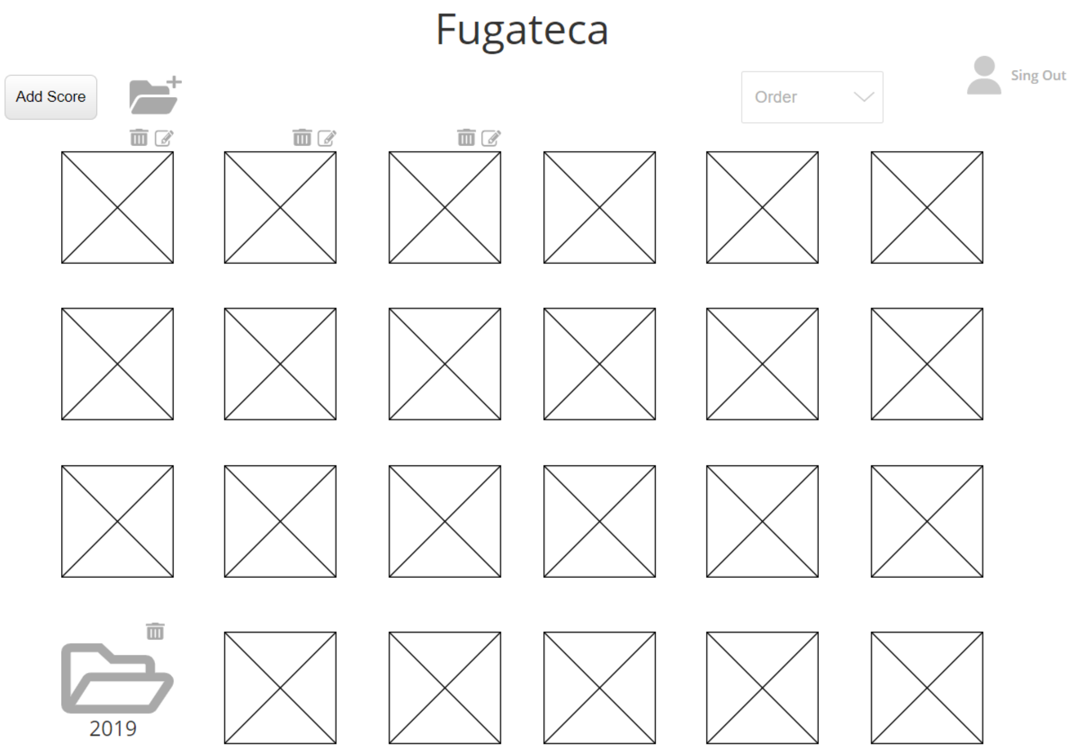


Figure A.4: Fugateca screen

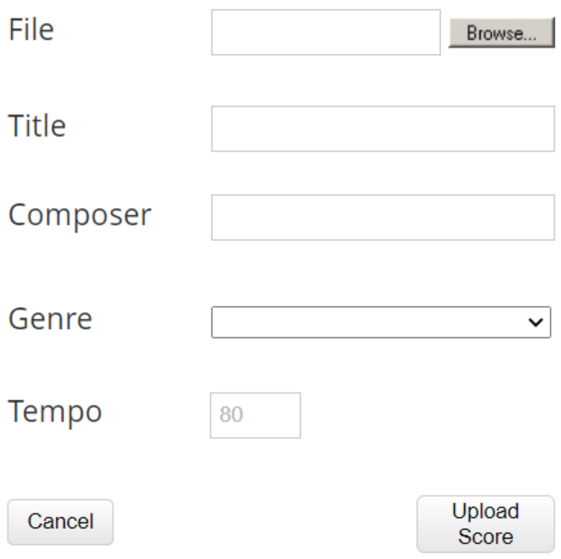


Figure A.5: Upload score screen

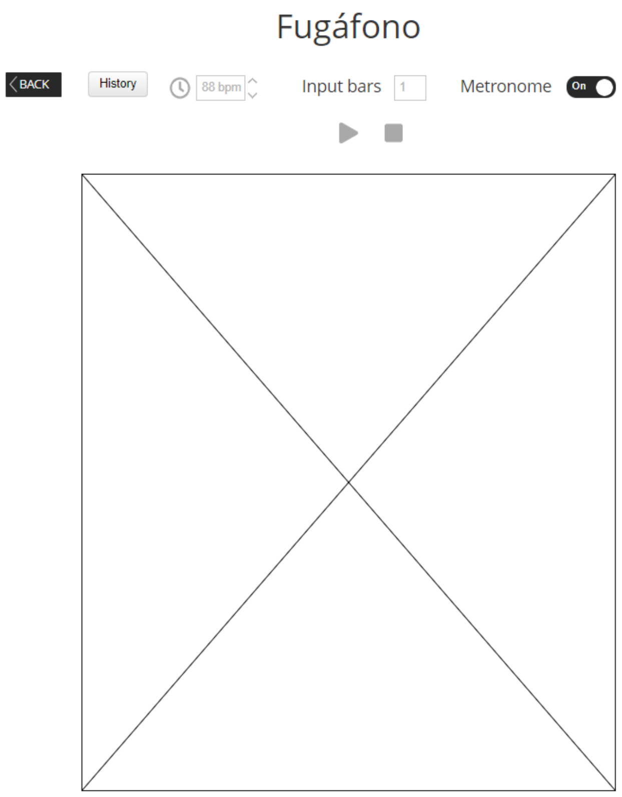


Figure A.6: Fugáfono screen

List of Acronyms

API Application Programming Interfaces. 28, 38, 39, 59, 78, 85

BPM beats per minute. 41, 43, 48, 59, 61

CRUD Create, Read, Update and Delete. 37

CSS Cascading Style Sheets. 37

DBMS DataBase Management System. 28, 34, 36

DDR4 Double Data Rate type four. 9

DTO Data Transfer Object. 24, 60

H Hours. 10

HTML HyperText Markup Language. 12, 33, 36, 37, 39

HTTP Hypertext Transfer Protocol. 33, 37

Hz Hertz. 49

JPA Java Persistence API. 35

JPG Joint Photographic Group. 40

JS JavaScript. 37, 39

JSON JavaScript Object Notation. 33, 51, 52, 60

MIDI Musical Instrument Digital Interface. 1, 9, 12, 28, 38, 59–62, 77, 78

MVC Model-View-Controller. 27

MVCC Multi-Version Concurrency Control. 36

NPM Node Package Manager. 37

Nº Number. 10

ODL Object Definition Language. vii, 31

ORM Object-Relational Mapping. 35

OSMD Open Sheet Music Display. 12, 39, 58, 60, 78

PDF Portable Document Format. 12, 40, 78, 79

PNG Portable Network Graphics. 12, 40

POM Project Object Model. 35

REST Representational State Transfer. 28, 33, 37

SRS Software Requirements Specification. 8

SSD Solid State Drive. 9

SSO Single Sign-On. 35

SVG Scalable Vector Graphics. 12, 39

UI User Interface. 37

UML Unified Modeling Language. 39

URL Uniform Resource Locator. 37, 54

XML Extensible Markup Language. 35, 39, 47

Glossary

Canvas HTML element incorporated in HTML5 that allows the generation of graphics dynamically by means of scripting. 39

DIN cable It is a type of connector, originally standardised by the"Deutsches Institut für Normung" (DIN), which is the "German Institute for Standardisation". 38

Fugalogo Application user. vi, 6, 29, 82

Fugateca User's score library. v, vi, 6, 13, 15, 52, 54, 58, 75, 82

Fugáfono Section of the application where the user studies (plays the piece of music and errors are displayed). v, vi, 6, 13, 15, 55, 58–60, 76, 83

Input bars Blocks of rhythm prior to the start of a performance that indicate the tempo and entrance to the musician. 11, 23

Metronome Device that produces an audible click at a regular interval that can be set by the user. 11, 13, 22, 23

mxl Compressed MusicXML format. 20

Promise An object used for asynchronous computations that represents a value that may be available now, in the future, or never. 37

Tempo Speed at which a musical piece should be performed. 3, 11, 20–23, 87

Bibliography

[1] MoonPiano, "Moon piano information web." [Online]. Available: https://moonpiano.praisethemoon.org/home

[2] Pianu, "Pianu web page." [Online]. Available: https://pianu.com/

[3] P. Sessions, "Playground sessions web page." [Online]. Available: https://www.playgroundsessions.com/

[4] Flowkey, "Flowkey web page." [Online]. Available: https://www.flowkey.com/es?utm_campaign=aff_futureplc&utm_source=website&utm_medium=general

[5] Skoove, "Skoove web page." [Online]. Available: https://www.skoove.com/es

[6] B. Williams, "Best apps for learning piano." [Online]. Available: https://rolandcorp.com.au/blog/best-apps-for-learning-piano

[7] D. C. MusicRadar, "Best online piano lessons 2021: recommended piano lesson apps, software and websites." [Online]. Available: https://www.musicradar.com/news/the-best-online-piano-lessons

[8] I. Wikimedia, "Do (nota)." [Online]. Available: https://es.wikipedia.org/wiki/Do_(nota)

[9] K. Beck, M. Beedle, A. van Bennekum, A. Cockburn, M. F. Ward Cunningham, J. Grenning, J. Highsmith, A. Hunt, R. Jeffries, J. Kern, B. Marick, R. C. Martin, S. Mellor, K. Schwaber, J. Sutherland, and D. Thomas, "Principles behind the agile manifesto." [Online]. Available: https://agilemanifesto.org/principles.html

[10] C. Walshaw, "Abc notation." [Online]. Available: https://abcnotation.com/

[11] I. MakeMusic, "Musicxml." [Online]. Available: https://www.musicxml.com/

[12] ——, "Makemusic." [Online]. Available: https://www.makemusic.com/

[13] ——, "Finale." [Online]. Available: https://www.finalemusic.com/

[14] LilyPond, “Lilypond.” [Online]. Available: https://lilypond.org/index.html

[15] PhonicScore, “Open sheet music display.” [Online]. Available: https://opensheetmusicdisplay.org/

[16] M. M. Cheppudira, “Vexflow.” [Online]. Available: https://www.vexflow.com/

[17] Indeed, “Average salaries spain.” [Online]. Available: https://es.indeed.com/career/salaries

[18] Varun, “What are the major features of java programming?” [Online]. Available: https://www.tutorialspoint.com/What-are-the-major-features-of-Java-programming

[19] Guru99, “Spring tutorial: What is spring framework.” [Online]. Available: https://www.guru99.com/spring-tutorial.html

[20] E. Geek, “¿qué es java hibernate? ¿por qué usarlo?” [Online]. Available: https://ifgeekthen.everis.com/es/que-es-java-hibernate-por-que-usarlo

[21] J. Garzas, “Simple y rápido. entiende qué es maven en menos de 10 min.” [Online]. Available: https://www.javiergarzas.com/2014/06/maven-en-10-min.html

[22] I. Wikimedia, “Database management system.” [Online]. Available: https://en.wikipedia.org/wiki/Database#Database_management_system

[23] ——, “Postgresql.” [Online]. Available: https://en.wikipedia.org/wiki/PostgreSQL

[24] J. van Niekerk, “How do html, css and javascript work together?” [Online]. Available: https://www.itonlinelearning.com/blog/how-do-html-css-and-javascript-work-together/

[25] I. Wikimedia, “Vue.js.” [Online]. Available: https://en.wikipedia.org/wiki/Vue.js

[26] E. S. M. Morote, “Vue router.” [Online]. Available: https://router.vuejs.org/

[27] Vuetify, “Vuetify - material design framework.” [Online]. Available: https://vuetifyjs.com/en/

[28] Axios, “Axios.” [Online]. Available: https://github.com/axios/axios

[29] D. Hristov, “Simple vue.js form validation with vuelidate.” [Online]. Available: https://vuejsdevelopers.com/2018/08/27/vue-js-form-handling-vuelidate/

[30] Kazupon, “Vue i18n.” [Online]. Available: https://kazupon.github.io/vue-i18n/

[31] M. Association, “Midi - musical instrument digital interface.” [Online]. Available: https://www.midi.org/

[32] W3C, “Web midi api.” [Online]. Available: https://www.w3.org/TR/webmidi/

[33] C. I. use, “Web midi api compatibility.” [Online]. Available: https://caniuse.com/midi

[34] Jazz-soft, “Jzz: Midi library for node.js and web-browsers.” [Online]. Available: https://www.npmjs.com/package/jzz

[35] I. Wikimedia, “Gitlab.” [Online]. Available: https://es.wikipedia.org/wiki/GitLab

[36] ——, “Equal temperament.” [Online]. Available: https://en.wikipedia.org/wiki/Equal_temperament

[37] W3C, “Compressed .mxl files.” [Online]. Available: https://www.w3.org/2021/06/musicxml40/tutorial/compressed-mxl-files/

[38] I. Wikimedia, “Índice acústico.” [Online]. Available: https://es.wikipedia.org/wiki/%C3%8Dndice_ac%C3%BAstico

[39] ——, “Índice acústico científico.” [Online]. Available: https://es.wikipedia.org/wiki/%C3%8Dndice_ac%C3%BAstico_cient%C3%ADfico

[40] J. Wolfe, “Note names, midi numbers and frequencies.” [Online]. Available: https://newt.phys.unsw.edu.au/jw/notes.html

[41] I. Wikimedia, “Piano key frequencies.” [Online]. Available: https://en.wikipedia.org/wiki/Piano_key_frequencies

[42] ——, “Swing (jazz performance style).” [Online]. Available: https://en.wikipedia.org/wiki/Swing_(jazz_performance_style)

[43] I. MakeMusic, “Justinmind.” [Online]. Available: https://www.justinmind.com/