

ScoreFlow.

Aplicación web para validar sistemas de reconocimiento de partituras musicales(OMR)

Marc Martín Martínez

Resumen— Este proyecto surge de la necesidad de corregir partituras musicales en formato MusicXML creadas a partir de *Optical Music Recognition* (OMR). El usuario sube a la aplicación un archivo MusicXML junto con una o varias imágenes de la partitura que usará como *ground truth* para revisar la pieza musical. A continuación, se muestra el grupo de imágenes que se hayan subido junto con un conjunto de compases establecidos por el archivo MusicXML. Se puede navegar libremente entre las diferentes imágenes que haya subido y los diferentes grupos de compases existentes. Usando una barra de herramientas, el usuario puede añadir, cambiar o eliminar: notas, accidentes, silencios... y cambiar características del compás: armadura, clave y signatura de tiempo. Finalmente, cuando se haya terminado de corregir la partitura, el usuario puede descargarse un nuevo archivo MusicXML con todos los cambios que haya hecho previamente.

Palabras clave— Partitura, Corrección, OMR, MusicXML, NodeJS, Vue.js, MySQL.

Abstract— This project comes up from the necessity of correcting musical scores in MusicXML format that were created from OMR *Optical Music Recognition*. The user uploads to the application a MusicXML file along with one or more images of the score that will be used as a ground truth to review the musical piece. Then, the group of images that have been uploaded are shown along with a set of measures that are established by the MusicXML file. You can freely navigate between the different images that were uploaded and the set of measures that exist. Using a toolbar, the user can add, modify or remove: notes, accidentals, rests... and change characteristics of the measure: key signature, clef and time signature. Finally, when the score is corrected, the user can download a new MusicXML file with all the changes that were made.

Keywords— Score, Correction, OMR, MusicXML, NodeJS, Vue.js, MySQL.

1 INTRODUCCIÓN

DURANTE la década de los 50 se funda la primera compañía dedicada al desarrollo y venta de dispositivos OCR (*Optical Character Recognition*), creada por David H. Shepard[1], desarrollador del primer

dispositivo OCR. Con el paso de los años surge un nuevo concepto de reconocimiento de caracteres, OMR (*Optical Music Recognition*), dedicado obviamente a la lectura de elementos musicales. La primera aplicación desarrollada es MIDISCAN, por *Musitek Corporation*[2]. Actualmente, aunque la tecnología de OMR ha avanzado, aún existen errores de lectura por parte del software de reconocimiento que da lugar a resultados incorrectos. De ahí la importancia del desarrollo de este proyecto.

- E-mail de contacto: marcmartin315@gmail.com
- Mención realizada: Enginyeria del Software
- Trabajo tutorizado por: Dr. Alicia Fornés Bisquerra (Ciencias de la Computación)
- Curso 2019/20

2 ESTADO DEL ARTE

Existen varias aplicaciones web con funcionalidades similares a la de este proyecto, aunque no dedicado exclusivamente a corregir archivos creados a partir de OMR, como *float.io*[3] y *Noteflight*[4].

Desde la perspectiva de visión por computadores de OMR, hay varios investigadores dedicados a la mejora de la tecnología de reconocimiento y a la divulgación de esta, como Arnau Baró-Mas , autor de [17] y [18] o Alicia Fornés, autora de , [19] y [20], ambos de ellos siendo colaboradores de la página *OMR Research*[21]

3 OBJETIVOS

En esta sección se explicarán los objetivos tenidos en cuenta para poder realizar este proyecto:

- Base de datos.
Lugar donde se guardarán todos los usuarios que usen la aplicación y todos sus proyectos.
- Gestor de Proyectos.
Entorno donde el usuario puede revisar todos los proyectos que haya creado, y crear de nuevos, y seleccionar uno de ellos para trabajar con él.
- Visualización de la partitura.
A partir de la API *VexFlow*[6], representar un conjunto de compases respectivos al fichero MusicXML. El espacio asignado a los compases en la página se adapta al tamaño de la pantalla del navegador.
- Edición de la partitura.
Usando *Vue.js*[7] se crea una interfaz de usuario que permite añadir, cambiar o eliminar elementos de la partitura.
- Exportar nuevo archivo MusicXML.
Una vez el usuario considere oportuno, se puede descargar un nuevo archivo con todos los cambios hechos previamente.

4 PLANIFICACIÓN

La planificación del proyecto se divide en 3 fases. El diagrama de Gantt correspondiente se encuentra en la figura 1. Para una versión mas detallada puede encontrarse la figura 10 en el apéndice.

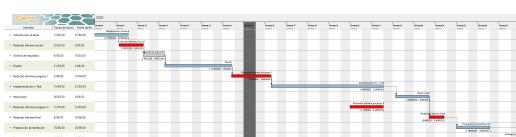


Fig. 1: Diagrama de Gantt.

4.1. Introducción al tema

Esta fase consistía en hacer un primer contacto con la materia.

- Funcionamiento de MusicXML.

Estudio del formato universal de representación de partituras. Al tratarse de un archivo XML, todo funciona con tags.

- Creación de glosario.

Documento con un listado de elementos musicales básicos y sus descripciones correspondientes. Útil en las primeras fases del proyecto donde aún no se dominaba completamente los conocimientos musicales requeridos para crear partituras coherentes.

- Comprobar el estado del arte.

Estudio de cual era la situación actual referente al OCR y OMR, además de aplicaciones similares al proyecto.

4.2. Análisis y Diseño

La meta de esta fase era identificar las funciones básicas que necesitaba el proyecto y como implementar estas de la manera mas óptima posible.

4.2.1. Análisis de Requisitos

Listar y clasificar los requisitos en funcionales y no funcionales, como puede verse en las tablas 1 y 2.

Funcionales	
ID	Descripción
RF-1	Subir un archivo MusicXML a la aplicación y su correspondiente imagen de la partitura escaneada.
RF-2	Visualizar la partitura e imagen en la pantalla.
RF-3	Seleccionar elemento que se quiere modificar con un click de ratón.
RF-4	Seleccionar notación musical: anglosajona (A, B, C, D, E, F, G) o latina (do, re, mi, fa, sol, la, si).
RF-5	Añadir, modificar o eliminar nota al compás, seleccionando <i>tempo</i> y nota.
RF-6	Añadir o quitar a nota existente un puntillo o calderón.
RF-7	Añadir una ligadura entre dos notas.

ID	Descripción
RF-8	Añadir o quitar accidente a una nota: sostenido (#), bemol (b) y becuadro (h).
RF-9	Seleccionar tipo de armadura. Puede tenerla o no.
RF-10	Seleccionar clave: Sol (G), Fa(B) y Do(C).
RF-11	Seleccionar la firma de compás: 4 3 4, 4, etc.
RF-12	Añadir, modificar o eliminar silencios.
RF-13	Cambiar tipo de barra del compás: simple, doble o de repetición.
RF-14	Exportar cambios realizados en la partitura a un archivo MusicXML.

TABLA 1: REQUERIMIENTOS FUNCIONALES

Después, usando el modelo de Kano, se volvieron a clasificar en *dissatisfiers*, *satisfiers* y *delighters*. Las tablas correspondientes similares a las anteriores pueden encontrarse en las tablas 5, 4 y 3 del apéndice.

4.2.2. Diseño

Creación de elementos considerados útiles para poder hacer un buen diseño de la aplicación.

- Diagramas UML

Creación de diagramas de casos de uso, de secuencia y de clase, como por ejemplo las figuras 2 y 3. Otros diagramas pueden verse en la sección del apéndice: Diseño. Utilización de *VisualParadigm* [8] para comprender estos diagramas.

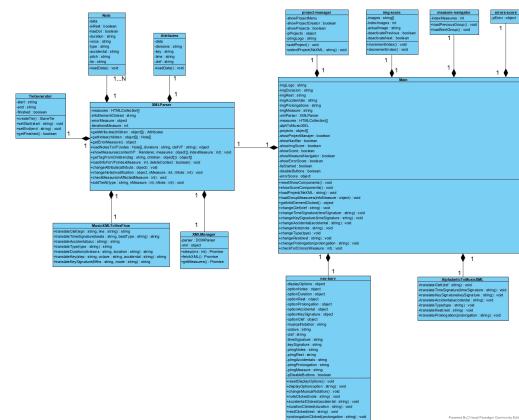


Fig. 2: Diagrama de Clase

No funcionales	
ID	Descripción
RNF-1	El sistema debe de usar mensajes de error que describan el problema de manera precisa y que el usuario final pueda entenderlo (sin tecnicismos).
RNF-2	El feedback de la interfaz de usuario debe producirse en menos de 1 segundo. En caso de que sea mayor, indicar al usuario de que su petición se está cargando.
RNF-3	La aplicación debe de contar con un diseño <i>responsive</i> .
RNF-4	Uso de diálogos de confirmación para evitar acciones accidentales por parte del usuario en los que es difícil recuperar el estado anterior a la acción.

TABLA 2: REQUISITOS NO FUNCIONALES

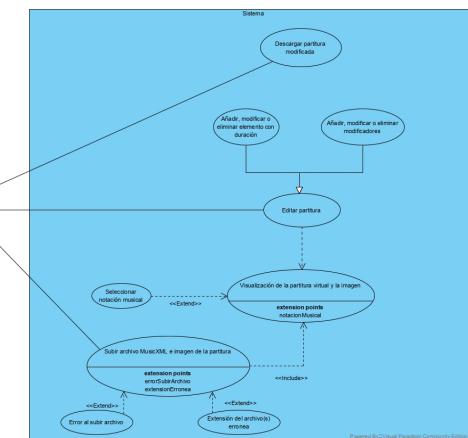


Fig. 3: Diagrama de Casos de Uso

- Prototipo

Creación de un prototipo simple para representar el editor de partituras usando la herramienta de edición de imágenes digitales *GIMP*[9]. Este prototipo sería desecharlo mas adelante, ya que se decidiría usar una barra de herramientas para la edición de la partitura. El prototipo esta disponible en el apéndice, en la página 7.

- Persona

Desarrollo de una *persona* para facilitar que la interfaz del usuario estuviera enfocada en un posible usuario de

la aplicación, y no en el desarrollador de la aplicación. Utilización de la herramienta *Xtensio*[10] para crear la persona, representada en la figura4.

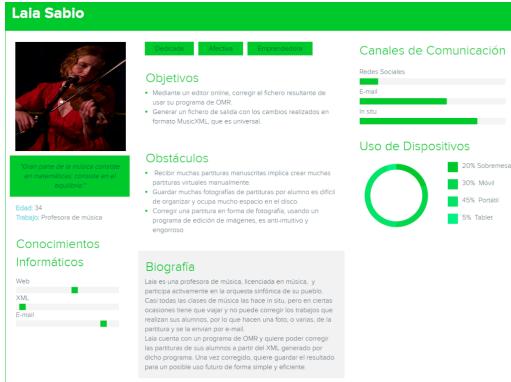


Fig. 4: Persona

4.3. Implementación

La planificación inicial de esta fase consistía en visualizar la partitura y después poder editarla, para finalmente exportar el nuevo archivo MusicXML. Todo esto siguiendo el paradigma de testing *Test-Driven Development*. Una vez acabado esto se esperaba tener hasta 3 iteraciones de una semana de duración cada una. En estas iteraciones se detectarían errores mediante *user testing* con usuarios con conocimientos musicales para después modificar requisitos y/o el diseño, e implementar estos cambios. Esta fase sufrió algunos cambios con respecto lo planificado al principio del proyecto por una planificación demasiado optimista, no teniendo suficiente tiempo para las iteraciones. Además, la parte de testing del proyecto también se tuvo que omitir.

5 METODOLOGÍA

La metodología a seguir del proyecto era iterativa. A medida que el proyecto avanzaba en el tiempo se iban modificando elementos del mismo, como por ejemplo el prototipo anteriormente mencionado. El diseño del programa también iba sufriendo pequeños cambios, al ver que surgían nuevas necesidades en la interfaz de usuario o se detectaban características obsoletas o daban lugar a dudas.

6 RESULTADOS

La aplicación está desarrollada en Nodejs. Esta está dividida en *backend*, la parte del servidor, y *frontend*, la del cliente.

6.1. Backend

En este apartado se mencionan las funcionalidades desarrolladas en el servidor y su funcionamiento. Se utilizan las extensiones:

- *Express*[11]. Framework para crear el servidor y hacer *routing*.
- *Express-fileupload*[12]. Módulo para *express* para la subida de archivos del cliente al servidor.

- *Express-session*[13]. Módulo para la gestión de sesiones que identifican a los usuarios.
- *bcrypt*[14]. Módulo para encriptar cadenas de caracteres. Utilizado para cifrar la contraseña de los nuevos usuarios y para comprobar que la contraseña de un usuario que se *loggea* es la misma que en la base de datos.
- *mysql*[15]. Driver para conectarse a una base de datos MySQL y hacer consultas en ella.

El servidor tiene dos propósitos principales:

1. Servir los archivos o redirigir al usuario según las peticiones que haga el cliente. Posible gracias al framework mencionado anteriormente, *express*.
2. Comunicarse con la base de datos para incluir nueva información o solicitarla.

La clase **DataBase** alberga toda la información necesaria para acceder a la base de datos: usuario, contraseña, host y nombre de la base de datos. La clase cuenta con varias funciones para hacer consultas sql contra la base de datos, como crear usuarios o proyectos u obtenerlos. El diagrama entidad-relación puede verse en la figura 5.

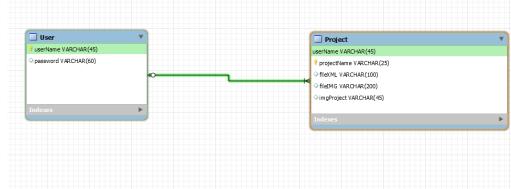


Fig. 5: Diagrama Entidad-Relación de la base de datos.

6.2. Frontend

Esta sección muestra los elementos creados para el frontend. Para crear una interfaz de usuario reactiva a los cambios hechos por el usuario se utiliza el framework *Vue*.

6.2.1. SignIn y LogIn

Esta página contiene un formulario para que el usuario pueda crear una cuenta en la aplicación. A la hora de confirmar la contraseña de la nueva cuenta, la aplicación comprueba que las dos contraseñas añadidas sean iguales antes de enviarlas al servidor. En la figura 6 se puede observar el formulario para crear una nueva cuenta.

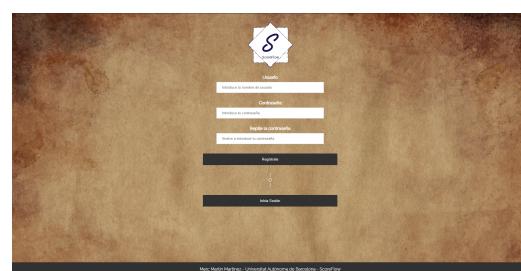


Fig. 6: Formulario de creación de usuario

Asimismo, la página también incluye otro formulario para poder iniciar sesión. El usuario puede cerrar la sesión en cualquier momento una vez *logueado* usando el botón en la esquina superior derecha.

6.2.2. Gestor de Proyectos

Este componente es el responsable de mostrar al usuario sus proyectos y de darle la opción de crear nuevos.

Para poder crear un nuevo proyecto el usuario ha de seleccionar un nombre para el proyecto, un archivo XML, y la imagen o imágenes correspondientes a la partitura. El nombre del proyecto ha de ser único.

Los proyectos se muestran usando el nombre del proyecto, el nombre del archivo XML y una imagen de la partitura, como se puede ver en la figura 7.

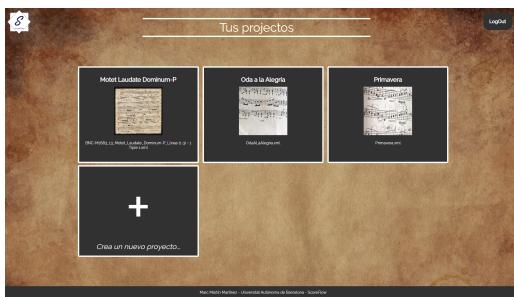


Fig. 7: Gestor de Proyectos

Una vez el usuario selecciona un proyecto el gestor de proyectos desaparece y se muestra el editor de partituras. El usuario siempre puede volver al gestor de proyectos usando el logo en la esquina superior izquierda.

6.2.3. Visualización de la partitura

Se crea la clase **XMLParser**, responsable de representar la partitura a partir del archivo MusicXML y la de aplicar los cambios a este hechos por el usuario.

Inicialmente se lanza una petición al servidor para obtener el archivo MusicXML del usuario utilizando un *fetch*. Despues, se parsea todo el documento para extraer los tags necesarios para crear una partitura. Dado que es un XML, se utiliza la interfaz *Document* [16] para poder modificar el árbol *DOM*. La información obtenida es almacenada en otras dos clases:

- **Attributes.** Contiene la información de los tags del compás: clave, signatura de tiempo y armadura. Esta clase solo puede tener una instancia, por lo que en una partitura solo puede haber una armadura, clave y signatura de tiempo. Es decir, el usuario no puede añadir nuevos atributos en mitad de la partitura. La aplicación solo tiene en cuenta el primer tag de atributos que ha de encontrarse en el primer compás.

- **Note.** Contiene la información de los tags de una nota: duración, tono y si es o no un silencio. Además de los elementos que se le pueden añadir a una nota: accidentes, *legatos* y ligaduras y prolongaciones (puntillo o calderón).

Para gestionar el funcionamiento de los *legatos* y ligaduras, que son representados de la misma forma dentro de la aplicación, se crea la clase **TieGenerator**. Su función básica es almacenar la primera y la última nota que forma una ligadura. Una vez se tienen las dos notas, crea un objeto de VexFlow que representa dicha ligadura. Además, controla si la clase **XMLParser** ha de mostrar un compás adicional en el caso de que la nota inicial y la final se encuentren en dos compases distintos.

Actualmente la aplicación solo puede mostrar correctamente una ligadura si el número de compases entre los que se encuentra el inicio y final de una ligadura es de uno. Además, una nota no puede pertenecer a una ligadura si esta está entre otras dos notas que forman una ligadura diferente. Por lo tanto, no pueden haber ligaduras dentro de otras ligaduras.

Para poder representar toda esta información en forma de partitura se utiliza la API VexFlow. Para poder construir la partitura, esta información ha de ser *traducida* para que la API pueda utilizarla. Para ello se crea otra clase, **MusicXMLToVexFlow**. En ella se compara cada tag con todas las posibles opciones que puede tener el tag que este disponibles en la API y devuelve ese equivalente.

Finalmente, se inicializan los valores necesarios para VexFlow: tamaño de los compases, del SVG(Scalable Vector Graphics) donde se encontrará la partitura... y se crean los objetos de VexFlow para después dibujarlos y así conseguir representar la partitura. La aplicación representa hasta 5 compases a la vez, 6 en el caso de una ligadura con notas en diferentes compases.

A cada objeto que representa una nota se le asignan los atributos *measure* y *note*, que representan el número compás donde se encuentra la nota y el índice de la nota dentro de ese compás. Además de *event listeners* a *clicks*. Esta información será usada mas adelante cuando el usuario quiera modificar una nota.

Existen algunos elementos que la aplicación no es capaz de representar por que son poco utilizados.

Claves:

- Bass Clef (Down Octave)
- Vocal Tenor Clef

Armaduras fuera de la escala diatónica:

- | | | |
|------------|--------------|-----------|
| ■ Dorian | ■ Mixolydian | ■ Locrian |
| ■ Phrygian | ■ Aeolian | |
| ■ Lydian | ■ Ionian | |

Armaduras teóricas:

- | | |
|-------------|----------|
| ■ F \flat | ■ G $\#$ |
| ■ d \flat | ■ e $\#$ |

Duración:

- Máxima
- Longa
- Cuadrada
- Garrapatea
- Semigarrapatea

6.2.4. Editor de Partituras

La interfaz se compone de: barra de herramientas, imagen o imágenes de la partitura, errores recogidos de la partitura y el navegador de partituras, que consiste en dos botones para avanzar o retroceder en el grupo de compases. Cada elemento de la interfaz de usuario se representa en componentes de *Vue*. Estos componentes y la representación de la partitura pueden verse en la figura 8.

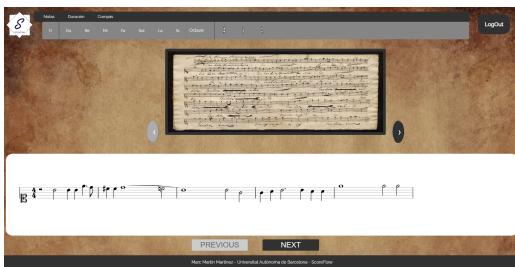


Fig. 8: Editor de Partituras

El componente *nav-bar*, o la barra de herramientas, permite al usuario modificar la nota *clickada* anteriormente o los atributos del compás. La tonalidad de una nota puede cambiarse utilizando uno de los botones que designa un tono mas la octava que se quiera utilizar. Los accidentes y prolongaciones pueden añadir o quitarse. La duración puede modificarse desde la redonda, 4 pulsos, hasta la semisíntesis, $\frac{1}{64}$ pulsos. En el compás puede modificarse la clave, la signatura del compás y la armadura. Además, la barra de herramientas cuenta con dos modos de visualización de sus elementos: notación musical latina y anglosajona, para que el usuario pueda usar la notación con la que se sienta más cómodo. La aplicación funciona internamente con la notación anglosajona.

Para poder aplicar los cambios que haga el usuario al archivo MusicXML se ha de volver a traducir la información transmitida por el usuario al formato MusicXML. Para ello se crea otra clase, **AlphabeticToMusicXML**, muy similar a la anteriormente mencionada MusicXMLToVexFlow.

Para visualizar las imágenes subidas por el usuario se cuenta con otro componente que cuenta con dos botones para avanzar o retroceder en las diferentes imágenes, y la imagen que quiera visualizar el usuario. La imagen seleccionada puede ser ampliada para ayudar al usuario a leerla mejor. En el caso que el usuario modifique un compás, y este no respete el número de pulsos necesario dictaminado por la signatura de compás, se mostrará un mensaje de error mostrando el número de pulsos actual del compás y el número necesario para que este sea correcto. Un ejemplo de este error se muestra en la figura 9.



Fig. 9: Mensaje de error del compás

Finalmente se muestra el conjunto de compases mas otros

dos botones para poder avanzar o retroceder a lo largo de la partitura.

7 CONCLUSIONES

Como se ha visto a lo largo del documento, la aplicación tiene un gran margen de mejora: poder añadir atributos en cualquier momento de la partitura, añadir mas elementos representables en la partitura, hacer que un proyecto sea colaborativo y que 2 o mas personas sean capaces de editar un mismo proyecto... Pero la planificación, y el seguimiento de ella, no se ha hecho de la mejor manera posible. Por consecuencia se han perdido funcionalidades u oportunidades que podrían haber dado lugar a una mejor aplicación. Otro factor que también ha afectado al desarrollo del proyecto de forma negativa ha sido subestimar la importancia de un diseño exhaustivo. Al no dedicar mas tiempo a él, se crearon nuevas funciones o clases sobre la marcha, dando como resultado a una situación caótica en el código, desarrollando funciones poco eficientes o que al poco tiempo quedaban obsoletas, teniendo que reprogramar funcionalidades que se consideraban ya completas.

AGRADECIMIENTOS

REFERENCIAS

- [1] *David H. Shepard*. Character reading techniques. <https://patents.justia.com/patent/4021777>
- [2] *Musitek Corporation*. SmartScore. <https://www.musitek.com/>
- [3] *flat.io* <https://flat.io>
- [4] *Noteflight*. <https://www.noteflight.com/>
- [5] *Make Music*. MusicXML. <https://www.musicxml.com/>
- [6] *Mohit Muthanna Cheppudira*. VexFlow www.vexflow.com
- [7] *Evan You*. Vue.js <https://vuejs.org/>
- [8] *Visual Paradigm*. <https://www.visual-paradigm.com/>
- [9] *GIMP*. <http://www.gimp.org>
- [10] *Xtensio*. [https://xtensio.com/](https://xtensio.com)
- [11] *OpenJS Foundation*. Express. expressjs.com
- [12] *Richard Girges and Roman Burunkov*. Express-fileupload. <https://github.com/richardgirges/express-fileupload>
- [13] *Douglas Christopher Wilson*. Express-session. <https://github.com/expressjs/session>
- [14] *Nicholas Campbell*. Bcrypt. <https://github.com/kelektiv/node.bcrypt.js>
- [15] *Felix Geisendorfer and contributors*. MySQL. <https://github.com/mysqljs/mysql>

- [16] Document. <https://developer.mozilla.org/en-US/docs/Web/API/Document>
- [17] Arnau Baro-Mas. Recognition of handwritten music scores. http://www.cvc.uab.es/people/aforres/students/TFG_2016_ABaro.pdf
- [18] Arnau Baro-Mas. Optical Music Recognition by Long Short-Term Memory Recurrent Neural Networks. http://www.cvc.uab.es/people/aforres/students/Master_2017_ABaro.pdf
- [19] Alicia Fornés Bisquerra. Analysis of Old Handwritten Musical Scores. <http://www.cvc.uab.es/people/aforres/publi/AFornes-Master.pdf>
- [20] Alicia Fornés Bisquerra. Writer Identification by a Combination of Graphical Features in the Framework of Old Handwritten Music Scores. <http://www.cvc.uab.es/people/aforres/publi/PhDAliciaFornes.pdf>
- [21] OMR Research. <https://omr-research.net/>

APÉNDICE

A.1. Planificación

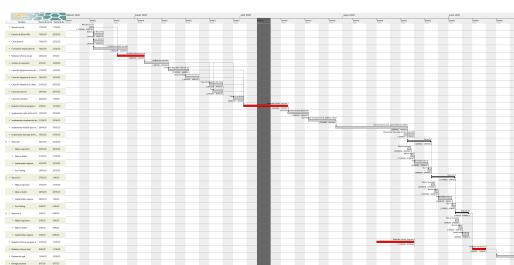


Fig. 10: Diagrama de Gantt detallado.

A.2. Diseño

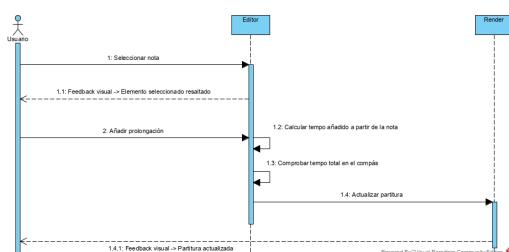


Fig. 11: Diagrama de Secuencia: Añadir prolongación.

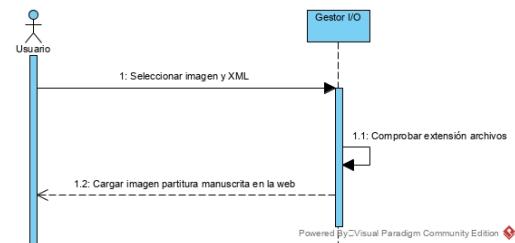


Fig. 12: Diagrama de Secuencia: Cargar partitura.

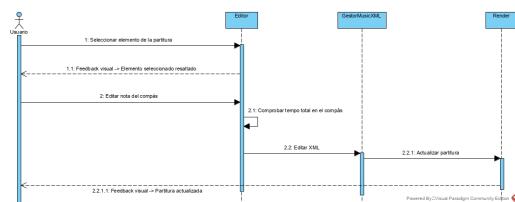


Fig. 13: Diagrama de Secuencia: Editar nota.

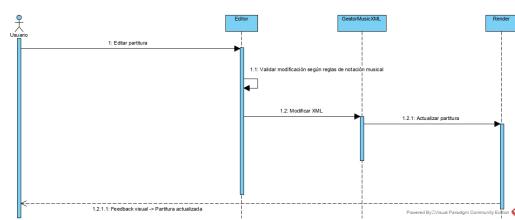


Fig. 14: Diagrama de Secuencia: Editar partitura.

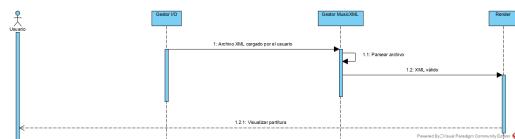


Fig. 15: Diagrama de Secuencia: Generar XML.

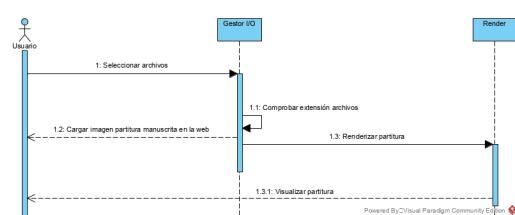


Fig. 16: Diagrama de Secuencia: Subir archivos.

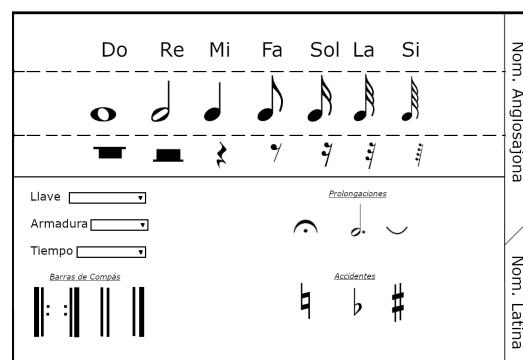


Fig. 17: Prototipo de la interfaz de usuario.

A.3. Análisis de Requisitos

Dissatisfiers	
ID	Descripción
DS-1	Subir un archivo MusicXML a la aplicación y su correspondiente imagen de la partitura escaneada.
DS-2	Visualizar la partitura e imagen en la pantalla.
DS-3	Seleccionar elemento que se quiere modificar con un click de ratón.
DS-4	Añadir, modificar o eliminar nota al compás, seleccionando <i>tempo</i> y nota.
DS-5	Seleccionar tipo de armadura. Puede tenerla o no.
DS-6	Seleccionar clave: Sol (G), Fa(B) y Do(C) y sus variantes.
DS-7	Seleccionar la signatura de compás: $\frac{4}{4}$, $\frac{3}{4}$, etc.
DS-8	Añadir, modificar o eliminar silencios.
DS-9	Exportar cambios realizados en la partitura a un archivo MusicXML.

TABLA 3: DISSATISFIERS SEGÚN EL MODELO DE KANO.

Satisfiers	
ID	Descripción
S-1	Cambiar tipo de barra del compás: simple, doble o de repetición.
S-2	Añadir o quitar a nota existente un puntillo (.)
S-3	Añadir ligadura entre dos o más notas.
S-4	Añadir o quitar accidente a una nota: sostenido (#), bemol (b) y becuadro (bz).

TABLA 5: SATISFIERS SEGÚN EL MODELO DE KANO.

Delighters	
ID	Descripción
D-1	Seleccionar notación musical: anglosajona (A, B, C, D, E, F, G) o latina (do, re, mi, fa, sol, la, si).

TABLA 4: DELIGHTERS SEGÚN EL MODELO DE KANO.