UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Gº en Ingeniería Informática

TFG Ingeniería Informática:

Digitalización de Partituras Antiguas

Presentado por Martín González Saiz
en Burgos el 7 julio de 2024
Tutor D. José Manuel Galán Ordax

Índice General

Índice de Figuras

Índice de tablas

Anexo A Planificación del proyecto

1.1 Introducción

En este apartado de los anexos se va a explicar detalladamente el desarrollo del proyecto durante los sprints así como su viabilidad económica.

1.2 Planificación temporal

En cada uno de los sprints ha habido una reunión previa en la que se ha acordado las tareas y objetivos a alcanzar. Para llevar a cabo esta gestión se ha utilizado el gestor de proyectos Trello, para administrar tanto la categoría, como el estado de cada tarea.

- Sprint 1: https://trello.com/b/s2npoqgv/tfg-digitalizacion-de-partituras-sprint-1-15-02-2024-01-03-2024
- Sprint 2: https://trello.com/b/SEnQJM6h/tfg-digitalizacion-de-partituras-sprint-2-01-03-2024-17-03-2024
- Sprint 3: https://trello.com/b/5qrTSteY/tfg-digitalizacion-de-partituras-sprint-3-18-03-2024-15-04-2024
- Sprint 4: https://trello.com/b/W4oJQ7Ah/tfg-digitalizacion-de-partituras-sprint-4-15-04-2024
- Sprint 5: https://trello.com/b/XWatG1p6/tfg-digitalizacion-de-partituras-sprint-5-23-05-2024-20-06-2024
- Sprint 6: https://trello.com/b/OuDMWHQl/tfg-digitalizacion-de-partituras-sprint-6-21-06-2024

Al inicio del proyecto se plantearon unos objetivos a alcanzar y como podría llegar a ser el desarrollo, así como las posibles herramientas a utilizar. Con ello se empezó la primera reunión en la que se centró su enfoque en los preparativos del proyecto.

1.2.1 Sprint 1: 15/02/2024 - 01/03/2024

En la primera reunión para el primer sprint planeamos y acordamos preparar todos los repositorios en cada una de las plataformas, además de todos los programas, que son utilizados a lo largo del desarrollo del proyecto. Por tanto en estas semanas creé repositorios para GitHub y Trello.

GitHub para poder reflejar la evolución del trabajo a nivel de código y programación así como para tener un buen control de las versiones y cambios, mientras que Trello reflejaría el trabajo que hay detrás de todo el tiempo de investigación, desarrollo y estudio de diferentes aspectos del proyecto.

Además instalé y configuré Mendeley para poder registrar todas las referencias a las que haya accedido para posteriormente poder implementarlo en la memoria. También me serviría para poder acceder a paginas útiles siempre que quiera durante el desarrollo de la herramienta.

Otro aspecto importante fue en estudio del entorno en el que se utilizaría la aplicación. Hice una investigación para elegir entre una aplicación web o una de escritorio, valorando sus ventajas e inconvenientes.

Por último me decanté por desarrollar la memoria en Word por su simplicidad en comparación con Latex.

Tabla 1 Sprint 1

Fecha	Tarea	Etiqueta	Estado	Tiempo estimado	Tiempo real
16/02/2024	Creación y preparación de la tabla de Trello	Configuration	Completado	0.5	0.25
19/02/2024	Creación del repositorio de GitHub	Configuration	Completado	0.5	0.25
20/02/2024	Instalación y configuración de Mendeley	Configuration	Completado	1	1
21/02/2024	Estudio de GitHub	Research	En curso	1.5	2
22/02/2024	Estudio de GitHub	Research	En curso	1.5	1.5
23/02/2024	Estudio de GitHub	Research	Completado	1	1
26/02/2024	Investigar sobre aplicaciones web	Research	Completado	2	1.5
27/02/2024	Investigar sobre aplicaciones escritorio	Research	Completado	2	1.5
28/02/2024	Investigación general de herramientas OMR	Research	Completado	2	1.5
29/02/2024	Decidir entorno en el que desarrollar la memoria	Research	Completado	1	0.75

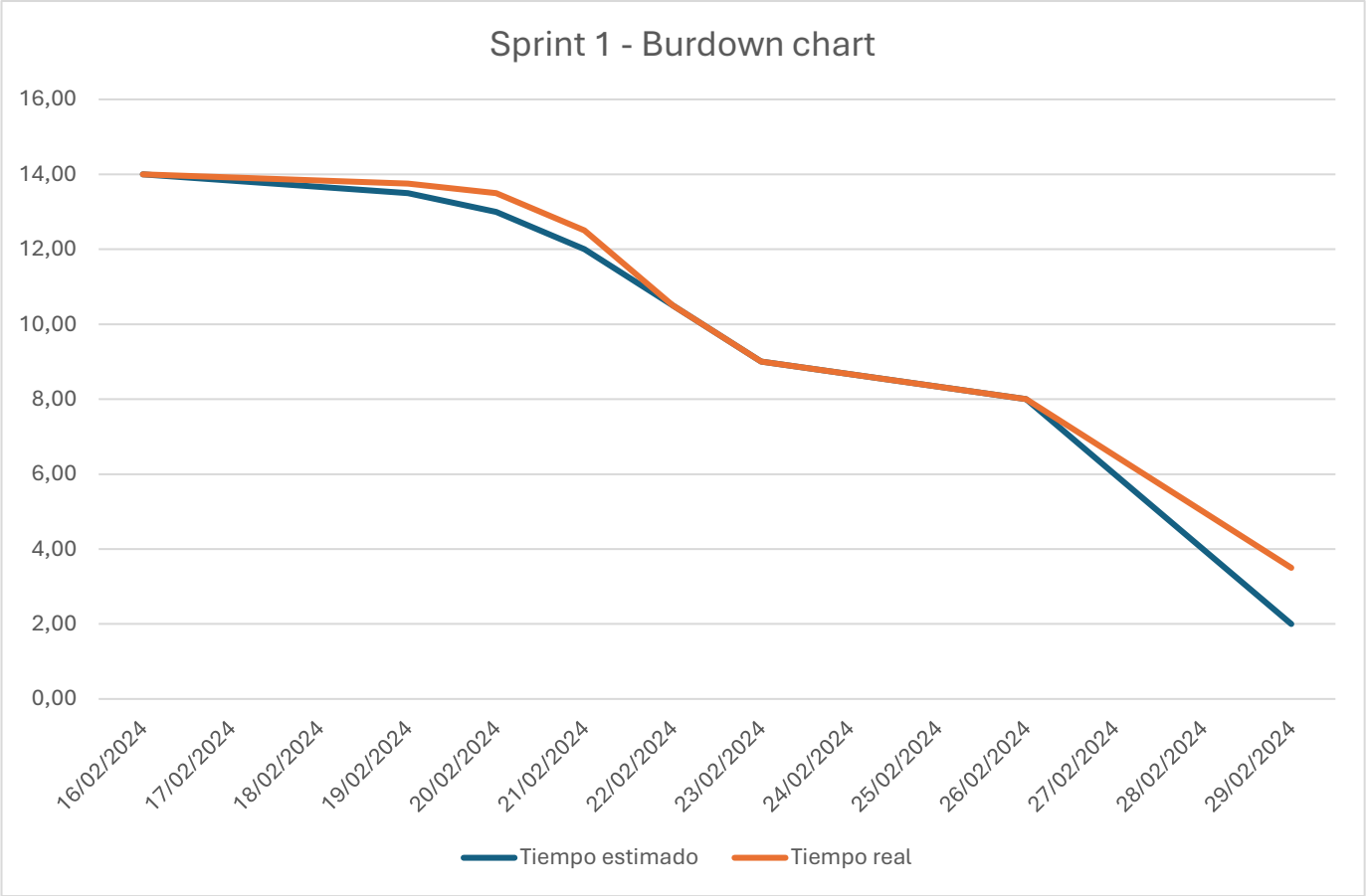

Figura 1 Burndown Report Sprint 1

1.2.2 Sprint 2: 01/03/2024 –17/03/2024

Este sprint se dedicó el trabajo a la investigación y el estudio de las herramientas a utilizar en el proyecto para levantar un servidor web y desarrollar la aplicación.

Entre ellas se valoró Flask o Django. Flask ofrece mucha simplicidad y flexibilidad dejando libertad para la implementación de otras herramientas y librerías, además, su aprendizaje es muy rápido permitiendo tener un mayor control en el desarrollo de la aplicación. Sin embargo, podría llegar a ser un problema si el proyecto crece lo suficiente con demasiadas funcionalidades, ya que habría que implementar muchas características. Por otro lado, Django es un framework que ofrece una amplia variedad de funcionalidades integradas de serie, también no permite un desarrollo tan rápido ya que está enfocado en aplicaciones complejas haciendo uso de las características de este framework, así como escalabilidad para cualquier proyecto. Valorando todos estos aspectos y analizando la presente y futura situación he considerado que la mejor opción es usar Flask para mi proyecto.

También se realizó una investigación de los paquetes OMR más sólidos y de código abierto en la actualidad. La elección más llamativa es Audiveris debido a su larga trayectoria y la comunidad que hay detrás ya que es muy activa y grande, además de por su amplia documentación, explicado cada punto con detalle, pero el motivo principal es que se considera la OMR con mejores resultados. Otras tecnologías que se valoró son SharpEye Music Reader, MuseScore.

En este sprint también se cambió el plan estructural de las tablas de Trello y la base de datos donde guardo las referencias de Mendeley a Zotero.

Tabla 2 Sprint 2

Fecha	Tarea	Etiqueta	Estado	Tiempo estimado	Tiempo real
06/03/2024	Cambiar el plan estructural de Trello	Documentation	Completado	0.5	0.3
07/03/2024	Cambiar la base de datos de referencias de Mendeley a Zotero	Documentation	Completado	0.25	0.25
10/03/2024	Documentar Sprint 1	Documentation	En curso	1	2
11/03/2024	Documentar Sprint 1	Documentation	Completado	1	1
12/03/2024	Documentar Sprint 2 y avances de los paquetes OMR en la memoria	Documentation	En curso	1	1.5
12/03/2024	Investigar servidor web: Django vs Flask	Research	En curso	1	1
13/03/2024	Investigar servidor web: Django vs Flask	Research	En curso	2	2
14/03/2024	Investigar servidor web: Django vs Flask	Research	Completado	1	2
16/03/2024	Investigar paquetes OMR para mi proyecto y elegir el más ideal	Research	Completado	2	2.5

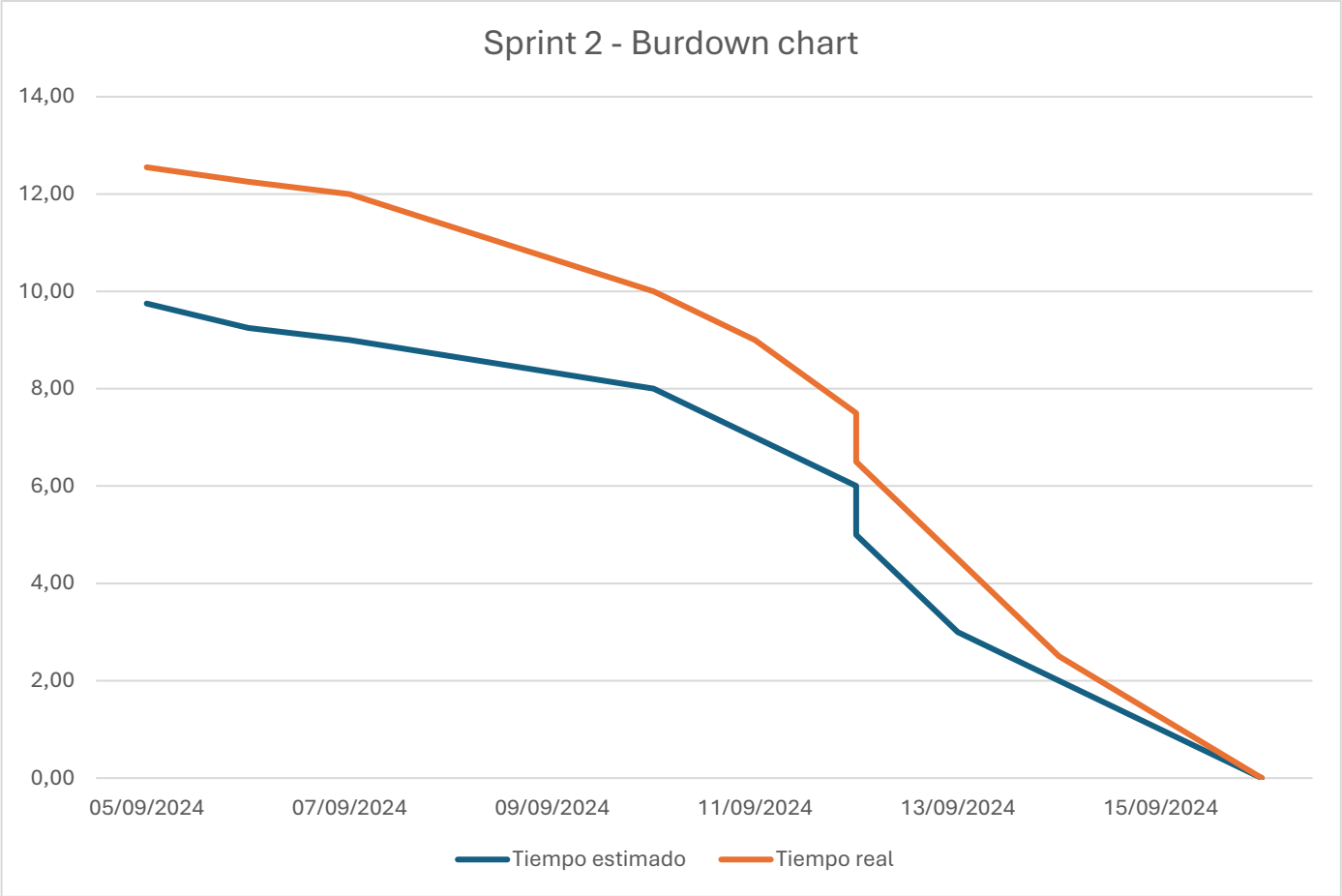

Figura 2 Burndown Report Sprint 2

1.2.3 Sprint 3: 18/03/2024 – 15/04/2024

En esta etapa se planteó comenzar con el desarrollo de la aplicación web, enfocando el trabajo en las funcionalidades más básicas de esta.

En primer lugar se planteó una primera versión de los tipos de usuarios que habría en la aplicación, empezando por un usuario corriente y un usuario registrado. Además se asignó unas primeras funcionalidades básicas a estos usuarios. El usuario sin registrarse podría entrar en la ventana home, iniciar sesión y registrarse y en el caso del usuario registrado, podría subir ficheros y cerrar sesión como funciones extra además de las funciones nuevas que se añadirían.

Tras ello se creó un entorno virtual en el que se trabajaría para el desarrollo de la aplicación. Se tomó esta decisión con el fin de evitar conflictos de dependencias de paquetes y versiones con otros proyectos.

Después se creó una primera versión del proyecto siguiendo una estructura de ficheros común en este tipo de proyectos, luego se instaló Flask. Gestionar el fichero de configuración en este punto fue clave para poder hacer las migraciones del modelo de las tablas a la base de datos.

Se hizo una primera implementación de un login y register haciendo uso de la librería de FlaskWTF para facilitar hacer los forms. Como se tuvo problemas con el uso adecuado de esta herramienta se cambió el código, retirando esta librería.

Para poder almacenar las entradas en base de datos los usuarios registrados se instaló xampp y se configuró la base de datos en phpadmin. Se eligió este entorno debido a su facilidad de uso y su libre acceso a cualquier navegador web.

Para reflejar los modelos creados en código en base de datos, se usó migraciones mediante comandos en la terminal para que las tablas se crearan automáticamente. Estos pasos dieron varios problemas debido a que la carpeta de migraciones almacenó más de un cambio y se solaparon entre ellos. Además los ficheros de la caché almacenaban datos antiguos apareciendo atributos fantasma. Para arreglarlo se borró la carpeta y se recreó la base de datos.

Por último se terminó de vincular las ventanas y de estructurar la aplicación.

Tabla 3 Sprint 3

Fecha	Tarea	Etiqueta	Estado	Tiempo estimado	Tiempo real
19/03/2024	Documentar Sprint 2	Documentation	Completado	1	1.5
20/03/2024	Cambiar las gráficas de los sprints	Documentation	Completado	1	1
21/03/2024	Plantear los tipos de usuarios, sus roles y funcionalidades	Research	Completado	4	0.5
21/03/2024	Preparar la aplicación web	Development	Completado	2	1.5
24/03/2024	Desarrollo de ventana login y register	Development	En curso	2	2
01/04/2024	Desarrollo de ventana login y register	Development	En curso	2	1.6
02/04/2024	Desarrollo de ventana login y register	Development	En curso	3	3
03/04/2024	Instalación e implementación de xampp	Configuration	Completado	1	1.5
03/04/2024	Instalación e implementación de phpMyAdmin	Configuration	Completado	2	2
04/04/2024	Desarrollo de ventana login y register	Development	En curso	2.5	2.5
07/04/2024	Desarrollo de ventana login y register	Development	En curso	2	2

08/04/2024	Desarrollo de ventana login y register	Development	En curso	3	3
09/04/2024	Desarrollo de ventana login y register	Development	En curso	3	3
10/04/2024	Desarrollo de ventana login y register	Development	Completado	1	1
10/04/2024	Vinculación entre ventanas	Development	Completado	1	1
10/04/2024	Desarrollo de registro de partituras	Development	Completado	1	1
11/04/2024	Gestión de usuarios y sesiones	Development	Completado	2	2
15/04/2024	Documentación del sprint 3	Documentation	Completado	3	3

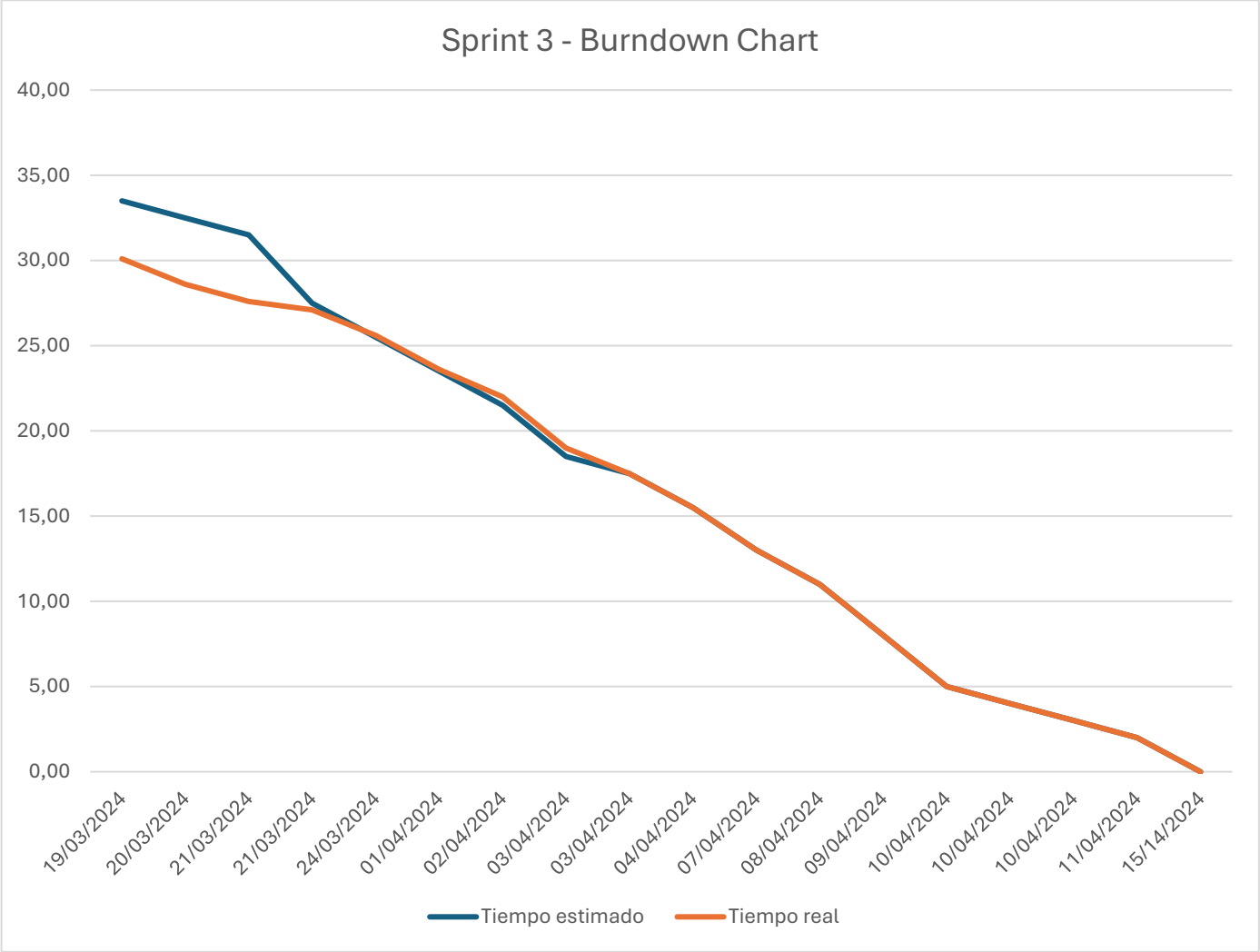

Figura.3 Burndown Report Sprint 3

1.2.4 Sprint 4: 15/04/2024 – 22/05/2024

Este sprint está enfocado en la importación de la tecnología OMR Audiveris en la aplicación.

Para ello, primero, se tuvo que investigar y revisar bien la documentación que aportan los autores de la herramienta o los desarrolladores interesados en esta. Se clonó la herramienta en el sistema y se ejecutó.

Se tuvieron problemas para ejecutarlo porque el nombre de los ficheros y archivos no se correspondían del todo con los ejemplos de la documentación.

Se analizó e investigó los diferentes aspectos del programa para saber cómo se trabaja con esta, a la par que fue necesaria una documentación con el pdf de ayuda para los desarrolladores.

Luego se tuvo que cambiar el formato en el que se aceptaba los archivos subidos a la aplicación (solo se podía pdfs) para darle flexibilidad a la aplicación.

Se creó un proyecto en Google Cloud para tener un servicio de almacenamiento para las partituras en la nube, que dio alguna dificultad adaptarlo al proyecto por lo que se desechó la idea por no malgastar tiempo.

Se intentó encapsular el proyecto junto con audiveris para moverlo a un servicio de nube que desplegara el proyecto públicamente para abrir su acceso a los usuarios, también aseguraría que el comportamiento es el mismo en un entorno de producción y localmente.

Se tuvo que registrar todas las dependencias del proyecto y registrar las instalaciones del terminal del docker en un fichero que ejecutaría más tarde. Se tuvieron problemas por la compatibilidad de los paquetes al construir el docker y, tras construirlo, el acceso a las rutas de las carpetas tuvo muchas ambigüedades, las cuales se tuvieron que resolver.

Otro inconveniente fue que en un inicio por cada cambio en el código del proyecto se tuvieron que construir nuevos contenedores e imágenes, por lo que el nombre de estos se pisaban, ralentizaban el dispositivo y colapsaron el almacenamiento. Se dejó aparcada la idea debido a la ralentización del proceso y la incertidumbre de si llegaría a algo funcional.

Se volvió con la programación de una función que ejecutara Audiveris como un miniservicio de la aplicación en segundo plano. Fue necesaria una documentación de las vías de ejecución que tiene Audiveris en su documentación para construir un comando que ejecutara el servicio sin interfaz de usuario, hiciera el proceso de digitalización y exportara el fichero resultante.

La construcción de este comando nuevamente dio problemas que llevaron un tiempo resolver. Esto se debe a que la ubicación de los ficheros de ejecución de java (.jar) no eran accesibles para la carpeta del proyecto. Se clonó el repositorio de Audiveris a el directorio de la aplicación web para facilitar el acceso. También se tuvo que cambiar la versión de la carpeta jdk de Java a la 17 y configurarlo en las variables de entorno para asegurar un buen comportamiento, como indican las especificaciones de la documentación de Audiveris.

Se mejoraron las vistas de la aplicación web con bootstrap para darle un mejor diseño a la aplicación y un toque más profesional.

Se exploraron las posibilidades para el acceso a metadatos en base de datos en Firebase. La configuración y adaptación de este servicio al proyecto llevaron tiempo, debido a las dificultades que se presentaron para la importación de paquetes así como adaptarme a la nueva estructura, sintaxis y funciones de este.

Tabla 4 Sprint 4

Fecha	Tarea	Etiqueta	Estado	Tiempo estimado	Tiempo real
16/04/2024	Documentación del sprint 3 y cambios en las tablas y gráficos	Documentation	Completado	0.5	0.5
23/04/2024	Investigación para la implementación de la tecnología OMR	Research	En curso	2	2
24/04/2024	Investigación para la implementación de la tecnología OMR	Research	En curso	2	2
25/04/2024	Investigación para la implementación de la tecnología OMR	Research	En curso	2	2
27/04/2024	Subir archivos en todos los formatos	Development	Completado	2	2
27/04/2024	Investigación para la implementación de la tecnología OMR	Research	En curso	3	3
30/04/2024	Investigación para la implementación de la tecnología OMR	Research	En curso	1.5	1
01/05/2024	Investigación para la implementación de la tecnología OMR	Research	En curso	4	4
03/05/2024	Implementación de servicio de almacenamiento	Development	En curso	3	2
04/05/2024	Implementación de servicio de almacenamiento	Development	En curso	5	5
05/05/2024	Implementación de un docker para Audiveris	Development	En curso	3	3

06/05/2024	Implementación de un docker para Audiveris	Development	En curso	5.5	5
07/05/2024	Implementación de un docker para Audiveris	Development	En curso	3	3
08/05/2024	Implementación de un docker para Audiveris	Development	En curso	2	2
09/05/2024	Implementación de un docker para Audiveris	Development	En curso	2.5	2.5
11/05/2024	Implementación de un docker para Audiveris	Development	Completado	1	1
11/05/2024	Implementación de una funcionalidad que ejecute audiveris	Development	En curso	2	2.5
12/05/2024	Implementación de una funcionalidad que ejecute audiveris	Development	En curso	7.5	8
13/05/2024	Implementación de una funcionalidad que ejecute audiveris	Development	En curso	5	5
14/05/2024	Implementación de una funcionalidad que ejecute audiveris	Development	En curso	3.5	3.5
15/05/2024	Implementación de una funcionalidad que ejecute audiveris	Development	En curso	4	4
17/05/2024	Implementación de una funcionalidad que ejecute audiveris	Development	En curso	3	3
18/05/2024	Implementación de una funcionalidad que ejecute audiveris	Development	Completado	3	4.5
18/05/2024	Mejora de las vistas	Development	Completado	2	2
18/05/2024	Implementación de Firebase	Research	En curso	2	2
19/05/2024	Implementación de Firebase	Development	En curso	2	2
19/05/2024	Implementación de Firebase	Development	En curso	3	3
20/05/2024	Implementación de Firebase	Development	En curso	1.5	2
20/05/2024	Implementación de Firebase	Development	En curso	3	5.5

21/05/2024	Implementación de Firebase	Development	En curso	5	5
22/05/2024	Implementación de Firebase	Development	Completado	5	7

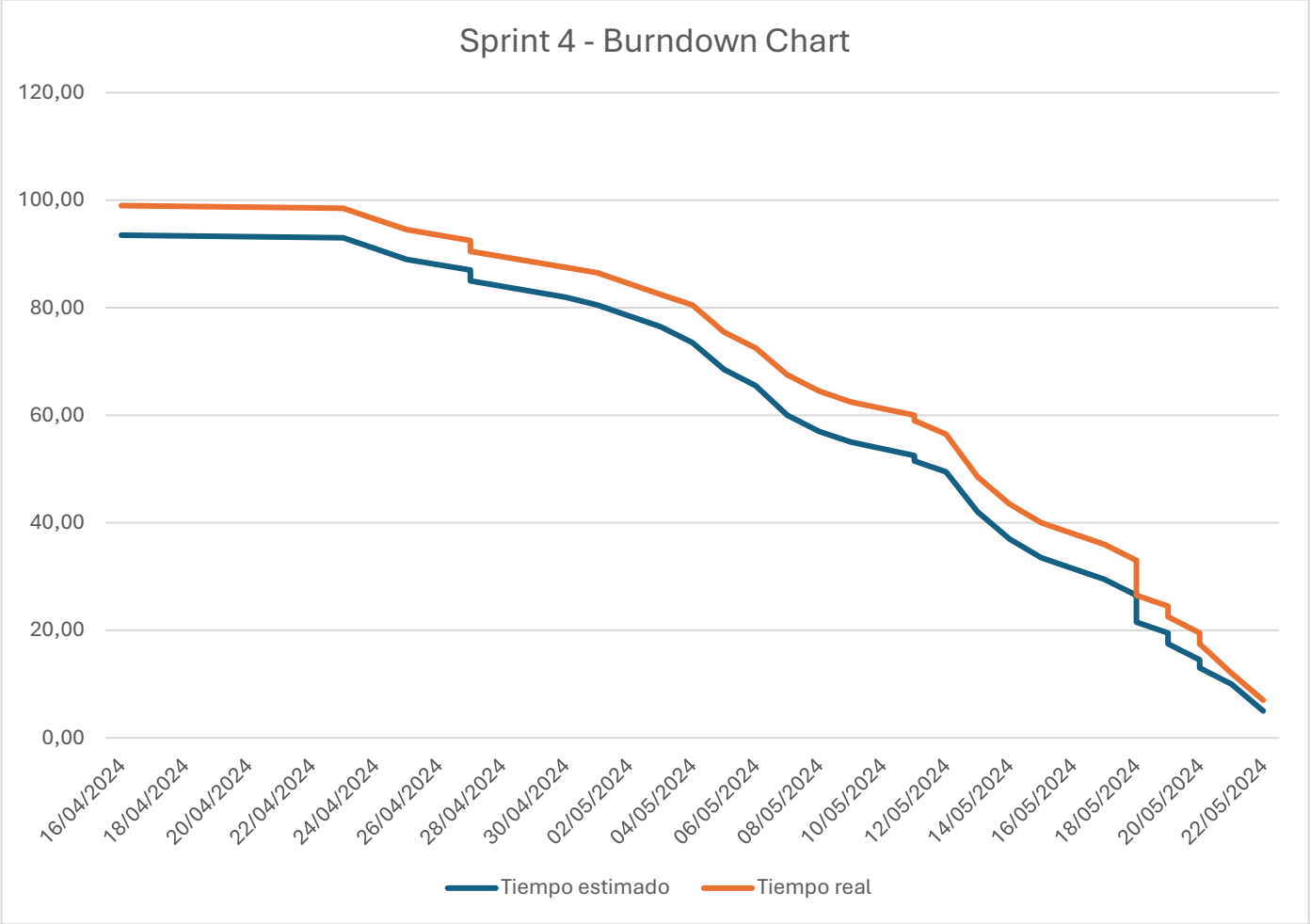

Figura 4 Burndown Report Sprint 4

1.2.5 Sprint 5: 23/05/2024 – 20/06/2024

En este sprint se valoraron otras alternativas para la digitalización de partituras. Se consideró hacer uso de Firebase ML que permite integrar aprendizaje automático en aplicaciones web. De esta forma se podría implementar en la web para que, en vez de usar Audiveris para transcribir las partituras, usara esta IA. Se valoró la posibilidad de usar un modelo preentrenado con esta funcionalidad para facilitar el proceso, pero no se obtuvieron resultados viables. Se investigó como se podría entrenar una en base a las preferencias, pero no se llevó adelante debido a la falta de experiencia en este sector y el tiempo que llevaría desarrollarlo y en base al tiempo que quedaba.

También se empezó la documentación de la memoria empezando por puntos esenciales y sencillos de implementar, como los objetivos de desarrollo sostenible o la introducción.

Se retomó la funcionalidad de ejecutar Audiveris construyendo un comando que se ejecutaría como batch en segundo plano, además también se replanteó la estructura de las funciones

de cada archivo del directorio, ordenando las definiciones, evitando que Firebase se inicializara más veces de las necesarias.

El siguiente paso fue implementar una funcionalidad que permitiera visualizar las partituras que se consiguieran digitalizar. Para ello se usó Verovio, instalándolo junto con sus dependencias en la máquina. Se tuvieron bastantes problemas en este punto ya que hubo muchos conflictos con la instalación de las dependencias y con las correcta configuración y establecimiento en las variables de entorno. Se incluyo la funcionalidad de visualizar las partituras y Verovio en las vistas con bibliotecas, mostrándose las partituras mediante bloques y scripts.

Luego se implementó una funcionalidad para que el usuario pudiera eliminar las partituras que desee en el listado de partituras registradas en la aplicación y Firebase. Se aparcó la implementación debido a que dio problemas con el acceso al bucket y al servicio de almacenamiento de Firebase.

Se añadieron alertas a las funcionalidades de la aplicación para que el usuario se mantuviera informado acerca de los procesos fallidos al intentar ciertas interacciones.

Aprovechando esta última implementación, se añadió una funcionalidad para que, en caso de que la digitalización de una partitura fallé, el usuario pueda alterar la imagen llegando a un proceso de preprocesamiento, facilitando a Audiveris poder procesar la imagen y llegar a una digitalización completa. Esta funcionalidad consiste en que el usuario inserta valores a funciones de preprocesamiento.

Por último, se empezó el desarrollo para desplegar la aplicación públicamente. Para ello se tomó dos caminos, el primero fue desplegar la aplicación haciendo uso de un archivo Procfile y buildpacks en Heroku. Esta opción dio problemas debido a que Heroku no permite la ejecución de scripts en tiempo de compilación sin configurar sus buildpacks, y ello conllevaba realizar las instalaciones necesarias de Audiveris en estos. Además los archivos que se generaban con los scripts no persistían por alguna razón. La opción que dio resultados fue utilizar el Docker creado para tener un mejor control del entorno de ejecución y usar Heroku para configurar la aplicación y desplegarla. También dio problemas por no estar bien adaptada al contexto la inicialización de la aplicación en código y los errores en las interpretaciones del puerto en el Dockerfile al desplegar la aplicación con Heroku. El despliegue está terminado pero la funcionalidad de digitalizar las partituras no se puede completar en la versión desplegada de Heroku ya que sobrepasa la memoria permitida en el plan gratuito.

Tabla 5 Sprint 5

Fecha	Tarea	Etiqueta	Estado	Tiempo estimado	Tiempo real
23/05/2024	Documentar sprint 4	Documentation	Completado	2	3
24/05/2024	Investigar herramientas de procesamiento de imágenes para Firebase	Research	En curso	0.4	0.4
24/05/2024	Investigar herramientas de procesamiento de imágenes para Firebase	Research	Completado	1	1

25/05/2024	Documentación de la memoria	Documentation	En curso	1	1.5
26/05/2024	Reimplementación de un docker	Development	Completado	1	2
26/05/2024	Documentación de la memoria	Documentation	En curso	1	1
27/05/2024	Reimplementación de audiveris	Development	En curso	2	2
28/05/2024	Reimplementación de audiveris	Development	Completado	2	3
28/05/2024	Reestructuración para Firebase	Development	Completado	2	3.5
31/05/2024	Documentar la memoria	Documentation	En curso	1	2.5
01/06/2024	Implementación de Verovio	Development	En curso	3	3
02/06/2024	Implementación de Verovio	Development	En curso	2	2
03/06/2024	Implementación de Verovio	Development	En curso	2	2
06/06/2024	Implementación de Verovio	Development	En curso	6	6
07/06/2024	Implementación de Verovio	Development	En curso	7	7
08/06/2024	Implementación de Verovio	Development	En curso	8	10
09/06/2024	Implementación de Verovio	Development	Completado	9	5
09/06/2024	Funcionalidad para eliminar partituras	Development	Bloqueado	2	2
10/06/2024	Arreglos en la aplicación	Development	Completado	3	6.5
11/06/2024	Alertas en las vistas	Development	Completado	4	6
12/06/2024	Función de preprocesamiento	Development	Completado	4	5.5
13/06/2024	Despliegue de la aplicación	Development	En curso	5	8
14/06/2024	Despliegue de la aplicación	Development	En curso	4	8
15/06/2024	Despliegue de la aplicación	Development	En curso	5	10
16/06/2024	Despliegue de la aplicación	Development	Bloqueado	5	7.5
17/06/2024	Documentación del sprint 5	Documentation	Completado	3	5
18/06/2024	Documentación de la memoria	Documentation	En curso	5	7
19/06/2024	Documentación de la memoria	Documentation	En curso	7.5	7.5

20/06/2024	Funcionalidad para eliminar partituras	Development	Completado	3	3
20/06/2024	Documentación de la memoria	Documentation	En curso	4	4

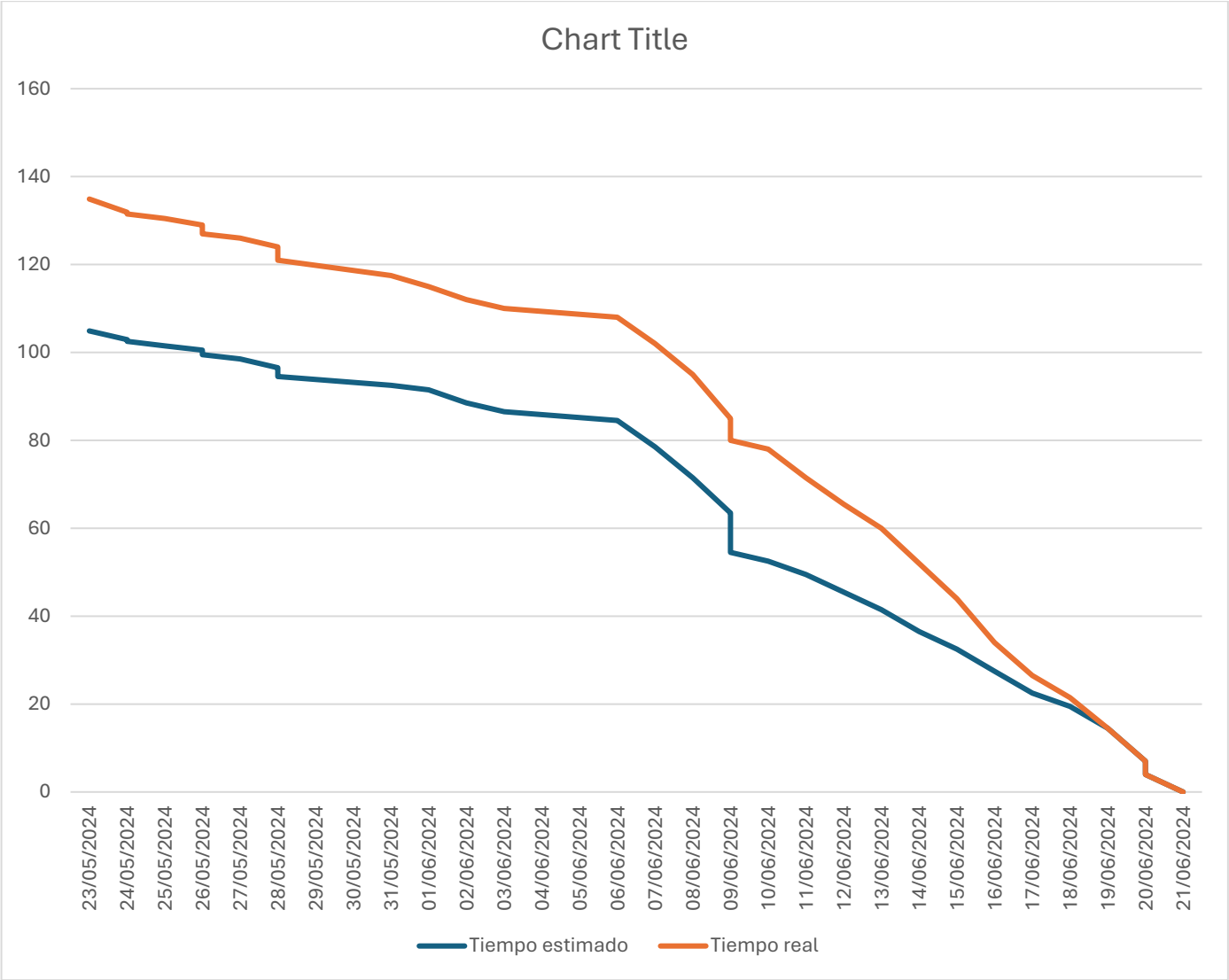

Figura 5 Burndown Report Sprint 5

1.2.6 Sprint 6: 21/06/2024 –08/07/2024

En esté sprint se planteó principalmente acabar con toda la documentación referente al proyecto. Además, se dedicó trabajo a la estilización de las vistas para dar un toque más atractivo para el usuario a la aplicación. Otra línea de trabajo que se realizó fue la simulación de pruebas en la web para la verificación del correcto funcionamiento de todas las funcionalidades. Además, también se planteó ordenar y comentar el código de la aplicación.

Tabla 6 Sprint 6

	Tarea	Etiqueta	Estado	Tiempo estimado	Tiempo real
21/06/2024	Documentación de la memoria	Documentation	En curso	8	8
22/06/2024	Documentación de la memoria	Documentation	En curso	8	9
23/06/2024	Documentación de la memoria	Documentation	En curso	10	10
24/06/2024	Documentación de la memoria	Documentation	En curso	5	6.5
26/06/2024	Documentación de la memoria	Documentation	En curso	6.5	6
27/06/2024	Documentación de la memoria	Documentation	En curso	7	7.5
28/06/2024	Documentación de la memoria	Documentation	En curso	7	9
29/06/2024	Documentación de la memoria	Documentation	Completado	1	3
29/06/2024	Documentación de los anexos	Documentation	En curso	5	5
30/06/2024	Documentación de los anexos	Documentation	En curso	7	7
01/07/2024	Documentación de los anexos	Documentation	En curso	7	7
02/07/2024	Documentación de los anexos	Documentation	En curso	7	8.5
03/07/2024	Estelización de las vistas	Development	Completado	6	9
04/07/2024	Documentación de los anexos	Documentation	En curso	4	4
04/07/2024	Arreglos en código	Development	Completado	3	3
05/07/2024	Comentar el código	Development	Completado	1	1
05/07/2024	Documentación de los anexos	Development	En curso	8	8
06/07/2024	Documentación de los anexos	Development	En curso	8	8
07/07/2024	Documentación de los anexos	Development	Completado	10	14

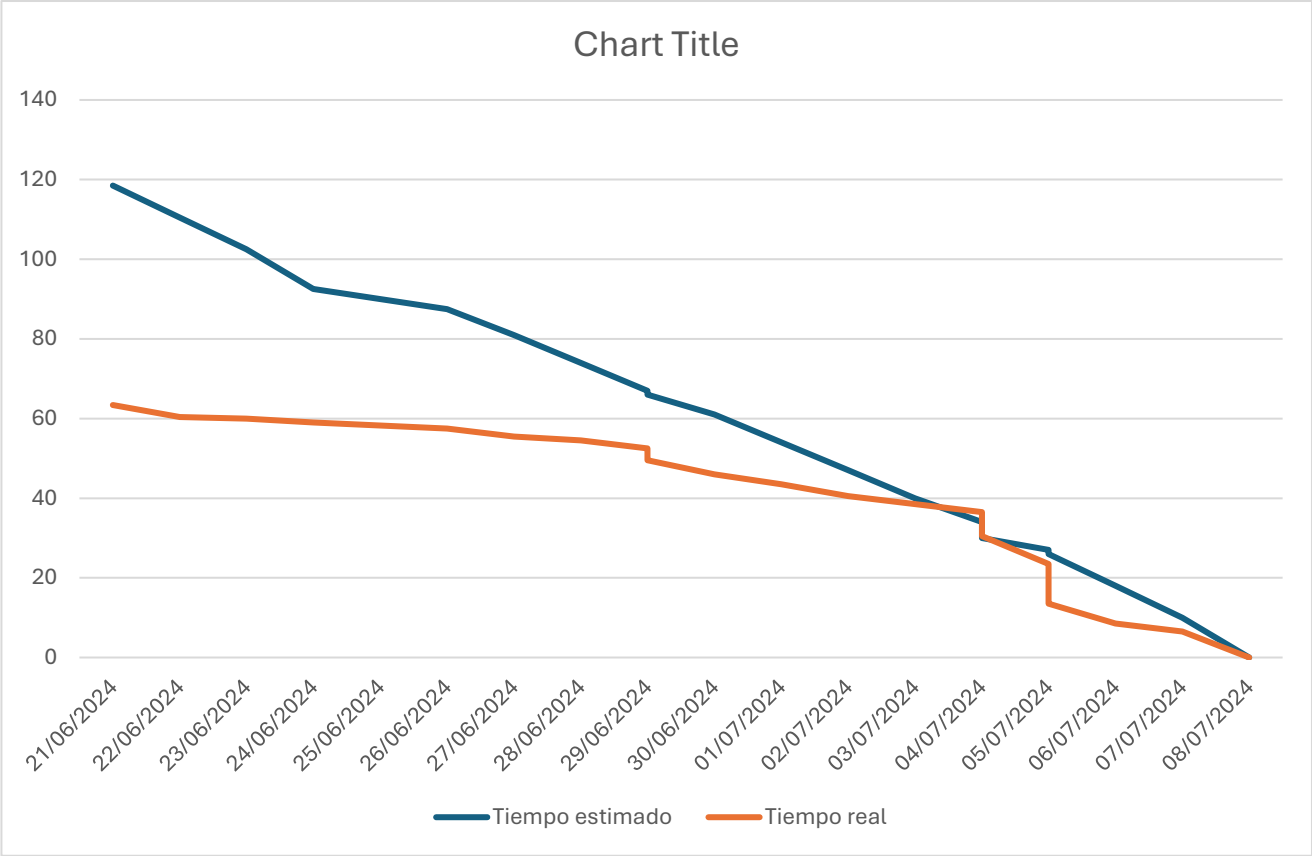

Figura 6 Burndown Report Sprint 6

1.2.7 Histórico de los sprints

Tabla 7 Sprint 7

Tarea	Etiqueta	Tiempo estimado	Tiempo real
Creación y preparación de la tabla de Trello	Configuration	0.5	0.25
Creación del repositorio de GitHub	Configuration	0.5	0.25
Instalación y configuración de Mendeley	Configuration	1	1
Estudio de GitHub	Research	4	4
Investigar sobre aplicaciones web	Research	2	1.5
Investigar sobre aplicaciones escritorio	Research	2	1.5

Investigación general de herramientas OMR	Research	2	1.5
Decidir entorno en el que desarrollar la memoria	Research	1	0.75
Cambiar el plan estructural de Trello	Documentation	0.5	0.3
Cambiar la base de datos de referencias de Mendeley a Zotero	Documentation	0.25	0.25
Documentar Sprint 1	Documentation	2	3
Documentar Sprint 2	Documentation	1.5	2.25
Avances de los paquetes OMR	Documentation	0.5	0.75
Investigar servidor web: Django vs Flask	Research	4	5
Investigar paquetes OMR	Research	2	2.5
Cambiar las gráficas de los sprints	Documentation	1	1
Plantear los tipos de usuarios, sus roles y funcionalidades	Research	4	0.5
Preparar la aplicación web	Development	2	1.5
Desarrollo de ventana login y register	Development	17.5	18.1
Instalación e implementación de xampp	Configuration	1	1.5
Instalación e implementación de phpMyAdmin	Configuration	2	2
Vinculación entre ventanas	Development	1	1
Desarrollo de registro de partituras	Development	1	1
Gestión de usuarios y sesiones	Development	2	2
Documentación del sprint 3	Documentation	3	3
Documentación del sprint 3 y cambios en las tablas y gráficos	Documentation	0.5	0.5

Investigación para la implementación de la tecnología OMR	Research	14.5	14
Implementación de servicio de almacenamiento	Development	8	7
Implementación de un docker para Audiveris	Development	17	16.5
Implementación de una funcionalidad que ejecute audiveris	Development	28	30.5
Mejora de las vistas	Development	2	2
Implementación de Firebase	Development	21.5	26.5
Documentar sprint 4	Documentation	2	3
Investigar herramientas de procesamiento de imágenes para Firebase	Research	1.4	1.4
Reimplementación de un docker	Development	1	2
Reimplementación de audiveris	Development	4	5
Reestructuración para Firebase	Development	2	3.5
Implementación de Verovio	Development	37	35
Funcionalidad para eliminar partituras	Development	5	5
Arreglos en la aplicación	Development	3	6.5
Alertas en las vistas	Development	4	6
Función de preprocesamiento	Development	4	5.5
Despliegue de la aplicación	Development	19	13.5
Documentación del sprint 5	Documentation	3	5
Documentación de la memoria	Documentation	72	82.5
Documentación de los anexos	Documentation	57.5	62.5
Estelización de las vistas	Development	6	9

1.3 Viabilidad económica

En este apartado se va a evaluar la rentabilidad que supondría el desarrollo de este proyecto en el supuesto de que fuera producido empresarialmente. Para ello se van a analizar los costes en base a datos reales de proyectos cotidianos.

1.3.1 Coste personal

En este apartado se van a evaluar los costes de personal que tendría el proyecto en base a los empleados. En el desarrollo de este proyecto han entrado en juego un único desarrollador junior trabajando más o menos 4 meses, con un total de 398.8 horas. (1)

Se toma el salario promedio de un programador junior, que es 21000 € brutos al año, siendo el precio por hora de 10,77 €. Calculamos el precio total en función de las horas invertidas en el proyecto y concluimos con un total de 4295,076 €, que vendrían siendo 1073,769€ mensuales brutos. (2)

Ahora hay que añadir al análisis los impuestos con los que tiene que contar la empresa. En España, las empresas tienen que pagar impuestos a la Seguridad Social por cada empleado. Según la Seguridad Social, la cotización para la empresa se distribuye en los siguientes conceptos (3):

- Contingencias comunes: 23,60%
- Desempleo: 5,50%
- FOGASA: 0,20%
- Formación profesional: 0,60%
- Mecanismo de Equidad Intergeneracional (MEI): 0,58%

Impuestos = 4295,076 € × 0,3048 = 1308,7427 €

Por lo tanto, el costo total del proyecto, incluyendo los impuestos, sería:

Costo total del proyecto = 4295,076 € + 1308,7427 € = 5603,8187 €

1.3.2 Hardware

En la sección de costes de hardware, es necesario calcularlos en función del material usado en su desarrollo.

En cuanto al ordenador se ha hecho uso de un portátil Lenovo Ideapad Gaming de 829 € y de pantalla se ha usado uno monitor Philips de 90€. Calculando que la amortización de todo el material va a llegar a los 4 años, el cálculo ser

$$919 / 4 = 229,75$$

Se hace la operación correspondiente para sacar su costo por mes y se multiplica por el número de meses de desarrollo:

$$(229,75€ \text{ al año} / 12 \text{ meses}) * 4 \text{ meses} = 76 €$$

1.3.3 Software

En este apartado se va a evaluar el coste que ha tenido el uso del software.

Al haber usado librerías y bibliotecas totalmente gratuitas, en lo que respecta a esto el coste es 0 €, si bien algunas de las herramientas usadas tienen planes de pago, no han sido usados en el proyecto por lo que no se va a tener en cuento.

El software utilizado en el proyecto es totalmente gratuito y público, teniendo un coste total de 0 €.

1.3.4 Beneficios

Este proyecto ha sido desarrollado sin intención de sacar beneficio económico de él, cuyo fin es aumentar las posibilidades al estudio de material musical.

Tabla 8 Beneficios

Concepto	**Coste (€)**
Personal	5583,5988
Hardware	76
Software	0
Total	5659,5988€

1.4 Viabilidad legal

En esta sección se va a analizar profundamente las leyes y regulaciones que tiene que cumplir el proyecto en el desarrollo y en el uso de la aplicación.

Para ello hay que analizar cada una de las librerías y herramientas usadas en el proyecto y comprobar y estudiar su licencia.

Tabla 9 Viabilidad legal

Herramienta	**Versión**	**Licencia**
GitHub	2.43.0.	MIT License
Scrum	-	CC BY-SA 4.0
GitHub	-	MIT License
Trello	-	-
Microsoft Word	-	-
GitHub Desktop	-	-
Zotero	-	AGPL
Python	3.11.4	PSF License
Flask	3.0.2	BSD-3-Clause
Bootstrap	5.3.2	MIT
Cloud Firestore	2.16.0	-
Google Cloud Storage	2.16.0	-
Bcrypt	4.1.3	ISC
pdf2image	1.17.0	GPL
OpenCV	4.10.0.82	Apache 2.0
NumPy	1.26.4	BSD-3-Clause
Audiveris	5.1.1	AGPL
Verovio	3.9.0	LGPL
Docker	24.0.5	Apache 2.0

Prácticamente la totalidad de las librerías y herramientas del proyecto son de libre uso y públicas, aunque algunas de ellas se someten a ciertas condiciones que la aplicación debe cumplir para su uso. Las licencias más restrictivas de las anteriores son:

General Public License (GPL)

Esta licencia exige que el software que incorpore GPL sea liberado con la misma licencia cuando se distribuya, con sus modificaciones y extensiones.

Affero General Public License (AGPL)

Es muy similar a GPL, pero en este caso al hacer una modificación y uso del software bajo APGL debe tener disponible el código fuente modificado a los usuarios a través de la red.

Lesser General Public License (LGPL)

Es más flexible que GPL, siendo necesario simplemente que cualquier modificación en el software en cuestión tenga que ser liberada bajo la misma licencia.

Dado el uso de librerías bajo GPL y AGPL, el proyecto se distribuirá bajo la licencia GPL para cumplir con las obligaciones de compartir el código fuente modificado y mantener la coherencia con las licencias de las herramientas utilizadas.

Anexo B Especificación de Requisitos

2.1 Introducción

A continuación, se va a desarrollar el apartado referente a los objetivos y requisitos necesarios para el proyecto. De esta manera se podrán encontrar las funcionalidades necesarias para crear los casos de uso de la aplicación.

2.2 Objetivos generales

Para crear los casos de uso, primero es necesario tener claros los objetivos generales del proyecto. Los objetivos generales son los siguientes:

- Incorporar una autenticación segura y efectiva.
- Dar un servicio de almacenamiento de partituras online para cualquier usuario con una interacción cómoda y rápida.
- Implementar un proceso de transcripción de partituras eficiente a través del uso del software de reconocimiento óptico de música (OMR).
- Proporcionar un sistema de almacenamiento independiente para las partituras que han sido digitalizadas.
- Implementar una funcionalidad de eliminación de partituras tanto subidas sin digitalizar, como partituras digitalizadas.
- Implementar una opción de preprocesamiento manual mediante el uso de tecnologías de procesamiento de imágenes.
- Implementación de la posibilidad de visualización de las partituras digitalizadas.

2.3 Catálogo de requisitos

El siguiente paso va a ser definir los requisitos funcionales y no funcionales que tiene el proyecto.

2.3.1. Requisitos funcionales

- R.F-01. Inicio de sesión: El usuario debe poder iniciar sesión.
- R.F-02. Registro de usuario: El usuario debe poder registrarse.
- R.F-03. Cerrar sesión: La aplicación debe permitir al usuario cerrar sesión.
- R.F-04. Subir partituras: El usuario debe poder subir una partitura en cualquier formato.
- R.F-05. Listar las partituras registradas: La aplicación debe permitir al usuario ver su lista de partituras registradas.
- R.F-06. Listar las partituras digitalizadas: La aplicación debe permitir al usuario ver su lista de partituras digitalizadas.
- R.F-07. Eliminar una partitura registrada: El usuario debe poder eliminar una partitura de su lista de partituras registradas.
- R.F-08. Eliminar una partitura digitalizada: El usuario debe poder eliminar una partitura de su lista de partituras digitalizadas.
- R.F-09. Digitalizar una partitura: El usuario debe poder digitalizar una partitura de su lista de partituras registradas.
- R.F-010. Preprocesar una partitura: El usuario debe poder preprocesar una partitura de su lista de partituras registradas.

- R.F-011. Visualizar una partitura: El usuario debe poder visualizar una partitura digitalizada.
- R.F-012. Descarga una partitura: El usuario debe poder descargar una partitura digitalizada.

2.3.2. Requisitos no funcionales

- R.NF-01. Seguridad: La aplicación debe asegurar la protección de los datos del usuario haciendo uso de métodos de encriptación para el almacenamiento de las credenciales.
- R.NF-02. Disponibilidad: La aplicación debe estar disponible durante el periodo de actividad.
- R.NF-03. Escalabilidad: La aplicación debe ser capaz de soportar y tener una capacidad de integración sencilla a modificaciones y aumento de funcionalidades.
- R.NF-04. Usabilidad: La interfaz de usuario debe ser intuitiva y fácil de usar, permitiendo a los usuarios realizar todas las operaciones con no más de tres clics.
- R.NF-05. Compatibilidad: La aplicación debe ser compatible con los principales sistemas operativos de escritorio, incluyendo Windows, macOS, Android e iOS.
- R.NF-06. Mantenibilidad: El código de la aplicación debe seguir una buena metodología y estructura de programación para asegurar que sea fácil de mantener y actualizar.

2.4 Especificación de requisitos

En esta sección se van a especificar los casos de uso en base a los requisitos descritos anteriormente.

2.4.1. Casos de uso

Casos de Uso de Autenticación de Usuario

Tabla 10 Caso de uso 1

CU-1	Inicio de sesión
Versión	1.0
Autor	Martín González Saiz.
Descripción	Permitir a los usuarios iniciar sesión de forma segura.
Actores	Usuario.
Precondiciones	El usuario debe estar registrado.
Acciones	1. El usuario introduce sus credenciales. 2. El sistema verifica las credenciales. 3. El usuario es autenticado y redirigido al menú de la aplicación.
Postcondiciones	El usuario ha iniciado sesión.
Requisitos asociados	R.F-01, R.NF-01

Tabla 11 Caso de uso 2

CU-2	Registro de usuario
Versión	1.0
Autor	Martín González Saiz.
Descripción	Permitir a los usuarios registrarse en el sistema.
Actores	Usuario.
Precondiciones	Ninguna.

Acciones	1. El usuario entra en el formulario de registro 2. El usuario completa todos los campos del formulario. 3. El sistema guarda los datos y registra al usuario.
Postcondiciones	El usuario ha sido registrado.
Requisitos asociados	R.F-02, R.NF-01

Tabla 12 Caso de uso 3

CU-3	Cerrar sesión
Versión	1.0
Autor	Martín González Saiz.
Descripción	Permitir a los usuarios cerrar sesión en el sistema.
Actores	Usuario.
Precondiciones	El usuario debe haber iniciado sesión.
Acciones	1. El usuario selecciona la opción de cerrar sesión. 2. El sistema cierra la sesión del usuario.
Postcondiciones	El usuario ha cerrado sesión.
Requisitos asociados	R.F-03, R.NF-01

Casos de Uso de Gestión de Partituras

Tabla 13 Caso de uso 4

CU-4	Subir una partitura
Versión	1.0
Autor	Martín González Saiz.
Descripción	Permitir a los usuarios subir una partitura en cualquier formato.
Actores	Usuario.
Precondiciones	El usuario debe estar autenticado.
Acciones	1. El usuario selecciona la opción de subir una partitura. 2. El usuario introduce el nombre para la partitura. 3. El usuario selecciona el archivo de la partitura en su sistema. 4. El usuario selecciona la opción de subir la partitura. 5. El sistema guarda el archivo con los datos y lo asocia al usuario.
Postcondiciones	La partitura es subida y registrada.
Requisitos asociados	R.F-04

Tabla 14 Caso de uso 5

CU-5	Listar las partituras registradas
Versión	1.0
Autor	Martín González Saiz.
Descripción	Permitir a los usuarios ver su lista de partituras registradas.
Actores	Usuario.
Precondiciones	El usuario debe estar autenticado.
Acciones	1. El usuario selecciona la opción de ver sus partituras registradas. 2. El sistema muestra la lista de partituras registradas.
Postcondiciones	La lista de partituras es visualizada.
Requisitos asociados	R.F-05

Tabla 15 Caso de uso 6

CU-6	Listar las partituras digitalizadas
Versión	1.0

Autor	Martín González Saiz.
Descripción	Permitir a los usuarios eliminar el registro de una partitura.
Actores	Usuario.
Precondiciones	El usuario debe estar autenticado.
Acciones	1. El usuario selecciona la opción de ver sus partituras digitalizadas. 2. El sistema muestra la lista de partituras digitalizadas.
Postcondiciones	La lista de partituras digitalizadas es visualizada.
Requisitos asociados	R.F-06

Tabla 16 Caso de uso 7

CU-7	Eliminar una partitura registrada
Versión	1.0
Autor	Martín González Saiz.
Descripción	Permitir a los usuarios eliminar el registro de una partitura registrada.
Actores	Usuario.
Precondiciones	El usuario debe estar autenticado y tener partituras registradas.
Acciones	1. El usuario selecciona la opción de eliminar una partitura 2. El sistema borra el registro de la partitura.
Postcondiciones	La partitura registrada es borrada.
Requisitos asociados	R.F-07

Tabla 17 Caso de uso 8

CU-8	Eliminar una partitura digitalizada
Versión	1.0
Autor	Martín González Saiz.
Descripción	Permitir a los usuarios eliminar el registro de una partitura digitalizada.
Actores	Usuario.
Precondiciones	El usuario debe estar autenticado y tener partituras digitalizadas.
Acciones	1. El usuario selecciona la opción de eliminar una partitura 2. El sistema borra el registro de la partitura.
Postcondiciones	La partitura digitalizada es borrada.
Requisitos asociados	R.F-08

Tabla 18 Caso de uso 9

CU-9	Digitalizar una partitura
Versión	1.0
Autor	Martín González Saiz.
Descripción	Permitir a los usuarios digitalizar una partitura de su lista de partituras registradas.
Actores	Usuario.
Precondiciones	El usuario debe estar autenticado, tener partituras registradas y estar en la lista de partituras registradas.
Acciones	1. El usuario selecciona el botón de digitalizar de la partitura en el listado. 2. El sistema digitaliza la partitura.
Postcondiciones	La partitura es digitalizada y guardada.
Requisitos asociados	R.F-09

Tabla 19 Caso de uso 10

CU-10	Preprocesar una partitura
Versión	1.0
Autor	Martín González Saiz.
Descripción	Permitir a los usuarios preprocesar una partitura de su lista de partituras registradas.
Actores	Usuario.
Precondiciones	El usuario debe estar autenticado, tener partituras registradas y estar en la vista de la lista de partituras registradas.
Acciones	1. El usuario selecciona el botón de preprocesar de la partitura en el listado. 2. El sistema muestra el formulario de los valores de las funciones de procesamiento. 3. El usuario introduce los valores de las funciones de procesamiento. 4. El sistema aplica las técnicas de procesamiento en la imagen de la partitura en cuestión. 5. El sistema comienza la descarga del nuevo archivo.
Postcondiciones	La partitura es preprocesada y descargada.
Requisitos asociados	R.F-10

Tabla 20 Caso de uso 11

CU-11	Visualizar una partitura digitalizada
Versión	1.0
Autor	Martín González Saiz.
Descripción	Permitir a los usuarios visualizar una partitura de su lista de partituras digitalizadas.
Actores	Usuario.
Precondiciones	El usuario debe estar autenticado, tener partituras digitalizadas y estar en la vista de la lista de partituras digitalizadas.
Acciones	1. El usuario selecciona el botón de visualizar de la partitura en el listado. 2. El sistema muestra una vista de la partitura.
Postcondiciones	La partitura es visualizada.
Requisitos asociados	R.F-11

Tabla 21 Caso de uso 12

CU-12	Descarga una partitura digitalizada
Versión	1.0
Autor	Martín González Saiz.
Descripción	Permitir a los usuarios descargar una partitura de su lista de partituras digitalizadas.
Actores	Usuario.
Precondiciones	El usuario debe estar autenticado, tener partituras digitalizadas y estar en la vista de la lista de partituras digitalizadas.
Acciones	1. El usuario selecciona el botón de descargar de la partitura en el listado. 2. El sistema comienza la descarga de la partitura.
Postcondiciones	La partitura es descargada.
Requisitos asociados	R.F-12

Casos de Uso del Menú

Tabla 22 Caso de uso 13

CU-13	Ir al menú
Versión	1.0
Autor	Martín González Saiz.
Descripción	Permitir a los usuarios visualizar el menú de acciones
Actores	Usuario.
Precondiciones	El usuario debe estar autenticado.
Acciones	1. El usuario completa el proceso de inicio de sesión. 2. El sistema redirige al usuario al menú principal.
Postcondiciones	El usuario ha sido redirigido al menú principal
Requisitos asociados	R.F-01

2.4.2. Diagrama de Casos de Uso

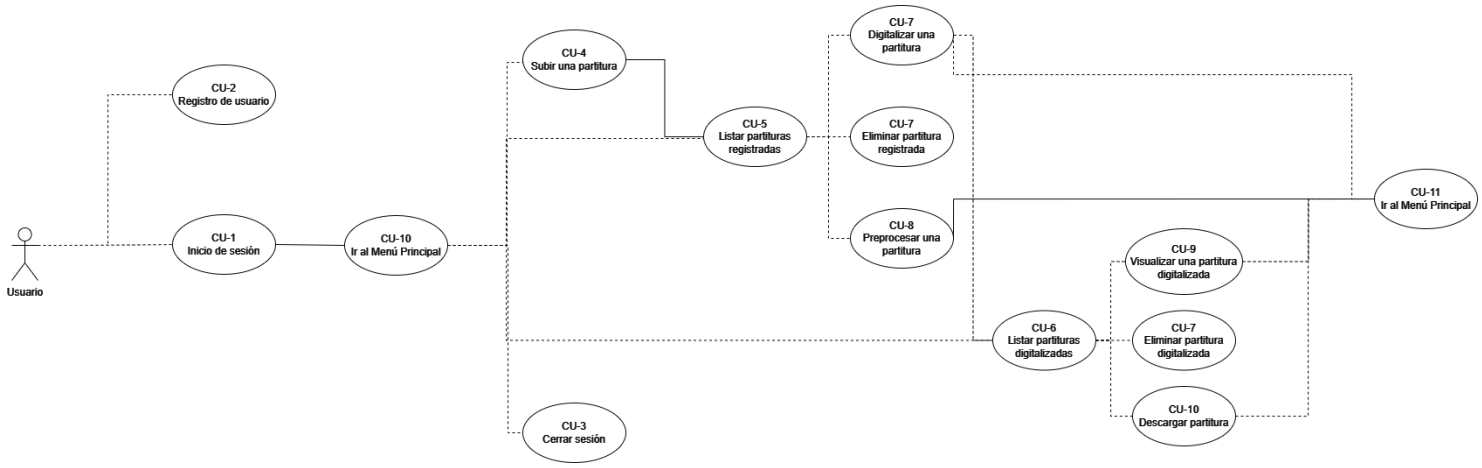

Figura 7 Diagrama de Casos de Uso

Anexo C Especificación de diseño

3.1 Introducción

En esta parte de la documentación del proyecto se va a explicar la estructura y modelo de datos que se ha seguido, teniendo en cuenta que para el desarrollo de esta aplicación se ha hecho uso de servicios externos.

3.2 Diseño de datos

En el desarrollo de la aplicación se utilizan distintos conceptos de estructura de datos, debido a que se hace uso de servicios externos como Firebase, por ello se va a seccionar el diseño de datos en distintos apartados.

3.2.1 Modelos de datos

User

Este modelo representa a un usuario que se ha registrado en la aplicación tras rellenar el correspondiente formulario, siendo apto en la autenticación y autorizado para interactuar con la aplicación en ciertas funcionalidades.

Sus atributos son los siguientes:

- id (str): Nombre del usuario único que actúa como clave primaria.
- email (str): Direccion de correo electrónico del usuario.
- passwordHash (str): Contraseña usuario encriptada para aumentar la seguridad.

Y sus métodos:

- __init__(username, email, passwordHash): Es el constructor del modelo de usuarios

SheetMusic

Este modelo representa los registros de datos referentes a una partitura subida por un usuario.

Sus atributos son los siguientes:

- title (str): Título de la partitura.
- file_path (str): Ruta del archive de la partitura almacenada en Firebase Storage.
- upload_date (datetime, opcional): Feha y hora de la carga del archivo.

Y sus métodos son:

__init__(title, file_path, upload_date=None): Es el constructor que inicializa una nueva partitura.

3.2.2 Almacenamiento y gestión de datos

Firebase Firestore

Esta herramienta es usada para almacenar los datos tanto de los usuarios como de las partituras en una base de datos NoSQL. En Firebase la base de datos de Firestore se organiza por colecciones y documentos, de manera similar al concepto de tablas, filas y columnas, siendo las colecciones lo equivalente a tablas y los documentos a las filas.

- Colección de users: Contiene los documentos los cuales cada uno representa los datos de un usuario en concreto. Cada documento tiene un id que es el nombre del usuario.
- Colección de sheet_music: Contiene los documentos los cuales cada uno representa una partitura, tanto para las registradas como para las digitalizadas.

Firebase Storage

Es la herramienta utilizada para el almacenamiento y gestión de los archivos que representan las partituras. Se ha seguido una estructura sistema de directorios que distinga los archivos por formato y usuario, manejando su acceso mediante rutas.

La estructura utilizada es: partituras/<user_id>/<file_type>/<filename> donde <user_id> es el ID del usuario, <file_type> es el formato del archivo de la partitura y <filename> es el nombre del archivo.

3.3 Diseño procedimental

En esta sección se va a mostrar con un diagrama de secuencias el proceso de un flujo de trabajo completo de la aplicación. Como el diagrama quedaría muy grande se va a seccionar.

3.3.1 Registro e inicio de sesión

Este diagrama muestra el proceso que sigue el programa para registrar una nueva cuenta de usuario. Para ello tras recibir la petición del formulario del usuario se encripta su contraseña en la capa del backend para luego enviar todos los datos al servicio Firestore. Seguido de este proceso, se realiza un inicio de sesión donde para autenticar un usuario, se verifican las credenciales. En ambos procesos se sigue un flujo de intercambio de datos entre los distintos servicios de Firebase.

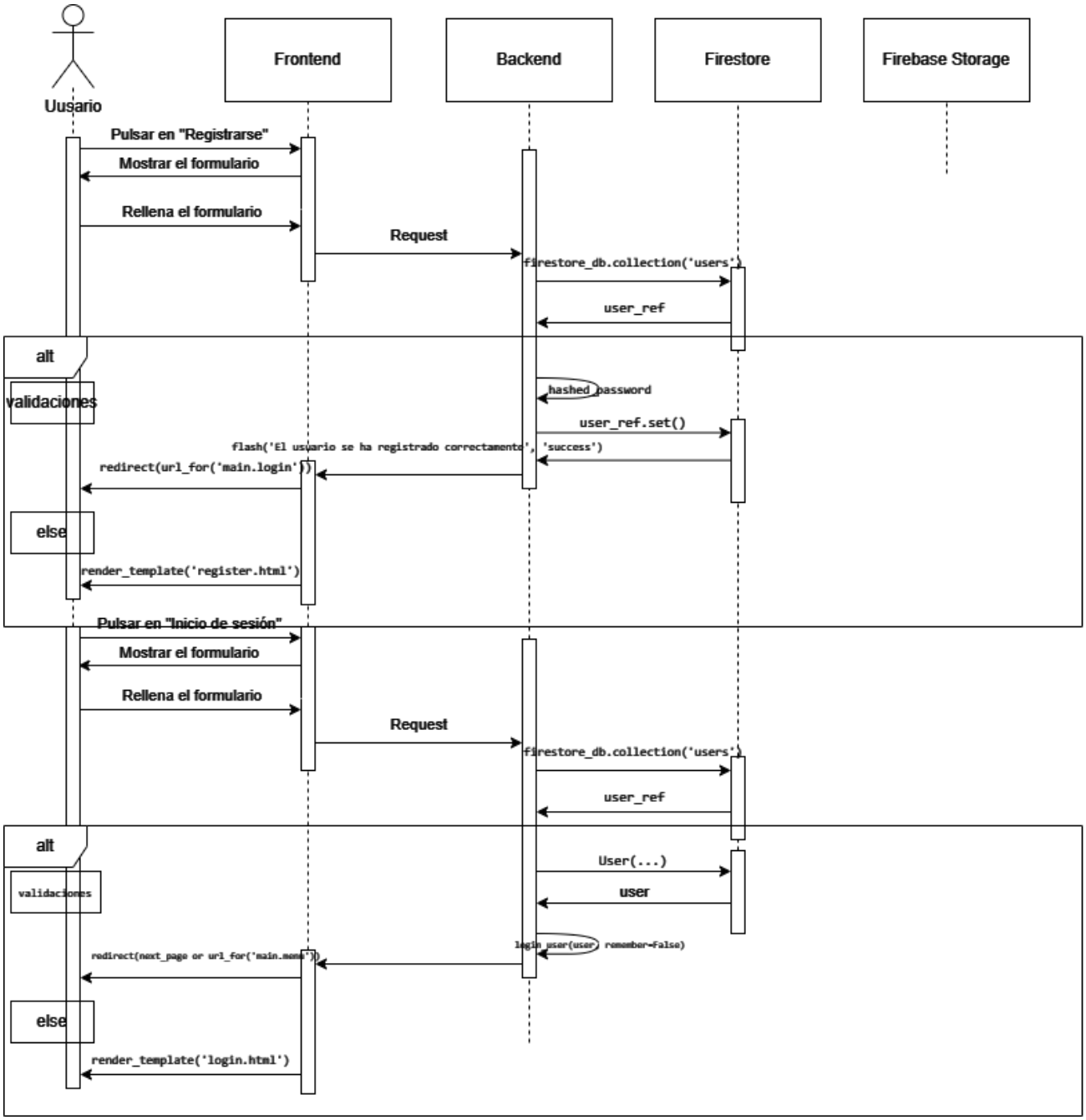

Figura 8 Registro e inicio de sesión

3.3.2 Subida y eliminación de una partitura

El siguiente diagrama es referente al flujo de subir una partitura a la aplicación y posteriormente eliminarla. Para ello el proceso se basa en esperar la petición del usuario de cada una de las acciones y hacer las comprobaciones correspondientes, siendo en el caso de subida verificar el formato de archivo y en el caso de la eliminación comprobar su existencia. Tras ello se intercambia información con los servicios de Firebase para actualizar los registros. En caso de excepción o éxito se le comunica al usuario.

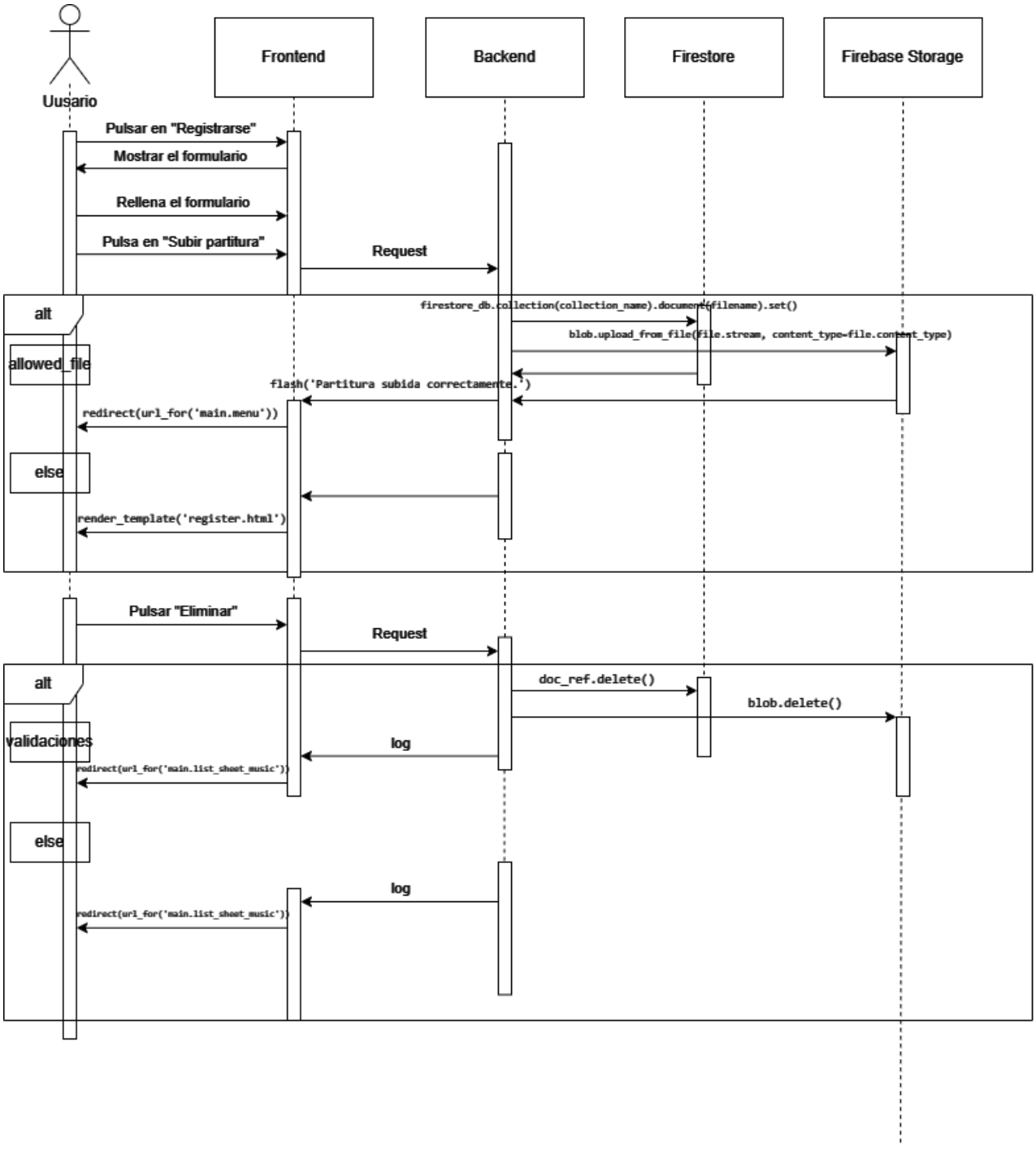

Figura 9 Subir y eliminar una partitura

3.3.3 Digitalización y preprocesamiento de una partitura

Ahora se va a mostrar un flujo de digitalización de una partitura con el lanzamiento de una excepción seguido de preprocesamiento, acabando con un intento de digitalización con éxito.

El usuario hace la petición de ver el listado de partituras, desde el backend se accede al listado iterando sobre los elementos del bucket de Firebase Storage y para cada elemento se busca en la colección de Firestore el nombre correspondiente añadiéndolo a un array de elementos. Esto se manda a la vista para mostrarlo al usuario.

A continuación, el usuario hace una petición para digitalizar una partitura en concreto, se descarga de Firebase Storage para luego intentar digitalizarla en otra función. El resultado del

proceso de digitalización es analizado buscando su log y cadenas de caracteres concretas para mandar alertas al respecto al usuario. El primer caso es el fallido por lo que se le envía al usuario un mensaje que indica que la partitura no es apta.

El siguiente intento es un éxito por lo que el resultado devuelto por audiveris es registrado en ambos servicios de Firebase y se envía un mensaje de éxito al usuario.

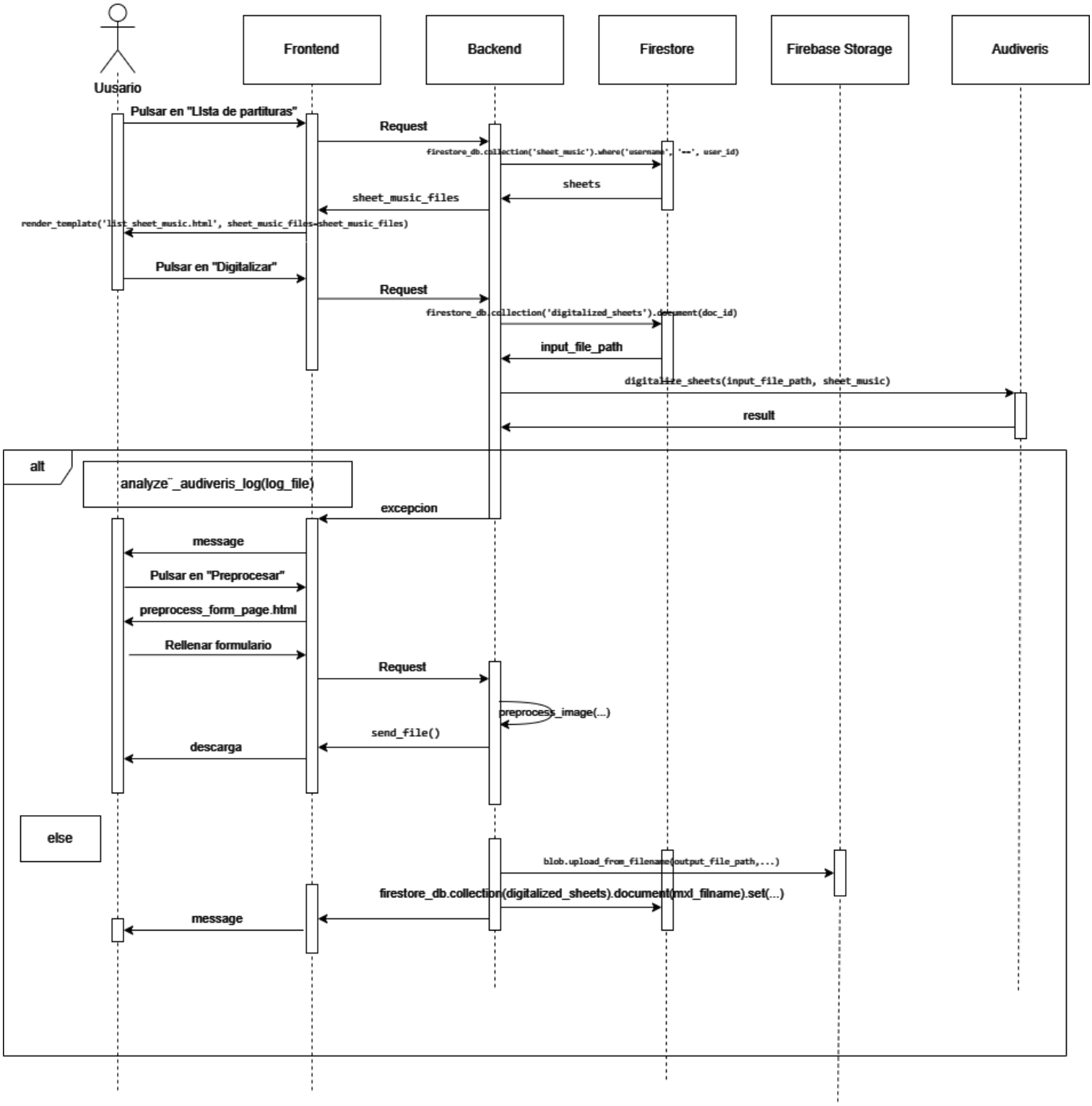

Figura 10 Digitalización y preprocesamiento de una partitura

3.3.4 Visualización de una partitura

Por último, se va a representar el flujo que se sigue cuando un usuario solicita visualizar una partitura. El usuario solicita ver la lista de partituras digitalizadas, la request llega al backend y este accede al listado a través de la colección de Firestore, y de los registros de Firebase Storage. Tras esto se envían las partituras a la vista para que el usuario las visualice y elija cual debe mostrarse. Realiza la solicitud para visualizar la partitura y el backend accede a esta a través de Firebase Storage para visualizarla haciendo uso de Verovio.

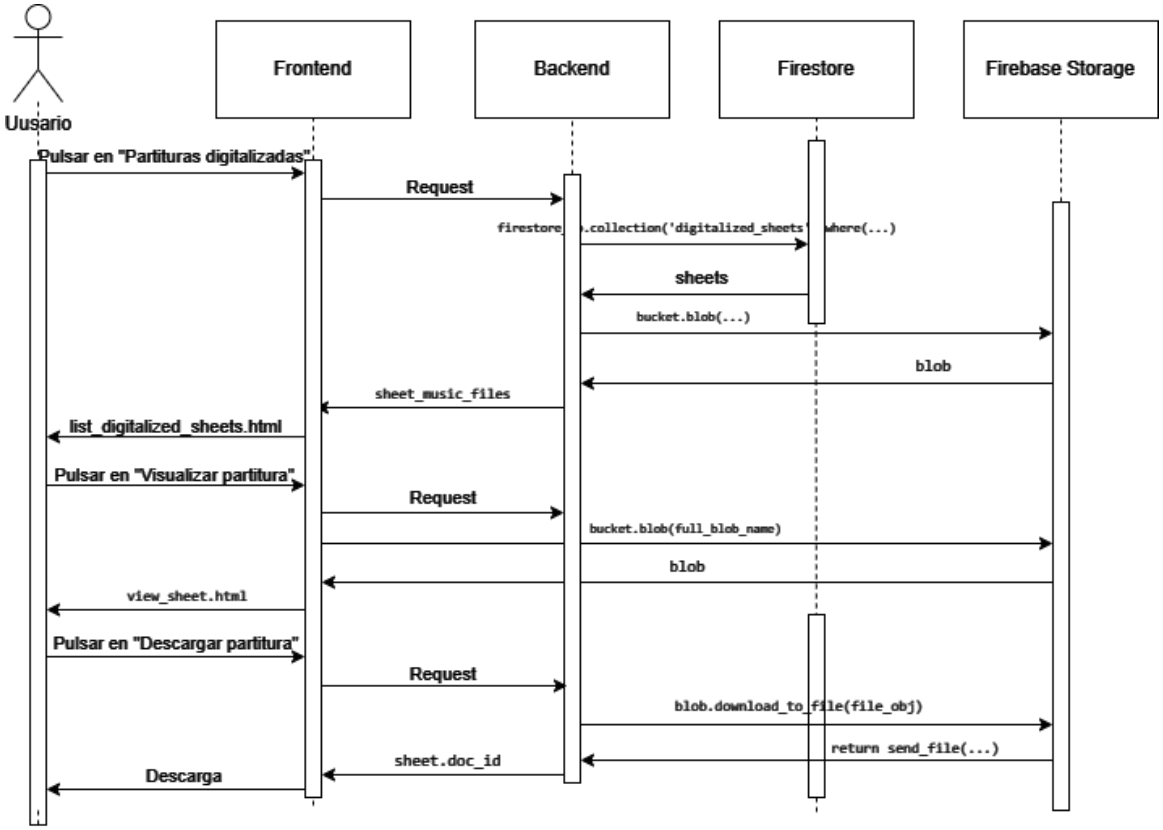

Figura 11 Visualización y descarga de una partitura digitalizada

3.4 Diseño arquitectónico

En esta sección se va a detallar el patrón de diseño que se ha seguido en el desarrollo del proyecto para mantener una escalabilidad.

3.4.1 Estructura Modular

En primer lugar, se va a hablar de la estructura modular seguida en el proyecto. Se ha hecho uso de *blueprints* para modularizar las rutas y vistas, de manera que se gestionen las áreas de la aplicación de manera distintiva y separada. A su vez, también se han separado los archivos y módulos en función de su funcionalidad, pudiendo ser modelos, rutas, configuraciones, extensiones, servicios de Firebase, etc.

3.4.2 Configuración Externa

Gran parte de los parámetros de configuración de la aplicación se han detallado y gestionado en un archivo externo, de manera que cualquier modificación y manejo en el

comportamiento de la aplicación se facilite en gran medida. Además, se integran variables de entorno para las credenciales de Firebase y Google Cloud para mantener una buena seguridad en todos los entornos, sobre todo públicamente.

3.4.3 Gestión de Usuarios

Toda la gestión de sesiones de usuario se ha hecho mediante una combinación de librerías de Flask con los servicios que proporciona Firebase para el almacenamiento y validación de los datos del usuario.

3.4.4 Servicios Externos

Principalmente para el manejo de datos se ha utilizado servicios externos.

- Firebase Admin: Se ha utilizado los servicios de Firebase para la gestión y almacenamiento de datos, siendo Firestore la base de datos y Firebase Storage el servicio de almacenamiento de las partituras de la aplicación, de esta manera se centraliza de forma adecuada los datos para cada usuario.
- Manejo de archivos: la carga, subida y manipulación de los datos se gestiona de forma distribuida y validada según el formato de estos y los usuarios, además se intenta hacer un manejo seguro.

3.4.5 Tratamiento de errores

Se ha hecho uso de excepciones como retroalimentación del flujo de la aplicación, así como de interacción informativa para el usuario con el fin de mantenerlo informado con cada acción en todo momento.

3.5 Diseño de interfaces

Para el diseño de interfaces de la aplicación se ha hecho uso de plantillas HTML que implementan la librería Bootstrap como framework de frontend. Todas las páginas utilizan Bootstrap para hacer que visualmente se mantenga una consistencia, ajustándose a varios dispositivos y tamaños de pantalla, esto ha ayudado a mantener en todas las vistas una estética uniforme.

Además, se ha ajustado la estructura de las vistas con el uso de secciones de estilo css para mantener una homogeneidad entre las ventanas.

En las vistas se hace uso de varios componentes propios de Bootstrap, ya sean botones, alertas y formularios, para mejorar la usabilidad de la aplicación para el usuario.

Anexo D Documentación técnica de programación

4.1 Introducción

En este apartado se va a explicar y mostrar la organización y estructuración que tiene el proyecto, así como un manual del programador para dar pautas e instrucciones a los usuarios de cómo preparar su entorno e instalar la aplicación correctamente.

4.2 Estructura de los directorios

- DigitalizaciónPartiturasApp: Es el directorio en el que se ha trabajado en la aplicación. Aquí se encuentran las carpetas y directorios que contienen el código de la aplicación, así como recursos necesarios para el funcionamiento de ciertas funcionalidades.
    - .vscode: Es la carpeta que contiene configuraciones y ajustes del proyecto utilizado por VS Code.
    - __pycache__: Es una carpeta generada automáticamente por Python con versiones optimizadas del código.
    - app: Es la carpeta principal que contiene los archivos principales de la aplicación.
        - templates: Contiene todos los archivos html necesarios para que se muestren las vistas
        - __init__: Utilizado para inicializar y configurar la aplicación web, ya sea de extensiones o servicios.
        - extensions.py: Este archivo se encarga de cargar las credenciales de Firebase y de inicializar esta herramienta en la aplicación.
        - firebase_utils.py: Se encarga de toda la interacción que tiene la aplicación con Firebase y con la base de datos NoSQL.
        - models.py: Contiene la definición de los modelos de la aplicación.
        - routes.py: Contiene todas las funcionalidades y lógica de la aplicación.
    - audiveris: Directorio clonado del software de Audiveris.
    - audiveris_output: Directorio de salida para las partituras digitalizadas.
    - logs: Carpeta que contiene los archivos del logging utilizados a lo largo del desarrollo del proyecto.
    - migrations: Directorio que contiene las migraciones realizadas durante el desarrollo del proyecto.
    - verovio: Directorio clonado de Verovio utilizado para visualizar las partituras.
    - config: Archivo en el que se realiza toda la configuración necesaria para el correcto funcionamiento de la lógica de la aplicación. Se definen variables para guardar la configuración de los servicios, como Firebase, claves y archivos .json utilizados posteriormente en otros módulos, así como la configuración de los logs para controlar el flujo de la aplicación.
    - DockerFile: Archivo creado para la creación del Docker donde se especifica las base, herramientas y dependencias necesarias para el correcto funcionamiento de la aplicación, así como la compilación de Audiveris y la ejecución final de la aplicación.

- requirements: Contiene todos los requerimientos necesarios para el correcto funcionamiento de la aplicación. Este archivo es creado automáticamente mediante el comando: pip freeze > requirements.txt.
- run.py: Este archivo se encarga de la ejecución del servidor de Flask con el host 0.0.0.0 para que sea accesible para cualquier IP.
- run_audiveris.sh: Este archivo construye el comando necesario para ejecutar Audiveris sin interfaz de usuario, exportando los resultados a una carpeta definida.
- sheet-transcribe-firebase-adminsdk: Son las credenciales json del servicio Firebase.

– memoria: Este directorio alberga los documentos referentes a la documentación.
– generate_base64: Este script se encarga de codificar las credenciales de Firebase.

4.3 Manual del programador

En este apartado se van a detallar los programas y configuraciones utilizados para el desarrollo del proyecto. Por lo tanto, los requisitos y pasos de instalación son:

– Python 3.11
– Git
– Docker
– Visual Code

A continuación, se va a explicar cómo instalar cada programa paso a paso en el sistema operativo Windows, que es en el que se ha llevado a cabo el desarrollo del proyecto.

Python 3.11

Python es el lenguaje de programación en el que ha sido desarrollada la aplicación.

Es necesario usar la versión de Python 3.11. Para su descarga es necesario acceder a la página oficial de Python (4). Tras esto, durante la instalación hay que marcar la opción “Add Python to PATH”.

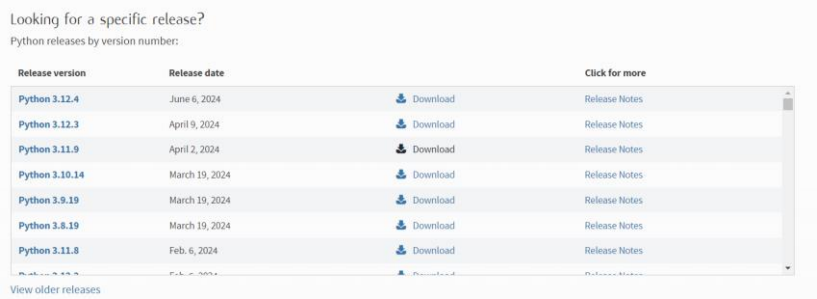

Figura 12 Descarga de Python (1)

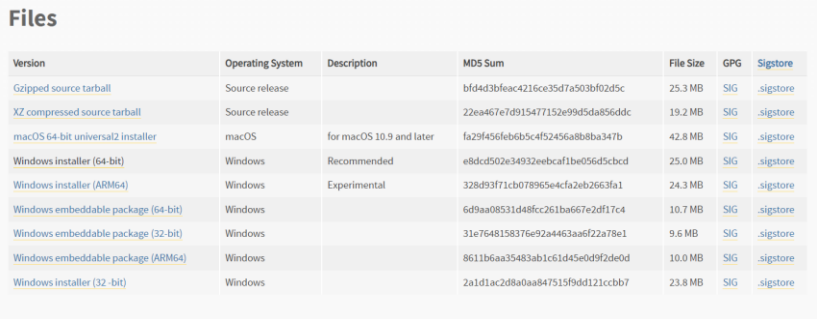

Figura 13 Descarga de Python (2)

Git

Es una herramienta de control de versiones usado para la gestión del desarrollo de la aplicación.

Es necesario descargar e instalar Git, por ello hay que acceder a la página oficial de Git (5).

Figura 14 Descarga de Git

Docker Desktop

Para la importación y ejecución del Docker es necesario tener instalado Docker Desktop que es el programa de gestión de contenedores e imágenes Docker. Para ello hay que acceder a la página oficial de Docker (6).

Figura 15 Descarga de Docker Desktop

Visual Studio Code

El editor de código utilizado para construir y trabajar en el desarrollo ha sido Visual Studio Code. Para ello es necesario acceder a la página oficial de Visual (7) para descargarlo e instalarlo. Al igual que antes en esta instalación es necesario activar la casilla de "Agregar al PATH". Algunas extensiones de Visual Studio que han resultado útiles son Python, Docker, GitLens y Prettier.

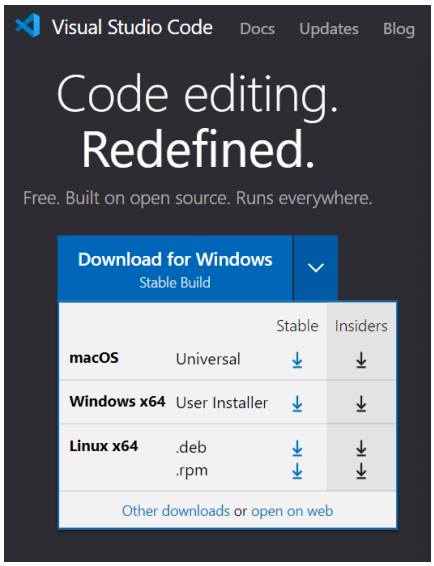

Figura 16 Descarga de Visual Studio Code

Heroku CLI

Es necesario descargar e instalar Heroku CLI para llegar al despliegue de la aplicación si se consigue optimizar el Docker lo suficiente. Para realizar la descarga hay que acceder a la página oficial de Heroku (8).

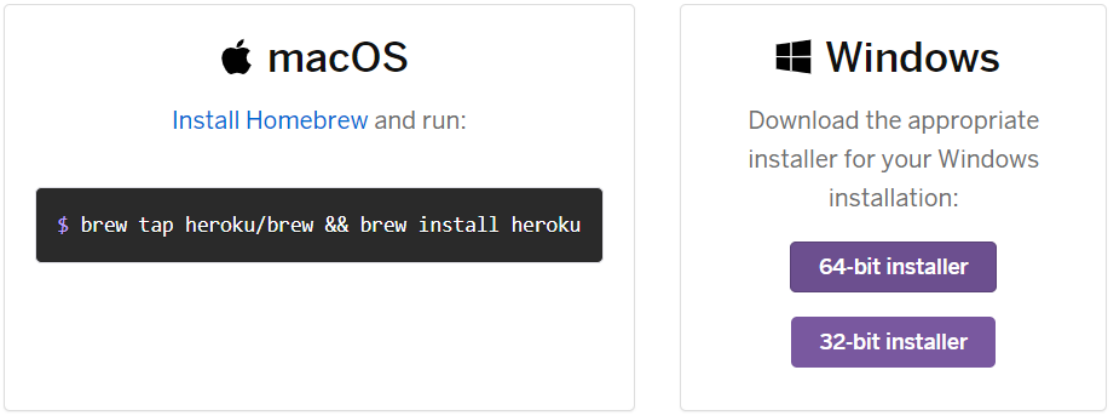

Figura 17 Descarga de Heroku CLI

4.4 Compilación, instalación y ejecución del proyecto

En primer lugar, sería necesario acceder al repositorio del proyecto para clonarlo https://github.com/martingonzsaiz/Digitalizacion-Partituras. Para ello hay que realizar el comando siguiente en la terminal CMD o en el PowerShell:

git clone https://github.com/martingonzsaiz/Digitalizacion-Partituras.git

Luego sería necesario moverse al directorio en cuestión para seguir con las siguientes instrucciones.

cd Digitalizacion-Partituras

Después se creó un entorno virtual en el que se trabajó y se llevó el desarrollo. La creación del entorno virtual se alcanza con los siguientes comandos:

python -m venv entorno_partituras

entorno_partitiuras\Scripts\activate

De esta manera se facilita el manejo del entorno y se asegura el aislamiento de las dependencias y su mayor gestión. Así las dependencias de este proyecto no entran en conflicto con dependencias externas. Si el entorno está activado aparece su nombre al principio de la línea de comandos.

Tras esto, es necesario instalar las dependencias del proyecto, para llevarlo a cabo se utiliza el comnado:

pip install -r requirements.txt

A continuación, es necesario tener claros ciertos aspectos de la aplicación. En esta aplicación se utilizan valores de variables para preparar el entorno de manera segura y efectiva. Estos valores son FIREBASE_CREDENTIALS_JSON_BASE64 y FIREBASE_BUCKET_NAME.

Las credenciales de Firebase sirven para que la aplicación se autentique y pueda acceder a los servicios de Firebase. En el caso de esta aplicación se hace uso de Firestore y Firebase Storage, por ello es necesario generar estas claves y definirlas como variables. Para ello habría que seguir unos pasos extra:

En Firebase Storage:

1. Ir a Firebase Console (9)
2. Crear o seleccionar el proyecto.
3. Ir a Configuración del proyecto > Cuentas de servicio.
4. Seleccionar “Generar nueva clave privada” y guardar el archivo JSON generado.
5. Se podría codificar las credenciales con el siguiente comando: **certutil -encode input.json output.txt**

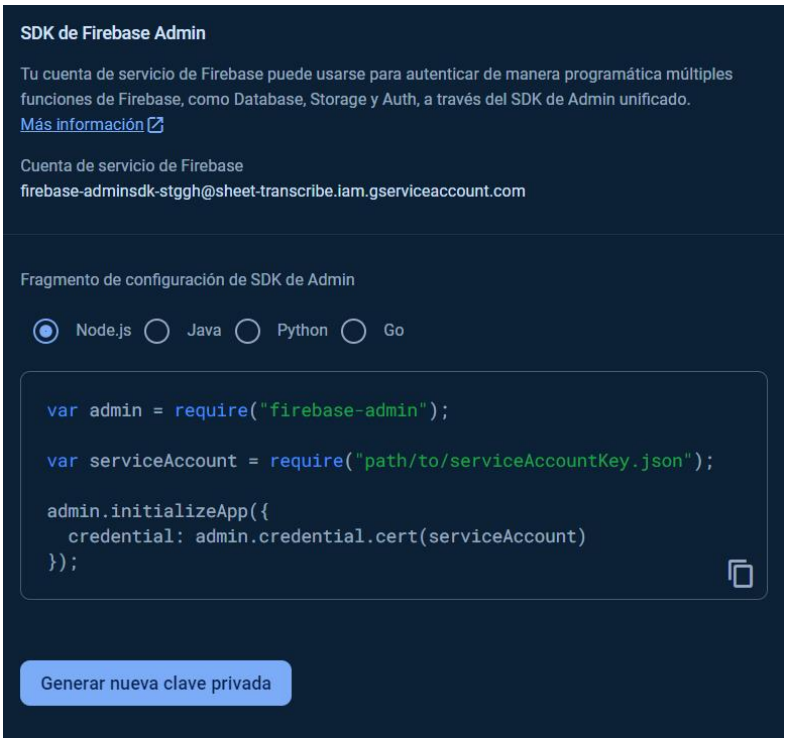

Figura 18 Descargar credenciales Firebase Storage

En Firebase Bucket:

1. Acceder a Firebase Console y entrar en el proyecto.
2. Entrar en Storage.
3. Copiar el nombre del bucket.

Ahora es necesario configurar las variables de entorno para poder cargar y preparar las credenciales desde el entorno del sistema haciendo uso de la biblioteca os, tal y como se hace en el archivo config.py.

1. Ir a Sistema>Configuración avanzada del sistema>Variables de Entorno
2. En Variables del sistema, es necesario crear las nuevas variables de entorno: FIREBASE_CREDENTIALS_JSON_BASE64, FIREBASE_BUCKET_NAME.
3. Aplicar los cambios

Con esta configuración se podría reutilizar el archivo config.py del repositorio.

Ahora, tras asegurarse de que el entorno virtual esta ejecutado, se podría ejecutar la aplicación con el comando:

python run.py

Y acceder a ella a través del navegador con http://127.0.0.1:5000, pudiendo interactuar con todas las funcionalidades de la aplicación localmente.

Construcción del Docker

Para la construcción del Docker habría que revisar y actualizar todas las nuevas dependencias que tiene la aplicación tras un desarrollo de una nueva funcionalidad para la aplicación. Todas estas nuevas dependencias se deberían añadir en la sección del Docker referente al RUN:

**RUN apt-get update && apt-get install -y --no-install-recommends **

**git **

**curl **

**build-essential **

**libtesseract-dev **

**tesseract-ocr **

**libfreetype6-dev **

**ghostscript **

**imagemagick **

**openjdk-17-jdk **

**dos2unix **

**poppler-utils && **

apt-get clean && rm -rf /var/lib/apt/lists/*

Tambien sería necesario añadir todas las nuevas variables de entorno que se requieran. También sería necesario realizar cambios en la imagen si se han añadido archivos o se han modificado antiguos.

Finalmente, se podría construir el Docker y subir la imagen. Para ello hay que usar el siguiente comando:

docker build -t melodymatrix .

Despliegue en Heroku

La configuración de los archivos y servicios para el despliegue de la aplicación se llevó a cabo para el despliegue de la aplicación con Heroku. Sin embargo, a pesar de que la aplicación está desplegada, no es posible realizar todas las funcionalidades de la aplicación debido a un exceso de memoria dentro del plan gratuito de este servicio.

Debido a esto dentro de la aplicación desplegada no es posible digitalizar las partituras ya que este proceso excede la memoria. No obstante, se va a explicar el despliegue para llegar a este futuro desarrollo si cabe.

Tras la construcción del Docker, sería necesario iniciar sesión en Heroku CLI y en el registro de contenedores mediante los comandos:

heroku login.

heroku container:login

Luego sería necesario crear una aplicación en Heroku, construir la imagen y publicarla en Heroku:

heroku create melodymatrix

heroku container:push web -a melodymatrix

Finalmente se podría lanzar el contenedor a heroku para después abrir la aplicación públicamente, dando acceso a cualquier usuario con el enlace de esta:

heroku container:release web -a melodymatrix

heroku open -a melodymatrix

En caso de que haya cualquier fallo o si se quiere revisar el flujo de la aplicación se puede usar el siguiente comando para visualizar los logs:

heroku logs --tail -a melodymatrix

4.5 Pruebas del sistema

Durante todo el desarrollo del proyecto se han realizado pruebas de integración para comprobar el correcto funcionamiento de los servicios utilizados de Firebase. Estas pruebas han consistido en comprobaciones e impresiones en el flujo de la aplicación para comprobar que las interacciones con Firestore funcionan correctamente y que el intercambio de datos con Firebase Storage se realiza como se desea.

Muchas de estas pruebas han sido un proceso de hacer varias pruebas de registro e inicio de sesión para comprobar que todo funcionaba correctamente con Firebase en diferentes contextos. Otro tipo de pruebas que se han realizado ha sido con el manejo de cargas y descargas de partituras con Firebase Storage.

A su vez también se ha hecho varias pruebas para comprobar el comportamiento de las funciones ajenas a los servicios utilizados en la aplicación. Dentro de este grupo, las funcionalidades en las que más se ha probado su funcionamiento es con las funciones de preprocesamiento de imágenes. El proceso ha consistido en manejar distintos valores para los campos del formulario de preprocesamiento de imágenes para comprobar visualmente los resultados obtenidos en la imagen. De esta manera se ha podido valorar cuales son las mejores combinaciones o valores. Otro aspecto en el que se ha hecho pruebas es la comprobación de los formatos en los que se admiten partituras, teniendo que ser .pdf, .png, .jpg, .jpeg.

Otro tipo pruebas que se ha realizado es referente al proceso de digitalización de las partituras. Debido al ratio de fallo que tiene Audiveris con lo que respecta a digitalizar partituras en ciertas circunstancias o con ciertas características, ha sido necesario afianzar y agrupar cierto grupo de ejemplares de partituras que valen como testeo y llegan al fin. Para ello ha habido que realizar un proceso de prueba y error hasta dar con una colección.

En general se han hecho pruebas para verificar que todas las rutas de la aplicación devuelven las respuestas y esperadas en diferentes contextos y condiciones. Además, cada vez que se ha implementado una funcionalidad nueva se ha comprobado que no alteraba el correcto funcionamiento del resto, haciendo un proceso de flujo completo de trabajo a lo largo de la aplicación.

Anexo E Documentación de usuario

5.1 Introducción

En este apartado se va a explicar cómo debe ser la descarga e instalación de la aplicación para cualquier usuario casual interesado en el uso de la aplicación. Además, se explicarán los requisitos necesarios para que el usuario pueda hacer uso de ella. Tras ello se explicará cómo es la interacción de la aplicación con el usuario visualmente.

En el caso de los requisitos de usuarios y la instalación se van a reflejar dos escenarios, debido a que la aplicación está desplegada pero no se puede hacer uso de la funcionalidad de digitalización, por ello se va a plantear la alternativa de descargar y ejecutar la imagen.

5.2 Requisitos de usuarios

Navegador Web

- Google Chrome
- Mozilla Firefox
- Microsoft Edge
- Safari
- Opera
- Internet Explorer

Docker

- Docker: Para el segundo caso es necesario tener instalado Docker para poder ejecutar los contenedores Docker.
- Terminal o línea de comandos: Además, es necesario tener la posibilidad de utilizar la terminal o línea de comandos en el dispositivo.

5.3 Instalación

Docker

Teniendo en cuenta que la imagen está disponible en el Docker Hub, para preparar el entorno en el que poder ejecutar la imagen e interactuar con la aplicación primero es necesario iniciar sesión en Docker Hub mediante el siguiente comando en un terminal:

docker login

Dado que la imagen es publica y se puede acceder a ella,

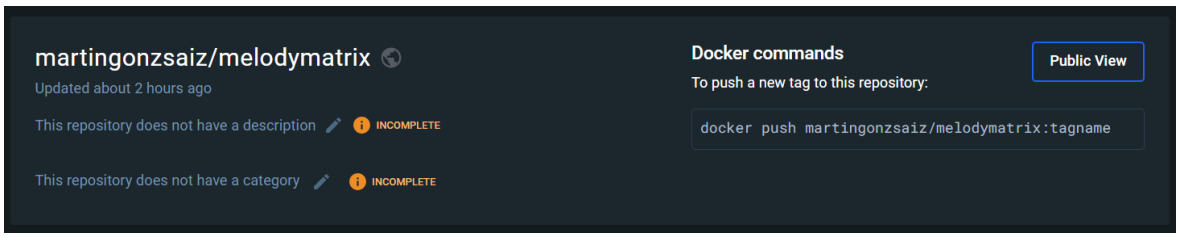

Figura 19 Imagen subida en Docker Hub

el trámite sería descargarla mediante el siguiente comando:

docker pull martingonzsaiz/melodymatrix:latest

El siguiente paso sería ejecutarla mediante el siguiente comando:

docker run -p 5000:5000 martingonzsaiz/melodymatrix:latest

Finalmente, como se mapea el puerto 5000 del contenedor al puerto 5000 del host, para acceder a la aplicación habría que acceder a http://localhost:5000 en el navegador.

Navegador Web

También es posible acceder a la página web a través de https://melodymatrix-abf474f50a15.herokuapp.com/ pero como se ha explicado anteriormente, no está disponible la funcionalidad de digitalización en esta versión desplegada debido a que excede la memoria del plan gratuito de Heroku.

5.4 Manual de usuario

En este apartado se va a explicar y detallar las distintas posibilidades de la aplicación, yendo paso a paso por las diferentes vistas analizando sus elementos.

Home

Es la página principal de la aplicación en la que se puede iniciar sesión y registrar a un usuario. En el pie de página se ha indicado información acerca de la universidad, el trabajo de fin de grado y el mes de entrega y defensa.

Figura 20 Pagina principal

Login

Esta es la página en la que inicia sesión un usuario. El usuario debe haberse registrado previamente para poder tener éxito en su inicio de sesión, sino aparecerá una alerta indicándolo.

Figura 21 Inicio de sesión

Por ello primero es necesario registrarse previamente para poder gozar de las funcionalidades de la aplicación.

Registro

Esta es la página de registro de la aplicación. El usuario debe rellenar el formulario para poder completar el proceso de registro correctamente. La primera validación que se debe cumplir es que el usuario que se está registrando no exista. El input introducido en el campo de correo electrónico debe cumplir con las validaciones de un correo electrónico y además no puede coincidir con el correo de un usuario existente. Otra validación que se debe cumplir es que la contraseña debe coincidir con la contraseña repetida para tener éxito en el proceso.

Figura 22 Registro de usuario

Caso de que un usuario exista:

Figura 23 Caso de que un usuario exista

Caso de correo no permitido:

Figura 24 Caso de correo no permitido

Caso de correo electrónico existente:

Figura 25 Caso de correo electrónico existente

Caso de contraseñas que no coinciden:

Figura 26 Caso de contraseñas que no coinciden

Menú

En esta vista el usuario puede optar por acceder a diferentes vistas o cerrar sesión y volver a la vista del home.

Figura 27 Menú

Subir Partitura

En esta vista el usuario puede subir un archivo referente a una partitura en los formatos: '.pdf', '.png', '.jpg', '.jpeg', '.mxl' y dar un nombre que represente al archivo en el campo de texto. En el caso de que el archivo no sea MusicXML este archivo se registrará y aparecerá en la vista de partituras registradas, en caso contrario aparecerá en la vista de partituras digitalizadas. El usuario debe dar al archivo un nombre al igual que debe registrar un archivo para que se complete el proceso.

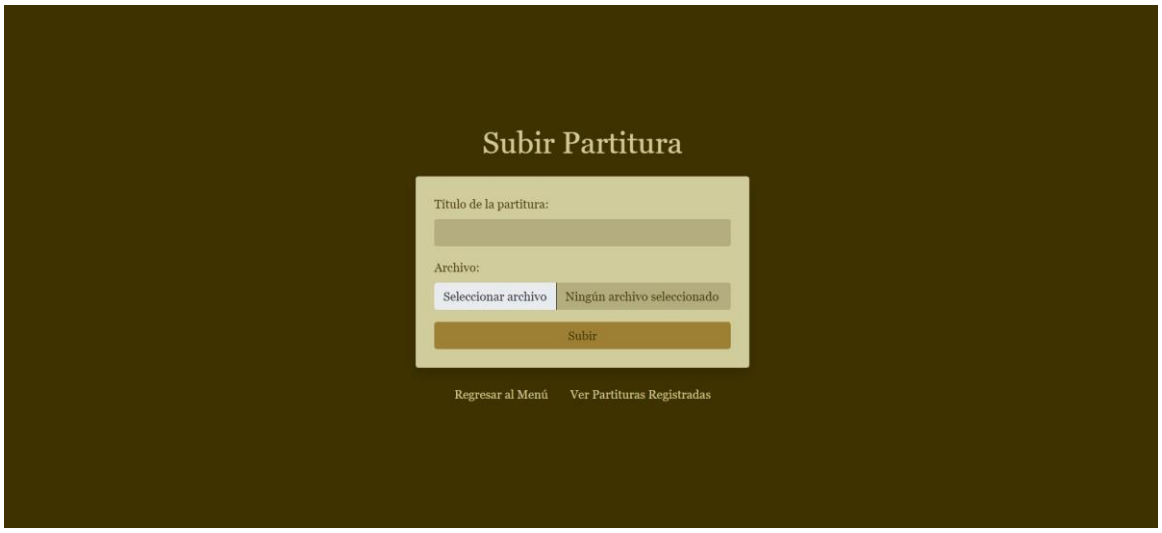

Figura 28 Subir una partitura

Caso en el que no se dé un nombre al archivo:

Figura 29 Caso en el que no se dé un nombre al archivo

Caso de que no suba un archivo:

Figura 30 Caso de que no suba un archivo

Partituras registradas

En esta vista el usuario puede optar por digitalizar, preprocesar o eliminar la partitura. En el primer caso, debido a que el proceso lleva tiempo de procesamiento, aparecerá una alerta indicando que el proceso está cargando. Si la digitalización fracasa se le comunicará con una alerta al usuario, indicando el numero registrado en los logs de Audiveris y pidiendo que se preprocese previamente. Cuando elimina una partitura también se le comunica con una alerta.

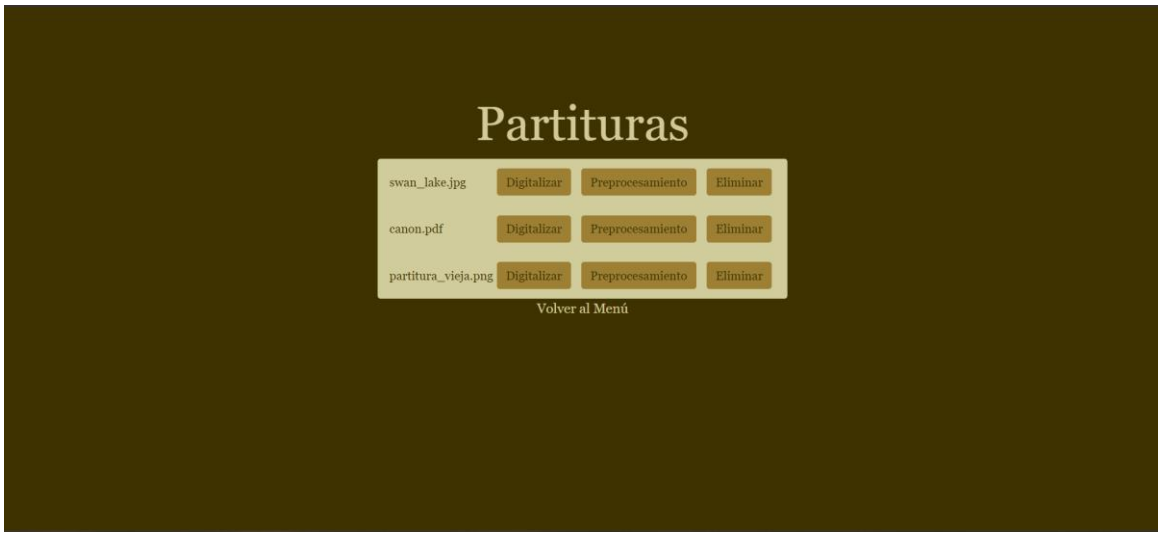

Figura 31 Partituras registradas

Digitalizando:

Figura 32 Digitalizando

Digitalización fallida:

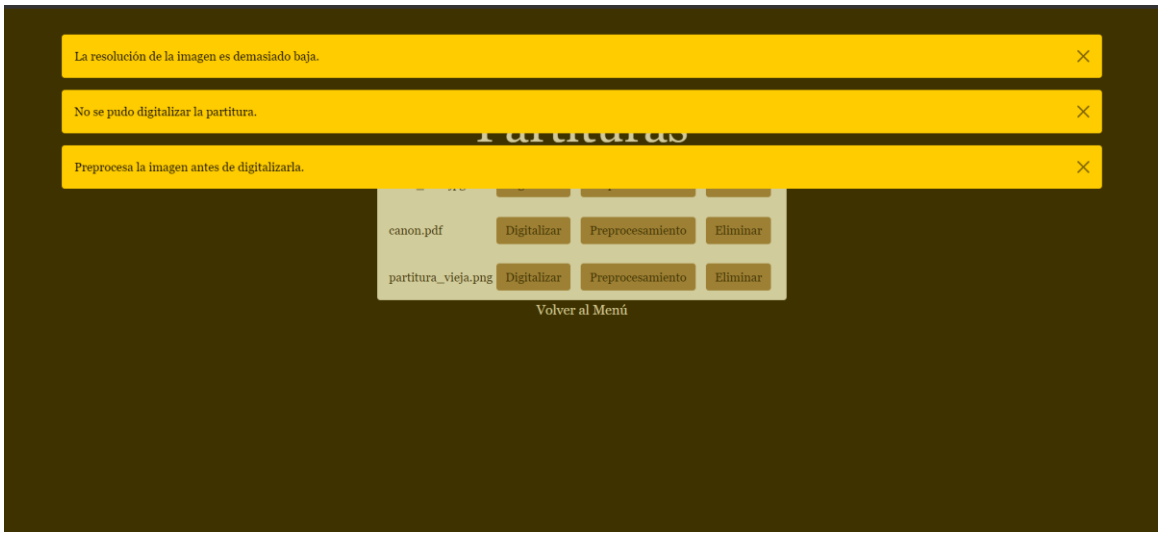

Figura 33 Digitalización fallida

Eliminación de una partitura:

Figura 34 Eliminación de una partitura (1)

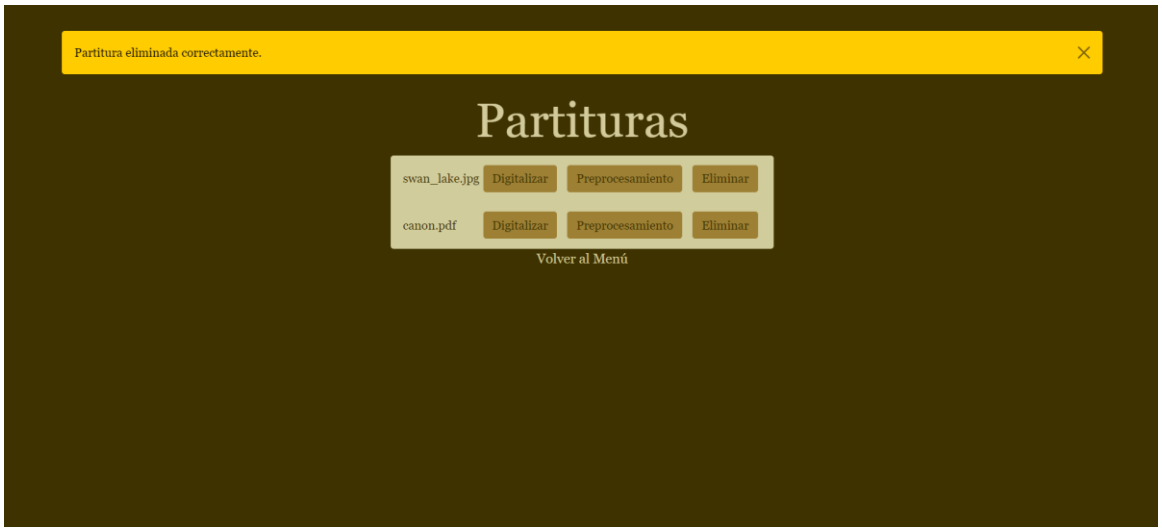

Figura 35 Eliminación de una partitura (2)

Preprocesamiento

En esta ventana se mostrará un formulario al usuario para que introduzca diferentes valores que pueden variar en las diferentes funciones de procesamiento de imágenes. La idea de esta funcionalidad es que el usuario encuentre una combinación con la que conseguir una partitura lo suficientemente legible como para poder llegar a digitalizarla. Por defecto se muestran unos valores bastante óptimos para cualquier imagen para mejorar la calidad de la imagen. El usuario debe introducir valores válidos para las funcionalidades ya que estas funciones responden a valores dentro de un rango lógico dentro del trabajo que hacen en el procesamiento. En el caso de Tamaño del Kernel Mediano el valor debe de ser impar y positivo que sea mayor que 1, ya que representa la dimensión de la matriz para aplicar el filtro. Esto ocurre también con el Tamaño del Bloque para el Umbral Adaptativo. Para el caso de la Erosión y la Dilatación se acepta cualquier valor entero que no sea negativo.

Figura 36 Preprocesamiento

Caso Tamaño del Kernel Mediano y Tamaño del Bloque para Umbral Adaptativo par:

Figura 37 Tamaño del Kernel Mediano par

Figura 38 Tamaño del Bloque para Umbral Adaptativo par

Caso Iteraciones de Erosión y Dilatación negativo:

Figura 39 Iteraciones de Erosión negativo

Figura 40 Iteraciones de Dilatación negativo

Caso de sobrecarga de datos por valores altos:

Figura 41 Caso de sobrecarga de datos por valores altos

Partituras digitalizadas

En esta vista se manejan las partituras digitalizadas. Cuando el usuario es redirigido a esta vista tras digitalizar una partitura se le comunica con una alerta. El usuario puede optar visualizar la partitura, descargarla o eliminarla. En caso de eliminarla también se le comunica con una alerta.

Figura 42 Partituras digitalizadas

Figura 43 Partitura digitalizada correctamente

Eliminación de partitura digitalizada:

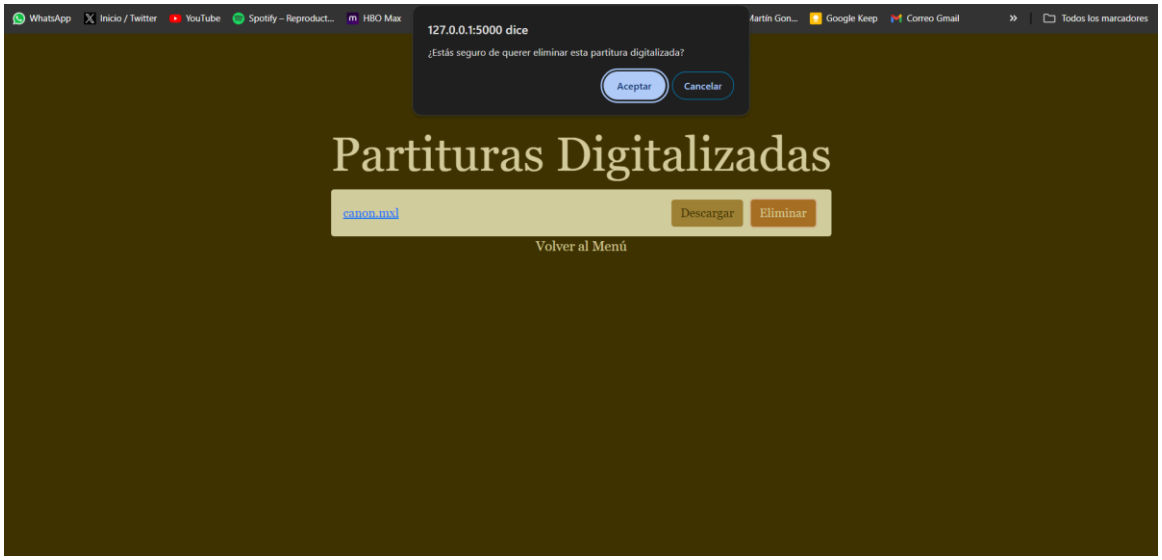

Figura 44 Eliminación de partitura digitalizada (1)

Figura 45 Eliminación de partitura digitalizada (2)

Visualización de partitura

En esta vista se visualiza una partitura entera. Como muchas partituras son de escalas grandes y se buscaba visualizar los elementos con detalle se decidió ampliar mucho el punto de visión y moverse mediante los scrolls. El usuario puede volver al menú mediante un botón en la parte baja de la vista.

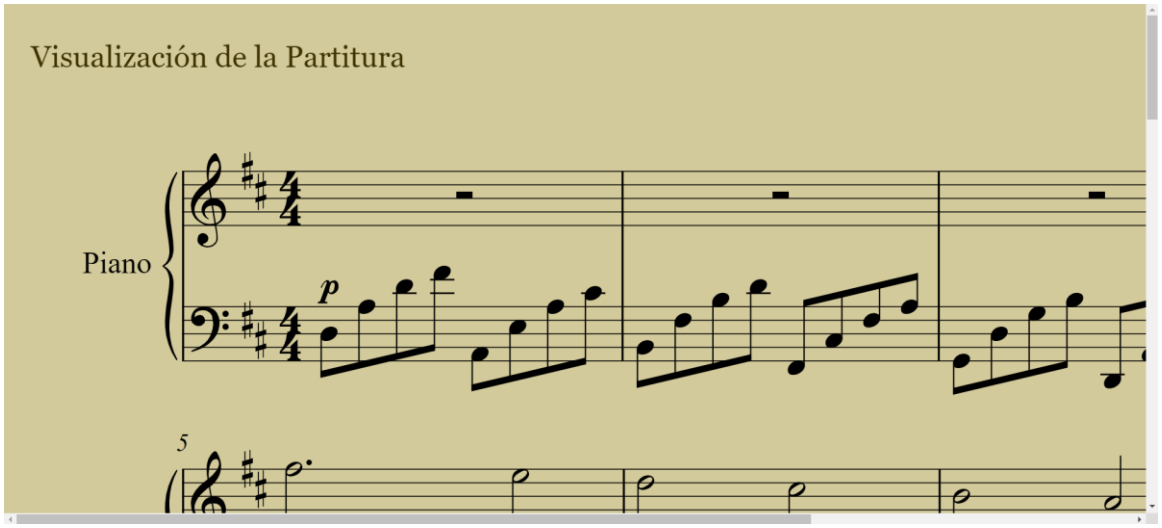

Figura 46 Visualización de partitura (1)

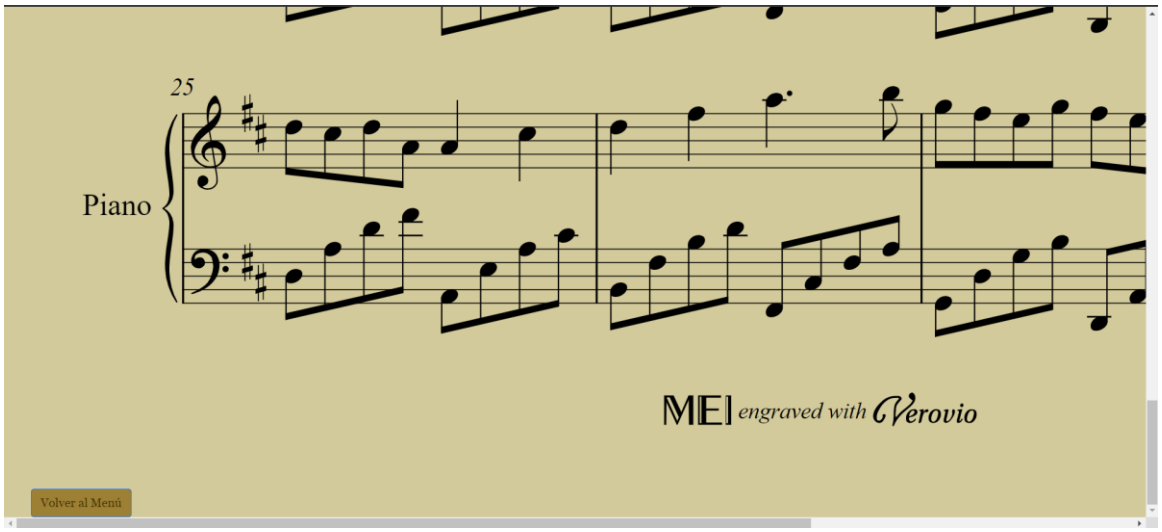

Figura 47 Visualización de partitura (2)

Apéndice F Anexo de sostenibilización curricular

En este apartado se van a listar los objetivos de desarrollo sostenible que alcanza este proyecto. (10)

Objetivo 4: Educación de calidad

El proyecto contribuye al Objetivo de Desarrollo Sostenible 4, 'Educación de calidad', al aportar en gran medida un recurso funcional para la educación musical. Siendo su acceso libre, permite la edición y manipulación de las partituras en formato digital, abriendo múltiples posibilidades a profesores, alumnos y personas autodidactas para gestionar diferente material del sector.

Objetivo 9: Industria, innovación e infraestructura

El proyecto contribuye al Objetivo de Desarrollo Sostenible 9, 'Industria, innovación e infraestructura', al implementar una tecnología para la digitalización y edición de partituras, mejorando la infraestructura tecnológica y fomentando la innovación en el sector.

Objetivo 12: Producción y consumo responsables

El proyecto contribuye al Objetivo de Desarrollo Sostenible 12, 'Producción y consumo responsables', mediante el uso de la herramienta los usuarios reducirán el consumo de papel, al optar por un formato digital. Esta práctica fomenta prácticas más sostenibles que alienta por un recurso más eco-friendly y eficiente.

Bibliografía

1. Talent.com. «Salario para Programador Junior en España - Salario Medio». Accedido 29 de junio de 2024. https://es.talent.com/salary.

2. Agency, Monster Digital. «Sueldos de un desarrollador junior: Salario de un developer junior». Epitech Spain (blog), 10 de diciembre de 2021. https://www.epitech-it.es/sueldo-desarrollador-junior/.

3. «Seguridad Social: Cotización / Recaudación de Trabajadores». Accedido 8 de julio de 2024. https://www.seg-social.es/wps/portal/wss/internet/Trabajadores/CotizacionRecaudacionTrabajadores/36537.

4. Python.org. «Download Python». Accedido 8 de julio de 2024. https://www.python.org/downloads/.

5. «Git - Downloads». Accedido 8 de julio de 2024. https://git-scm.com/download.

6. «Docker: Accelerated Container Application Development», 10 de mayo de 2022. https://www.docker.com/.

7. «Visual Studio Code - Code Editing. Redefined». Accedido 8 de julio de 2024. https://code.visualstudio.com/.

8. «The Heroku CLI | Heroku Dev Center». Accedido 8 de julio de 2024. https://devcenter.heroku.com/articles/heroku-cli.

9. «Firebase console». Accedido 8 de julio de 2024. https://console.firebase.google.com/u/0/.

10. «Objetivos y metas de desarrollo sostenible - Desarrollo Sostenible». Accedido 8 de julio de 2024. https://www.un.org/sustainabledevelopment/es/objetivos-de-desarrollo-sostenible/.