UNIVERSIDAD DE MÁLAGA

E.T.S. INGENIERÍA INFORMÁTICA
UNIVERSIDAD DE MÁLAGA

Graduado en Ingeniería Informática

Desarrollo de un sistema WEB para el proyecto LIMBO Space: Social Sound XP

Development of a WEB system for the LIMBO Space project: Social Sound XP

Realizado por
Miguel Alfonso Moretón Schulz

Tutorizado por
José Francisco Chicano García

Departamento
Lenguajes y Ciencias de la Computación

MÁLAGA, septiembre 2022

UNIVERSIDAD DE MÁLAGA

E.T.S. INGENIERÍA INFORMÁTICA
UNIVERSIDAD DE MÁLAGA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

GRADUADO EN INGENIERÍA INFORMÁTICA

Desarrollo de un sistema WEB para el proyecto LIMBO Space: Social Sound XP

Development of a WEB system for the LIMBO Space project: Social Sound XP

Realizado por

Miguel Alfonso Moretón Schulz

Tutorizado por

José Francisco Chicano García

Departamento

Lenguajes y Ciencias de la Computación

UNIVERSIDAD DE MÁLAGA

MÁLAGA, SEPTIEMBRE DE 2022

Fecha defensa: octubre de 2022

Dedicatoria y agradecimientos

Le dedico este trabajo a mis padres que siempre han sido mis referentes en la vida. Sin ellos no hubiera podido llegar tan lejos y espero poder retribuirles en vida todo el amor, apoyo, cariño, y comprensión; especialmente en mis años universitarios y durante los momentos más difíciles. Espero que estén orgullosos, porque yo lo estoy de ustedes.

Agradecimiento a todos los miembros de la Free Software Foundation, Stack-Overflow, YouTube, GitHub y en especial a Junior Toussaint creador de la página getarrays.io. Sin el apoyo desinteresado, información y proyectos de carácter público que han compartido no hubiera sido posible el desarrollo de este trabajo. Espero próximamente contribuir a esta gran comunidad y poder así ayudar a estudiantes, desarrolladores e ingenieros de todo el mundo.

¡Gracias totales!

Entre lo bueno y lo excepcional hay un paso…Lo importante es darlo.

Resumen

LIMBO es un proyecto financiado por el Vicerrectorado de Smart Campus, cuyo objetivo es el de crear una estructura artística sonora y de forma circular ubicada en un espacio público del campus de la Universidad de Málaga (UMA). Dicha estructura tiene altavoces dispuestos en su periferia que apuntan hacia el centro y donde, uno o varios usuarios, pueden estar en el centro de esta. Los usuarios podrán interactuar con la estructura mediante el uso de un sistema de información, interfaz principal y componentes de manejo de efectos de audio que dan soporte a la experiencia sonora. Mediante gestos en la interfaz principal, los usuarios serán capaces de interactuar con los audios y efectos disponibles en el sistema de información generando efectos de sonido 3D interactivos.

El concepto del proyecto, su funcionalidad multimedia y características, requieren de un sistema de información WEB que dé respuesta tanto a sus necesidades presentes como futuras, tomando en cuenta la continua evolución y cambios que puede sufrir. Este trabajo fin de grado se centra en el análisis, diseño y desarrollo de dicho sistema de información. Para ello se hizo elección de una metodología ágil e iterativa de desarrollo "SCRUM", donde una vez extraídos los requisitos, se dividió el trabajo en "Sprints" de desarrollo para ir trabajando en cada uno de ellos.

Desde el principio, todos los requisitos (de información, funcionales y no funcionales) fueron cambiando a medida que se iban asentando las bases del proyecto; y con ello también el proceso de análisis, diseño y desarrollo del sistema. Para poder hacer frente a dichos cambios durante todo el proceso, fue clave la correcta elección de herramientas y arquitecturas que permitieran dicho dinamismo y adaptación a lo largo de su duración,

sin perder de vista los objetivos adicionales de reutilización, mantenibilidad y mejora. Siguiendo las pautas de diseño marcadas por las herramientas elegidas y moldeadas por los requisitos del proyecto, se pudo desarrollar un sistema de información WEB desacoplado con alto valor estructural a nivel de software que da respuesta a las necesidades multimedia y de información de los futuros usuarios.

El resultado obtenido no solo es un sistema de información WEB escalable, mantenible y desacoplado; sino que también permite la adición de más componentes , a la fecha todavía en desarrollo. De igual forma, el sistema logró responder a todos los requisitos obligatorios inicialmente planteados y modela la arquitectura para la implementación de los futuros. Si bien hay amplio margen de mejora y trabajo restante por parte de los demás equipos pertenecientes al proyecto, la consecución del sistema de información marca un antes y un después para lograr el éxito de LIMBO como proyecto artístico, educativo, interactivo e innovador.

Palabras claves: Sistema de información, WEB, Arquitectura Software, Multimedia, Experiencia sonora, Sonido 3D, Software, Backend, Frontend

Abstract

LIMBO is a project financed by the Smart Campus Vice-Rectorate, whose objective is to create a circular artistic sound structure located in a public space on the campus of the University of Malaga (UMA). That structure has loudspeakers arranged on its periphery pointing toward its center and where, one or more users can be in the center of it. Users will be able to interact with the structure through an information system, the main interface, and audio effects management components that support the sound experience. Through gestures in the main interface, users will be able to interact with audio and effects, available in the information system, generating real-time interactive 3D sound effects.

The concept of the project, its multimedia functionality, and its characteristics, require a WEB information system that responds to both its present and future needs, considering the continuous evolution and changes that it may undergo. This Final degree project focuses on the analysis, design, and development of the information system. For this, "SCRUM" was chosen as the development methodology where, once the requirements were extracted, the work was divided into development "Sprints" to work on each one of them.

Since the beginning, all the requirements (information, functional and non-functional) were changing while the foundations of the project were laid; and with it also the process of analysis, design, and development of the system. To be able to deal with these changes, a correct set of tools and architectures were chosen that allowed the dynamism and adaptation required, without losing the focus on the additional objectives of reusability, maintainability, and upgradability. Following the design guidelines marked by the chosen

tools and shaped by the project requirements, it was possible to develop a decoupled WEB information system with high software structural value that responds to the multimedia and information needs of future users.

The result obtained is not only a scalable, maintainable, and decoupled WEB information system; but also, one that allows the addition of more components, which to date are still under development. Likewise, the system managed to respond to all the initially established mandatory requirements and models the architecture needed for future ones. Although there is room for improvement and work remaining by other teams that belong to the project, the accomplishment of the information system marks a before and after to achieve the success of LIMBO as an artistic, educational, interactive, and innovative project.

Keywords: Information system, WEB, Software Architecture, Multimedia, Sound Experience, 3D Sound, Software, Backend, Frontend

Índice

Diccionario 10

Términos 10

Acrónimos 11

1. Introducción 13

1.1. Estudio del estado arte 14

1.2. Tecnologías 15

1.3. Metodología de trabajo empleada 16

1.4. Fases de Desarrollo 18

2. Diseño del sistema 21

2.1. Descripción general del sistema 21

2.2. Análisis de requisitos 22

2.2.1. Requisitos de Información 23

2.2.2. Requisitos Funcionales 25

2.2.3. Requisitos No Funcionales 27

2.3. Instalación y configuración de las herramientas de desarrollo 28

2.4. Distribución de los requisitos funcionales en diferentes entregas parciales 31

3. Desarrollo del sistema 35

3.1. Sprint 1 36

3.2. Sprint 2 41

3.2.1. Autenticación 41
3.2.2. Autorización 44
3.2.3. Comprobaciones y otras funciones de seguridad 45
3.3. Sprint 3 46
3.4. Sprint 4 48
3.5. Sprint 5 51
3.5.1. Prototipo de manejo de audio. 51
3.5.2. Integración del prototipo de audio en el sistema de información 55
3.6. Sprint 6 57
3.6.1. Descripción general del formato y uso de los puntos de conexión 60
3.6.2. Puntos de acceso 61
3.7. Sprint 7 68
3.8. Sprint 8 70
3.9. Sprint 9 71
4. Conclusiones y Líneas Futuras 73
4.1. Conclusiones 73
4.2. Líneas Futuras 74
Bibliografía 77
Apéndice A 80
I. Imágenes del sistema de información 80
I.I. Vistas principales 80

I.II. Vistas del área de usuario.....83

Diccionario

Términos

- *API RESTful*: "es una interfaz que dos sistemas de computación utilizan para intercambiar información de manera segura a través de Internet" (Amazon Web Services, s. f.).
- API: acrónimo de interfaz de programación de aplicaciones, o *application programming interface* por sus siglas en inglés. Es un conjunto de subrutinas, funciones y procedimientos que ofrece cierta biblioteca para ser utilizada por otro software como una capa de abstracción (Wikipedia, s. f.).
- Back-End: capa de acceso y lógica de negocio de una aplicación.
- *Framework*: "estructura conceptual y tecnológica de asistencia definida, normalmente, con artefactos o módulos concretos de software, que puede servir de base para la organización y desarrollo de software" (Wikipedia, s. f.).
- Front-End: capa de presentación e interacción de datos de una aplicación.
- *Full stack*: se puede referir a un perfil de desarrollo o aplicación que incluye todos los componentes tecnológicos de un sistema: desde el *Frontend*, hasta el *Backend* y todas las conexiones y operaciones intermedias (Torralba, 2021).
- MIME: *Multi-purpose Internet Mail Extensions* por sus siglas en inglés es un identificador de dos partes para formatos de archivo transmitidos por Internet (Yuan, 2020).
- Pila tecnológica: mejor conocido como stack tecnológico es el conglomerado de tecnologías usadas en aplicaciones, desde la base de datos hasta la interfaz de usuario y todos los sistemas intermedios.

- Sprint: núcleo central de la metodología de trabajo 'scrum'. Se trata de un miniproyecto de no más de un mes (ciclos de ejecución muy cortos entre una y cuatro semanas), cuyo objetivo es conseguir un incremento de valor en el producto a construir (BBVA, s. f.).
- *Streaming:* proceso para enviar y recibir paquetes de datos en un flujo continuo a través de una red. Permite que se inicie la reproducción de contenido mientras el resto de los datos todavía está en tránsito sin descargar archivos que ocupan espacio de almacenamiento en los dispositivos (Lato, 2021).

Acrónimos

- BD: base de datos.
- CRUD: creación, lectura, modificación y eliminación, o *create, read, update* y *delete* por sus siglas en inglés.
- IDE: acrónimo de entorno de desarrollo integrado, o *integrated development enviroment* por sus siglas en inglés.
- MB: megabytes.
- N/A: acrónimo de "no aplica".
- OS: sistema operativo u *operacional system* por sus siglas en inglés.
- S. F.: sin fecha.
- TIC: tecnologías de la información y la comunicación.

1. Introducción

El presente trabajo consiste en el análisis y desarrollo de un sistema de información WEB para la implementación de las funcionalidades del proyecto LIMBO Space (Social Sound Experience). Dicho proyecto está concebido como una futura instalación artística sonora dispuesta en un espacio público dentro del campus universitario, con el cual se puede interactuar mediante una interfaz web. En el sistema de información los usuarios deben ser capaces de realizar todas las funciones de control y manejo de cuentas, además de poder añadir audios e interactuar con la estructura a través de una interfaz WEB.

Surge entonces la necesidad de un sistema WEB capaz de controlar la funcionalidad de reproducción de audio en los altavoces integrados en la estructura, al igual que una interfaz que permita a los usuarios interactuar con la misma bajo unos parámetros definidos por "usuarios administradores". Dicho sistema debe poder ser modificado y actualizado por los usuarios autorizados para controlar las funcionalidades de reproducción e interacción con la estructura, además del acceso de los demás usuarios a las mismas.

El proyecto LIMBO Space: Social Sound XP, es un proyecto multidisciplinar que reúne a equipos de diferentes facultades de la Universidad de Málaga y con un alcance extenso que abarca todos los componentes del diseño, desarrollo y puesta en marcha. Por lo cual es preciso acotar el alcance del presente trabajo fin de grado que abarca exclusivamente el **análisis y desarrollo del sistema de información WEB** necesario para dar soporte a las funcionalidades del proyecto. A pesar de ello a lo largo del trabajo se mencionan distintas interacciones con los grupos de trabajo relacionados en el proyecto que, de forma directa o indirecta, influyeron sobre el análisis y desarrollo del sistema.

Teniendo en cuenta el apartado anterior los objetivos principales del TFG son: el análisis exhaustivo del presente sistema extrayendo todos los requisitos funcionales, no funcionales y de información; el desarrollo y prueba del sistema de información; y la documentación tanto del código como la elaboración de todos los entregables asociados con el sistema y su funcionamiento. El sistema resultante debe ser mantenible, escalable y seguro con el fin de garantizar su continuidad, mejora y posible utilización para futuros proyectos académicos. Estas características influyeron tanto en la elección del stack o "pila tecnológica" como en la implementación de las funcionalidades y documentación con el fin de garantizar la mayor calidad posible.

1.1. Estudio del estado arte

El proyecto Limbo Space: Social Sound XP nace de la mano de un grupo multidisciplinar de profesores de la UMA con el objetivo de ampliar los espacios públicos de carácter múltiple: social, cultural y de participación ciudadana dentro de la Universidad de Málaga. Su objetivo es ser un espacio interactivo de carácter sonoro y cuya funcionalidad se plantea en constante redefinición para un uso abierto, múltiple y flexible (Vicerrectorado de Smart-Campus, s. f.). Este tipo de espacios artísticos interactivos surgieron a inicios del siglo XX para potenciar las nuevas formas de expresión en las cuales se requería la interacción del espectador (Waelder Laso and G. Díaz, 2019), siendo esta interacción pieza fundamental para enlace entre artista, obra y espectador. Esta interacción se ha visto reforzada en los últimos años gracias al uso de las Tecnologías de la Información y la Comunicación (TIC), un ejemplo de ello es el éxito que ha tenido una de las grandes fuentes de inspiración de este trabajo "**Reactable**", cuya propuesta es la de un instrumento musical electrónico que permite experimentar con sonido de una manera cautivadora, divertida y visualmente atractiva (Reactable Systems SL , s. f.).

Figura 1. *Reactable Live! Table Structure and figures.* (Reactable Systems SL, s. f.)

1.2. Tecnologías

Para el desarrollo de sistemas WEB existen múltiples tecnologías que se podían emplear, sin embargo, y dentro del marco de objetivos del proyecto, la elección de estas se basó en algunos factores claves como: documentación disponible, interoperabilidad entre tecnologías, soporte actual y futuro por parte de las compañías y la comunidad; y por último y más importante tecnologías y lenguajes de programación empleados actualmente en asignaturas de la Universidad de Málaga. Este último punto se ha considerado de especial relevancia para la mantenibilidad y reusabilidad del código en proyectos futuros. Tomando en cuenta las consideraciones del apartado anterior las tecnologías y herramientas elegidas fueron:

Apartados	Tecnología	Herramientas	Lenguajes
Base de Datos	Base de datos relacional	MySQL MySQL Workbench	SQL
Back-End	*Framework* de desarrollo	Spring	Java 7
Front-End	*Framework* de desarrollo	Angular	Typescript JavaScript HTML5

			CSS3
Entorno de ejecución	Entorno de ejecución	Nodejs	n/a
IDEs de desarrollo	Herramienta de desarrollo	IntelliJ IDEA Visual Studio Code	n/a
Herramienta API	Herramienta de testing	Advanced REST client	n/a
Servidor de correo	Servidor SMTP	mailDev	n/a
Diseño web	Herramienta de diseño	Bootstrap Studio	n/a
Repositorio de código	Repositorio de almacenamiento de Código	GitHub	Git
Herramientas adicionales	Herramienta de autocompletado	GitHub Copilot	n/a

Tabla 1. Tecnologías y herramientas de desarrollo escogidas.

1.3. Metodología de trabajo empleada

Desde el principio del proyecto existía la necesidad de emplear una metodología ágil de desarrollo que se adaptará rápidamente a los cambios, ya que, si bien la idea y concepto base estaban definidos, muchas de las funcionalidades técnicas para dar soporte a la funcionalidad deseada no lo estaban, o si quiera se habían definido. Por ello se eligió la metodología ágil SCRUM, qué consiste en un marco de trabajo para el desarrollo ágil de software basado en el desarrollo incremental e iterativo de requisitos en bloques cortos y fijos priorizando las funcionalidades claves del sistema (Drumond, s. f.). Dicho marco se adapta perfectamente a las necesidades del proyecto y a su naturaleza flexible, similar a un proyecto de negocio donde el cliente sabe lo que quiere, pero no tiene definido el "cómo", y es allí donde SCRUM ayuda a subsanar las carencias de requisitos técnicos, mediante prototipos iterativos donde se pueda ir probando el sistema y modificando a demanda.

Siguiendo las directrices de SCRUM se definieron Sprints de dos o cuatro semanas, dependiendo de los requisitos a implementar, con retroalimentación directa del tutor para la verificación del proceso de desarrollo, consecución de requisitos y corrección o adición de requisitos y funcionalidades adicionales a cada entrega. Las entregas constan de una reunión inicial para definir los objetivos del presente Sprint, un control a mitad del periodo (siete o catorce días naturales) para la supervisión del progreso realizado; y una reunión al final del Sprint para la verificación de los resultados obtenidos y cuáles deben ser postergados u adelantados para futuras entregas. Las entregas se definieron con dicho carácter cíclico para permitir las posibles modificaciones del proyecto a lo largo de su ejecución, pruebas manuales para la verificación del funcionamiento, y revisión de objetivos pasados y futuros.

Debido al carácter multidisciplinar del proyecto y teniendo en cuenta que este iba a ser diseñado y desarrollado inicialmente únicamente por mi persona bajo supervisión del tutor, la metodología SCRUM fue la herramienta perfecta para iniciar el mismo, y posteriormente permitir la incorporación de otros equipos que fueron trabajando en paralelo y por separado. La retroalimentación constante ayudaría a la adaptación sobre la marcha de los requisitos cambiantes y reduciría posibles errores de requisitos que podrían surgir debido a la falta de definición y carácter cambiante del proyecto. Sin embargo, también podían ralentizar el desarrollo por las esperas necesarias para obtener retroalimentación de lo hecho hasta el momento.

1.4. Fases de Desarrollo

Inicialmente se plantean siete fases de desarrollo, que podrían estar sujetas a cambios dependiendo del avance del proyecto, cambio de requisitos y restricciones de tiempo y/o recursos económicos, tecnológicos o técnicos. Las Fases del desarrollo fueron las siguientes:

1. Análisis de requisitos

Consiste en la extracción y documentación de los requisitos funcionales, no funcionales y de información del proyecto. Es uno de los pasos más importantes debido a que estos requisitos vienen marcando la ruta de desarrollo del sistema y una definición deficiente podría acarrear cambios en la lógica del sistema afectando a varios componentes de esta.

2. Instalación y configuración de las herramientas de desarrollo

Debido a que el sistema está concebido como una aplicación "full stack" hay múltiples tecnologías y herramientas necesarias para su desarrollo, y la configuración de estas puede acarrear dificultades técnicas y consumo de tiempo que hay que reflejar.

3. Distribución de los requisitos funcionales en diferentes entregas parciales

En esta fase el objetivo es ponderar y distribuir de la forma más equitativa posible las funciones a desarrollar en los diferentes Sprints, si bien esta distribución es altamente volátil en proyectos dinámicos como LIMBO, donde los requisitos pueden volverse más complejos o eliminarse en función de la retroalimentación de cada Sprint.

4. Desarrollo de los requisitos funcionales

Fase más extensa del trabajo donde se contempla la implementación y verificación iterativa de los requisitos hasta la consecución de los resultados. Al igual que el punto anterior está sujeto a cambios debido a los posibles cambios de requisitos o adición de nuevos.

5. Pruebas manuales

Se plantean pruebas manuales, durante todo el proceso de desarrollo, de las funciones implementadas hasta el momento para corregir posibles fallas y errores e ir comprobando el correcto funcionamiento del sistema en general. Esta fase puede involucrar a otros equipos encargados de otras funcionalidades o apartados del proyecto LIMBO.

6. Documentación del desarrollo

Una vez finalizado el desarrollo del sistema de información se fija una etapa de documentación donde se añaden las descripciones necesarias de la mayoría de los componentes del sistema de información a vista de posibles reutilizaciones para proyectos futuros, mantenibilidad y corrección de errores.

7. Elaboración y revisión de documentos

Se precisa aportar las indicaciones de acceso al repositorio de datos donde se encuentra alojado el código del sistema. En esta etapa se considera también la redacción de la memoria del presente TFG como parte de los documentos técnicos a entregar al equipo del proyecto LIMBO.

2. Diseño del sistema

2.1. Descripción general del sistema

Previa a la redacción de los requisitos funcionales se realizaron múltiples reuniones virtuales para asentar los requisitos del sistema de información y la aplicación en general. El tutor explicó los objetivos del proyecto haciendo referencia en múltiples ocasiones a la "Memoria del proyecto de investigación del plan propio de smart-campus", pero añadiendo comentarios adicionales y sugerencias de desarrollo. En este punto se construyó una descripción general del sistema que se describe a continuación:

El sistema de información debe permitir a los usuarios darse de alta, baja, iniciar y cerrar sesión de uso. El sistema debe ser multiusuario capaz de atender a varias peticiones al mismo tiempo y donde el acceso se base en función de roles que posea el usuario, limitando aquellas funciones de administración a usuarios con la autorización necesaria. Una vez dados de alta los usuarios deben poder modificar y añadir información en su perfil al igual que añadir, modificar, eliminar y escuchar audios. El sistema debe dar soporte a la autenticación ya sea mediante IDUMA o uno temporal mientras esta es configurada. En el segundo caso se debe dar soporte a la autenticación de direcciones de correo electrónico y funcionalidades de recuperación y reinicio de contraseña.

El sistema debe implementar una funcionalidad de espera que gestione el acceso a la interfaz de interacción, a realizar por otros miembros del proyecto, y dar soporte a los efectos de audios que de igual forma están a cargo de otro equipo. La interfaz de interacción se basará en figuras que tendrán audios asociados que el usuario podrá mover, y en donde, en función del movimiento y elección de las figuras, se modificarán los audios mediante

efectos sonoros asociados a las figuras. El sistema de información no contempla el desarrollo de esta funcionalidad, pero sí deberá dar soporte a la misma para que esta pueda implementarse. Finalmente, los audios serán reproducidos mediante altavoces en la estructura conectados a una tarjeta de audio la cual el sistema de información deberá utilizar para enviar la información de los audios a la estructura.

Vale la pena acotar que el servidor va a ser un ordenador cerca de la estructura sonora, por lo cual se tienen ciertas limitantes de memoria y capacidad de procesamiento. Se valora la opción de separar las funciones del servidor de las de la interfaz del usuario, para no sobrecargar a este. Por lo cual se plantea un sistema de información desacoplado que separe el *Backend* del *Frontend*, en vez de integrarlo todo en una sola aplicación.

2.2. Análisis de requisitos

Si bien la idea de tener un sistema de información que permitiera a los usuarios subir audios y manejarlos de forma interactiva para producir efectos sonoros a través de figuras mediante una tarjeta de audio conectada a altavoces en una estructura estaba claro, el cómo realizar dichas funciones, requisitos técnicos e incluso de seguridad no estaban definidos. Lo anterior derivó en unos requisitos abiertos a cambios y que debían ser modelados de forma tal que una modificación implicara el menor esfuerzo posible de edición de código y lógica de funcionamiento.

Por las presentes restricciones de tiempo se fijaron adicionalmente unos requisitos deseables que añadirían valor al sistema resultante, pero por falta de experiencia y recursos no se tenía fijado su consecución. Los requisitos definidos fueron los siguientes:

2.2.1. Requisitos de Información

La siguiente tabla detalla la información que el sistema almacenará:

Referencia	Entidad	Información
RI-1	Usuario	• id • nombres • apellidos • fecha de nacimiento • correo electrónico • contraseña • rol de usuario • permisos • Imagen de perfil • supervisor
RI-2	Librería	• id • **usuario** • **audios**
RI-3	Información de archivo	• id • nombre • tipo de archivo MIME • tamaño en MB • fecha de subida • indicador de archivo público • **usuario**
RI-4	Audio	• id • formato • duración en minutos • **información de archivo**
RI-5	Figura	• id • nombre • Imagen • **audios**

Tabla 2. Requisitos de información del sistema.

Es conveniente describir los atributos de las entidades detalladas en la tabla 2 y su función dentro del sistema:

- Usuario: usuario del sistema
    - Id: identificador único.

    - Nombre: nombres del usuario. Primero y segundo de tenerlo, si no solo primero.
    - Apellidos: apellidos del usuario.
    - Fecha de nacimiento: fecha de nacimiento del usuario.
    - Correo electrónico: email del usuario.
    - Contraseña: contraseña de acceso a la aplicación.
    - Rol de usuario: rol asociado al usuario, por ejemplo: usuario, administrador, etc.
    - Permisos: permisos dentro del sistema.
    - Imagen de perfil: Imagen de perfil del usuario.
    - Supervisor: id del supervisor de dicho usuario en caso de tenerlo. Su objetivo es el de controlar quién ha cambiado el rol de un usuario o lo ha creado manualmente (se guarda siempre el último cambio).
- Librería: entidad que sirve de almacenaje de los archivos de audio de un usuario.
    - Id: Identificador único
    - Usuario: usuario asociado a la librería.
    - Audios: audios asociados a la librería.
- Información de archivo: entidad que almacena metadatos del archivo subido.
    - Id: Identificador único
    - Nombre: nombre del archivo.
    - Tipo de archivo MIME: tipo de archivo con propósitos de registro.
    - Tamaño en MB: tamaño del archivo en MB.
    - Fecha de subida: fecha en la cual se subió y guardó el archivo al sistema. Se guardará con propósitos de registro.

    - Indicador de archivo público: que detalla si el archivo lo puede visualizar todos los usuarios registrados de la plataforma o no.
    - Usuario: usuario que ha subido el archivo.
- Audio: representa un audio subido al sistema por un usuario.
    - Id. Identificador único
    - Formato: formato de audio origen (mp3, WAV, etc.)
    - Duración en minutos: duración en minutos del archivo de audio.
    - Información de archivo: información de archivo asociado
- Figura:
    - Id: Identificador único
    - Nombre: nombre de la figura.
    - Imagen: Imagen asociada a una figura.
    - Audios: audios asociados a dicha figura.

2.2.2. Requisitos Funcionales

La próxima tabla contiene las funciones que debe desempeñar el sistema de información:

Referencia	Título	Descripción
RF-1	CRUD Usuario	Un usuario debe poder crear, modificar, eliminar y ver la información de su cuenta dentro de la aplicación.
RF-2	CRUD Librería	Los usuarios deben poder crear, ver, modificar y eliminar sus librerías personales que contengan audios.
RF-3	CRUD audio	Un usuario debe poder añadir, reproducir, actualizar y eliminar un archivo de audio que le pertenezca.
RF-4	CRUD usuarios externos	Un administrador debe poder realizar las operaciones CRUD en las cuentas de usuarios de la aplicación.
RF-5	Validación de correo electrónico	Una vez el usuario haya creado una cuenta debe validar su cuenta para usar la aplicación mediante un enlace enviado a su bandeja de correo electrónico.
RF-6	Recuperación de contraseña	Un usuario debe ser capaz de recuperar su contraseña mediante un enlace enviado a su correo electrónico.
RF-7	Inicio de sesión	El usuario podrá iniciar sesión dentro de la aplicación y esta debe permanecer abierta hasta que esté la cierre o pase un

		tiempo configurado. Una vez inicie sesión deberá ser redirigido al sistema de información.
RF-8	Cierre de sesión	Un usuario podrá cerrar su sesión en cualquier momento, saliendo del sistema de información al realizar esta operación.
RF-9	Bloquear cuenta por repetidos intentos de inicio de sesión fallidos.	Si un usuario realiza un número definido de inicios de sesión fallidos en el sistema, este debe bloquear la cuenta hasta que el usuario o administrador la reactive.
RF-10	Reactivación de cuenta	Un usuario podrá reactivar su cuenta después de ser bloqueada por múltiples inicios de sesión fallidos mediante un enlace enviado al correo electrónico asociado a la cuenta.
RF-11	Habilitar y deshabilitar de audios personales	Los administradores podrán habilitar o deshabilitar la creación y uso de audios personales a los usuarios base (todos aquellos que no sean administradores).
RF-12	Reproducción de audio	La página web del sistema debe permitir escuchar los audios subidos previa utilización en la estructura.
RF-13	Cambio de formato del audio	El sistema debe ser capaz de cambiar el formato de almacenamiento de los audios según lo definan en la configuración de arranque.
RF-14	Soporte a la tarjeta de audio.	El sistema debe poder enviar a la tarjeta de audio los audios en el formato requerido para su reproducción y realizar cualquier conversión necesaria para soportar la función.
RF-15	Asociar audio a figura	El administrador podrá asignar uno o múltiples audios a una de las figuras definidas.
RF-16	Desvincular audio de figura	El administrador podrá desvincular un audio de una figura.
RF-17	Solicitud de turno	Cualquier usuario podrá solicitar un turno para usar la interfaz gráfica de la aplicación e interactuar con las figuras y audios.
RF-18	Paso a la interfaz de interacción	Permitir el paso al usuario correspondiente en la cola para que pueda usar la aplicación mediante notificación.
RF-19	Informar del estado de la cola de espera e interfaz gráfica	El sistema debe informar a los usuarios del estado actual de uso de la interfaz gráfica y la cola de espera.
RF-20	Creación de usuarios predefinidos	Cuando el sistema sea iniciado por primera vez o luego de reinicios, debe crear o comprobar que existan usuarios administradores predefinidos en la configuración para acceder al sistema.
RF-21	Creación de figuras predefinidas	Cuando el sistema sea iniciado por primera vez o luego de reinicios, debe crear o comprobar que existan las figuras predefinidas en la configuración para que estas puedan ser usadas en el sistema.

Tabla 3. Requisitos funcionales del sistema.

Requisitos funcionales adicionales

A continuación, se presenta la tabla que detalla las funciones deseables, pero no obligatorias de implementar del sistema:

Referencia	Título	Descripción
RFA-1	Autenticación con IDUMA	Usar el sistema IDUMA para todas las funcionalidades de autenticación y roles de usuario del sistema.
RFA-2	Limitar uso por proximidad	Permitir el uso de la interfaz gráfica solo si el usuario se encuentra cerca de la estructura.

Tabla 4. Requisitos funcionales adicionales del sistema.

2.2.3. Requisitos No Funcionales

La siguiente tabla describe las características del sistema y el código que lo compone.

Referencia	Título	Descripción
RNF-1	Uso de tecnologías actuales y empleadas en la Universidad	El sistema debe emplear tecnologías actualizadas, mantenidas y mantenibles usadas por la Universidad de Málaga en las asignaturas de la carrera de ingeniería informática.
RNF-2	Documentación	El sistema debe estar documentado lo máximo posible al igual que el código que lo implementa.
RNF-3	Modularización	Se debe modularizar el sistema de información para permitir su actualización y mejora constante.
RNF-4	Seguro	El sistema debe tener en cuenta las directrices de seguridad de la LOPD
RNF-5	Multiusuario	El sistema debe ser capaz de manejar 1000 usuarios de manera simultánea.
RNF-6	Mantenibilidad, escalabilidad y reutilización	El sistema debe ser mantenible, escalable y reutilizable a lo largo de su vida útil y para futuros proyectos.
RNF-7	Monitorización	Se debe ser capaz de monitorizar el sistema y la interacción de los usuarios con el mismo.

Tabla 5. Requisitos no funcionales del sistema.

2.3. Instalación y configuración de las herramientas de desarrollo

A continuación, se detalla una recopilación de las herramientas instaladas y usadas para el desarrollo del sistema. No se menciona la mayoría de las versiones utilizadas ya que estas fueron actualizándose constantemente durante el tiempo de desarrollo del sistema.

Lenguajes de programación y *Frameworks* de desarrollo

Lenguajes:

- **Java**: como lenguaje de programación para el *Backend* se usó **Java** en su versión **8** ya que es ampliamente usada en la Universidad de Málaga en varias asignaturas de la carrera. Sin embargo, y debido a la retrocompatibilidad de Java, no deberían realizarse grandes cambios si en un futuro se quiere actualizar la versión.
- **Typescript**: lenguaje usado por el *framework* Angular basado en JavaScript, pero añadiendo tipos estáticos y objetos basados en clases.
- **HTML5**: lenguaje de marcado de texto usado para la configuración de las vistas web del sistema.
- **CSS 3**: lenguaje de definición de estilos de los elementos que componen el HTML y por ende la vista de la aplicación.

Frameworks de desarrollo:

- **Spring**: *framework* de desarrollo usado en el *Backend* del sistema que utiliza Java como lenguaje de programación.
- **Angular**: *framework* usado en el *Frontend* del sistema que usa Typescript como lenguaje de programación. Para instalar dicho *framework* se instaló y usó el entorno de ejecución que se detalla a continuación.
- **Nodejs**: entorno de ejecución para JavaScript.

Es de aclarar que la decisión de usar Spring y Angular como *Frameworks* de desarrollo, viene dada por el objetivo de hacer un sistema web integral desacoplando el *Backend* y *Frontend* de tal forma que se pudieran realizar cambios en ellos de forma independiente sin afectar al otro. De esta manera ambas "partes" del sistema interactuarían entre sí mediante una API implementada por el *Backend*, y que da cabida a actualizar o incluso sustituir cualquiera de los dos componentes en un futuro, sin llegar a afectar al funcionamiento del otro siempre que se mantengan los mismos puntos de conexión. Esto aportaría una flexibilidad presente y futura para el desarrollo y mantenimiento del sistema.

Base de Datos

Como BD se eligió **MySQL** no solo por su sencillez de uso, sino también, por el extenso uso en el ámbito académico y comunitario. Para un manejo más directo se instaló tanto el motor de BD como la aplicación de manejo mediante interfaz "Workbench". Una vez instalada en el sistema y habiendo comprobado la conexión desde la interfaz se procedió a instalar las demás herramientas.

Herramientas de desarrollo

- **IntelliJ IDE**: IDE de desarrollo de java que soporta múltiples *Frameworks* de desarrollo y es ampliamente compatible con Spring.
- **Visual Studio Code**: editor de código que soporta múltiples lenguajes de programación y entornos. Incluye la posibilidad de ampliar extensiones que complementan las capacidades de este.
- **Bootstrap Studio:** aplicación de diseño y desarrollo web. Ofrece gran cantidad de componentes y herramientas para agilizar el desarrollo de interfaces y contiene plantillas que pueden ser reutilizadas y modificadas.

- **GitHub:** es un repositorio de código GIT donde se almacena el código del sistema de información y los distintos componentes que conforman el sistema LIMBO.
- **GitHub Copilot:** herramienta que emplea inteligencia artificial para completar y sugerir código "sobre la marcha" en el IDE de desarrollo. Compatible con IntelliJ, Visual Studio Code, otros IDEs y editores de código.
- **Advanced REST Client:** herramienta para probar APIs mediante uso de solicitudes REST.
- **MailDev:** Servidor SMTP de correo que permite interceptar todos los correros enviados por una aplicación, previa configuración.

Como se puede apreciar se utilizaron numerosas tecnologías, aplicaciones y herramientas para el desarrollo del sistema y a primera vista no se puede apreciar la relación entre ellas, para ello se detallan en la figura 2 a continuación:

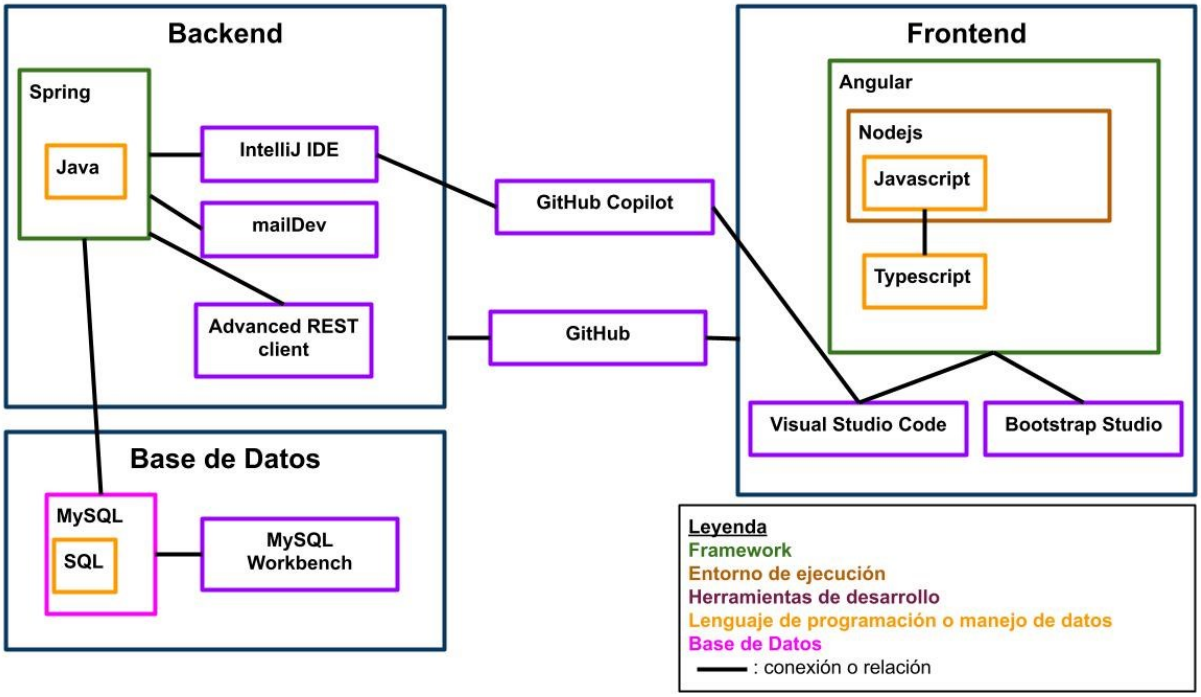


Figura 2. Relación entre tecnologías y herramientas de desarrollo.

2.4. Distribución de los requisitos funcionales en diferentes entregas parciales

La distribución de los requisitos fue una labor difícil y poco precisa ya que se tenía poca experiencia en el desarrollo de sistemas web, sin embargo, el tutor ayudó a orientar esta labor indicando qué puntos se tenían que desarrollar primero. La distribución en Sprints de las funcionalidades del sistema fue la siguiente:

- Sprint 1
    - Objetivos:
        - Modelado de las entidades y relaciones del sistema y programación en el *Backend*.
        - Configurar conexión con la BD.
    - Duración estimada: 4 semanas.
- Sprint 2
    - Objetivos:
        - Implementación de funciones de seguridad, manejo de cuenta y contraseña.
        - Configuración de funciones de administración.
    - Duración estimada: 6 semanas.
- Sprint 3
    - Objetivo:
        - Diseño de la interfaz gráfica.
    - Duración estimada: 2 semanas.

- Sprint 4
    - Objetivos:
        - Implementación de la interfaz gráfica en Angular
        - Conexión del *Backend* con el *Frontend*
        - Configuración de seguridad de conexión entre el Backend y el *Frontend*
    - Duración estimada: 4 semanas.
- Sprint 5
    - Objetivos:
        - Implementar prototipo de manejo del audio del *Backend*.
            - Manejo de conversión de audio.
        - Presentación del prototipo al equipo encargado de los efectos sonoros.
    - Duración estimada: 4 semanas.
- Sprint 6
    - Objetivos:
        - Desarrollo y pruebas de las funcionalidades básicas de la aplicación.
        - Integrar el manejo de audios y figuras dentro de la aplicación
    - Duración estimada: 4-6 semanas
- Sprint 7:
    - Objetivo: manejo del sistema de espera de usuarios para el uso de la interfaz
    - Duración estimada: 2 semanas
- Sprint 8:
    - Objetivo: Realizar documentación y finalizar redacción de la memoria.
    - Duración estimada: 6 semanas

- Sprint 9:
    - Objetivos
        - Revisión de la documentación
        - Realizar presentación de la memoria
    - Duración estimada: 2 semanas

3. Desarrollo del sistema

El inicio del desarrollo del sistema viene marcado por la programación del *Backend*, planteado como una aplicación, cuyo objetivo es el de implementar una interfaz API RESTful para la comunicación con el *Backend* y toda la funcionalidad a la cual da soporte. Para ello, se siguió la filosofía y arquitectura de Spring que define una separación entre **modelos**, **recursos**, **servicios** y **repositorios**. Los **modelos** representan las entidades que interactúan en el sistema, en este caso usuarios, librerías, audios, figuras entre otros. Los **recursos** definen los endpoints o puntos de comunicación, URL de la API que consultará el *Frontend* para realizar peticiones. Los **servicios** implementan toda la lógica del sistema y realizan casi todas las operaciones necesarias para dar respuesta a las solicitudes. Los **repositorios** se encargan de comunicar los servicios con la BD para realizar las operaciones necesarias de búsqueda, inserción, modificación y eliminación de objetos en BD.

De igual forma se decidió usar el inglés como idioma para definir no solo los nombres, sino cualquier comentario o documentación dentro del código del sistema, con el objetivo de favorecer la reutilización y no generar discrepancias entre notaciones del *framework* inglesas y comentarios o variables en español. El objetivo final es desarrollar un sistema mantenible, reutilizable y sobre todo "entendible" por todos. A pesar de ello se mantuvieron los mensajes de log del sistema y notificaciones con el usuario en español, dado que la base de usuarios y administradores seria en su mayoría de habla hispana.

A continuación, se detalla el desarrollo del sistema de información dividido en los Sprint que lo componen. Cada Sprint indica los objetivos a completar y la descripción de todas

las actividades realizadas para la consecución de estos. Dicha descripción contiene información técnica clave para entender el desarrollo y funcionamiento del sistema de información.

3.1. Sprint 1

Objetivos:

- Modelado de las entidades y relaciones del sistema y programación en el *Backend*. **Modelos** de Spring.
- Configurar conexión con la BD y el *framework* del *Backend* Spring.

El sprint 1 comienza con el modelado de las entidades que van a interactuar en la aplicación y la relación que existe entre ellas. Habiendo extraído los requisitos de información ya se tenía una aproximación inicial de las entidades del sistema, lo que quedaba era modelarlas para después empezar a traspasar dichos modelos a clases y atributos en el código del sistema. Para ello se decidió realizar un diagrama entidad relación también conocido como modelo entidad relación o ER, que es diagrama que ilustra cómo las "entidades", como personas, objetos o conceptos, se relacionan entre sí dentro de un sistema (Lucid Software Inc, s. f.). El diagrama resultante se detalla en la figura 3.

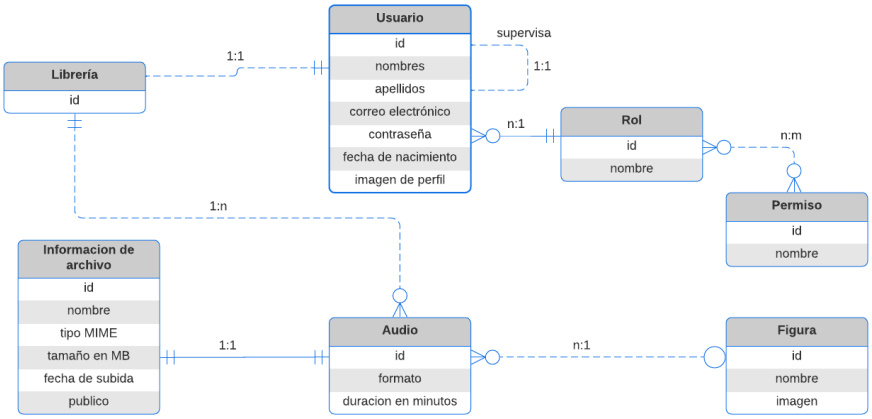


Figura 3. Diagrama ER del sistema de información de LIMBO.

Sin detallar la notación del diagrama ER, lo que pretende representar la figura 3 son las siguientes relaciones:

- Cada usuario puede tener un supervisor.
- Cada usuario puede tener una librería: la opcionalidad de esta relación es la de evitar tener que crear una librería si el usuario no sube ningún audio y usa solamente los audios públicos.
- La librería puede contener audios.
    - Cada audio debe tener una librería asociada.
- Cada audio debe tener información de archivo (que describe sus datos).
- Cada audio puede estar asociado con una figura.
- Cada figura puede tener múltiples audios asociados.
- Un usuario debe tener un rol asociado.
    - Un rol puede tener usuarios asociados.
- Cada rol puede tener múltiples permisos.
    - Cada permiso puede tener múltiples roles.

Una vez definido el ER se procedió a traspasar dichos requisitos a clases y atributos dentro del código configurando apropiadamente la BD siguiendo las indicaciones de la página oficial del *framework* del *Backend* (Spring). El manejador de persistencia de JPA se encargaría de transformar dichas clases, atributos y relaciones a sus homónimos necesarios en BD. Durante este proceso se discutieron varios aspectos relacionados con el modelaje de las entidades del sistema. Esto derivó en radicales cambios con respecto al ER de la figura 3, donde algunas entidades pasaron a ser entidades débiles, se agregaron atributos adicionales y múltiples cambios más. El siguiente Diagrama ER ampliado o EER (*Enhanced entity-relationship*) presentado en la figura 4, viene reflejando dichos cambios

además de todos los componentes, atributos y tablas necesarios para la operación del sistema de información del *Backend* y la BD.

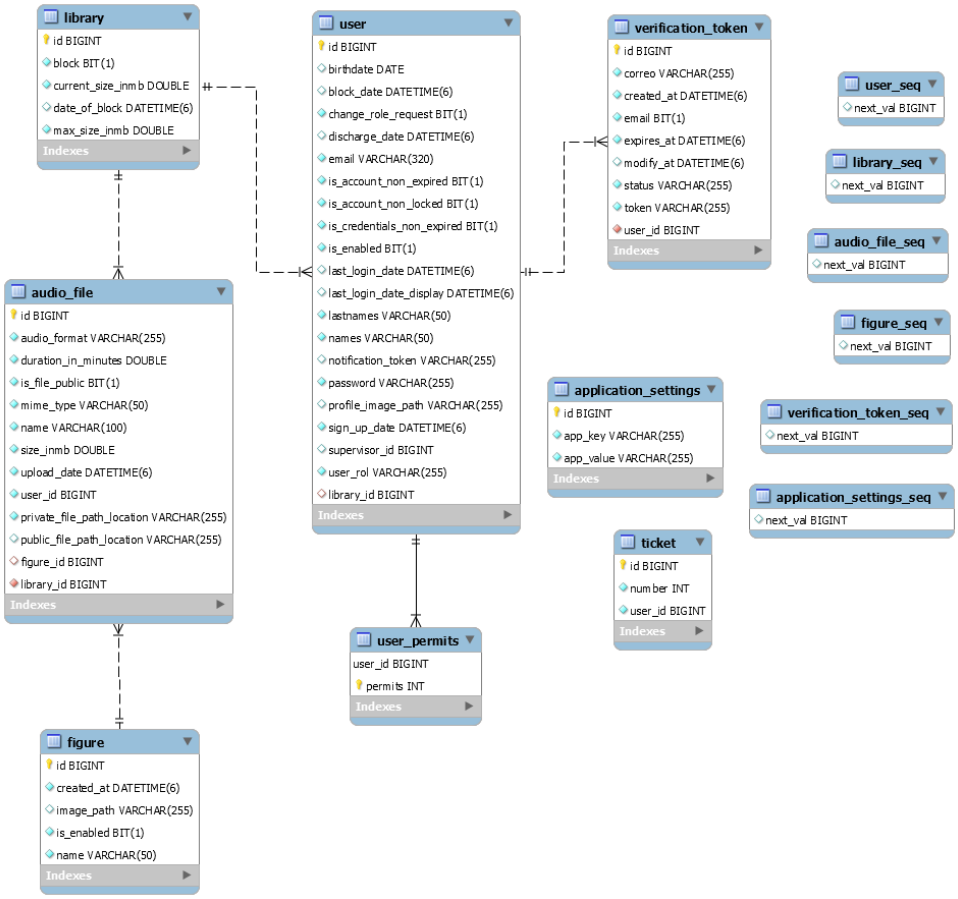


Figura 4. Diagrama EER de la BD del sistema de información de LIMBO.

Como se puede apreciar en la figura 4 en comparación con la figura 3, existen múltiples modificaciones como tablas adicionales, modificación de relaciones, entre otros. Esto se debe a que hubo múltiples cambios de diseño y a la propia interpretación que hace JPA de cómo deberían modelarse las relaciones en la BD para dar soporte al sistema. No se pretende entrar en detalle a explicar el diagrama ya que, explicar la conversión que realiza JPA no entra dentro del objetivo de la presente memoria, y muchas de las decisiones de diseño se explican más adelante, pero sí es conveniente aclarar algunos puntos para dar mayor sentido al EER representado por la figura 4:

- Se añaden tablas con el posfijo "_seq" (por ejemplo "user_seq") por cada tabla "principal" de la BD. Esto permite a la BD hacer una asignación secuencial e incremental de id empezando desde 1 a las entidades creadas, que facilita labores de desarrollo, mantenimiento y manejo de errores. De no configurarse se asigna un identificador único, pero aleatorio.
- Se añaden tres tablas adicionales:
    - verification_token: usado para el manejo de la comprobación de correo y reinicio de contraseña.
    - application_settings: usado para guardar en forma de clase las variables de configuración del sistema.
    - ticket: que controla el orden de uso de los usuarios de la interfaz principal de interacción con el sistema.

Es importante acotar que lógicamente se realizó una división en dos ámbitos diferentes, dentro de las carpetas del *Backend*:

- app: que contiene principalmente los archivos necesarios para el manejo de "funciones de negocio" específicas del sistema LIMBO. La figura 6 detalla el diagrama UML de las clases que componen dicha carpeta.
- backend: que contiene los archivos necesarios para las funciones que dan soporte a las funciones de negocio, más generales del sistema de información.

El diagrama UML de **todas** las clases que componen el *Backend* se adjunta al presente documento. No se incluye dentro de la presente memoria debido a su gran tamaño.

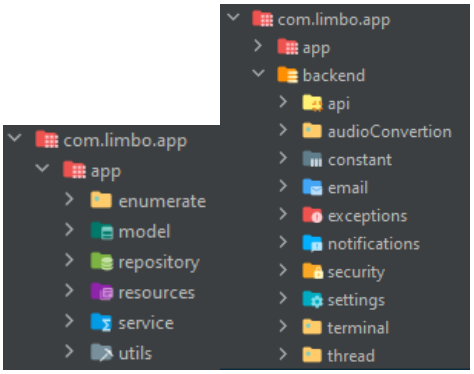


Figura 5. Estructura definitiva de ficheros bajo la carpeta "app" y "backend" en el *Backend* del sistema de información.

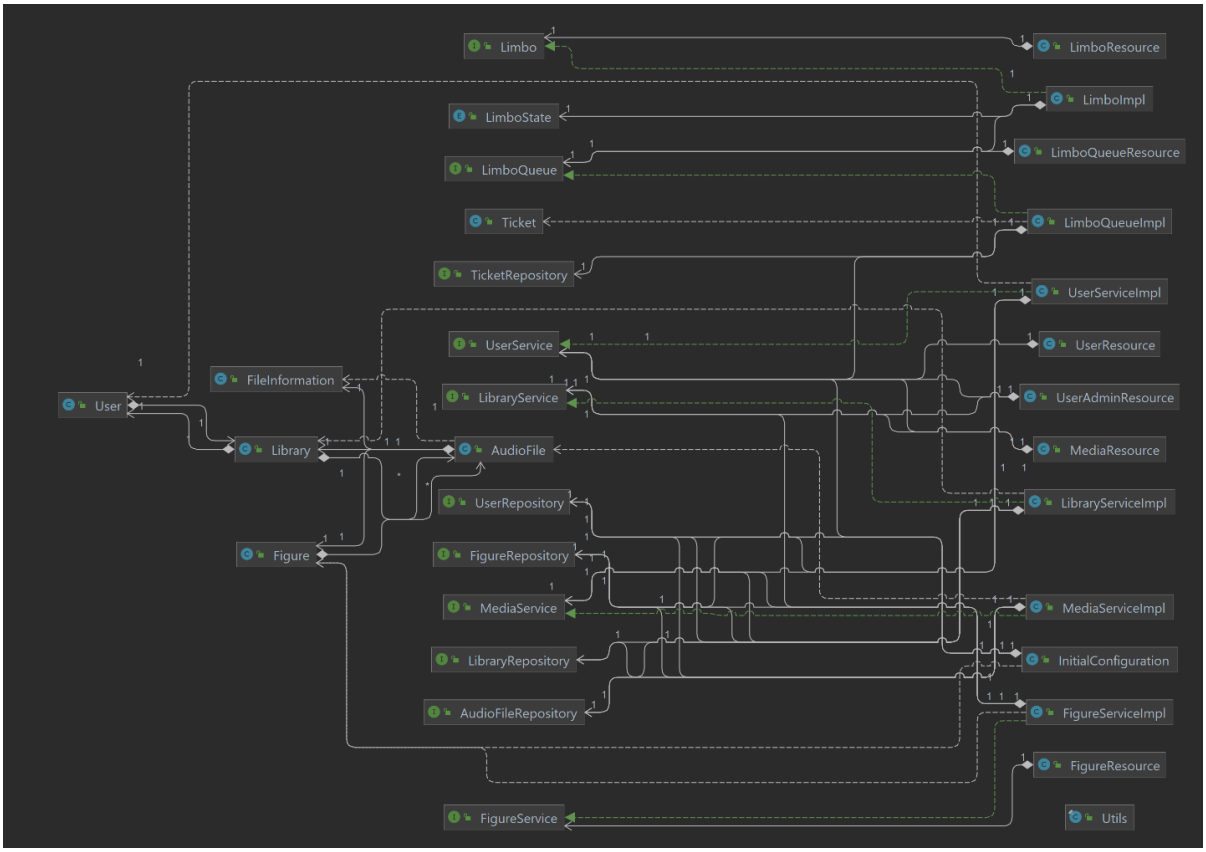


Figura 6. Diagrama de clases de los archivos que componen la carpeta "app" del *Backend*.

Una vez programadas las entidades, se realizaron una serie de pruebas manuales, para verificar su configuración y corregir errores antes de continuar con el siguiente Sprint. La figura 6 y la estructura de carpetas vista en la figura 5, ya refleja los cambios realizados a

consecuencia de la retroalimentación por parte del tutor al final del Sprint y adaptaciones realizadas más adelante en el desarrollo.

3.2. Sprint 2

Objetivos:

- Implementar las funciones de seguridad, manejo de cuenta y contraseña.
- Configuración de funciones de administración.

Inicialmente se tenían varias opciones para implementar las funciones básicas de seguridad, incluyendo el manejo de roles de usuario y permisos dentro de la aplicación. También se tenía presente la posibilidad de usar el servicio de identificación de la UMA iDUMA. Debido a que el manejo de funciones de seguridad y acceso es un punto amplio, se dividió en dos apartados principales: autenticación y autorización.

3.2.1. Autenticación

Para la autenticación se tenían inicialmente tres opciones: autenticación basada en formulario, usar JSON Web Tokens (JWT) o implementar desde el comienzo el servicio de autenticación de la UMA iDUMA. Esta última se rechazó en esta etapa temprana del desarrollo ya que dificultaría las labores de desarrollo y resolución de errores debido a que el manejo de la autenticación no estaría en manos del desarrollador, sino la propia UMA y sus administradores. Por ello quedaban dos opciones: autenticación por formulario y JWT.

Se consultó con el tutor en este punto cuál era la mejor opción y **se decidió implementar JWT**, debido a que en esta etapa se estaba valorando la futura implementación de una aplicación móvil y JWT es compatible con cualquier dispositivo y plataforma, a diferencia

de la autenticación por formulario que solo es compatible con aplicaciones web. Adicionalmente, JWT ofrece una gran flexibilidad de uso que permite propagar la identidad y privilegios a un "token" único, con un tiempo de uso determinado, pudiendo ofrecer privilegios temporales a usuarios para realizar acciones en el sistema. Este token se envía con cada solicitud del usuario al *Backend* y *Frontend*, el cual "está firmado por la clave del servidor, así que el cliente y el servidor son ambos capaces de verificar que el token es legítimo" (Wikipedia , s. f.).

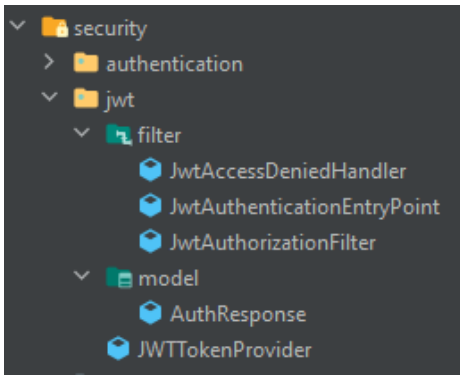


Figura 7. Carpetas y clases relacionadas con el manejo de JWT en el *Backend*.

Una vez el usuario inicia sesión y habiendo comprobado sus credenciales de acceso, se le otorga un JWT único que contiene la información necesaria para verificar la identidad del este durante un tiempo predeterminado fijado por la fecha de expiración del token. El JWT o token es enviado con cada petición y respuesta entre el *Backend* y *Frontend*, y mientras no esté expirado, permite al usuario dar autoría en las peticiones que realice en el sistema. Para el uso de JWT como herramienta de autenticación se desarrollaron todos los modelos, servicios y filtros necesarios descritos a continuación:

- La clase JWTTokenProvider es la que se encarga de generar el token JWT una vez comprobadas las credenciales del usuario. Dicho token es firmado con la clave

secreta almacenada en el *Backend* usando el algoritmo HMAC512. De igual forma define otros parámetros del token como organismo que lo emite, realiza funciones de comprobación del token entre otras.

- Las clases contenidas en la carpeta "filter", se encargan del manejo del token cuando se reciben peticiones, descompone las partes del token para comprobar sus datos y crean el contexto de seguridad que contiene al usuario y datos de la petición entrante.
- Por último, la clase AuthResponse, define la estructura y atributos del token, necesarios para poder realizar los manejos anteriormente mencionados.

En el *Frontend* se hizo uso de los guardias de Angular, que son filtros que se ejecutan cuando se accede a ciertas vistas del sistema. En concreto se creó un guardia llamado "AuthenticationGuard" que, en las vistas donde se requiere estar autenticado, hace una decodificación del JWT y verifica su expiración. El guardia es comprobado en cada vista donde se requiera que el usuario esté autenticado. Aproximaciones similares se usan para verificar que un usuario sea administrador para limitar el acceso a las vistas.

Para mayor detalle sobre la implementación se recomienda ver el código fuente del sistema de información y la descripción del estándar JWT disponible en internet. Otros parámetros de seguridad adicionales se describen a continuación:

- El tiempo de vida/uso del JWT antes de que el usuario tenga que volver a autenticarse en el sistema es de 24 horas, el cual también es modificable desde la configuración del sistema.
- Se obliga al uso de contraseñas de longitud mayor a 8 caracteres y mediante el *Frontend* que el patrón de esta contenga: al menos una letra mayúscula, al menos

una letra minúscula y al menos un carácter especial (!@#$%^&*_=+-). Este aspecto es mejorable ya que el *Backend* también debería realizar dicha comprobación de patrón.

- La contraseña se recibe en texto plano y se cifra al guardarse en BD y para comprobaciones de inicio de sesión usando el método proporcionado en Java BCryptPasswordEncoder con una fortaleza de 10. El método admite entre 4 y 31 de fortaleza, donde a mayor mejor, pero impacta el rendimiento. Este número es configurable, pero de cambiarse puede afectar contraseñas ya generadas, teniendo que los usuarios con la fortaleza anterior reiniciar sus contraseñas.

3.2.2. Autorización

Para la autorización se decide hacer uso de roles de usuario que pueden tener permisos asociados a los mismos. Debido a los requerimientos del sistema solo se discrimina por rol de usuario y no se hace uso de los permisos, sin embargo, estos están modelados e implementados, pero vacíos, para una posible configuración en el futuro. Se definieron tres roles: USER, ADMIN y SUPER_ADMIN.

Ya que no se esperan grandes cambios en este apartado, se decidió no implementar y persistir los roles en BD, sino que se modelaron como una clase enumerada **UserRol.** Sin embargo, en caso de ser necesario en un futuro la portabilidad de estos a una clase persistida en BD es sencilla. De igual forma y porque no se usan actualmente en el sistema, no se definen permisos que vayan a ser guardados en BD.

Se pretende que si un usuario es administrador (ADMIN o SUPER_ADMIN) este pueda realizar ciertas operaciones en el sistema tanto en el *Backend* como en el *Frontend*. En

varias posiciones del *Backend* se comprueba de forma constante dicha condición antes de ejecutar este tipo de funciones sólo disponibles para los administradores. De igual forma en el *Backend* y mediante un punto de conexión se verifica el estatus de administrador para acceder a ciertas vistas y funcionalidades. Otras comprobaciones y funciones relacionadas con la seguridad de la aplicación se detallan a continuación:

3.2.3. Comprobaciones y otras funciones de seguridad

- Se realiza una comprobación estática de inyección SQL en todos los campos de entrada del usuario en el sistema.
- El nombre de las carpetas de los usuarios guardados en el sistema de ficheros del servidor *Backend* se le hace una operación de hash, así que no hay forma directa de saber a qué carpeta corresponde cada usuario.
- Se realizan comprobaciones de nombre en los audios almacenados para evitar disrupciones en el sistema de archivos. Por ejemplo, nombres que contengan un "."en cualquier punto de la cadena, podrían afectar al sistema de archivos.
- Se contempla la implementación de conexión entre el *Backend* y el *Frontend* usando HTTPS, pero como depende de los certificados a usar en producción, no se implementa en el desarrollo del sistema de información.
- Se desarrollan las funciones de activación de cuenta, recuperación y cambio de contraseña mediante correo electrónico.
- Se utiliza un simulador de servidor SMTP de correo electrónico "MailDev" para interceptar todos los correos enviados por la aplicación. Antes de que el sistema se ponga en producción se debe cambiar la configuración y asignar un servidor real como Gmail o cualquier otro.

3.3. Sprint 3

Objetivo:

- Diseño de la interfaz gráfica.

Para el diseño de la interfaz gráfica se tomaron en cuenta una serie de requisitos que esta debía cumplir para garantizar la usabilidad del sistema de información. Entre dichos requisitos se destacan: atractivo visual, adaptabilidad a distintos tamaños de pantalla, en especial dispositivos móviles y capacidad de mejora o actualización en el futuro. Para agilizar el proceso se usó la herramienta Bootstrap Studio que incluye una serie de plantillas y funciones de edición mediante interfaz gráfica, reduciendo el tiempo de desarrollo.

Bootstrap Studio es una herramienta de pago, pero que ofrece licencias gratuitas a estudiantes. Su interfaz permite editar componentes HTML sin escribir líneas de código y las plantillas incluyen efectos y otros componentes "atractivos". Además, son adaptables a distintos tamaños de pantalla por lo cual se adapta perfectamente a las necesidades presentes. Se utilizaron dos plantillas principales: una para la página web principal del sistema y otra para el área de usuarios una vez estos hayan iniciado sesión. Las plantillas usadas fueron "Clean Sky" y "SB admin" expuestas en la figura 8. El resultado final se detalla en las siguientes figuras 9, 10 y las imágenes adicionales de la interfaz en el Apéndice de la memoria.

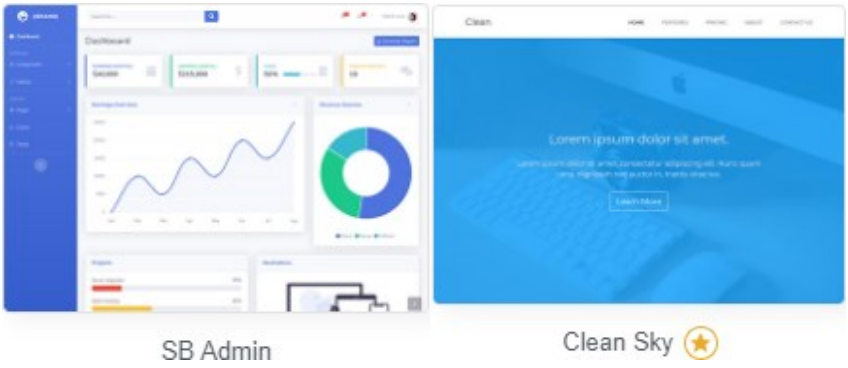


Figura 8. Plantillas de Bootstrap Studio utilizadas. (Bootstrap Studio, s. f.)

Figura 9. Vista de la página principal con formato de dispositivo móvil.

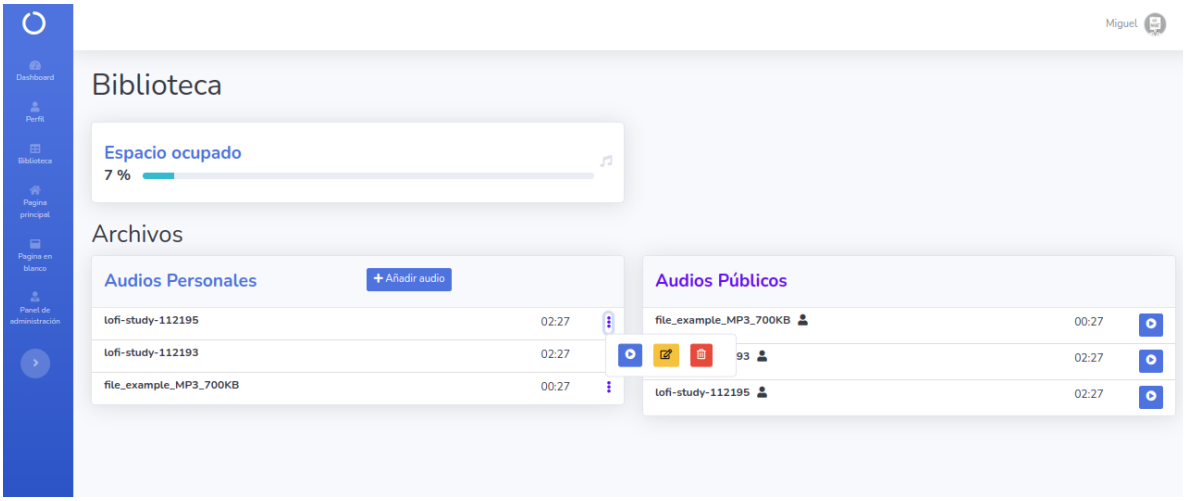


Figura 10. Vista de la página de la biblioteca del usuario en su área personal con formato de escritorio.

3.4. Sprint 4

Objetivos:

- Implementación de la interfaz gráfica en Angular.
- Conexión del *Backend* con el *Frontend.*
- Configuración de seguridad de conexión entre el *Backend* y el *Frontend.*

El Sprint 4 marca el inicio de la programación del *Frontend* de la aplicación. Una vez diseñada la interfaz se tenían que exportar los modelos de las páginas diseñadas en Bootstrap Studio a Angular. Este paso involucra generar todos los componentes necesarios para hacer que la interfaz funcione. De igual forma se tuvo que modificar algunos archivos generados por Bootstrap Studio para integrarlos correctamente en el *Framework.*

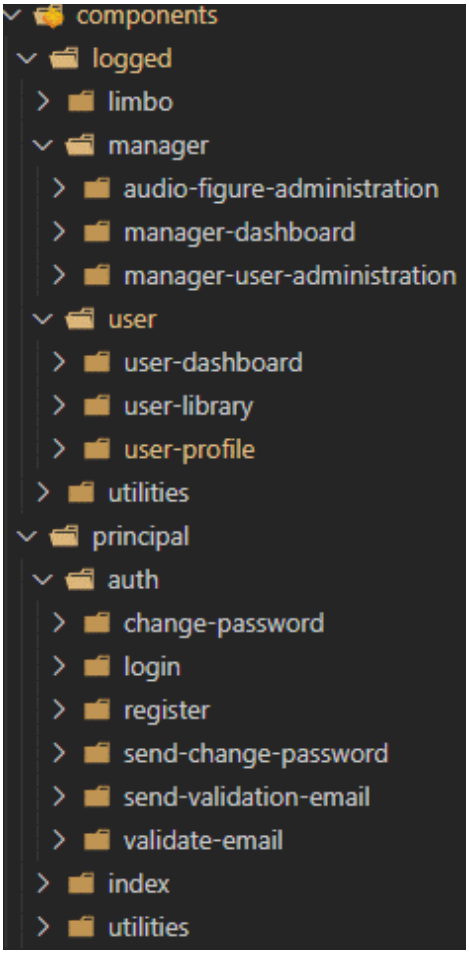


Figura 11. Estructura de carpetas de los componentes Angular del *Frontend*.

La filosofía y arquitectura de Angular varían ligeramente de la usada por Spring. En Angular la aplicación se divide en **componentes**, **modelos** y **servicios**. Los **modelos** al igual que Spring son los que, como su nombre indica, modelan las entidades que interactúan en la aplicación. Los **componentes** son cada una de las vistas de la aplicación o partes que la componen y a su vez cada una está integrada por un archivo HTML, un archivo Typescript y un archivo CSS. Los **servicios** son el equivalente en Spring a los recursos; estos realizan las peticiones al *Backend* a los puntos de conexión definidos y devuelven las respuestas.

Los componentes se dividen de forma lógica en dos carpetas: una para los componentes de la vista principal "principal" y otra para los componentes de las vistas del área de usuario "logged". A su vez estas se subdividen en función de su utilidad y el tipo de usuario que va a usar dicha vista. Angular permite la reutilización de componentes por lo cual elementos en comunes de las vistas, como menús de navegación y cabeceras, son también componentes inyectados en otros, favoreciendo la reutilización. Dichos componentes reutilizables se encuentran en las carpetas "utilities" en cada una de las dos carpetas principales del *Frontend*.

Por supuesto existen muchos más componentes lógicos y funcionales dentro de la aplicación como:

- Módulo de enrutamiento "app-rounting-module"
    - Define las rutas de las vistas de la aplicación, los componentes y guardias (introducidos anteriormente) asociados.
- Pipes o procesadores de flujo de información
- Enumerados para el manejo de atributos enumerados del sistema.
- *Assets* o Activos estadísticos del sistema
    - Que almacenan las fuentes de texto, archivos CSS, archivos js usados por las vistas e imágenes de las figuras estáticas definidas.
- Variables de entorno "enviroments"
    - Definen variables de configuración como dirección de la API donde se ejecuta el *Backend* (en desarrollo en http://localhost:8088/api) y dirección del *Frontend* (http://localhost4200).
- Paquete de librerías "package.json"
    - Donde se definen: el nombre de la aplicación *Frontend*, versión de aplicación, dependencias de librerías y versión de librerías.
- Y muchos más componentes y clases propias de aplicaciones Angular.

El diagrama UML de **todas** las clases que componen el *Frontend* se adjunta en los archivos adicionales de la presente memoria. No se incluye dentro de la presente memoria debido a su gran tamaño.

Para la conexión entre ambas "partes" de la aplicación se usó la documentación oficial de Spring, la cual guía con facilidad el proceso. Para ello se tuvo que definir la configuración

de control de acceso HTTP CORS, principalmente en el *Backend* quien sería quien recibiría las peticiones provenientes del *Frontend*. El Intercambio de Recursos de Origen Cruzado (CORS en inglés) es un mecanismo que utiliza cabeceras HTTP adicionales para permitir que un agente de usuario obtenga permiso para acceder a recursos seleccionados desde un servidor, en un origen distinto (dominio) al que pertenece (Mozilla Foundation, s. f.). Este es el caso del sistema de información, donde debido al desacoplamiento entre *Backend* y *Frontend* el servidor de Spring se encuentra funcionando en una dirección y puerto diferentes al servidor Angular.

Una vez realizada la configuración CORS tanto en el *Backend* como en el *Frontend* y comprobado su conexión, se procedió a iniciar el siguiente Sprint.

3.5. Sprint 5

Objetivos:

- Implementar prototipo de manejo del audio del *Backend*.
- Manejo de conversión de audio.
- Presentación del prototipo al equipo encargado de los efectos sonoros.

3.5.1. Prototipo de manejo de audio.

En este punto del proyecto ya se estaban formando los equipos adicionales que iban a participar y desarrollar otros componentes del sistema, en concreto el equipo encargado de desarrollar los efectos sonoros conformado por estudiantes y personal de la Escuela de Telecomunicaciones de la UMA, en adelante "equipo de audios". El Equipo de audios precisaba un prototipo funcional que fuese capaz de reproducir audio usando la interfaz de audio de Java, Java API sound y saber las limitantes de esta. El prototipo debía ser

capaz de elegir un archivo de audio en cualquier formato (mp3, mp4, WAV, etc.) y reproducirlo por la tarjeta de audio elegida para el proyecto: GIGAPORT eX, en adelante tarjeta de audio. Para ello se programó una aplicación Java que respondiera a los requisitos del equipo de audios.

Figura 12. Tarjeta de audio empleada en el sistema LIMBO GIGAPORT eX. (ESI, s/f)

Hasta el momento se había desarrollado el sistema usando las herramientas de desarrollo en un entorno Windows, sin embargo y a pesar de la alta compatibilidad de la tarjeta de audio en dicho OS, muchas de las pruebas realizadas con la aplicación Java tenían problemas, debido a que Windows modifica las instrucciones de audio que enviaba la aplicación a la tarjeta de Audio. Por ello se tuvo que migrar el desarrollo a un entorno Linux, que finalmente iba a ser el OS del servidor que usaría la tarjeta de audio, por lo cual no representó un gran cambio, pero sí llevó un tiempo portar el código de un OS a otro al igual que la mayoría de las herramientas software usadas hasta el momento.

Una vez en un entorno Linux, se continuó desarrollando y probando de forma manual el prototipo a entregar al equipo de audios encontrando en el proceso una gran limitante en la API de sonido de Java: de forma nativa sólo es capaz de reproducir audios en formato WAV. El formato WAV o *Waveform audio file format*, es un formato de audio digital que no aplica ninguna compresión al flujo de bits y almacena las grabaciones de audio con diferentes tasas de muestreo y tasas de bits (Aspose Pty Ltd , s. f.). La obligación de usar un **único** tipo de archivo de audio trae consigo varias limitantes y más si se trata de un archivo del tipo WAV:

- El formato WAV al ser un formato sin compresión de audio es hasta 10 veces más grande en tamaño que un MP3 (Red, s. f.). Si almacenamos todos los archivos de audio en el servidor en este formato podríamos quedarnos sin espacio en un punto dado.
- El formato WAV no es compatible con la mayoría de los reproductores web de sonido. Además, debido a su tamaño ralentiza la transmisión o descarga del mismo.
- El formato de audio que el usuario sube a la aplicación probablemente no sea del tipo WAV, ya que no es un tipo de audio muy común, por lo cual para su uso habría que transformarlo.

Surge entonces la pregunta de cómo transformar decenas de formatos diferentes de audio a formato WAV para ser usados en la aplicación. Igualmente, importante es la pregunta sobre cómo almacenar los archivos de audio en el servidor para evitar quedarnos sin espacio. Para resolver ambas cuestiones se usó la librería FFmpeg quien se encargaría de realizar las operaciones de conversión necesarias. FFmpeg es una solución completa y multiplataforma para grabar, convertir y transmitir audio y video (FFmpeg, s. f.), ampliamente mantenida por la comunidad y usada por reproductores populares como VLC y navegadores como Google Chrome.

Invocando una consola de comando del OS Linux dentro de la aplicación Java y mediante los parámetros adecuados podemos transformar cualquier tipo de audio entrante a WAV, y más importante aún, al usar esta librería se da la posibilidad de emplear cualquiera de sus utilidades disponibles al equipo de audios. Para solucionar el problema de almacenamiento del sistema se decidió guardar todos los archivos en formato MP3 y realizar la conversión a WAV cuando fuese necesario, dado que esta consume muy poco

tiempo y recursos del sistema. Con ello se solucionaron todos los problemas encontrados con la reproducción de audios del prototipo.

Un ejemplo del comando ejecutado para la conversión de audios es el siguiente:

ffmpeg -y -i nombre.mp3 nombre.wav

- ffmpeg es la ubicación, ruta a la carpeta dentro del sistema Linux, del binario de la librería ffmpeg almacenada en el sistema.
- Las opciones "-y -i" indican que se sobrescribirá el archivo de salida sin preguntar y que se indica la ruta del archivo respectivamente.
- Se mantiene el nombre original en ambos casos.
- El formato de entrada en este caso es mp3 y el de salida WAV, pero podría cambiarse por cualquier par de formatos soportados por la librería.

Al final del sprint se desarrolló una aplicación Java que, mediante línea de comandos, permitía elegir un archivo de audio de una carpeta específica del sistema, realizar la conversión a formato WAV en caso de ser necesario y reproducirlo con funciones de pausa, reanudar y cambio de archivo. El diagrama UML resultante del programa se detalla en la figura 13. Dicho prototipo fue entregado, presentado y explicado al equipo de audios en una reunión mantenida en la UMA con presencia del tutor. Esta reunión marcó el final del presente Sprint para dar inicio al próximo, sin embargo, y durante todo el desarrollo se estuvo asesorando al equipo de audios con las consultas y dudas técnicas relacionadas con la API de sonido de Java, la librería FFmpeg y demás apartados técnicos del sistema de información.

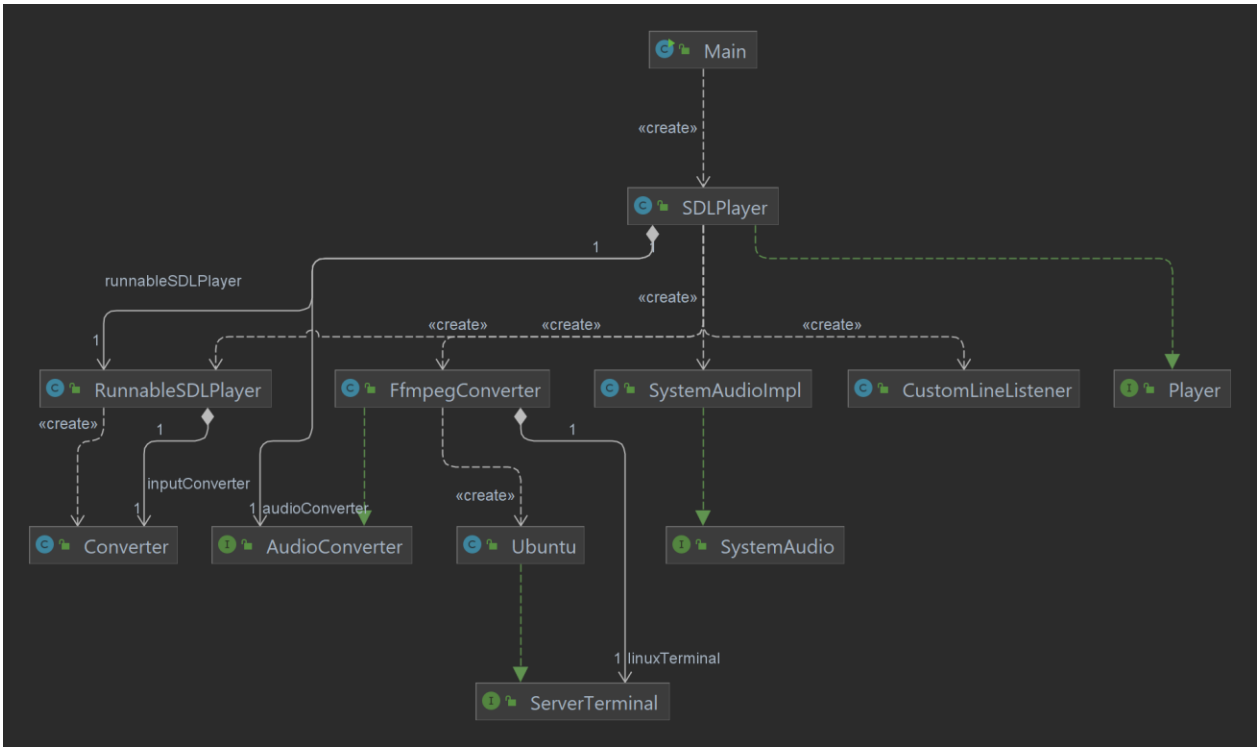


Figura 13. Diagrama de clases del prototipo de reproducción y transformación de audios.

3.5.2. Integración del prototipo de audio en el sistema de información.

Luego de entregar el prototipo al equipo de audios, se adaptó lo aprendido para implementar las funcionalidades de conversión necesarias para el sistema de información. Para ello se implementó un interfaz que define una función de conversión de audio. Y una clase que implementa dicha interfaz llamada FfmpegConverter que implementa dicha función, ubicando primero el binario de la librería ffmpeg y después ejecutando el comando anteriormente descrito con los parámetros necesarios.

En este Sprint se definió de igual forma como se iba a manejar los archivos multimedia del sistema de información en el sistema tanto para los archivos de audios como la imagen de perfil de los usuarios. Debido a la limitante de tener que ejecutar comandos de transformación de audios donde estos estén ubicados al momento de la conversión en el

sistema de ficheros del *Backend*, se decidió guardar los datos de los audios en el sistema de ficheros del sistema, al igual que la imagen de perfil de los usuarios. Esto conlleva ventajas y desventajas, pero generalmente las BD no suelen almacenar grandes archivos multimedia, pero en cambio sí metada relacionada con los mismos, incluyendo su ubicación (Singh, 2019).

Para los audios se decidió crear una clase "AudioFile" que almacenaría información relacionado con el audio y su ubicación en el sistema de ficheros del servidor. Nótese que los datos del audio no se guardan en BD, pero en cambio sí los metadatos relacionados con ellos, incluyendo la ubicación de este en el sistema de ficheros. Esta ubicación no es enviada al *Frontend* cuando se pide información del audio, para evitar revelar la estructura de ficheros del *Backend*.

Para las imágenes se usa una aproximación similar, solo que esta vez no se guarda información sobre la imagen, pero si su ubicación dentro del sistema de ficheros como atributo en la clase del usuario "User". De igual forma no se envía esta información al *Frontend* por motivos de seguridad. La misma aproximación se usó para las imágenes de las figuras explicadas más adelante.

La estructura del sistema de ficheros del servidor *Backend*, se muestra en más detalle en la figura 14. Principalmente tenemos dos carpetas: una almacena las imágenes de las figuras del sistema de información "figure" y otra que almacena los audios e imagen de perfil de los usuarios "users". Nótese como el nombre de la carpeta de cada usuario es producto de la aplicación de una función hash, por lo cual, y como se mencionó en los

apartados de seguridad, no hay forma directa de saber a qué carpeta pertenece cada usuario.

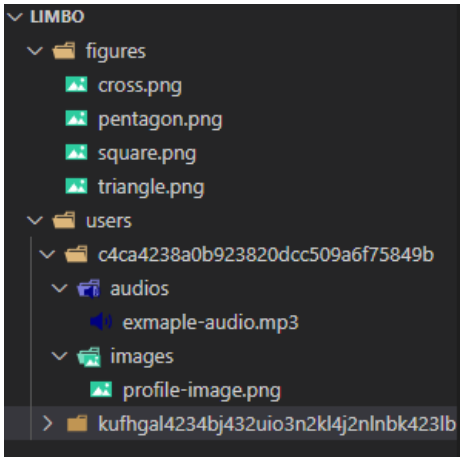


Figura 14. Estructura del sistema de ficheros de las carpetas del sistema de información.

Cuando se realizan peticiones al *Backend* solicitando archivos de audios o imágenes, primero se solicita el objeto que lo representa y luego los datos del archivo relacionados con este. En el caso de los audios se pide la clase "AudioFile" y luego con el identificador de dicho objeto, los datos de audio relacionados. En el caso de las imágenes de Perfil y Figuras, primero la clase "User" o "Figure" respectivamente, y luego las imágenes asociadas a los mismos.

3.6. Sprint 6

Objetivos:

- Desarrollo y pruebas de las funcionalidades básicas de la aplicación.
- Integrar el manejo de audios y figuras dentro de la aplicación.

El sprint inicia con la incorporación de nuevos miembros y equipos al proyecto, entre ellos un estudiante de informática encargado de la interfaz de interacción del sistema y alumnos de bellas artes que diseñarían distintos componentes visuales, colorama y demás a apartados gráficos de la aplicación. Se inicia el sprint con una reunión involucrando a la mayoría de los participantes del proyecto, donde se definen algunas prioridades y planes a seguir. Una vez extraídos los cambios a realizar y sabiendo los objetivos actualizados del proyecto se procede a la continuación del sistema de información.

Hasta la fecha las pruebas manuales que se habían hecho del código tomaban tiempo ya que no había una interfaz gráfica definida con la cual interactuar con la aplicación. Una vez finalizado el Sprint 4 ya se tenía dicha interfaz y se pudo agilizar tanto el desarrollo como la prueba del código. Se plantearon desarrollar pruebas unitarias y de integración al principio del proyecto, pero duplicaban el tiempo de desarrollo, y una vez implementadas las funciones de seguridad, se hacía cada vez más complejo y extenso de realizar.

Teniendo una interfaz definida con los componentes necesarios se procedió a realizar las funcionalidades restantes del sistema de información. En esta etapa ya algunas habían sido adelantadas y probadas de forma manual, mediante consola o con la herramienta de solicitudes API REST "Advanced REST client". Se divide el desarrollo de funcionalidades en tres apartados separados:

Funcionalidades Básicas

- Actualizar información de perfil.
- Subir audios a la plataforma y modificar el nombre de estos de forma opcional.
- Reproducción de audios subidos a la plataforma mediante reproductor web.

- Implementación de notificaciones para notificar al usuario del resultado de las operaciones que realiza en el sistema.
- Función de cambio de correo y reactivación de este.

Funcionalidades de Administración

- Dar de alta y baja, habilitar y bloquear usuarios de forma manual en la aplicación.
- Subir audios públicos al sistema
- Poder habilitar y deshabilitar la subida de audios a usuarios en la aplicación.
- Cambiar datos de usuarios de la aplicación en caso de ser necesario.
- Asociar, eliminar y modificar los audios de las figuras definidas en el sistema.

Corrección de errores y mejoras

- Corrección de errores a medida que se realizaban funciones del sistema.
- Refactorización de código para favorecer la reutilización del código.
- Refactorización de nombres de variables y funciones para mejorar la legibilidad.
- Implementación de notificaciones WEB mediante las herramientas de Firebase.
- Soporte en el desarrollo de la interfaz principal del sistema LIMBO.

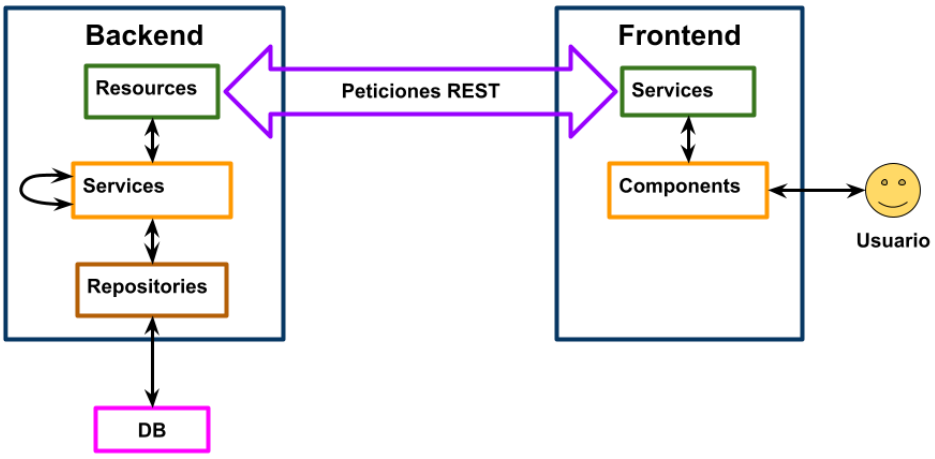


Figura 15. Flujo de información del sistema de información a través de los componentes

Debido a que el *Backend* y el *Frontend* están desacoplados, quiere decir que se ejecutan y pueden ubicarse en servidores separados, la interacción entre ellas se realiza **única y exclusivamente** a través de puntos de conexión. Dichos puntos de conexión están definidos en el *Backend* ya que es quien ofrece la funcionalidad del sistema de información al *Frontend*, el cual solo se limita principalmente a operaciones de interacción con el usuario. El flujo de información está detallado en la figura 15, donde se refleja como el *Frontend* realiza el envío de solicitudes al *Backend* y este responde después de haber procesado la información. Teniendo una base asentada de las funcionalidades en este punto se terminaron de definir los puntos de conexión o *endpoints* (por sus siglas en inglés) del sistema de información, los cuales se describen y clasifican a continuación:

3.6.1. Descripción general del formato y uso de los puntos de conexión

Los puntos de conexión están compuestos por los siguientes apartados:

<URL> / api / <Área> / <Funcionalidad>

- URL: es la dirección donde está alojado el servidor. En desarrollo esta dirección es http://localhost:8080 . Cuando la aplicación se ponga a disposición de los usuarios dicha dirección será la del servidor.
- api: contiene únicamente la palabra api, aunque se modela el uso de versiones para cuando el sistema de información esté en producción. En este caso estaría compuesto de la siguiente forma /api/v# siendo # el número de la versión.
- Área: divide en grupos el propósito o usuario que ejecuta usa dicho punto de conexión.
- Funcionalidad: indica la funcionalidad a ejecutar.

Por ejemplo, el punto de conexión para el inicio de sesión es el siguiente:

http://localhost:8080/api/auth/login

Cada punto de conexión se traduce en una función de distintas clases en *Frontend* *Backend*. En el *Frontend* bajo la carpeta **services** y en el *Backend* en la carpeta **resources**. Cada clase contiene múltiples métodos, uno por cada punto de conexión y como se menciona anteriormente son los encargados de pasar a las clases que los invocan la solicitud o respuesta de las peticiones.

3.6.2. Puntos de acceso

Puntos de acceso sin autorización o públicos

Dichos puntos de acceso no requieren de un JWT que autorice al usuario de realizar dichas peticiones. Por lo tanto, cualquier usuario puede mediante la interfaz o herramientas como Advanced REST client, hacer consultas a los mismos. Este es el comportamiento deseado ya que cuando un usuario que va a crear un usuario, iniciar sesión o recuperar su contraseña no se encuentra autenticado en el momento.

Los puntos de conexión públicos son los siguientes

- localhost:8080/api/auth/register
    - Para realizar el registro de usuario.
- .../api/auth/login
    - Para el inicio de sesión.
- .../api/auth/logout
    - No implementado, pero modelado por si se necesita hacer algún tipo de operación futura cuando el usuario cierra sesión en el sistema, Debido a que el cierre de sesión se realiza sólo en el *Frontend*, este endpoint no se utiliza.
- .../api/auth/send-validation-email/{userEmail}

    - Envía al correo electrónico del usuario (userEmail) el correo para validar su cuenta.
- .../api/auth/send-change-password/{userEmail}
    - Envía al email del usuario el correo para el cambio de contraseña.
- .../api/auth/check-token/{token}
    - Punto de conexión que se consulta cuando un usuario pulsa el enlace de validar email o cambiar contraseña.
    - Debe contener en el enlace el token a verificar. No confundir con JWT, este es un token usado como método de verificación de la petición adicional al JWT.
- .../api/auth/validate-email/{token}
    - Enlace para validar el email del usuario en el sistema.
    - Contiene un token para validar la operación como parámetro en el enlace (mismo caso que el anterior endpoint).
- .../api/auth/ change-password
    - Para cambiar la contraseña de un usuario.

Puntos de acceso con autorización o privados

Para que el sistema pueda acceder a estos puntos de conexión se debe otorgar el JWT en cada petición de estos. Esto sirve para autenticar al usuario e identificar el rol que tiene en el sistema (USER, ADMIN o SUPER_ADMIN). Algunos puntos de conexión solo están permitidos para ciertos roles dentro del sistema. La división de puntos de conexión es la siguiente:

Usuario

Puntos de acceso asociados a la información del usuario.

- localhost:8080/api/user/check-admin-status
    - Petición que verifica si un usuario es administrador.
- .../api/user/get-latest-info
    - Devuelve la información actualizada del usuario actual.
    - Mediante el JWT se extrae el usuario que hizo la petición y se devuelve únicamente su información más actualizada.
- .../api/user/update/info
    - Para actualizar la información del usuario
- .../api/user/delete-user
    - Marca el usuario actual para eliminar su cuenta.
    - Esta endpoint está modelado y configurado, pero no es accesible desde la interfaz debido a la futura implementación de iDUMA.
- .../api/user/profile-image
    - Devuelve la imagen de perfil de usuario actual.
- .../api/user/update/profile-image
    - Actualiza la imagen de perfil del usuario.
- .../api/user/delete/profile-image
    - Elimina la imagen de perfil del usuario.

Administradores

Puntos de acceso usados por los administradores para realizar operaciones solo disponibles para ellos.

- localhost:8080/api/admin/user-profile-image/{userID}

    - Devuelve la imagen de perfil de un usuario especificado
- .../api/admin/add-user
    - Añade de forma manual un usuario al sistema
- .../api/admin/edit-user
    - Edita de información de un usuario.
- .../api/admin/delete-user/{userEmail}
    - Elimina al usuario con el correo electrónico especificado (userEmail).
- .../api/admin/find/{email}
    - Devuelve la información del usuario con el correo especificado.
- .../api/admin/find-all-users
    - Devuelve todos los usuarios registrados en el sistema de información.
- .../api/admin/media/remaining-space-in-library-of-user/{userEmail}
    - Devuelve el espacio restante de la librería del usuario con el correo especificado.
- .../api/admin/media/update-audio
    - Actualiza la información de un audio.

Librería de audios de un usuario

Puntos de acceso relacionados con la librería de un usuario.

- localhost:8080/api/media/add-audio
    - Añade un archivo de audio a la librería del usuario.
- .../api/media/add-public-audio
    - Añade un audio marcado como público.
- .../api/media/user-audios
    - Devuelve los audios de un usuario

- .../api/media/delete-audio/{audioId}
    - Elimina el audio del usuario con el id especificado.
- .../api/media/update-audio
    - Actualiza la información de un archivo de audio.
- .../api/media/audio-file/{audioId}
    - Devuelve los datos de audio del archivo de audio especificado.
    - Esta petición devuelve los datos del archivo de audio, no su representación como objeto dentro del sistema.
- .../api/media/library-usage-percentage
    - Devuelve el porcentaje de espacio restante de la librería de un usuario.
- .../api/media/library-remain-space-in-mb
    - Devuelve el espacio restante en MB de la librería de un usuario.
- .../api/media/are-audios-enabled
    - Petición para determinar si los usuarios no administradores pueden subir audios al sistema.
- .../api/media/public-audios
    - Devuelve todos los audios marcados como públicos del sistema.
- .../api/media/audio-owner-info/{audioId}
    - Devuelve la información del usuario que le pertenece el audio.
    - Solo disponible para los administradores.

Notificación WEB.

Apartado relacionado con las notificaciones WEB de Firebase.

- localhost:8080/api/notify/firebase/save-notification-token
    - Guarda el token de identificación de notificaciones web del usuario.

    - No confundir con el JWT. Este token lo genera la API externa de Firebase para poder enviar notificaciones WEB al usuario.
- .../api/notify/firebase/notification
    - Envía una notificación web a un usuario específico.
    - Hace uso del token enviado en el cuerpo de la petición para realizar dicho envío.
- .../api/notify/firebase/topic/notification
    - Envía una notificación web a un tópico.
    - No utilizada actualmente en el sistema, pero está modelada e implementada para un uso futuro.
- .../api/notify/firebase/subscription
    - Suscribe al usuario actual a un tópico de notificaciones web.
    - No utilizada actualmente en el sistema, pero está modelada e implementada para un uso futuro.

Cola de espera para usar la interfaz principal.

Punto relacionado con los puntos de conexión asociados a la espera, estado y acceso a la interfaz principal.

- localhost:8080/api/limbo/queue/add-ticket
    - Añade al usuario a la cola de espera para usar la interfaz.
    - En este punto se le asigna un ticket de espera.
- .../api/limbo/queue/ticket-number
    - Devuelve la información del ticket de espera de un usuario.
- .../api/limbo/queue/current-user-name
    - Nombre del usuario que actualmente está usando la estructura.

- .../api/limbo/queue/estimated-wait-time
    - Tiempo de espera aproximado para usar la interfaz principal.
- .../api/limbo/queue/check-user-turn
    - Verifica si es el turno del usuario de usar la plataforma.
- .../api/limbo/queue/number-of-users-in-queue
    - Devuelve el número de usuarios esperando para usar la interfaz.
- .../api/limbo/system/state
    - Devuelve el estado actual del sistema: *Free*, *In use*, *Turn Off* o *In maintenance*.

Manejo de Figuras

Puntos de conexión asociados al manejo y configuración de las figuras de interacción del sistema usadas en la interfaz principal. Estos endpoints solo son accesibles por usuarios administradores.

- localhost:8080/api/figure/get-all
    - Devuelve todas las figuras configuradas en el sistema.

Debido al manejo estático actual de las figuras están definidas y configuradas desde el momento que inicia el sistema y no se pueden modificar dinámicamente. Sin embargo, se prevé un futuro uso de figuras dinámicas, por lo cual se implementaron los siguientes endpoints, a pesar de que no se usan actualmente, para dar cobertura a requisitos futuros.

- ❖ localhost:8080/api/figure/add
    - Añade una figura al sistema.
- ❖ .../api/figure/update
    - Actualiza la información de una figura en el sistema.
- ❖ .../api/figure/delete

    - Eliminar una figura del sistema.
- .../api/figure/image
    - Devuelve la imagen asociada a una figura.
- .../api/figure/delete-image
    - Elimina la imagen de una figura en el sistema.
- .../api/figure/iscomplete
    - Determina si una figura está apta para el uso en el sistema: tiene imagen asociada, información completa y se encuentra habilitada.

Debido a la gran cantidad de trabajo la duración del presente Sprint se duplicó y pasó de un tiempo estimado de 4 semanas a 8, con una semana adicional de reuniones y correcciones para hacer frente a los cambios surgidos en este periodo. Habiendo finalizado esta etapa quedaba únicamente el manejo del sistema de espera para el acceso a la interfaz principal y labores de documentación y manual de puesta en marcha del sistema para los administradores. La implementación de cambios de requisitos en este momento ya era mucho más complicada y los pocos que hubo, se analizaron detenidamente para evitar prolongar el desarrollo del sistema.

3.7. Sprint 7

Objetivo:

- Manejo del sistema de cola espera de usuarios para el uso de la interfaz

Debido al carácter multiusuario del sistema LIMBO, múltiples usuarios podrían ingresar al mismo momento para usar la interfaz principal e interactuar con los audios y efectos. Para ello es necesario controlar el acceso a la interfaz, dado que solo un usuario puede

estar usando la aplicación en un momento dado. Se plantea un sistema de espera por tickets gestionado por eventos de Spring.

Spring permite manejar de forma asíncrona eventos personalizados para publicación y consumo de mensajes (Paraschiv, 2021). El manejo de eventos define tres componentes fundamentales: los mensajes, los publicadores (*publisher*) y los oyentes (*listeners*). Los publicadores escriben y envían mensajes y los oyentes los reciben. Es una filosofía similar al de productor consumidor usando el paso de mensajes. En este caso el publicador son los usuarios que solicitan usar la aplicación, los mensajes son los tickets para acceder a la interfaz y el oyente sería el sistema que procesaría de forma secuencial las solicitudes de uso.

Una vez los usuarios hayan iniciado sesión en la aplicación, la secuencia de uso planteada es la siguiente:

1. Desde la página del "*Dashboard*" y presionando un botón los usuarios solicitan hacer uso de la interfaz principal.
2. a. Si no hay usuarios usando la interfaz principal actualmente este es redirigido directamente a ella.

2. b. Si hay un usuario usando la interfaz o más usuarios esperando este es colocado de forma secuencia en una cola de espera.

3. Una vez sea su turno el usuario recibe una notificación indicando que puede usar la aplicación. A partir de este momento tiene un tiempo configurable para acceder a la interfaz. De no hacerlo su solicitud es eliminada y se procede con el siguiente usuario.

Siguiendo la guía oficial de Spring relacionada con los eventos se implementó la funcionalidad. Seguidamente se realizaron una serie de pruebas manuales y corrección de errores para verificar el funcionamiento. Habiendo finalizado se dio por terminado el desarrollo del sistema de información y el presente Sprint, sin embargo, se continuaron realizando mejoras y corrección de errores a medida que se iba integrando con otras partes del sistema LIMBO.

Una de las últimas mejoras implementadas fue la implementación de una clase que gestiona la configuración inicial de la aplicación (RF-20 y RF-21). Dicha clase se asegura que tanto las figuras como los usuarios administrativos iniciales estén creados y en caso contrario los crea. Durante dicho proceso de creación también se asegura, en el caso de las figuras, que se tengan sus imágenes almacenadas en la carpeta correcta y con el nombre correspondiente. Esta funcionalidad es básica para el inicio de la puesta en producción del sistema de información.

3.8. Sprint 8

Objetivo:

- Realizar documentación y finalizar redacción de la memoria.

Durante todo el desarrollo del sistema se iba documentado el código generado, pero esto no se hizo de forma sistemática u ordenada, en gran parte debido a que cambios en los requisitos podrían reflejarse en cambios de código y por ende en su documentación. Para no desperdiciar tiempo documentando y seguidamente refactorizando no solo código, sino documentación de este, se decidió esperar hasta la finalización del desarrollo del sistema para empezar esta labor.

Se inicia el Sprint con la documentación de gran parte del código, generado. A pesar de que existen herramientas que determinan la cobertura de documentación del código, esta se descartó por restricciones de tiempo. La redacción de la memoria fue desarrollándose de forma paralela durante gran parte de los Sprint. A medida que se iba redactando la memoria el tutor fue sugiriendo cambios mientras se añadía, modificaba y eliminaba información en función de los cambios de requisitos y necesidades de información.

3.9. Sprint 9

Objetivos:

- Revisión de la documentación.
- Realizar presentación de la memoria.

El Sprint 9 marca el final del trabajo de fin de grado, realizando una revisión integral del sistema de información, en especial la documentación. Se incluye la realización de la presentación de la memoria, para reflejar el tiempo invertido en ello y que, nuevamente, esta puede servir para la comprensión general del sistema a administradores y futuros desarrolladores que usen o mantengan el código. Se comprende también la celebración de reuniones con el tutor para explicar los detalles del sistema final y organización de aspectos futuros como contacto en caso de dudas del código, fuera del marco del TFG.

4. Conclusiones y Líneas Futuras

4.1. Conclusiones

A lo largo del desarrollo del sistema de información se superaron muchos retos y dificultades técnicas y organizativas que derivaron en un tiempo de desarrollo más largo del inicialmente estimado. El proyecto LIMBO y su constante redefinición e interacción multimedia, representaron un gran desafío de diseño y programación, donde se tuvieron que utilizar múltiples tecnologías y herramientas, similares a los usados en proyectos de mayor escala en el campo profesional. Gracias al enfoque sistemático e iterativo con constante retroalimentación, fue que se pudo llegar a la consecución de los requisitos definidos y dejar el sistema preparado para los futuros cambios.

Desde el primer momento se buscó darle un enfoque de desarrollo lo más cercano al empleado en proyectos profesionales y académicos de mayor tamaño, alejándose del desarrollo lineal y de menor duración, habitual en proyectos dentro de las asignaturas de la universidad debido a las restricciones de tiempo y número de estudiantes. En dicho proceso se pusieron de manifiesto prácticamente todos los conocimientos adquiridos durante la carrera en materia de diseño, planificación y desarrollo de software; pasando por todos los niveles de abstracción lógica de una aplicación: desde la Base de datos, hasta la interfaz final del usuario. Fue también necesario aprender e implementar nuevas tecnologías y herramientas necesarias para las funcionalidades claves del sistema, que no se tenían en cuenta, pero el proyecto las demandaba.

Como resultado se obtuvo un sistema más estructurado, escalable, desacoplado, reutilizable, documentado y mejorable que el inicialmente estimado. Sistemas *Full stack* como el desarrollado son una gran oportunidad para poner en práctica los conocimientos aprendidos durante la carrera de ingeniería informática y nos acercan más a la herramientas y procesos usados en el ámbito profesional al cual ya los alumnos se han aproximado o están próximos a incorporarse. Sin lugar a duda ha sido una experiencia estresante, pero infinitamente más gratificante, y que me ha aportado una serie de experiencias y competencias que me ayudarán a ser un mejor profesional.

4.2. Líneas Futuras

Debido a la escala del sistema hay muchos aspectos mejorables del funcionamiento de este, no solo a nivel de código, sino también a nivel de documentación y reutilización de código. De igual forma y debido a su arquitectura, es ampliamente reutilizable para proyectos futuros y, sin dejar el ámbito del proyecto LIMBO, es un sistema que puede adaptarse más rápidamente a nuevas funcionalidades o requisitos que el proyecto demande. A continuación, listo una serie de posibles mejoras y usos del sistema desarrollado en este TFG y que pueden ser parte de trabajos futuros.

- Implementación de pruebas unitarias y de integración: un buen proyecto futuro sería realizar todas las pruebas unitarias y de integración necesarias, tanto en el *Backend* como en el *Frontend*, para tener la cobertura de pruebas deseadas de las funciones del sistema y medir la misma a medida que se van realizando.
- Refactorización del código del *Frontend*: el código del *Frontend* es mejorable y se puede reducir la repetición de código en favor de la reutilización de código y hacer un uso más estructurado y ordenado de los temas y archivos de estilo CSS.

- Mejorar la función de reproducción de audio: actualmente el navegador del usuario necesita descargar todo el archivo de audio para poder reproducirlo, lo cual conlleva un mayor uso de datos de transmisión y genera problemas puntuales en su reproducción. Implementar una reproducción bajo demanda usando *streaming*, sería más recomendable.

- Implementar requisitos opcionales: implementar la autenticación con iDUMA y limitar el acceso a la interfaz principal de la estructura a los usuarios cercanos a la misma fueron requisitos opcionales, pero aportan valor al proyecto LIMBO.

- Implementar paginación en la carga de usuarios del sistema: actualmente en la vista de administración de usuarios del sistema se cargan todos los usuarios en una sola página, lo cual no es recomendable. El uso de herramientas como la paginación, que divide lista de carga de datos en segmentos, es más recomendable, sobre todo cuando el número de usuarios va creciendo.

- Implementar un manejo dinámico de figuras: actualmente solo están configuradas 4 figuras en el sistema, y si bien estas se cargan de forma dinámica, no se pueden añadir o eliminar. El *Backend* implementa las funciones necesarias para conseguir este manejo dinámico, pero el *Frontend* no da soporte y tampoco la interfaz principal de uso (fuera del alcance de este TFG).
- Reutilización del sistema de información para otros proyectos: la reutilización del presente sistema en otros proyectos académicos y trabajos fin de grado puede

ayudar a mejorar el sistema LIMBO gracias a la retroalimentación generada por el uso del código y su modificación.

Bibliografía

Amazon Web Services. ¿Qué es API RESTful? (s. f.). Recuperado 27 agosto 2021, de https://aws.amazon.com/es/what-is/restful-api/.

Aspose Pty Ltd. WAV - *Waveform Audio File Format*. (s. f). Recuperado 29 agosto 2021, de https://docs.fileformat.com/audio/wav/.

BBVA. Metodología 'scrum': ¿Qué es un 'sprint'? (s. f.). Recuperado 24 agosto 2021, de https://www.bbva.com/es/metodologia-scrum-que-es-un-sprint/.

Drumond, C. Scrum. Atlassian. (s. f.). Recuperado 14 marzo 2021, de https://www.atlassian.com/es/agile/scrum.

ESI. GIGAPORT eX. (s. f.). Recuperado 30 agosto 2021, de https://www.esi-audio.com/products/gigaportex/.

FFmpeg. *A complete, cross-platform solution to record, convert and stream audio and video*. (s. f.). Recuperado 29 agosto 2021, de https://ffmpeg.org/.

Google. angular. (s. f.). Recuperado 14 de marzo de 2021, de https://angular.io/

inngest. MailDev. (s. f.). Recuperado 14 de septiembre de 2022, de https://github.com/maildev/maildev

IntelliJ. IntelliJ IDEA: IDE de Java eficaz y ergonómico (s. f.). JetBrains. Recuperado 14 de marzo de 2021, de https://www.jetbrains.com/es-es/idea/

Lato, N. (2021). ¿Qué es el streaming y cómo funciona? Recuperado 11 septiembre 2022, de https://www.avg.com/es/signal/what-is-streaming#topic-2.

Lucid Software Inc. Qué es un diagrama entidad-relación. Lucidchart. (s. f.). Recuperado 27 agosto 2021, de https://www.lucidchart.com/pages/es/que-es-un-diagrama-entidad-relacion.

Microsoft. Visual Studio Code - Code Editing. Redefined. (s. f.). Recuperado 14 de septiembre de 2022, de https://code.visualstudio.com/

Microsoft. *Your AI pair programmer*. (s. f.). Recuperado 14 de septiembre de 2022, de https://github.com/features/copilot

Mozilla Foundation. Control de acceso HTTP (CORS). (s. f.). Recuperado 29 agosto 2021, de https://developer.mozilla.org/es/docs/Web/HTTP/CORS.

OpenJS Foundation. nodejs. (s. f.). Recuperado 14 de septiembre de 2022 de https://nodejs.org/es/

Oracle. Java. (s. f.). Recuperado 14 de septiembre de 2022 de https://www.java.com/es/

Oracle. MySQL. (s. f.). Recuperado 14 de marzo de 2021, de https://www.mysql.com/

Paraschiv, E. (2021). *Spring Events*. baeldung. Recuperado 1 septiembre 2022, de https://www.baeldung.com/spring-events.

Reactable Systems SL. Reactable Experience. (s. f.). Recuperado 23 agosto 2021, de https://reactable.com/experience/.

Red, J. *What Is the Difference in Size Between MP3 & WAV?* Techwalla. Recuperado 29 agosto 2021, de https://www.techwalla.com/articles/what-is-the-difference-in-size-between-mp3-wav.

Singh, V. (2019). Should I use DB to store file ? medium. Recuperado 7 de agosto de 2021, de https://medium.com/@vaibhav0109/should-i-use-db-to-store-file-410ee22268c7

Smith, C. (2011). *What is the difference between a file system and a database*? Quora. Recuperado 13 de agosto de 2021, de https://www.quora.com/What-is-the-difference-between-a-file-system-and-a-database/answer/Christian-Smith-2

Torralba, A. (2021). ¿Qué es Full Stack? ID Digital School. Recuperado 24 agosto 2021, de https://iddigitalschool.com/bootcamps/que-es-full-stack/.

Uchida-Psztyć, P., Nasution, M., Musiorski, K., Klein, R., Schiltz, A., giper45, Luberti, W. & Badger, B. (s. f.). Advanced REST Client application. GitHub. Recuperado 14 de septiembre de 2022, de https://github.com/advanced-rest-client/arc-electron#readme

Vicerrectorado de Smart-Campus. Limbo Space: Social Sound XP. (s. f.). Recuperado 3 mayo 2021.

VMware. *Spring makes Java simple*. (s. f.). Recuperado 14 de marzo de 2021, de https://spring.io/

Waelder Laso, P., & G. Díaz, P. (2019). Arte computacional e interactivo. Recuperado 23 agosto 2021, de http://arts.recursos.uoc.edu/programacio-disseny-arts/es/4-4-1-origenes-del-arte-interactivo/.

Wikipeadia .*Framework*. (s. f.). Recuperado 26 agosto 2021, de https://es.wikipedia.org/wiki/*Framework*.

Wikipedia. Interfaz de programación de aplicaciones. (s. f). Recuperado 23 agosto 2021, de https://es.wikipedia.org/wiki/Interfaz_de_programación_de_aplicaciones.

Wikipedia. JSON Web Token. (s. f.). Recuperado 28 agosto 2022, de https://es.wikipedia.org/wiki/JSON_Web_Token.

Yuan, K. (2020*). Introduction to File MIME Types*. Baeldung. Recuperado 25 agosto 2021, de https://www.baeldung.com/linux/file-mime-types.

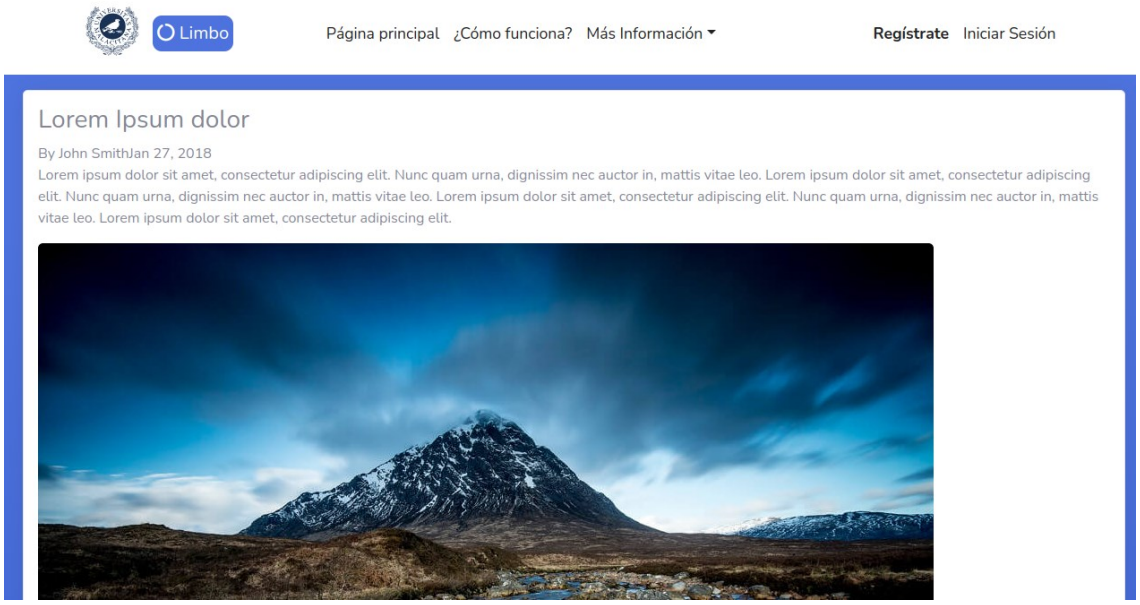
Zine EOOD. Bootstrap Studio. (s. f.). Recuperado 14 de septiembre de 2022, de https://bootstrapstudio.io/

Zine EOOD. Bootstrap Studio. (s. f.). Recuperado 29 agosto 2022, de https://bootstrapstudio.io/.

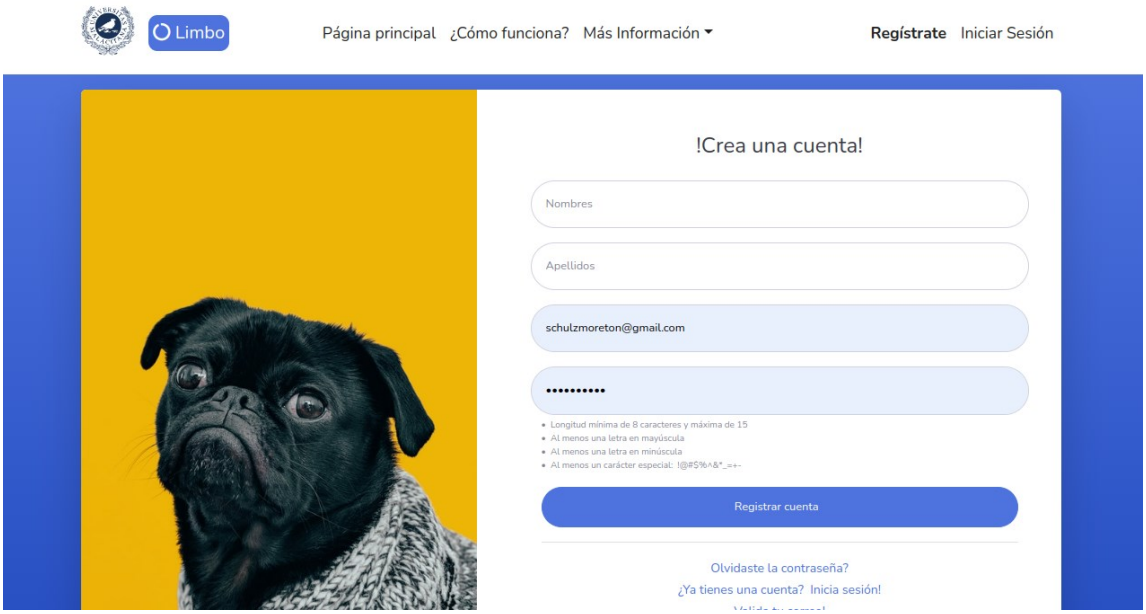
Apéndice A

I. Imágenes del sistema de información

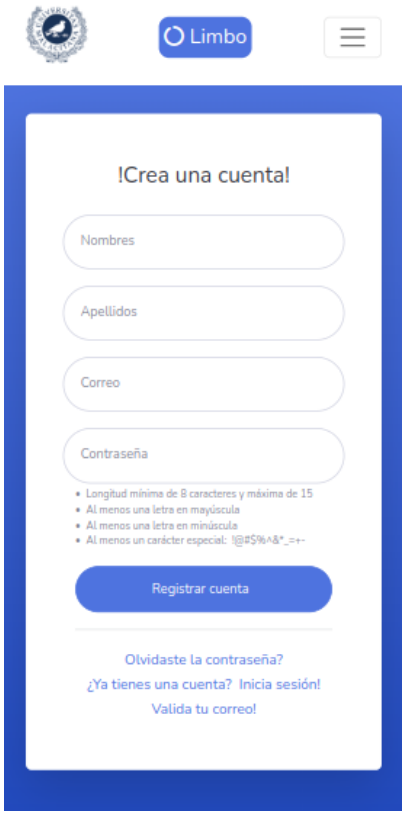
I.I. Vistas principales



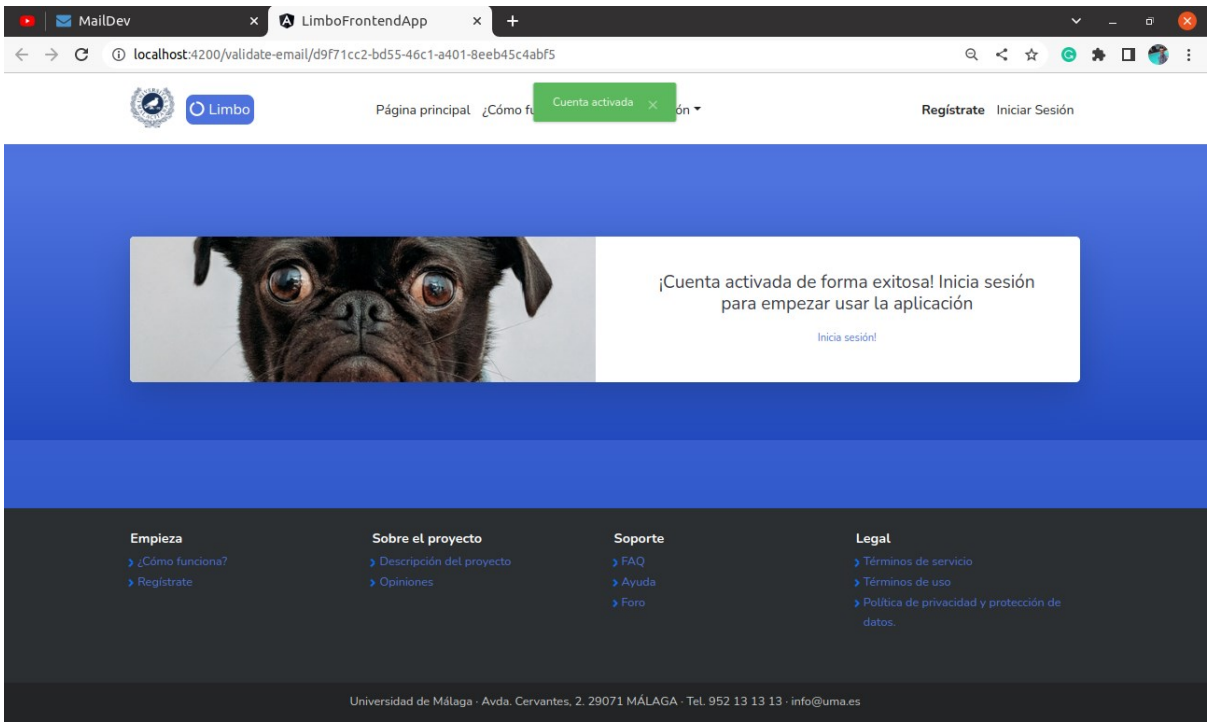
Vista principal del sistema de información con imagen e información de relleno.



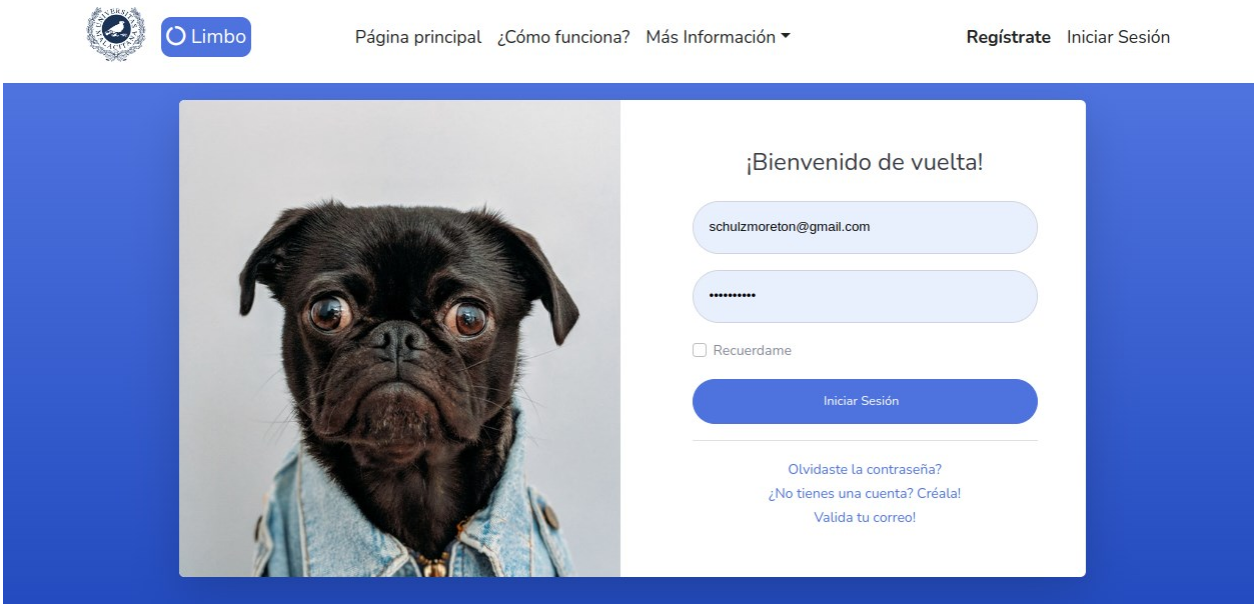
Vista para crear una cuenta en el sistema de información con imagen de prueba.



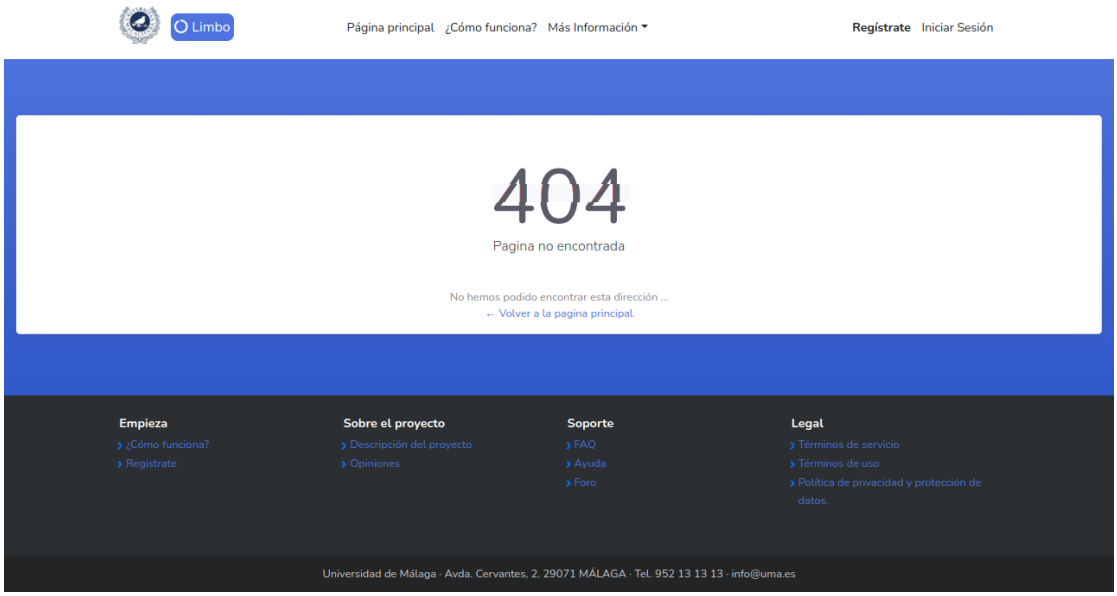
Vista para crear una cuenta en el sistema de información con imagen de prueba.



Vista de validación de cuenta con notificación de éxito con imagen de prueba.

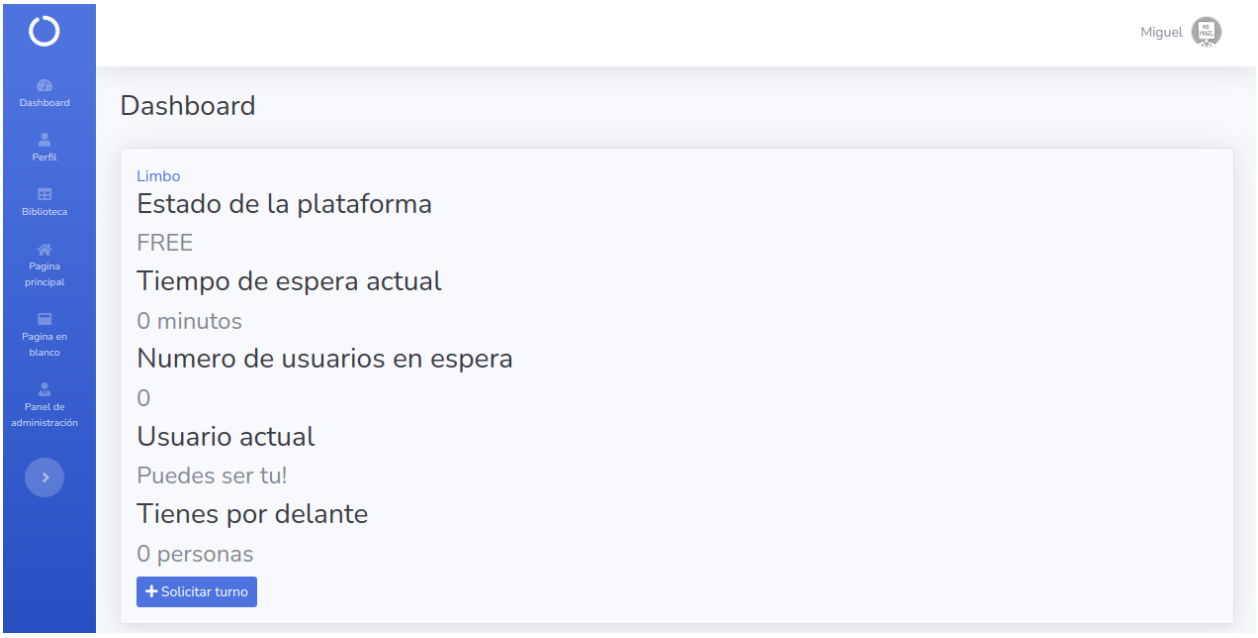


Vista de inicio de sesión con imagen de prueba.

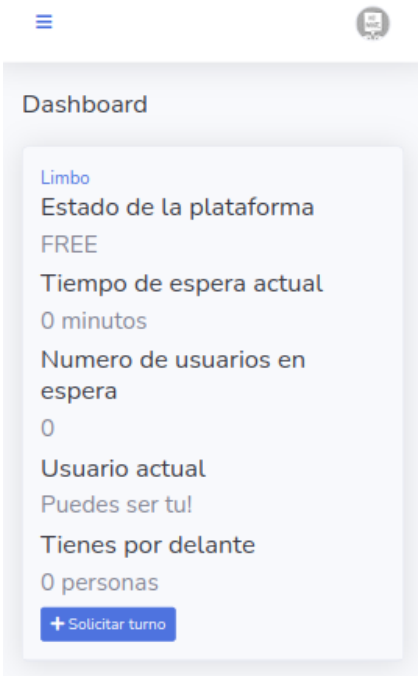


Vista de página no encontrada en la vista principal.

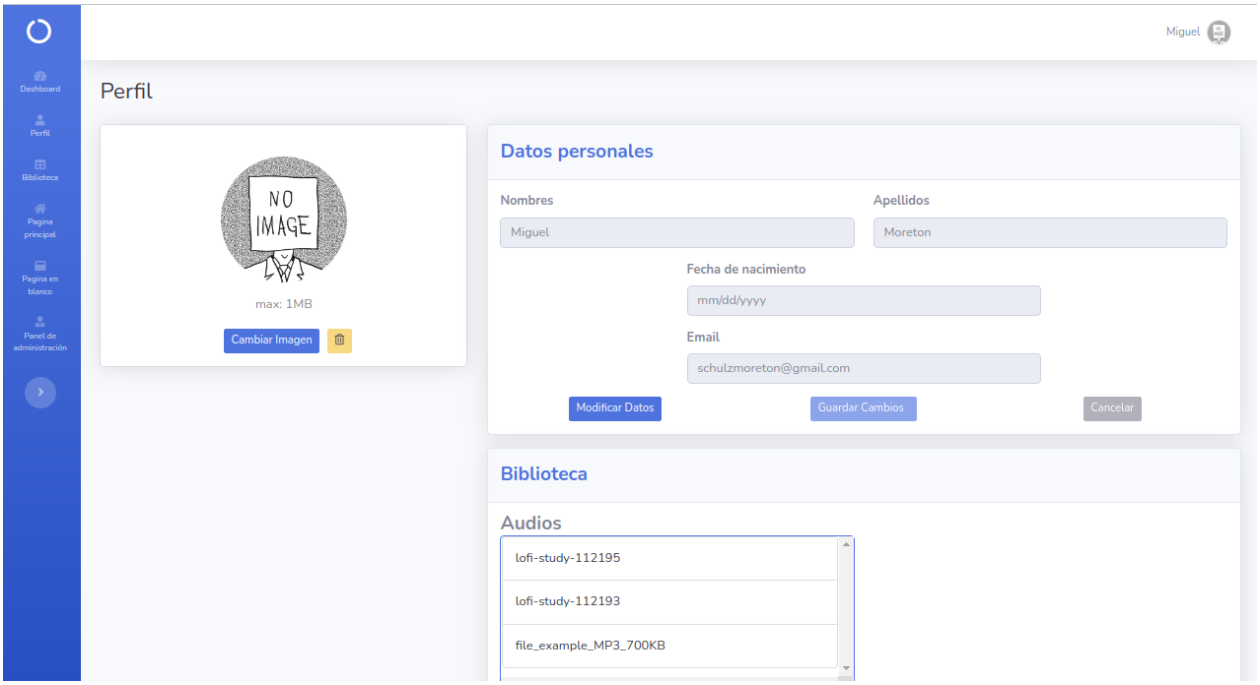
I.II. Vistas del área de usuario



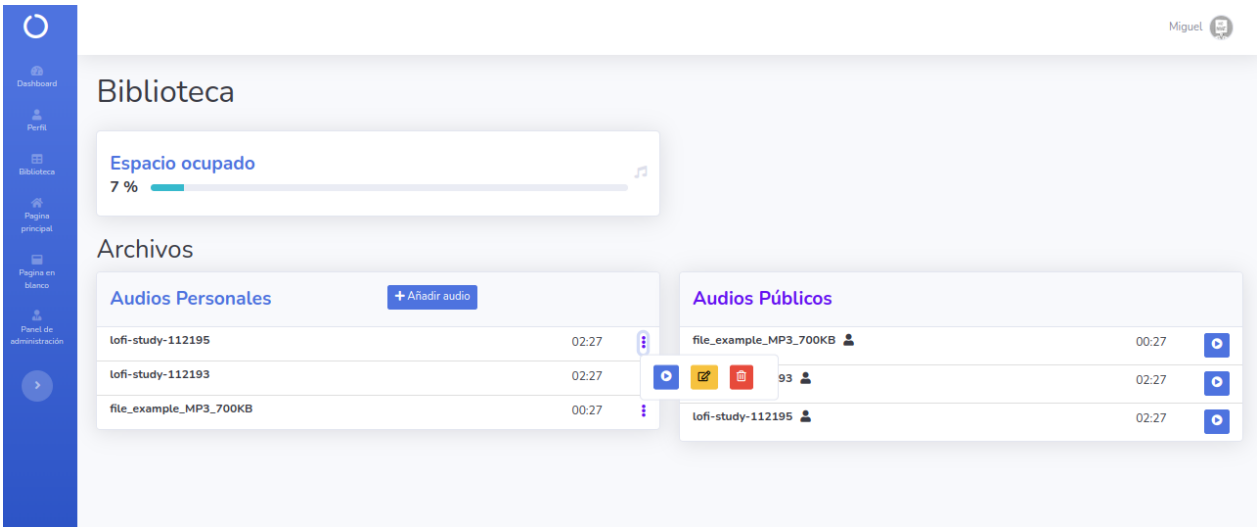
Vista del *Dashboard* en su versión de prueba.



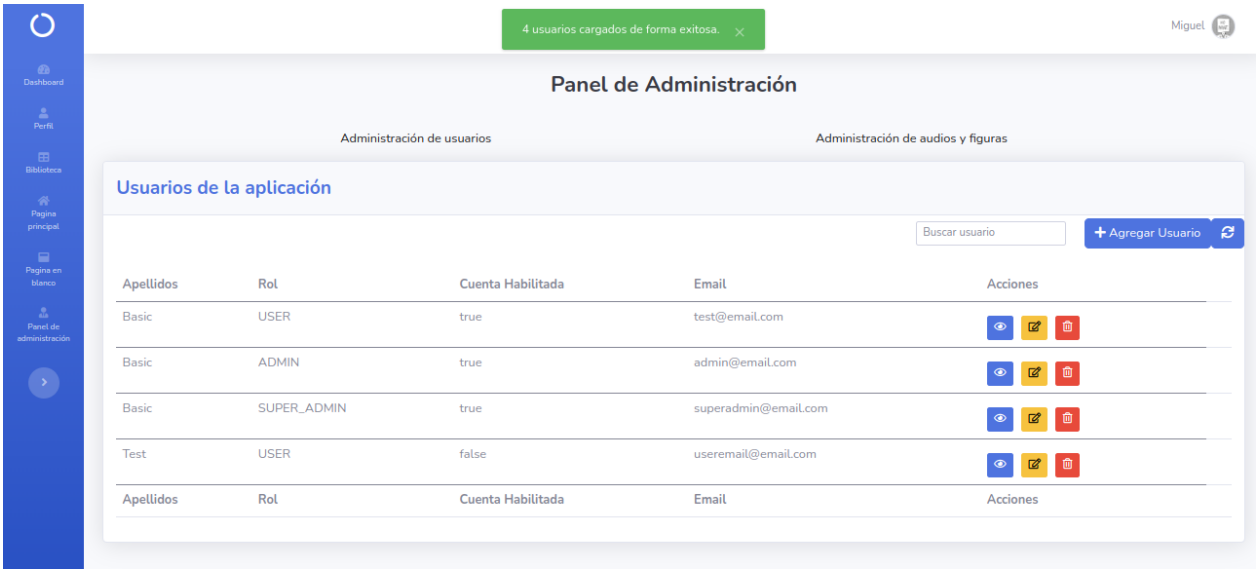
Vista del *Dashboard* en su versión de prueba en formato de dispositivo móvil.



Vista del perfil de un usuario.

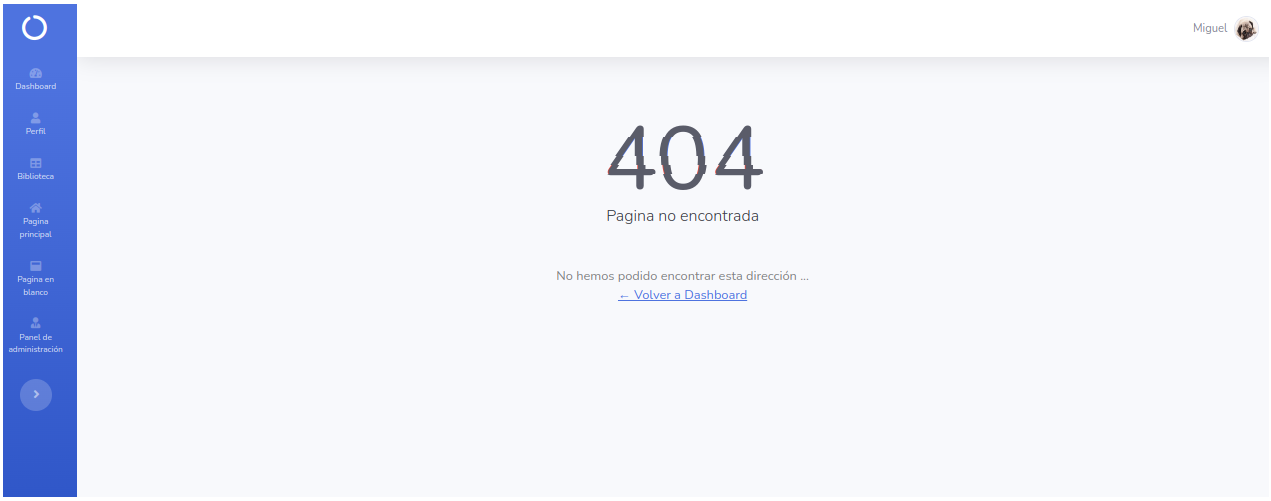


Vista de la biblioteca de audios de un usuario (Audios personales) y audios públicos con desplegable de reproducción, edición o eliminación.



Vista de la biblioteca de audios de un usuario (Audios personales) y audios públicos con desplegable de reproducción, edición o eliminación.

Vista del área de administración de audios públicos y figuras.



Vista de página no encontrada en el área del usuario.

UNIVERSIDAD DE MÁLAGA | uma.es

E.T.S de Ingeniería Informática
Bulevar Louis Pasteur, 35
Campus de Teatinos
29071 Málaga

E.T.S. DE INGENIERÍA INFORMÁTICA