Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Desarrollo de una plataforma social basada en audios

Trabajo Fin de Máster

Máster Universitario en Ingeniería Informática

Autor: Brandon Andrés Lara Erazo

Tutor: Joan Fons i Cors

2020/2021

Resumen

Este proyecto consiste en el desarrollo de la aplicación móvil de una red social completamente enfocada al contenido en formato de audio, el cual se trata de un enfoque diferente al que se utiliza en la mayoría de las redes sociales. La idea inicial es que tenga una funcionalidad similar al de estas redes sociales, por ejemplo, notificaciones, compartir, comentar, realizar publicaciones, perfiles de usuario, etc. Pero con la diferencia de que todo aquello que el usuario publique sea mediante notas de voz, sobre las cuales giraría el resto de las características de la aplicación. Aparte de esto también se diferenciaría de otras plataformas de audio, porque el enfoque de la aplicación permitiría compaginar el contenido profesional y el social, por las propias funcionalidades de la plataforma. Es decir que la propia naturaleza de la aplicación favorecería la creación de contenido general o cotidiano, y la distribución de contenido profesional, como podrían ser podcasts, canciones, o programas. El objetivo es cubrir el vacío que existe entre las redes sociales y las plataformas de audio, ya que estas últimas están enfocadas al ámbito profesional, mientras que las redes sociales se centran en un contenido visual, donde las funcionalidades son específicas para este tipo de contenido.

Para llevarla a cabo, se hace uso de las tecnologías de desarrollo más avanzadas de la actualidad, ya que nos proporcionan las ventajas que mejor se adaptan al trabajo que se realiza, como la escalabilidad, mantenibilidad o la capacidad de desarrollar aplicaciones móviles nativas para iOS o Android con el mismo código. Para poder alcanzar un mayor número de usuarios, el objetivo es publicar la aplicación móvil en la Play Store de Google, y en la App Store de Apple, ya que se tratan de las dos tiendas de aplicaciones más populares del momento debido a que la mayoría de los dispositivos móviles utilizan los sistemas operativos de estas empresas.

Palabras clave: red social, aplicación, Android, iOS, móvil, audios

Resum

Aquest projecte consisteix en el desenvolupament de l'aplicació mòbil d'una xarxa social completament enfocada al contingut en format d'àudio, el qual es tracta d'un enfocament diferent al que s'utilitza en la majoria de les xarxes socials. La idea inicial és que tinga una funcionalitat similar al d'aquestes xarxes socials, per exemple, notificacions, compartir, comentar, realitzar publicacions, perfils d'usuari, etc. Però amb la diferència que tot allò que l'usuari publique siga mitjançant notes de veu, sobre les quals giraria la resta de les característiques de l'aplicació. Apart d'això també es diferenciaria amb altres plataformes d'àudio, perquè l'enfocament de l'aplicació permetria compaginar el contingut professional i el social, per les pròpies funcionalitats de la plataforma. És a dir que la pròpia naturalesa de l'aplicació afavoriria la creació de contingut general o quotidià, i la distribució de contingut professional, com puguen ser podcasts, cançons, programes, etc. L'objectiu és cobrir el buit que existeix entre les xarxes socials i les plataformes d'àudio, ja que aquestes últimes estan enfocades a l'àmbit professional, mentre que les xarxes socials se centren en un contingut visual, on les funcionalitats són específiques per a aquesta mena de contingut.

Per a dur-la a terme, es fa ús de les tecnologies de desenvolupament més avançades de l'actualitat, ja que ens proporcionen els avantatges que millor s'adapten al treball que es realitza, com l'escalabilitat, mantenibilitat o la capacitat de desenvolupar aplicacions mòbils nadiues per

a iOS o Android amb el mateix codi. Per a poder aconseguir un major nombre d'usuaris, l'objectiu és publicar l'aplicació mòbil en la Play Store de Google, i en l'App Store d'Apple, ja que es tracten de les dues botigues d'aplicacions més populars del moment pel fet que la majoria dels dispositius mòbils utilitzen els sistemes operatius d'aquestes empreses.

Paraules clau: xarxa social, aplicació, Android, iOS, mòbil, àudios

Abstract

This project consists of the development of a mobile application for a social network completely focused on content in audio format, which is a different approach from that used in most social networks. The initial idea is that it has a similar function to these social networks, for example, notifications, sharing, commenting, publishing, user profiles, etc. But with the difference that everything that the user publishes is through voice notes, on which the rest of the application's features would revolve. Besides from this, it would also differ from other audio platforms, because the approach of the application would allow combining professional and social content, due to the platform's own functionalities. In other words, the very nature of the application would favour the creation of general or everyday content, and the distribution of professional content, such as podcasts, songs, programs, etc. audio platforms, since the latter are focused on the professional field, while social networks focus on visual content, where the functionalities are specific for this type of content.

To carry it out, the application is made of the most advanced development technologies of today, since it offers us the advantages that best adapt to the work that is done, such as scalability, maintainability, or the ability to develop native mobile applications for iOS or Android with the same code. To reach a greater number of users, the objective is to publish the mobiles application in the Google Play Store, and in the Apple App Store, since they are the two most popular application stores of the moment since most mobile devices use the operating systems of these companies.

Keywords: social network, application, Android, iOS, mobile, audio.

Índice de contenidos

Índice de Figuras

Índice de tablas

1. Introducción

Debido al éxito que han tenido las redes sociales en los últimos años, han surgido una gran cantidad y variedad de este tipo de aplicaciones, y es que hoy en día existen redes sociales que se enfocan en ciertas temáticas, o en particularidades específicas que las diferencian del resto[1]. Por otro lado, uno de los factores que ha influido en la popularidad de las redes sociales es el auge de los móviles inteligentes y las aplicaciones que se pueden instalar en estos, sobre todo en las plataformas iOS y Android, ya que se tratan de las más utilizadas en todo el mundo con una amplia diferencia [2].

En este proyecto se pretende crear una red social que se centre en el audio, para diferenciarse de las soluciones que ofrecen un contenido más genérico o totalmente diferente a lo que se plantea, además con esta nueva red social se quiere ofrecer una alternativa innovadora a las aplicaciones existentes en su ámbito, para ello, hay que tener en cuenta la importancia que tienen las aplicaciones móviles para el desarrollo y el crecimiento de la red social, y en consecuencia de esto, establecer la prioridad del desarrollo para las principales plataformas móviles por encima de cualquier otra.

Esta plataforma consiste en una red social en la que los usuarios van a poder publicar cualquier tipo de contenido únicamente en formato audio, además van a tener la opción de interactuar de distintas maneras, tanto con cualquiera de las publicaciones que se encuentren, como con otros usuarios de la plataforma, para ello se dispondrá de un sistema de seguidores, así como un listado temporal para visualizar las publicaciones propias, o las de los usuarios a los que se está siguiendo. Finalmente, para complementar las interacciones, también existirá la posibilidad de recibir notificaciones cada vez que ocurran.

En primer lugar, se debe considerar que el desarrollo se realiza con unos recursos limitados, debido a que solamente se dispone de una persona para realizar el trabajo y poco presupuesto, de modo que se debe tener como objetivo minimizar todo tipo de costes. Esto se tiene en cuenta a la hora de especificar la metodología de trabajo a seguir, ya que en gran medida de esto depende la planificación del proyecto y el coste temporal que va a tener. Una vez analizado el problema, tanto en diseño como la implementación de la solución se han de realizar utilizando tecnologías y técnicas que permitan ahorrar cualquier tipo de costes.

[1] La información sobre la existencia de los diferentes tipos de redes sociales se ha encontrado en la web https://blog.hootsuite.com/es/8-tipos-de-redes-sociales/ y https://sproutsocial.com/insights/tipos-de-redes-sociales/

[2] Los datos de uso de los sistemas operativos se pueden encontrar en la página web https://www.emagister.com/blog/cuales-son-los-sistemas-operativos-mas-utilizados-en-el-mundo/ y en https://codigoespagueti.com/noticias/mapa-plataformas-moviles-mas-usadas/

1.1 Motivación

En este capítulo se justifica el interés de todos los aspectos relacionados con el proyecto que se va a llevar a cabo, para ello se especifican las motivaciones personales que explican el contenido y la orientación del trabajo, las motivaciones técnicas, los objetivos de aprendizaje, y posibles propósitos profesionales posteriores. De esta manera, los motivos que han llevado a realizar este proyecto son los siguientes:

- Aprender una tecnología novedosa como React Native, ya que se trata de uno de los avances más interesantes de los últimos años porque se trata de un framework para desarrollar aplicaciones nativas multiplataforma con el mismo código, sin duda una de las características más importantes para los desarrolladores de aplicaciones móvil.
- Durante los años de grado y máster se han aprendido a utilizar tecnologías de diferentes tipos relacionadas con el desarrollo de software, por otra parte, se han estudiado diferentes conceptos de la gestión y desarrollo de proyectos software, así como maneras de implementar software de calidad, lo cual resulta verdaderamente interesante ya que significa poner en práctica gran parte de lo que se ha aprendido junto con otros conocimientos adquiridos fuera de los estudios cursados.
- Al tratarse de un proyecto complejo, durante su desarrollo además de poner en práctica lo aprendido también va a suponer un nuevo desafío técnico en el cual se van a aprender cosas nuevas e interesantes que motivaran el crecimiento personal, y el afán por seguir aprendiendo.
- Crear una plataforma novedosa que ayude a mejorar la comunicación y conexión de cualquier persona, haciendo uso de cualquier forma audio como elemento principal de comunicación.
- Publicar una aplicación en las tiendas más importantes de aplicaciones móviles.
- Desarrollar una aplicación interesante para los usuarios completamente enfocada a la comunicación mediante audios.
- Desarrollar desde cero toda la infraestructura de una red social.
- Utilizar NodeJS para desarrollar un servidor para una plataforma social, utilizando patrones y arquitecturas utilizadas en el ámbito empresarial o productivo a nivel mundial actualmente.
- Crear una red social que utilice una forma de interacción enfocada íntegramente a los audios, ya que todas las aplicaciones de este tipo son genéricas y no tienen funcionalidades enfocadas íntegramente a este tipo de interacción, lo que puede resultar atractivo a cierto público.
- Mejorar la forma de compartir contenido, y desarrollar una aplicación referente en esta materia.

Estas motivaciones dejan claro porque llevar a cabo este proyecto resulta de interés, y sirve como punto de partida para la elaboración de los objetivos a continuación.

1.2 Objetivos

El objetivo principal de este trabajo consiste en el desarrollo de una plataforma social, en la cual los usuarios van a poder compartir contenido en formato audio, e interactuar de diferentes formas, además, esta plataforma tiene que estar disponible para los principales sistemas operativos móviles, Android y iOS. Para poder cumplir con este objetivo se tienen que llevar a cabo los siguientes objetivos secundarios:

- Configurar y poner a punto la instancia en la nube que alojara el servidor y la base de datos de nuestra aplicación.
- Configurar los diferentes entornos para facilitar todo el ciclo de vida del desarrollo de la aplicación y alinearlo con las metodologías utilizadas.
- Implementar el sistema de integración y despliegue continuos.
- Desarrollar el backend con una *API REST* para proveer a la aplicación móvil una forma de integrar y realizar las operaciones necesarias sobre la base de datos.
- Implementar toda la plataforma siguiendo las buenas prácticas recomendadas para el desarrollo de aplicaciones empresariales.
- Utilizar un framework de Desarrollo que facilite la implementación de las aplicaciones móviles para Android y iOS.
- Desarrollar pruebas automáticas y planes de pruebas sobre el software que se ajusten al ciclo de vida teniendo en cuenta la integración y el despliegue continuos.
- Tener en cuenta la escalabilidad del sistema y utilizar las tecnologías que favorezcan este aspecto, esto es importante porque se trata de una red social.
- Publicar la aplicación en Google Play Store y la App Store de Apple.
- Asegurar la confidencialidad y seguridad del software, en especial en el backend de la plataforma.

Estos objetivos van a guiar el proceso que se llevará a cabo durante el desarrollo y la implantación de la red social, por lo tanto, son indispensables para poder especificar aquello que se desea conseguir con este trabajo y el propósito que servirá como punto de partida de todo el proyecto.

1.3 Objetivos de desarrollo sostenible: Sociedad inclusiva

La Agenda 2030 para el Desarrollo Sostenible plantea 17 Objetivos de Desarrollo Sostenible, para potenciar y promover la inclusión social, económica y política de todas las personas, independientemente de su edad, sexo, discapacidad, raza, etnia, origen, religión o situación económica u otra condición. Estos objetivos se tratan de una llamada mundial para adoptar medidas que logren acabar con los grandes problemas del planeta

En esta agenda la discapacidad está presente, ya que más de mil millones de personas de todo el mundo viven con alguna forma de discapacidad. En muchas sociedades, las personas con

discapacidad a menudo terminan desconectadas, viviendo aisladas y discriminadas. Además, estas personas tienen un acceso más limitado a las tecnologías de la información y las comunicaciones (TIC) que las personas sin discapacidad. Existe una brecha significativa entre las personas con y sin discapacidad en el uso de internet. Entre 14 países estudiados, solo el 19% de las personas con discapacidad en comparación con el 36% de las personas sin discapacidad utiliza Internet. Esto puede atribuirse a la falta de accesibilidad de dicha tecnología, así como la menor capacidad de los hogares con personas con discapacidad para acceso a Internet. Por ejemplo, más de un tercio de los portales nacionales en línea incluyen características que no son accesibles para personas con discapacidad.

Una de las barreras fundamentales que causan la exclusión de las personas con discapacidad es falta de disponibilidad o accesibilidad en entornos físicos y virtuales. Para ello, en informes de la organización de las naciones unidas se ha propuesto que, en cuanto a la tecnología, debe priorizarse la promoción de la tecnología accesible, siguiendo el enfoque del diseño universal. Además, hay que incentivar la investigación y el desarrollo de la tecnología de asistencia puede ayudar a acelerar la disponibilidad de estas tecnologías.

En la sección anterior se especificó que el objetivo principal de este proyecto es el desarrollo de una red social completamente basada en el audio, ya que se trata de un aspecto que la diferencia de la mayoría de las soluciones existentes en este ámbito. Estas mismas soluciones habitualmente suelen tener limitaciones para personas con discapacidad, lo cual es un factor en la exclusión de este tipo de medios para estas personas. Por ello, entre otras cosas, el desarrollo de la plataforma que se plantea en este trabajo aporta cierto valor para las personas que, debido ciertas discapacidades, solamente pueden consumir contenido en formato de audio, incluyéndolas en las actividades y comunidades sociales, o el consumo de contenido profesional, que se promueven por la naturaleza de la propia plataforma, que como se verá más adelante, tendrá un funcionamiento que permite y contempla un uso social genérico, además del profesional que se le pueda dar. Sin embargo, todavía quedaría un largo camino para poder proveer de un entorno ampliamente inclusivo, que se podría conseguir mediante futuras mejoras y versiones de la plataforma.

1.4 Metodología

En este proyecto no es estrictamente fundamental ofrecer flexibilidad en cuanto a las funcionalidades a entregar debido a que es un trabajo de fin de máster acordado, y por esta misma razón tampoco es importante realizar entregas continuas en cortos periodo de tiempo, ni realizar validaciones funcionales a medida que se va avanzando en el desarrollo. Al ser una sola persona la que se dedica al desarrollo completo de la aplicación, solamente se puede continuar con una etapa o tarea determinada del proyecto cuando se ha terminado la anterior, por lo tanto, no es importante que en la metodología de trabajo se paralelicen las etapas del desarrollo o las tareas a realizar.

Dadas estas circunstancias, lo mejor es desarrollar el software siguiendo un modelo en cascada [1] realizando un análisis y diseño previos al desarrollo, para especificar, validar, y organizar todo lo que se requiere del software, y con esto definir elementos arquitectónicos y la organización del código para comenzar con el desarrollo.

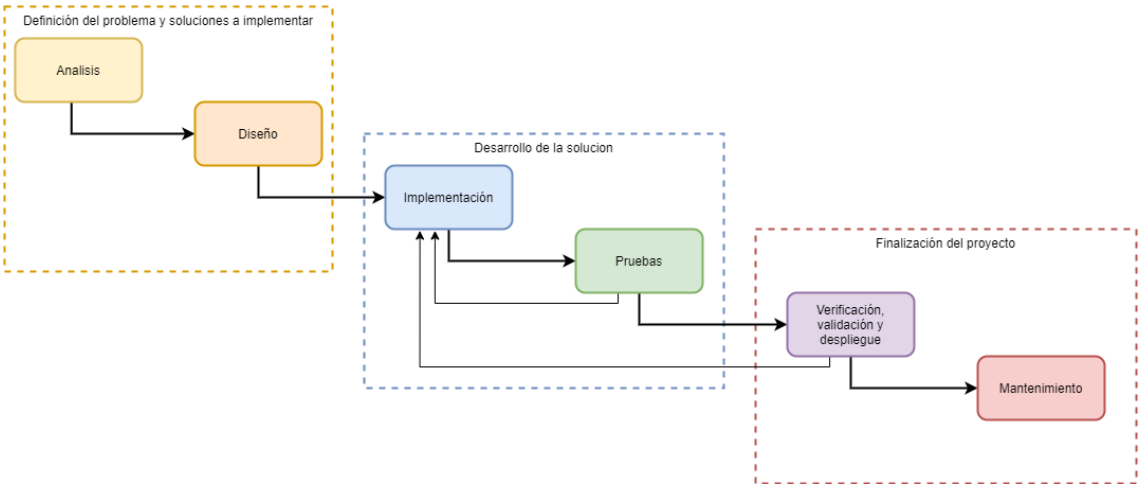


Figura 1.1: Modelo de desarrollo en cascada clásico

El modelo de desarrollo en cascada consiste en un progreso secuencial, dividido en varias fases de los proyectos software ordenadas rigurosamente [1], ya que para dar comienzo a cualquiera de las etapas es estrictamente necesario haber terminado las fases anteriores [2]. Se trata de un modelo ampliamente utilizado y a su vez muy criticado debido a los problemas que puede originar en cualquier etapa del proyecto, por ejemplo, habitualmente al desarrollar software el coste de corregir problemas es mucho más alto cuantas más fases se hayan avanzado, y esto claramente es una desventaja en el modelo de desarrollo en cascada, ya que las validaciones importantes se realizan en las últimas fases del proyecto, y en las cuales se pueden encontrar cualquier tipo de problema que requiera regresar a etapas anteriores, incluso a las fases más tempranas del proyecto, por ejemplo, en caso de haber un problema de análisis que se detecta al final del desarrollo, es muy probable que se tenga que volver a analizar, diseñar y desarrollar el software, lo cual es prácticamente empezar desde el principio.

En este proyecto no se necesita una validación por parte del cliente o de externos, ya que no se trata de un producto o servicio que se tenga que entregar, eso quiere decir que este tipo de problemas no pueden afectar al proyecto, en caso de realizarlo siguiendo la metodología de desarrollo en cascada, principalmente porque los requisitos están prácticamente cerrados desde el principio, y cualquier cambio se haría en tiempo de desarrollo, sin embargo, aunque se dedique solamente una persona a todas las fases del proyecto, y se considere que la fase de análisis se ha validado correctamente, en las siguientes fases es necesario realizar validaciones y pruebas para encontrar posibles problemas, y resolverlos para conseguir entregar un software de calidad.

La fase de implementación se llevaría a cabo por una sola persona, y en este caso, no es lo más adecuado terminar toda la implementación del software, y después continuar con las pruebas y validaciones de todo lo que se ha desarrollado, tal y como se nos indica en el modelo en cascada, porque ante cualquier problema el coste temporal de realizar correcciones sería muy elevado, además de que resulta totalmente contraproducente con las tecnologías que se quieren utilizar, ya que estas nos facilitan el desarrollo, integración, pruebas y despliegue del software.

Por lo tanto, la solución a esto sería realizar una pequeña variación en esta metodología, juntando la etapa de desarrollo y la de pruebas, para que, de esta manera al terminar cada tarea de desarrollo, se debería probar de forma unitaria e integrada con el resto del software ya implementado y/o desplegado.

Todo esto tiene un pequeño inconveniente, y es que se necesita una infraestructura que permita realizar las pruebas de integración con el resto de las tareas, sin afectar a todo lo que ya hemos comprobado que funciona correctamente, y que se encuentra desplegado en un entorno productivo, pero tampoco podemos realizar estas pruebas en local debido a que las diferencias entre el sistema en el que se desarrolla el software y en el que se despliega pueden afectar de manera considerable.

Para resolver este problema, lo más adecuado es crear un entorno de pruebas que consista en un clon exacto del entorno productivo ejecutándose sobre la misma máquina, y sobre el cual se realizaran todas las pruebas sin afectar a cualquier otro entorno. De esta forma la manera más adecuada de trabajar seria la desarrollar tareas, probarlas en local, luego desplegarlas en el entorno de pruebas y realizar las pruebas pertinentes y, por lo tanto, después de validarlo todo ya se podría subir ese código a producción.

Finalmente, por motivos de organización y control, todo lo probado y validado correctamente no se desplegaría directamente en el entorno productivo, sino que se agruparía con otras tareas terminadas para subirlas todas en un mismo paquete como una nueva versión del software, esta metodología de trabajo resulta especialmente conveniente a la hora de realizar la planificación y seguimiento del trabajo de desarrollo.

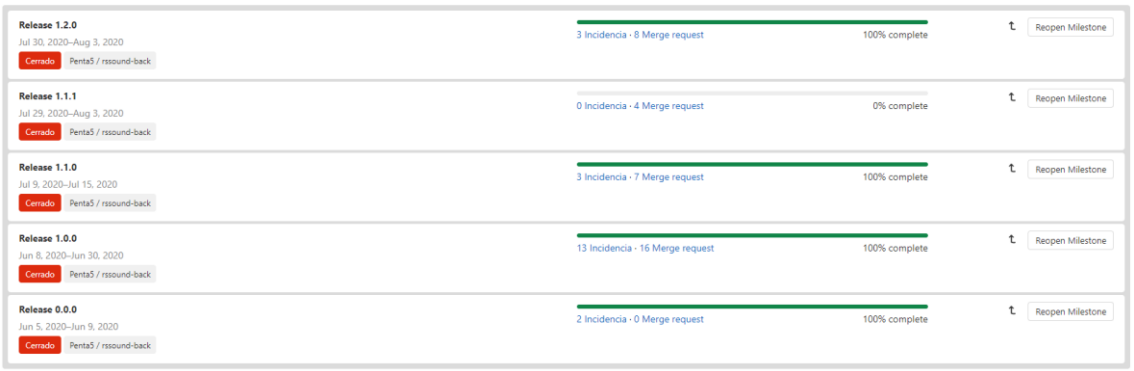


Figura 1.2: Versiones del servidor

Una de las ventajas que nos proporciona esta metodología, es la de estandarizar y optimizar los procesos de desarrollo, ya que en cada tarea existen muchas acciones que se repiten y que se pueden automatizar fácilmente, por eso apoyándose en las tecnologías de despliegue e integración continuos, que se utilizan en este proyecto, se puede conseguir un proceso de desarrollo agradable, metódico, organizado, y controlado, para poder llevar a cabo un software sobre el que se han realizado multitud de pruebas sobre cada funcionalidad antes de ser desplegado.

La metodología va a estar ligada principalmente al sistema de control de versiones, a las subidas de código, y a las fusiones en los repositorios remotos, para que, de esta manera, la integración

del código se haga mediante las propias versiones, que se confirmen mediante este tipo de software.

De esta forma en van a existir repositorios remotos para los entornos de pruebas y de producción, con la versión correspondiente de cada entorno, es por ello que, además del trabajo de desarrollo, se tiene que realizar una integración de código con el repositorio de pruebas, para probar aquello que se acaba de desarrollar, y después de validarlo, realizar esta misma tarea pero en el repositorio de producción, todo ello mediante el sistema de control de versiones, y las automatizaciones de integración y despliegue continuos.

Debido a la gran cantidad de tareas que se tienen que hacer, la integración y el despliegue continuos facilitan y apoyan ciertas partes del trabajo, y por lo tanto no es mala idea adaptar la forma de trabajar a estas tecnologías, y a su vez adaptar de ciertos aspectos configurables a la forma de trabajar. De esta manera se puede mejorar la metodología de trabajo, los procesos, y el software obtenido.

1.5 Colaboraciones

Este proyecto se ha realizado en colaboración con la compañera Luz Mélida García, para su trabajo de fin del máster (TFM) en producción artística de la facultad de Bellas Artes de la Universidad Politécnica de Valencia, titulado “Diseño de interfaz de una red social de audios: Soundn y sus necesidades gráficas” para marzo de 2021.

Esta colaboración se basa en la aplicación móvil que se va a desarrollar, ya que toda la parte de gestión y desarrollo del software se contemplan en este proyecto, mientras que la parte de diseño de la interfaz gráfica y la experiencia de usuario se corresponde al trabajo de Luz. Por lo tanto, el trabajo de cada uno está muy diferenciado, acotado y distribuido, donde Luz se encargaría de diseñar todas las pantallas y elementos gráficos de la aplicación mediante herramientas para el prototipado o para realizar mockups, como Adobe XD o Photoshop. En detalle, en este trabajo se realizarían los mockups y prototipos del diseño y maquetación de las pantallas, y de todos los elementos gráficos, como botones, iconos e imágenes.

El trabajo general es interesante, porque se ponen en práctica diferentes conocimientos de dos disciplinas que a menudo, en este tipo de proyectos suelen estar presentes, ya que hoy en día el aspecto visual de cualquier producto o servicio software es muy importante, y por lo tanto no es de extrañar que cada vez haya más programadores y diseñadores trabajen de forma conjunta para llevar a cabo proyectos como el que se trata en este documento.

2. Estado del arte

Actualmente el uso de las redes sociales se ha convertido en un fenómeno a nivel mundial debido a lo extendido que esta su uso y todo lo que han generado a su alrededor. Aunque la mayoría parten de bases muy similares entre sí, es cierto que la clave del éxito es la diferencia que tienen en aspectos específicos como la temática, las mecánicas propias de la red social, la forma de comunicar, el tipo de comunicación, enfoque, incluso el dispositivo desde el que se accede a la red social.

El rápido desarrollo de las tecnologías móviles también ha sido una parte fundamental en el éxito de este tipo de software, para empezar la aparición de teléfonos inteligentes proporciono dispositivos de bolsillo en los que se podía realizar muchas de cosas que hasta el momento solamente se podían hacer en ordenadores o dispositivos especializados.

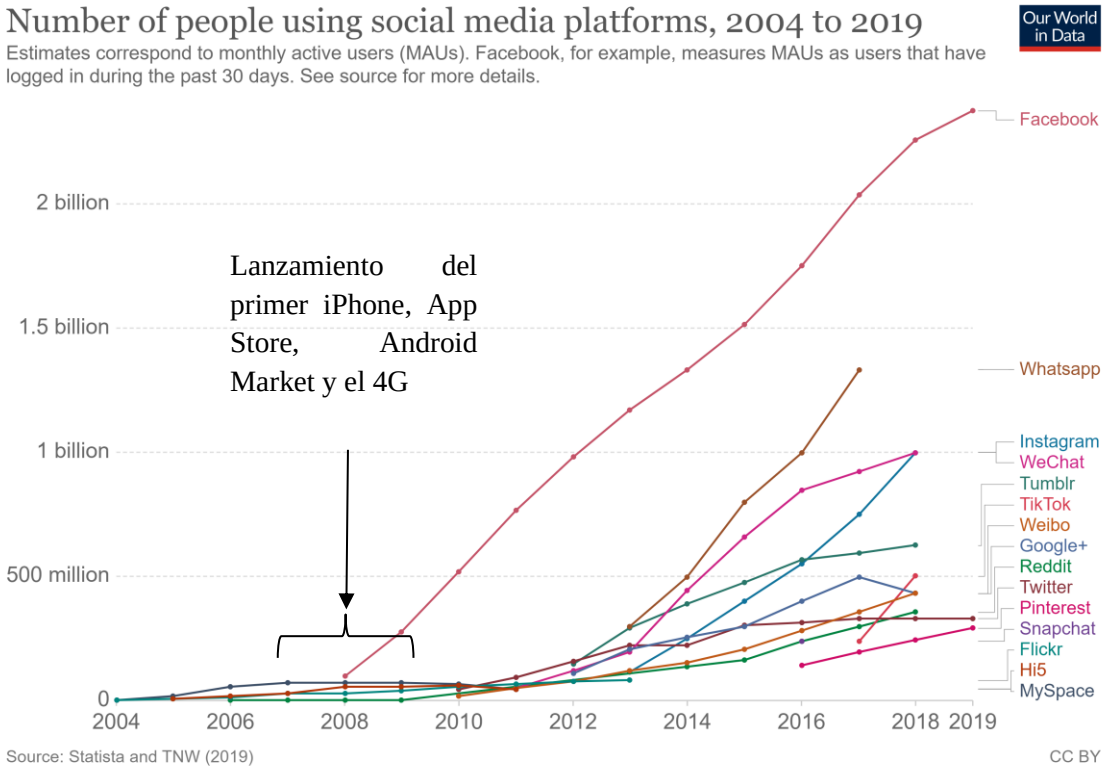


Figura 2.1: Uso de las principales plataformas sociales

Junto a los anuncios de los primeros smartphones empezaron a surgir tiendas de aplicaciones integradas para poder descargar e instalar aplicaciones móviles en estos, y el que mientras los smartphones aumentaban en popularidad por los usos que se le podía dar a través de sus aplicaciones, las tiendas se llenaban de más cantidad y variedad de aplicaciones debido al aumento de popularidad de los dispositivos móviles. Además, desde entonces ha habido mejoras en las telecomunicaciones que han proporcionado la posibilidad de transferir contenido cada vez más grande y con menor latencia.

Android se trata de un sistema operativo basado en el núcleo de Linux, y completamente diseñado para dispositivos móviles con los que se pudiese interactuar principalmente mediante una pantalla táctil, y se permite su uso en cualquier hardware de terceros. Actualmente se trata del sistema operativo móvil más utilizado móvil con una cuota de mercado superior al 70% en el último año[3].

Este sistema fue adquirido por Google en 2005 y posteriormente en 2007 se anuncia la primera versión del sistema operativo, sin embargo, los primeros dispositivos no estarían disponibles hasta 2008, y desde los primeros años su cuota de mercado creció a una gran velocidad. En el año 2020 ya se superaban los dos millones y medio de aplicaciones disponibles en su tienda oficial de aplicaciones[4], Play Store; y actualmente consta de una enorme comunidad de desarrolladores de aplicaciones en todo el mundo.

iOS es otro sistema operativo móvil desarrollado por Apple para sus dispositivos móviles, el cual está totalmente orientado al uso mediante pantallas táctiles, igual que en Android. La compañía Apple no permite el uso de este sistema operativo en hardware de terceros, por lo que se trata de un software propietario exclusivo para dispositivos Apple.

Este sistema fue presentado en 2007 con el nombre de iPhone OS, pero no sería hasta 2010 después de la inclusión de este sistema operativo en iPad que se le renombraría con su nombre actual.

Al igual que Android, iOS consta de una gran comunidad de desarrolladores de aplicaciones, y un ecosistema bastante maduro. En la actualidad la App Store, su tienda de aplicaciones oficial, tiene casi dos millones de aplicaciones disponibles.

Antes de la aparición de los smartphones las redes sociales ya eran ampliamente utilizadas, pero como podemos observar no fue hasta los lanzamientos de todos los avances explicados anteriormente que aumentó en gran medida el uso, la cantidad y la variedad de las redes sociales.

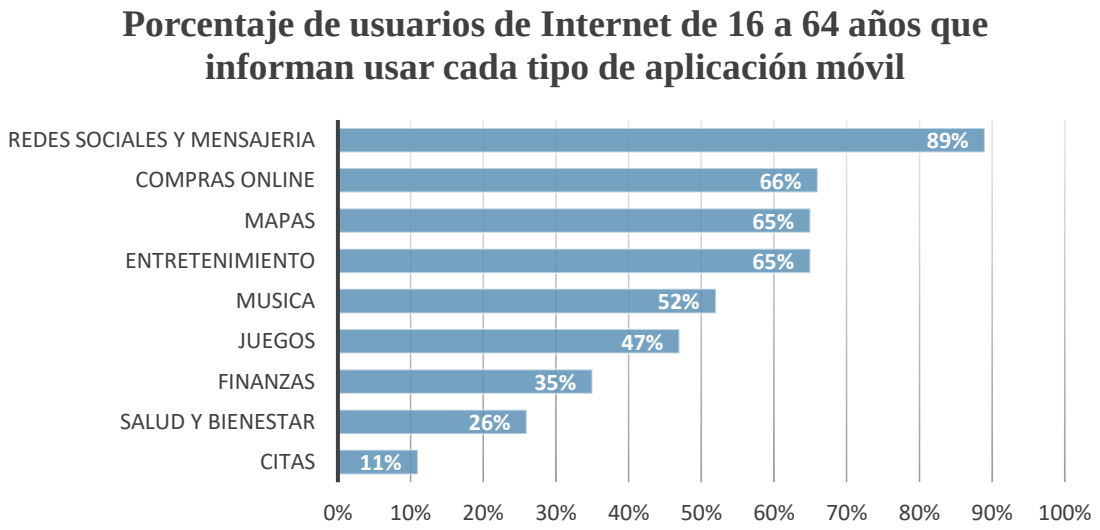


Figura 2.2: Uso de las categorías de aplicación móvil en el tercer cuarto de 2019

[3] Fuente: https://gs.statcounter.com/os-market-share/mobile/worldwide
[4] Fuente: https://www.statista.com/statistics/276623/number-of-apps-available-in-leading-app-stores/

Recientemente se puede observar que la mayoría de las personas suele utilizar sus dispositivos móviles para acceder a cualquier red social o aplicación de mensajería por encima de otras categorías. En el gráfico de la Figura 2.2, con datos de Globalwebindex, se puede observar este hecho, ya que nos dice que el 89 % de los usuarios de internet han utilizado por lo menos una vez en el tercer cuarto de 2019 alguna aplicación de la categoría de redes sociales o mensajería.

Como se ha visto hasta ahora, debido a múltiples circunstancias la importancia y popularidad de los móviles inteligentes ha ido aumentando en gran medida a lo largo de los últimos años, si a esto se le suma que, según la gráfica anterior, la categoría más utilizada en aplicaciones móviles es la de redes sociales y mensajería, podríamos decir que el desarrollo de aplicaciones móviles tiene una gran importancia para las redes sociales si se tiene en cuenta que desde el principio de la década del 2010, las redes sociales se utilizan mayoritariamente en dispositivos móviles tal y como se indica en los siguientes gráficos de The Wall Street Journal en 2013 y Statista para España en 2019.

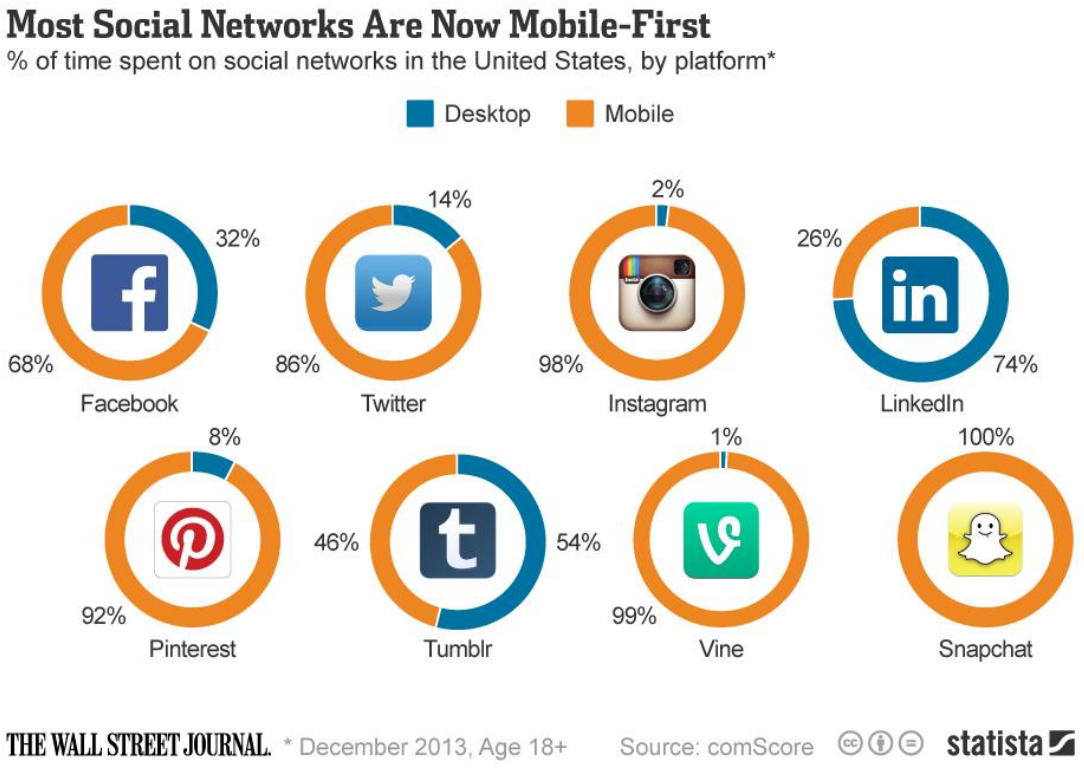


Figura 2.3: Porcentaje de personas que accedieron a redes sociales por dispositivo

Dentro de la gran variedad de redes sociales que existe actualmente, se puede observar que algunas son más genéricas y abarcan un poco de todo sin enfocarse mucho en cualquier característica, mientras que hay otras que se centran en ciertas particularidades que las diferencian del resto, o en nichos en los que sus usuarios suelen tener necesidades muy específicas, y donde estas plataformas se enfocan totalmente en satisfacer estas necesidades. Por ejemplo, en Facebook tiene bastantes funcionalidades muy variadas que no están especialmente enfocadas más allá de lo habitual en las redes sociales, por otra parte, LinkedIn tiene funcionalidades similares, pero las enfoca a la búsqueda de trabajo, además de características completamente enfocadas a la gente que busca trabajo o a los expertos de selección de personal para las empresas.

Finalmente, Instagram enfoca sus funcionalidades a la comunicación a través de contenido visual, ya que las publicaciones solo pueden ser fotos o videos que se acompañan con características que potencian específicamente esta forma de compartir contenido, por ejemplo, al subir una foto existe la posibilidad de añadirle varios tipos de filtros para estilizarla, también existe la posibilidad de modificar algún aspecto de la cara para publicarlo en video.

2.1 Soluciones existentes

Hay redes sociales que se enfocan al audio principalmente, y además de diferentes maneras, ya que es una forma de comunicación y de creación de contenido muy relevante hoy en día. Este tipo de redes existen para satisfacer ciertas necesidades dentro de la temática del audio, y se diferencian entre si según las necesidades que se cubran o la forma de cubrirlas. Por lo tanto, podemos encontrar plataformas sociales especializadas en podcasts, producción musical, o la comunicación mediante audios como las que se explican a continuación.

Limor

Consiste en una plataforma para realizar publicaciones mediante audios utilizando algunas de las mecánicas habituales de las redes sociales. Limor permite grabar, editar, compartir y escuchar audio desde su aplicación móvil para Android o iOS, el enfoque de esta plataforma es el audio social, ya que implementa varias funcionalidades de interacción social propias de las redes sociales, pero utilizando el audio como el centro de todo.

Twitter (Audios en iOS)

Es una de las redes sociales más conocidas actualmente, que se caracteriza porque permite enviar mensajes de texto de poca longitud. Los usuarios pueden suscribirse a los mensajes de otros usuarios, para que se les muestren en sus respectivas paginas principales. Actualmente el contenido de estos mensajes puede ser de otros tipos aparte de solo texto, como pueden ser videos, imágenes o imágenes animadas. Sin embargo, durante el desarrollo de este proyecto, Twitter anunció la posibilidad de publicar contenido en formato de audio solamente para su aplicación de iOS ya que previamente no se podía crear publicaciones donde su contenido fuese exclusivamente de audio. Esta nueva funcionalidad permite al usuario grabar audio mediante el micrófono del teléfono para posteriormente publicarlo en la plataforma con las mismas características que el resto de las publicaciones que se pueden realizar.

Koo

Se trata de una aplicación que parecía prometedora debido al funcionamiento que se mostraba en las demostraciones, y la visibilidad que obtuvo en plataformas conocidas como TechCrunch o Product Hunt. No obstante, parece que ha sido abandonada y que no se ha continuado con el proyecto, ya que sus enlaces y referencias no funcionan actualmente, y tampoco hay forma de encontrar esta aplicación en internet, ni más información acerca de esta.

Se describe como una red social basada en audios, que permite a los usuarios grabar y escuchar clips de audio inferiores a 1 minuto. La experiencia de usuario y el funcionamiento tenían

grandes similitudes con Snapchat, pero totalmente enfocado a la creación de audios. Según las demostraciones, esta aplicación parecía destacar por su sencillez en cuanto a funcionalidad e interfaz, sin descuidar el aspecto visual.

iVoox

Es una plataforma para distribuir, buscar y escuchar podcasts de un extenso catálogo. Esta aplicación es la que menos se enfoca en el aspecto social de todas las anteriores, ya que sus características están totalmente dirigidas a la reproducción de podcasts, ya que permite guardar favoritos, crear listas de reproducción, buscar por temáticas, etc. Por otra parte, cualquier persona puede subir sus podcasts, pero al tratarse de una de las plataformas de podcasting hispanohablante más importante, es posible encontrar particulares o empresas, más o menos conocidos, que se dedican a subir contenido de forma profesional y seria.

SoundCloud

Consiste en una red social para músicos, donde se les proporcionan canales para distribuir y promocionar sus creaciones. Dentro de su categoría es una de las plataformas más importantes de todo el mundo, donde muchos artistas conocidos publican sus proyectos musicales. Se caracteriza por la sencillez en su interfaz y sus funcionalidades, ya que los creadores simplemente tienen que subir el archivo de audio, y ponerle un título para publicarlo en la plataforma, y de esta manera cualquier persona pueda escucharlo mediante un reproductor para cada publicación. Finalmente, el componente social consiste en la posibilidad de seguir a un usuario para visualizar sus publicaciones en la página principal, comentar sobre el reproductor de las publicaciones, marcar cualquier canción publicada como me gusta, guardarla en favoritos, o compartirla con otros usuarios dentro de la aplicación.

2.1.1 Crítica al estado del arte

En las redes sociales que se han explicado anteriormente, hay varios aspectos que se pueden mejorar de cara a ganar popularidad, diferenciarse y realizar una propuesta de valor.

Para empezar, Limor es la aplicación más similar a lo que se intenta conseguir en este proyecto, pero su resultado tiene algunos inconvenientes para ser una propuesta de valor, e incluso para lograr diferenciarse de las redes sociales ya existentes. Lo primero que podemos ver en esta aplicación es que, a pesar de haber pasado dos años desde su lanzamiento no parece que tenga una gran comunidad, esto puede deberse a que durante todo este tiempo no se haya expandido a la plataforma de escritorio, para seguir creciendo con las personas que se dedican de forma profesional a crear contenido de audio, ya que estas suelen trabajar principalmente desde dispositivos de escritorio.

A pesar de que la facilidad y sencillez para comunicarse debe de ser uno de los aspectos clave en las redes sociales enfocadas en el audio, en algunas de las funcionalidades de Limor no se consigue alcanzar estos aspectos, y es que su experiencia de usuario en algunos puntos puede resultar tediosa, como el caso de editar y publicar una grabación de audio.

Finalmente, la experiencia de usuario y el aspecto visual de esta aplicación pueden no ser los más adecuados para los estándares de hoy en día, y provocar que lo usuarios no quieran utilizarla en un primer momento, o que usuarios actuales dejen de utilizarla por encontrar alternativas mejores y más modernas.

En el caso de la funcionalidad de subir clips de audio en Twitter, su principal punto negativo es que se trata de una implementación muy específica en una red social con un uso más genérico, y es que no hay nada de atractivo en utilizar esta funcionalidad en una red social en la que ya se puede escribir, subir videos o fotos, y que la gente ya está habituada a realizar ese tipo de publicaciones, tampoco no existen características específicas que fomenten las publicaciones de audios, y puede ser esta una de las razones por las que no es más popular, y es que dentro de Twitter ahora mismo está funcionalidad, es una posibilidad más que se ofrece al usuario pero sin ningún elemento especifico que la haga más útil, o atractiva para el usuario. Por último, no es ideal que las publicaciones de audios solamente están disponibles para iOS, ya que Android es la plataforma de móviles más utilizada y no debe de infravalorarse, aunque iOS se utilice mayoritariamente en esta red social, la cantidad de usuarios que utilizan Twitter para Android sigue siendo muy grande.

En el caso de Koo, no se puede saber con exactitud las razones por las cuales se abandonó el proyecto, ya que desde el principio parecía contar con suficientes apoyos para llevar a cabo la idea sin problemas. En este caso no se puede hacer uso de la plataforma, ya que actualmente no se encuentra en ningún sitio. Sin embargo, por entrevistas y demostraciones se puede ver que se centraban antes en conseguir más usuarios, que, en recibir ingresos con la aplicación, para ello su estrategia era centrarse en los creadores de contenido de YouTube, y otorgarle otro canal más de comunicación complementario. Esto puede tener varios problemas, uno de ellos puede ser que los creadores de contenido no tengan la necesidad de tener otro canal más de comunicación, para eso ya tienen YouTube, esta plataforma no les ofrece ninguna ventaja con respecto a las soluciones existentes.

La plataforma iVoox por su parte no apuesta tanto por el componente social, sino que se centra mucho más en el contenido, y al parecer no es una mala decisión, teniendo en cuenta que es una de las plataformas de podcasting más importantes de España. Pero esto puede conllevar algún riesgo como puede ser la aparición de competencia ofreciendo algo similar o más atractivo de alguna forma, un claro ejemplo de esto puede ser Spotify, que en los últimos años ha comenzado a ofrecer servicios similares y enfocados al podcasting profesional.

Por su parte SoundCloud ofrece un servicio bien definido para músicos y con ciertos elementos sociales que hacen crecer la comunidad, y promueven el uso de la aplicación al dar la opción de interactuar con la gente que publica canciones. Es evidente que esta fórmula ha funcionado y ha sido exitosa en los últimos años, y es por eso por lo que no ha sido necesario innovar o buscar formas para mejorar el servicio que provee SoundCloud. Esto es bueno de cierta manera, ya que si algo está bien lo mejor es no cambiarlo, pero como se ha visto hasta ahora la rápida expansión de las redes sociales y de los dispositivos móviles, están dando oportunidades para llevar a cabo plataformas con enfoques diferentes que resulten interesantes para el público general, y a su vez para ciertos nichos como ha intentado Limor, y es muy probable que SoundCloud no lo esté aprovechando.

Recapitulando, anteriormente se han analizado algunas soluciones especializadas en diferentes aspectos, de las plataformas centradas en el contenido de audio, sin embargo, de alguna forma u

otra no se está aprovechando el potencial que tienen las redes sociales, y de esta forma parece que no se está apostando por alternativas innovadoras, que junten las características interesantes de las plataformas de audio y de las redes sociales.

Es un hecho que debido a la popularidad de algunas, es completamente normal que se decida no invertir en nuevas funcionalidades o cambios en sus aplicaciones, ahora bien, la única aplicación con estas características que se ha encontrado es Limor, y según todo lo que se ha considerado previamente, teniendo en cuenta que las redes sociales dependen de su popularidad, y que esta aplicación tiene alrededor de mil descargas en la Play Store, se podría decir que no hay aplicaciones realmente llamativas con particularidades de las redes sociales y completamente basadas en el audio.

2.2 Contexto tecnológico

En este capítulo se analizan las herramientas, middlewares [3], frameworks y otras tecnologías que se van a utilizar en la plataforma, para ello se explica en qué consisten, cuáles son sus ventajas, cuál es su relación con la aplicación que se va a desarrollar, y finalmente se compara con otras tecnologías similares que resuelven el mismo problema de otra forma diferente.

React Native

Es un framework de desarrollo de aplicaciones móviles de código abierto y basado en JavaScript, el cual es utilizado para desarrollar aplicaciones para Android y iOS con las capacidades y el rendimiento de la plataforma nativa, y con los beneficios del desarrollo web. Además, React cuenta con una buena comunidad y el respaldo de una compañía como Facebook.

La principal ventaja de las aplicaciones nativas es que su rendimiento es considerablemente mejor que las aplicaciones desarrolladas mediante otras soluciones como por ejemplo las webapps. Sin embargo, el problema de desarrollar aplicaciones nativas es que es necesario realizar todo el proceso de desarrollo para todas de las plataformas, ya que cada una utiliza tecnologías y lenguajes diferentes, para solventar este problema se utiliza React Native ya que con un mismo código se pueden implementar aplicaciones para Android, iOS, y UWP.

En nuestro caso nos otorga una inmensa mejora en la mantenibilidad del código, con un rendimiento similar al de las aplicaciones nativas, sin tener que repetir el trabajo de desarrollo de aplicaciones para cada plataforma, y facilita el desarrollo de las aplicaciones debido al fast refresh, que permite comprobar los cambios en la aplicación instantáneamente en el momento en el que se guarda el código, sin tener que esperar a compilaciones o builds del código.

NodeJS

Es un entorno de ejecución de JavaScript multiplataforma, de código abierto, con una arquitectura orientada a eventos asíncronos, y está enfocado para crear aplicaciones en red escalables. NodeJS es similar en diseño y está influenciado por sistemas evento machine de Ruby y Twisted de Python, pero llevando este modelo de eventos más allá. Incluye un bucle de eventos como runtime de ejecución en lugar de una biblioteca[5].

Esta plataforma, ha ganado mucha popularidad en los últimos años entre desarrolladores y analistas. Además, es aceptado y apoyado por múltiples empresas reconocidas, lo que implica un uso importante en el ámbito empresarial para desarrollar diferentes tipos de productos y servicios [4].

Por lo general, el comportamiento se define mediante devoluciones callbacks de llamada al iniciarse un script y al final se inicia un servidor a través de una llamada de bloqueo. Al contrario que la mayoría del código JavaScript, no se ejecuta en un navegador, sino en el servidor.

En NodeJS no existe como tal la llamada de inicio del evento de bucle, simplemente se entra en el bucle de eventos después de ejecutar el script de entrada y sale cuando no hay más devoluciones. Se comporta de forma similar a JavaScript en el navegador.

PM2

Se trata de un gestor de procesos para aplicaciones de diferentes plataformas en entornos productivos, y que cuenta con balanceador de carga incorporado. Nos permite mantener las aplicaciones operativas siempre, recargarlas sin tiempos de caída y nos facilita las tareas comunes de la gestión de sistemas. Las principales características de esta tecnología son:

- Configuración del comportamiento
- Soporte de mapa de fuentes
- Integración con contenedores
- Observación y recarga
- Gestión de logs
- Monitorización
- Sistema de módulos
- Recarga en máxima memoria
- Modo clúster
- Recarga en caliente
- Flujo de trabajo de desarrollo
- Scripts de inicio
- Flujo de trabajo del despliegue
- Compatible con PaaS
- Monitorización de métricas clave

[5] Sobre NodeJS en la web oficial: https://nodejs.org/es/about/

En este proyecto nos resulta esencial utilizar PM2 debido a que, por motivos económicos, el servidor de la aplicación se despliega en diferentes entornos sobre el mismo VPS, y PM2 nos proporciona todas las herramientas para que la gestión de los entornos sea sencilla y escalable.

Express

Es un framework de aplicaciones web para NodeJS, de código abierto y bajo la licencia MIT. Está diseñado para el desarrollo de aplicaciones web y APIs fácilmente. Se le ha nombrado como el framework de servidor estándar de facto para NodeJS, y proporciona funcionalidades de enrutamiento de peticiones, gestión de sesiones y cookies, etc.

Express está basado en Connect, el cual es un framework que a su vez se basada en el módulo http de NodeJS, por lo tanto, Express va a tener todas las funcionalidades de Connect y el módulo http.

Realmente no se ha planteado utilizar cualquier alternativa a Express, ya que se trata uno de los frameworks más importantes y populares de NodeJS, que cuenta con una documentación detallada, un buen soporte, y una amplia comunidad con la que poder resolver los problemas que puedan surgir. Incluso existen algunos frameworks de más alto nivel como LoopBack, que facilitan y aceleran el desarrollo en gran medida al abstraer el código, pero esto suele tener el inconveniente de reducir la flexibilidad durante el desarrollo. Finalmente, también se ha tenido en cuenta que Express se ha utilizado en los estudios realizados.

MySQL

Se trata del sistema de gestión de bases de datos de código abierto más popular del mundo, y uno de los más populares teniendo en cuenta aquellos sistemas que no son de código abierto. Este sistema se utiliza para administrar bases de datos relacionales, que utilizan múltiples tablas para almacenar y organizar la información. Se trata de un sistema muy utilizado por los desarrolladores de aplicaciones web porque al ser de código abierto es fácilmente accesible, y también por contar con una gran comunidad en la que apoyarse a la hora de implementar bases de datos.

Sus características más destacadas son:

- Arquitectura cliente - servidor
- Compatibilidad con SQL
- Vistas
- Procedimientos almacenados
- Desencadenantes
- Transacciones

MySQL se trata de una opción para tener en cuenta al usarlo en el ámbito empresarial. Al ser de código abierto permite disponer a desarrolladores y pequeñas empresas de una solución fiable y estandarizada para la puesta en marcha de sus bases de datos.

La principal razón por la que se ha utilizado esta tecnología es porque se trata de un software totalmente gratuito pero de gran calidad, pero existen ciertas soluciones que cumplen estas características como PosgreSQL que se han descartado debido a la poca experiencia con estas

tecnologías en comparación con el conocimiento en MySQL, es interesante destacar que aunque los lenguajes pueden ser muy similares, en otros aspectos pueden haber diferencias que necesiten de un tiempo de aprendizaje previo para poder llevar a cabo el trabajo.

PassportJS

Es un middleware de autenticación para NodeJS, flexible y modular. Passport puede utilizarse sin problemas con cualquier aplicación web basada en Express, además nos proporciona un enorme catálogo de estrategias de autenticación que podemos utilizar según las necesidades del proyecto.

JWT

Es la abreviación de JSON Web Token, el cual es un estándar abierto de la industria que define la forma de transmitir información de forma segura entre diferentes partes mediante un objeto JSON. Esta información puede ser verificada y confiable porque está firmada digitalmente. Los tokens firmados pueden verificar la integridad de los reclamos que contiene, mientras que los tokens cifrados ocultan esos reclamos de otras partes.

Visual Studio Code

Se trata de un editor de texto diseñado específicamente para editar código fuente de cualquier tipo. Es un programa de código abierto, gratuito y desarrollado por Microsoft. Entre sus características destacan la sencillez de la interfaz, la facilidad de uso, y la amplia compatibilidad con lenguajes. Además, también destaca porque se pueden añadir características extra mediante extensiones que se pueden descargar del *marketplace* que tiene integrado dentro de la aplicación, y donde se publican extensiones creadas por la comunidad Open Source que se han utilizado en el desarrollo de este proyecto, ya que en muchos escenarios son de mucha utilidad, han mejorado la experiencia, la productividad y la comodidad durante las sesiones de trabajo.

Git

Es un software de control de versiones, pensado en la eficiencia y la confiabilidad del mantenimiento de versiones de aplicaciones cuando estas tienen un gran número de archivos de condigo fuente. Su propósito es llevar el registro de los cambios en los archivos y coordinar el trabajo que varias personas realizan sobre estos archivos compartidos. Inicialmente fue diseñado como un núcleo de sistema de control de versiones de bajo nivel sobre el cual se podría implementar interfaces, sin embargo, desde entonces Git se ha convertido en un sistema de control de control de versiones completo con las siguientes características:

- Enfocado al desarrollo no lineal, lo que significa que facilita la gestión de ramas y la unión de diferentes versiones.
- La gestión es distribuida, debido a que cada programador tiene una copia del código fuente, y los cambios se propagan entre los repositorios locales.
- Los repositorios se pueden publicar por HTTP, FTP, rsync o mediante un protocolo nativo, y puede ser mediante una conexión TCP/IP siempre o cifrado SSH.

La alternativa más importante a esta tecnología es Subversion, que ha sido estudiada durante el grado en ingeniería informática, pero finalmente se ha decidido no utilizar por diferentes motivos.

Primero al tener al trabajar con repositorios centralizados Subversion complica la colaboración, aunque sí que es cierto que de momento no es un gran inconveniente debido a que no hay colaboración en el desarrollo del software, pero es uno de los objetivos es desarrollar una aplicación profesional que cumpla con los estándares empresariales de hoy en día, y ser apto para el trabajo en equipo es una característica interesante que aporta valor.

Por otra parte, el tener un repositorio central obliga a tener conexión para poder guardar cambios en este, por el contrario, Git es distribuido y por lo tanto se pueden realizar confirmaciones de forma local para mantener un control del código sin conexión, lo que resulta adecuado debido a que solamente hay un desarrollador trabajando en el proyecto y si en algún momento no hubiese conexión al repositorio remoto simplemente con guardarlo de forma local ya se podrían gestionar las versiones del código.

Gitlab y Gitlab CI/CD

Es una plataforma de desarrollo software colaborativo y control de versiones basado en Git. Para algunos aparatados de desarrollo en este proyecto son necesarias varias herramientas que están incluidas en la versión gratuita, también llamada Gitlab Community.

Para empezar, es necesario administrar el proyecto y para ello Gitlab tiene una utilidad avanzada para desarrollar ideas, resolver problemas y planificar el trabajo de forma colaborativa, aunque puede ser usada para otros propósitos si se personaliza de acorde a las necesidades y el flujo de trabajo.

En el ámbito del desarrollo de software profesional existen algunas herramientas como Jira o Trello para cubrir estas necesidades, y es que se tratan de herramientas que cuentan con más funcionalidades que Gitlab. A pesar de ello, esta es mucho más simple y sus características son más que suficientes para satisfacer ciertas necesidades del proyecto, ya que solamente se va a gestionar y planificar las tareas sin entrar en detalle.

En Gitlab las tareas estarían totalmente integradas con el repositorio del proyecto, es decir que tendría funcionalidades que facilitan la conexión entre las tareas y el código que se desarrolla o modifica en esas tareas, un ejemplo de esto sería la vinculación de ramas a una tarea para poder controlar el código fuente que se modifica durante y después del desarrollo de dicha tarea.

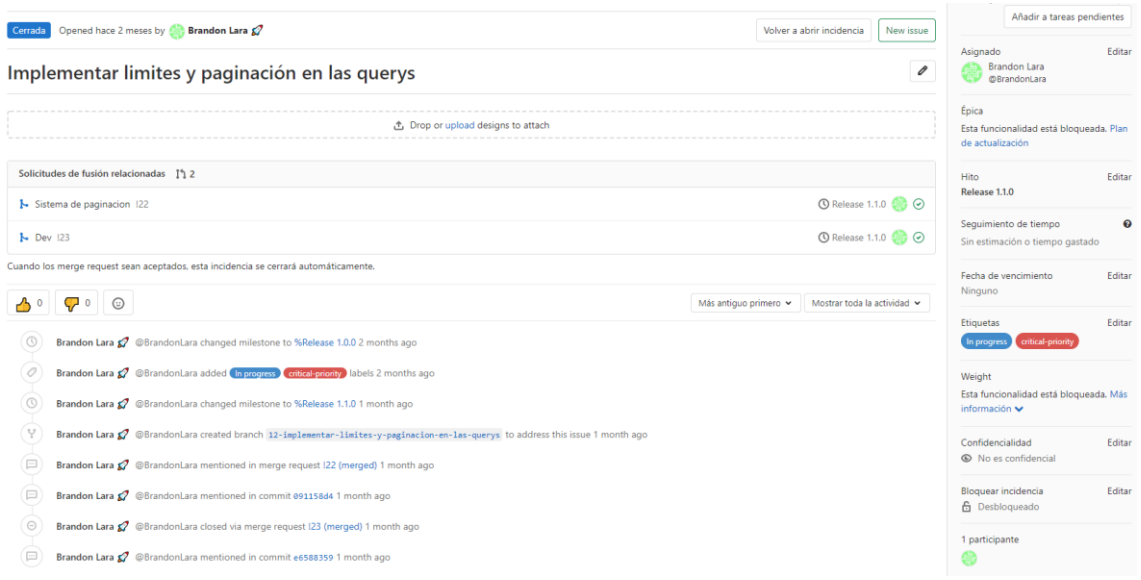


Figura 2.4: Ejemplo de una tarea en Gitlab.

Por otra parte, en Gitlab existe una herramienta llamada Gitlab CI/CD, para desarrollar software siguiendo metodologías continuas, como integración (CI), entrega (CD) y despliegues continuos (CD). Que es utilizada principalmente para la gestión de entornos, y el despliegue continuo de la aplicación en entorno de prueba para validar y posteriormente desplegar en producción cuando el software este en una versión estable.

Para ello se dispone de ejecutores compartidos, que son una especie de contenedores donde se ejecutan ciertos *pipelines* que contienen trabajos ordenados y gestionados por etapas, estos trabajos contienen todos los comandos necesarios para realizar cualquiera de las acciones que se requieran durante cada etapa, por ejemplo, en la etapa de despliegue existirán dos trabajos, uno para el despliegue en producción y otro para el entorno de pruebas, los cuales se diferencian por los comandos que ejecutan ya que en este ejemplo primero se mueven ficheros del software a la carpeta correspondiente de cada entorno debido a que no utilizan la misma.

Por otra parte, es necesario ejecutar el programa con diferentes configuraciones en función del entorno para el que se ejecuten, por lo tanto, los comandos que ejecuten estas acciones deben de ser diferentes en cada trabajo.

En este proyecto es necesario este tipo de utilidades debido a que se juntan las fases de desarrollo, pruebas y despliegue en una misma fase basada en las tareas que hay que realizar para llevar a cabo el sistema, y por lo tanto será imprescindible que cada tarea conste de mecanismos de integración, entrega y despliegue continuos. Esto es algo que GitHub no ofrece de forma gratuita.

SourceTree

Es una aplicación para simplificar y facilitar la interacción con repositorios de Git mediante una interfaz gráfica que permite visualizarlos y gestionarlos de diferentes maneras. Se trata de un cliente desarrollado por Altassian y distribuido de forma gratuita, a pesar de una de las interfaces graficas de Git más completas que existen, por esta razón se ha elegido esta opción por encima de otras como GitHub Desktop, pero teniendo esto en cuenta GitKraken sería una opción más que viable, sin embargo, para gestionar repositorios privados como en el caso de

este proyecto es necesario pagar la licencia de la aplicación, por lo tanto la mejor opción sigue siendo SourceTree.

Sequelize

Se trata de una herramienta de mapeo objeto-relacional para NodeJS basada en promesas de JavaScript que facilita el mapeo entre las estructuras de la base de datos relacional tradicional y el modelo de datos de la aplicación NodeJS. Esta herramienta es compatible con PosgreSQL, MySQL, MariaDB, SQLite, Microsoft SQL Server y permite realizar las operaciones CRUD abstrayendo la lógica de acceso a la base de datos y el procesamiento de los datos que se consultan. Por lo tanto, lo hace una herramienta ideal para agilizar el desarrollo de este proyecto, ya que solamente nos tenemos que ocupar del modelado de los datos en la aplicación.

Una alternativa a esta tecnología es TypeORM, que viene a cubrir la misma necesidad, sin embargo, Sequelize simplemente se clasifica como un traductor objeto – relación, mientras que TypeORM se considera como un 'microframework' del backend. Eso quiere decir que esta última tiene una infraestructura que va mucho más allá que la de un ORM, esto puede ser un inconveniente porque puede tener una curva de aprendizaje demasiado alta, debido a la mayor complejidad que supone el uso de un framework frente a un ORM. Por otra parte, el soporte de JavaScript que ofrece TypeORM no es tan bueno como el que puede ofrecer Sequelize.

Npm

Proviene de Node package mánager, y como su nombre indica se trata del gestor de paquetes de NodeJS por defecto, el cual es el registro de software más grande del mundo y donde muchos de los desarrolladores de código abierto comparten su software. En este proyecto lo utilizaremos para gestionar algunas de las tecnologías y librerías que se utilizan en este proyecto de una manera sencilla, y las dependencias que se tiene de estas, ya que con Npm es sencillo gestionarlas porque es capaz de instalar todas las librerías, y tecnologías que utiliza el proyecto con un solo comando, siempre y cuando las dependencias a este software se hayan guardado en el momento de la instalación.

JsDoc

Se trata de un lenguaje utilizado basado en etiquetas para documentar el código fuente de JavaScript similar a Javadoc, de esta manera los programadores pueden añadir comentarios en el fichero describiendo el código que se está desarrollando, para después escanear el código y generar la documentación en formato HTML.

APIDocJS

Se trata de un generador de documentación similar a JSDoc, pero totalmente enfocado a documentar las APIs de la aplicación. Su funcionamiento es igual que JSDoc, ya que genera la documentación en formato HTML a partir de los comentarios que escribe el desarrollador describiendo la funcionalidad mediante unas etiquetas específicas para este tipo de software.

Multer

Consiste en un middleware para NodeJS que se encarga de las peticiones multipart/form-data que habitualmente sirven para subir archivos al servidor. En nuestro caso se utiliza precisamente para subir los audios y fotos desde la red social al servidor y almacenarlos en el disco.

ESLint

Es una herramienta de análisis para identificar problemas de estilo y patrones a evitar en cualquier código ECMAScript/JavaScript, con el objetivo de evitar errores y hacer que el código sea coherente. La característica más interesante de ESLint es que es completamente configurable y modular, por lo que cada regla se trata de un módulo o complemento que se pueden añadir en el tiempo de ejecución, de esta manera según estas reglas se visualizarían los posibles problemas en el código instantáneamente.

3. Soundn: una plataforma social basada en audios

3.1 Propuesta

Este trabajo plantea la creación de una red social en la que todos sus aspectos están completamente enfocados al audio, ya que sería el único medio para intercambiar información o compartir contenido dentro de la plataforma, pero la solución que se quiere realizar no limita el uso de la plataforma a aquellos profesionales de la producción musical, la industria radiofónica, o del podcasting, si no que la intención es abarcar más casos de uso cotidianos o propios de una red social genérica, y convertirse en una plataforma en la que cualquier persona pueda realizar crearse un perfil y crear cualquier tipo publicación que se le venga a la mente, sin limitaciones, o incluso si se desea realizar contenido profesional, poder hacerlo y en el futuro poder cobrar por ello directamente desde la plataforma.

Una de las propuestas de valor es realizar esta red social multiplataforma, ya que se ha visto que en los últimos tiempos el uso de las redes sociales es muy alto en la mayoría de las plataformas que existen, de esta manera se intenta conseguir el mayor número de usuarios potenciales, y alcanzar el propósito de concentrar más casos de uso genéricos, también es cierto que las plataformas móviles es donde más se utilizan este tipo de aplicaciones, sin embargo existe una mayor segregación tecnológica, debido a la multitud de sistemas operativos móviles que se utilizan en la actualidad, es por ello que sería recomendable acotar a los dos sistemas operativos mayoritarios, o el más usando en el peor de los casos, y a partir de ahí expandirse al resto de plataformas.

Para conseguir todo lo anterior, gran parte de las funcionalidades de plataforma seguirán siendo las habituales de la mayoría de las redes sociales, para mantener de cierta manera el enfoque genérico de la red social, sin olvidar el audio como elemento principal de la plataforma. Por lo tanto, se trata de una alternativa a la mayoría de las redes sociales existentes porque la comunicación se realiza de una forma totalmente diferente, y se integran las funcionalidades totalmente centradas e ideadas para este tipo de comunicación mediante audios con aquellas que resultan más interesantes y que son propias de las redes sociales convencionales.

Sin embargo, actualmente existen diferentes plataformas que, a pesar de no tener la misma intencionalidad que la solución que se propone, sí que comparten varias similitudes con esta, para empezar, la idea de centrarse en el audio también se observa en otras plataformas como las que se han comentado en el capítulo 2 del Estado del arte. Además de esto, las funcionalidades centradas en el audio también se encuentran en todas estas plataformas.

La intención de no limitar ni focalizar la red social a la producción musical, o la creación de contenido, se trata de una manera original y novedosa de implementar la idea, en cualquier caso, la aplicación Limor se aproxima a lo que se desea obtener funcionalmente al finalizar este

proyecto, pero por diversas razones no ha funcionado correctamente, y muchos de los usuarios son personas que se dedican al podcasting.

3.2 Plan de trabajo

La planificación del trabajo de este proyecto se ha realizado de acorde a la metodología de trabajo que se utiliza, la cual se basa en el modelo de desarrollo en cascada tal y como se especifica en el capítulo 1.4, por lo tanto, la planificación se va a realizar por etapas y de manera secuencial para un solo desarrollador.

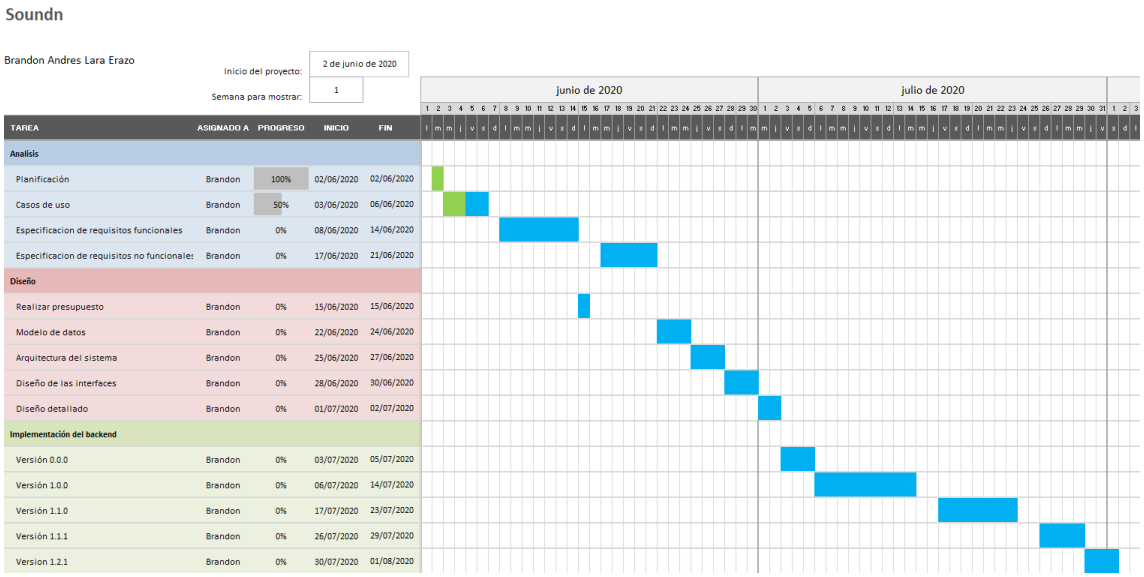


Figura 3.1: Planificación del proyecto.

De esta manera el proyecto empieza en 2 de junio con el análisis del software, el cual dura 21 días en los que se detallan los casos de uso de la aplicación para especificar los requisitos funcionales y no funcionales. La planificación de esta fase es relajada, y por esta razón entre cada tarea de la fase hay un periodo de tiempo en los que no se trabaja pero que sirven de colchón por si cualquiera de las tareas se alargase más de lo previsto.

Por su parte la fase de diseño empieza con la tarea de investigar el presupuesto del proyecto, y la cual al tratarse de una tarea pequeña se puede hacer durante los periodos de tiempo que no tienen una tarea asignada de la fase de análisis, por otro lado las otras dos tareas del diseño son más complejas y llevan más tiempo, por lo tanto se tienen que realizar después de terminar el análisis de la aplicación, pero no se establece ningún periodo de tiempo sin trabajo ya que no se tratan de tareas lo suficientemente grandes como para necesitarlo.

La fase de desarrollo de este proyecto se ha dividido en dos partes de acorde a la arquitectura general de la aplicación, una parte para todo lo relacionado con el servidor y el backend, y otra parte para el frontend de la aplicación. En estas circunstancias lo más adecuado es empezar por la implementación de la parte del backend y servidor, para después concluir con todas las tareas del frontend. Este orden es debido a que posiblemente las tareas del servidor sean la parte más crítica e indispensable, y, por lo tanto, el desarrollo y funcionamiento del resto de la aplicación depende de haber completado correctamente estar tareas.

El bloque de trabajo del backend se planifica en 5 versiones durante todo el mes de Julio, en estas versiones se realiza el incremento de la funcionalidad del software, sin embargo, la primera versión es la más corta porque se trata de una versión previa a la implementación, en la que se inicializa y prepara todo aquello necesario para el desarrollo del software.

De las 4 versiones restantes, 3 son versiones principales en las que se desarrollan principalmente nuevas funcionalidades, mientras que 1 versión se dedicara exclusivamente a la corrección de los fallos que no se detecten durante las pruebas. En cualquier caso, entre cada versión se planifican 2 días sin trabajos que sirven como colchón en caso de cualquier imprevisto durante el desarrollo, o como descanso si las cosas van bien.

Las aplicaciones móviles se tienen que desarrollar a un ritmo diferente, debido a que por una parte se implementa toda la funcionalidad, y por otra se diseña el aspecto visual que va a tener la aplicación, y se trabaja sobre su experiencia de usuario.

Por lo tanto, después de las vacaciones de agosto se ha planificado el desarrollo completo de las aplicaciones móviles empezando por toda la funcionalidad, y terminando en el aspecto visual de esta. A pesar de haber entregado todas las versiones del backend, también resulta importante planificar una versión de este para corregir todos los problemas que pudiesen surgir al integrar las dos partes del sistema durante el desarrollo del frontend o la conclusión del proyecto.

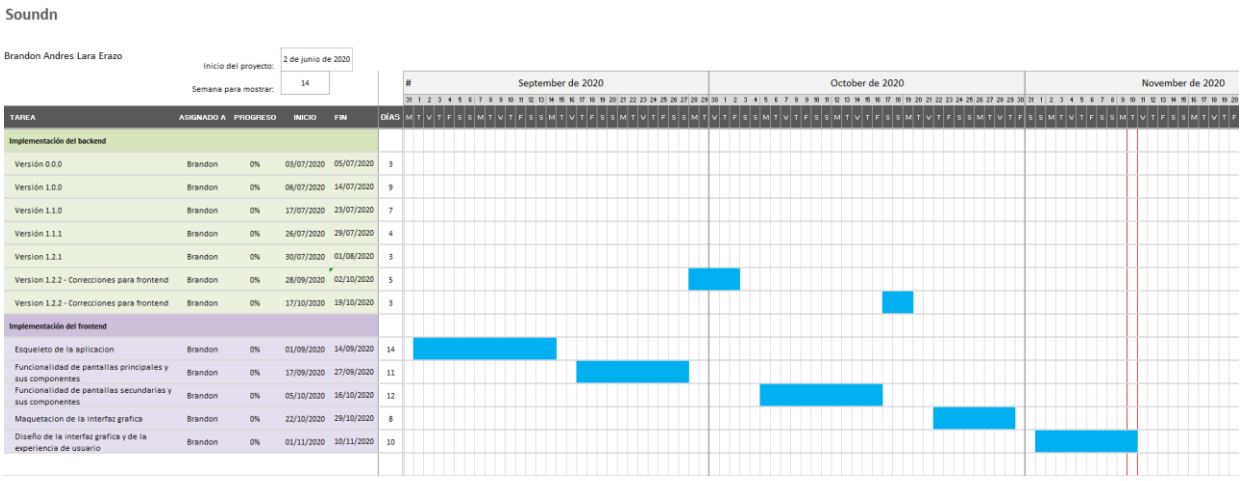


Figura 3.2: Diagrama de Gantt del frontend.

La primera etapa del desarrollo del frontend consiste en la organización de la aplicación y el desarrollo de su esqueleto, lo cual servirá como base para la implementación de toda la aplicación, al terminar esta tarea no se prevé que se necesiten correcciones sobre el servidor ya que solamente se llevan a cabo los aspectos más básicos, superficiales y tempranos de Soundn.

A continuación de esta etapa se ha planificado el inicio del desarrollo completo y funcional de las pantallas principales, y todos los componentes que las forman. Se estima que la duración de estas tareas sea de 11 días, entre la propia implementación y las pruebas de integración, que se realizan a medida que se va avanzando.

A diferencia de la etapa anterior, durante la implementación de las tareas de esta fase existe una mayor probabilidad de encontrar errores o problemas con el servidor y por esta razón se ha planeado un periodo de 5 días para realizar las correcciones necesarias en el backend. En el diagrama de Gantt de la Figura 3.1, se aprecia que estas correcciones se realizan a continuación de la etapa de pantallas principales, debido a las limitaciones técnicas del software para realizar el diagrama. Sin embargo, esto no es así porque las correcciones se realizarán a medida que se encuentren fallos o problemas durante la implementación de las tareas de la etapa, se realiza de esta manera, porque lo más probable es que estos fallos bloqueen el avance de toda la fase, y esta forma de trabajar nos evita contratiempos, en los resultados finales de las tareas, provocados por aplazar las correcciones, ya que los errores en la entrega de datos pueden derivar en una implementación errónea de las tareas del frontend.

De la misma manera se ha establecido, que a continuación, se realicen las pantallas secundarias de una forma muy similar, a como se había planteado el desarrollo de las principales, con la diferencia que esta etapa dure un poco más, debido a que la cantidad de pantallas secundarias de la aplicación es mayor que la de principales, en concreto de planifica que la duración de esta fase dure un total de 12 días.

Finalmente, después de haber realizado y probado todas las pantallas con sus funcionalidades, es necesario distribuir los elementos de la interfaz, de una forma apropiada para mejorar el aspecto visual en conjunto y la experiencia del usuario, por esta razón se ha planificado un total de 8 días, para diseñar e implementar la maquetación de las aplicaciones, y de esta manera proceder con el diseño de la interfaz gráfica y la experiencia de usuario durante 10 días.

Esta etapa se centrar solamente en realizar el trabajo visual de la aplicación, y por lo tanto no se modifica la lógica de la aplicación, ni se llevan a cabo tareas complejas que necesiten un periodos de pruebas para seguir avanzando, solamente se tiene que revisar que los cambios se visualizan correctamente en dispositivos móviles de diferentes tamaños, por esta razón no se ha alargado más la planificación de estas fases, tampoco se han programado en diferentes versiones, para realizar pruebas al final de cada una, ni se ha pensado en añadir fases de correcciones o pruebas.

3.3 Presupuesto

Para realizar todo el trabajo se cuenta solamente con una persona de perfil técnico, y por lo tanto se va a implicar completamente a todas y cada una de las fases que se han planificado excepto al diseño, de lo cual se hablará en capítulos posteriores. Por otra parte, para llevar a cabo este proyecto se ha optado por minimizar los costes de dinero, porque debido a las circunstancias, se trata de un recurso muy limitado, por lo tanto, las tecnologías y herramientas que se han utilizado, en su mayoría son de código abierto y permiten su uso de forma gratuita.

A pesar de esto, para poder cumplir con ciertas características del proyecto, se tienen que realizar varios gastos para contratar servicios indispensables, que no tienen una alternativa gratuita. El primer gasto consiste en una suscripción para utilizar un servidor privado virtual o VPS, que en este proyecto se utilizaría poder alojar el backend de la aplicación, antes que nada, hay que comparar, y analizar las plataformas más populares, para contratar los servicios más adecuados a las necesidades del proyecto.

Para realizar la comparativa, se tienen que establecer las características que necesita la instancia para alojar toda la infraestructura del proyecto. Por experiencia, se sabe que este tipo de software no suele consumir muchos recursos, por lo tanto, con las opciones más básicas de cada plataforma se podría seguir adelante. Existe una gran cantidad de plataformas que ofrecen estos servicios, pero las mejores opciones siempre van a ser aquellas más populares. Además, de esta forma se facilita la comparativa al acotar el número de plataformas que se comparan.

Tabla 3.1: Comparativa de los servicios en la nube de plataformas populares.

Características	Google Cloud Platform	Amazon Web Services	Digital Ocean	Microsoft Azure	OVHcloud	Vultr	A2 Hosting
Núcleos del procesador	1	2	1	1	1	1	1
Memoria RAM	4 GB	2GB	1 GB	1,75 GB	2 GB	1GB	512 MB
Cantidad de almacenamiento	20GB	40GB	25 GB	70 GB	40GB	25 GB	20 GB
Ancho de banda	-	500 MB/s	-	-	250 MB/s	-	-
Límite de trafico	-	-	1TB	-	Ilimitado	1 TB	2 TB
Precio al mes	16,88 €	7,69€	4,24 €	7,43 €	4,6 €	4,24 €	4,25 €

Se ha determinado que opción *value* de OVH, se trata de un servicio con una buena relación entre los recursos que ofrece, y su precio, ya que ofrece por 4,60 euros al mes, una instancia con un procesador de 1 núcleo, 2 gigabytes de memoria RAM, 40 gigabytes de almacenamiento, un ancho de banda de 250 Mb/s y varios servicios adicionales, incluidos[6] con este mismo precio.

Para terminar, hay otro coste obligatorio en este proyecto para tener en cuenta, y se trata de la tarifa de 20 euros para poder publicar la aplicación en la Play Store de Google. Entonces, si se tiene en cuenta que el desarrollo del proyecto va a durar alrededor de seis meses, hay que sumar el coste del servidor durante los seis meses, al coste de subir la aplicación a la tienda. Por lo tanto, el presupuesto total de este proyecto es de 47,6 euros, que se desglosan en 27,6 euros por los seis meses de servidor, teniendo en cuenta que cada mes cuesta 4,6 euros, y a esto finalmente, se le suman los 20 euros de subir la aplicación a la tienda de aplicaciones de Android.

[6] Características de los servicios que ofrece OVH https://www.ovhcloud.com/es-es/vps/compare/

4. Análisis

Para las primeras versiones de esta red social, se implementarán solamente las funcionalidades habituales y esenciales de este tipo de software, seleccionando las que mejor se adapten a las particularidades del sistema. Esto es así para poder cumplir con los plazos establecidos, ya que las redes sociales hoy en día tienen una gran cantidad y variedad de funcionalidades interesantes, y llevarlas a cabo requeriría de un esfuerzo que no se puede asumir de momento. Por otra parte, también se busca diferenciarse de lo que existe actualmente, e innovar a partir de un enfoque diferente a las redes sociales tradicionales.

Lo primero que se debe de tener en cuenta en una aplicación de este estilo, es un sistema de cuentas de usuario, para poder autenticarse en el sistema y recibir autorización para acceder al resto de la aplicación, por esta razón al usuario se le debe proporcionar la manera de registrarse e iniciar sesión en el sistema. Este punto está estrechamente ligado con otra característica fundamental de las redes sociales, como son los perfiles de usuario, y es que resulta imprescindible que la aplicación cuente con esta característica, para mostrar la información asociada a cada usuario, personalizar su experiencia en la aplicación, y facilitar la comunicación y conexión entre los diferentes usuarios registrados en la aplicación. Como consecuencia de la existencia de perfiles de usuario, también debería poderse editar la información que se muestra en estos perfiles, pudiendo así otorgar flexibilidad al usuario a la hora de personalizar su perfil.

Otra característica importante de la plataforma, son las publicaciones que van a realizar los usuarios, las cuales se tratan de uno de los rasgos diferenciadores sobre el resto de las plataformas, debido a que estas solamente pueden contener audio, son simples y personalizables. Además, estas publicaciones tienen la funcionalidad estándar que tendrían publicaciones de otras plataformas, pero enfocadas al mismo audio, como, por ejemplo, las respuestas a una publicación también serían de este tipo. Las publicaciones se crearían de una forma sencilla, grabando o subiendo el audio, subiendo la imagen de fondo del reproductor, y estableciendo unas etiquetas identificativas de la aplicación. Al simplificar la creación de las publicaciones se incita a que su contenido también pueda ser genérico, o cotidiano, como el del resto de redes sociales. La idea es que las publicaciones sean reproductores de audio personalizados, que estén acompañados con datos del usuario, y botones para interactuar con la publicación. Habitualmente estas publicaciones estarían dispuestas en listados con diferentes filtros en función de lo que se necesite mostrar.

En el caso de la visualización principal, el listado debería mostrar todas las publicaciones que han realizado los usuarios que se están siguiendo en ese momento. Además, lo ideal es que este listado se ordene por la fecha de creación de la publicación, ya que resulta una forma de organizar la información más interesante para el usuario.

Tal y como se ha mencionado anteriormente, se va a poder interactuar con las publicaciones que aparezcan, y su contenido. En un principio, para cumplir con la funcionalidad básica de las redes sociales, se ha pensado que las formas de interacción sean las que se especifican a continuación:

- **Respondiendo** a una publicación mediante otra publicación que sirva de respuesta, es decir que se podrán realizar publicaciones de audio que sirvan como respuesta a una

publicación de audio anterior, por lo tanto, cada publicación tendrá su propia sección de respuestas con un listado de las respuestas de audio que se han realizado.

- **Reaccionando** a una publicación, mediante diferentes emoticonos que reflejen el sentimiento que ha provocado la publicación.
- **Compartiendo** la publicación en la propia aplicación, esto significa que dentro de la aplicación se va a poder compartir internamente cualquier publicación para que las puedan reproducir los seguidores de la persona que la comparte.
- **Comentando** una publicación, esto significa que se pueden realizar pequeños comentarios de texto en cualquier audio, y por lo tanto cada publicación contendrá su propio apartado con su listado comentarios

Una red social, por definición, se compone por usuarios relacionados entre sí en función de diferentes características o criterios, esto significa que es estrictamente necesario, que todos los usuarios de la plataforma puedan conectarse entre ellos de alguna manera. Entonces para satisfacer esta característica sería necesario implementar un sistema de seguidores, para que los usuarios puedan seguirse entre ellos, e interactuar fácilmente unos con otros.

Se ha optado por esta solución basada en seguidores, porque es la que mejor se adapta la plataforma que se tiene en mente desde un principio, debido a que se trata de una solución ampliamente utilizada en redes sociales enfocadas a un público general, pero a su vez se trata de un punto fuerte para los creadores de contenido, que puedan utilizar la plataforma de forma profesional o seria, ya que cualquier usuario podría conectar con ellos y llegar a un número mayor de personas.

Otra de las funcionalidades que se heredan de las redes sociales son las notificaciones, ya que con estas el usuario puede estar al día de las interacciones, que el resto de los usuarios tiene con él, y las acciones que realizan los usuarios a los que sigue. En primera instancia, al usuario se le notificaría cuando se realice cualquiera de las interacciones antes descritas sobre sus publicaciones, o cuando un usuario comience a seguirle. Además, también se le notificaría cuando un usuario al que está siguiendo realiza una de estas interacciones, o cuando comienza a seguir a otro usuario.

Para fomentar el descubrimiento de publicaciones interesantes, o de usuarios que las realicen, se ha ideado una forma de explorar las publicaciones, cuyas etiquetas están teniendo más repercusión en la plataforma, para ello se mostrarían las etiquetas más utilizadas recientemente, y las publicaciones que contienen estas etiquetas.

Por último, resulta interesante que los usuarios dispongan de un buscador para que puedan localizar las publicaciones y/u otros usuarios. En el caso de las publicaciones, la búsqueda se realizaría por sus etiquetas, y se mostraría en un listado de publicaciones, igual que en el resto de la aplicación. Por otro lado, los usuarios se buscarían por su nombre de usuario, ya que, al ser único, facilitaría la búsqueda de usuarios específicos.

4.1 Requisitos funcionales

Los requisitos funcionales describen el comportamiento del sistema, y las funcionalidades que debe proporcionar este para realizar las acciones requeridas, se trata de una descripción detallada de las funciones, servicios y restricciones del sistema.

La especificación de requisitos funcional debe ser precisa, y definir exactamente qué es lo que se va a implementar, en definitiva, estos requisitos se tratan de declaraciones de los servicios que debe proporcionar el sistema, es decir, cómo se debe comportar en situaciones particulares, y qué respuestas debe dar a determinadas entradas. En algunos casos, estos requisitos pueden declarar explícitamente lo que el sistema no debe hacer. [2]

Apoyándose en la descripción general previa, en este caso se han podido extraer los siguientes requisitos funcionales del sistema:

Tabla 4.1: Requisito funcional de inicio de sesión.

Código	RF-01
Nombre	Iniciar sesión
Dependencias	
Descripción	Se debe poder acceder a la aplicación mediante un acceso autenticado por nombre de usuario/correo electrónico y contraseña.
Importancia	Esencial
Restricciones	

Este requisito provee de un punto de entrada a la aplicación en caso de que el usuario no se haya autenticado todavía, por lo tanto, va a ser lo primero que va a ver un usuario sin identificar en la plataforma, por otra parte, aunque parezca que el desarrollo del resto de funcionalidades dependa de este requisito, en realidad no es necesario que exista el inicio de sesión. Finalmente, se trata de una funcionalidad esencial por la naturaleza de las redes sociales, además también es necesario para cumplir con los requisitos de seguridad que se especifican en el capítulo 4.2.

Tabla 4.2: Requisito funcional de registro de usuario.

Código	RF-02
Nombre	Registro de usuario
Dependencias	
Descripción	El sistema debe permitir el registro de usuarios mediante formulario de registro, el acceso a esta funcionalidad debe de ser a partir del inicio de sesión, estas dos funcionalidades están relacionadas, por lo tanto, siempre se debe de poder acceder a una de ellas desde la otra.
Importancia	Esencial
Restricciones	En el formulario de registro debe haber diferentes campos obligatorios, para poder rellenar cierta información esencial para el correcto funcionamiento de otras funcionalidades, como por ejemplo el inicio de sesión, los campos obligatorios son: • Nombre de usuario • Alias (Nombre a mostrar) • Contraseña • Correo electrónico

Este requisito existe para dar la posibilidad, a cualquier usuario, de registrarse en la red social. Además, esta funcionalidad está muy relacionada con el inicio de sesión, y en este caso en el mismo lugar de inicio de sesión se va a poder acceder al registro del usuario, pero a pesar de esto no existe ninguna dependencia con la funcionalidad de inicio de sesión. Por motivos evidentes, se trata de un requisito fundamental de la aplicación, el cual tiene restricciones de obligatoriedad en ciertos campos del formulario, ya que hay ciertos datos que deben de ser obligatorios para que la cuenta de usuario se cree correctamente.

Tabla 4.3: Requisito funcional de datos del perfil.

Código	RF-03
Nombre	Datos de perfil
Dependencias	RF-02
Descripción	Al terminar de registrarse, el usuario debe de poder rellenar la información no esencial de su perfil mediante otra pantalla, la cual podrá omitir si lo desea. La información que de momento se va a poder introducir en el sistema será: • Foto de perfil • Descripción • Alias
Importancia	Media
Restricciones	

Tabla 4.4: Requisito funcional del perfil de usuario.

Código	RF-04
Nombre	Perfil de usuario
Dependencias	Ninguna
Descripción	Cada usuario va a tener un perfil en el que se tiene que mostrar los siguientes datos del usuario: • El nombre de usuario. • Su alías. • La descripción. • Su foto de perfil. • Sus publicaciones. • El audio de perfil • Las publicaciones que han compartido.
Importancia	Esencial
Restricciones	Ninguna

Este perfil de usuario muestra diferentes datos públicos que el usuario decida, se trata de una tarea esencial, ya que es una de las características principales de las redes sociales, además, esta tarea no depende de ninguna de las otras funcionalidades o tareas que se tienen que desarrollar.

Tabla 4.5: Requisito funcional de editar datos del usuario.

Código	RF-05
Nombre	Editar datos de usuario
Dependencias	RF-04
Descripción	La aplicación debe permitir que cada usuario pueda editar su nombre de usuario, alias, descripción, foto de perfil y audio de perfil
Importancia	Esencial
Restricciones	El usuario podrá editar sus datos solamente desde su perfil de usuario.

Con esta funcionalidad el usuario va a poder editar sus datos públicos, su imagen de perfil y su audio de perfil, para ello cada usuario desde su propio perfil va a poder modificar los datos, lo

que significa que, para poder llevar a cabo esta funcionalidad, es necesario que previamente se implemente el perfil de usuario.

Tabla 4.6: Requisito funcional de crear publicaciones.

Código	RF-06
Nombre	Crear publicaciones
Dependencias	
Descripción	Se va a disponer de un sistema de creación de publicaciones, en el cual los usuarios van a poder grabar el contenido, o subirlo directamente desde su sistema de almacenamiento. Por otra parte, también van a poder escoger el estilo del fondo de la publicación, de entre una serie de fondos por defecto, o fondos que suban desde su galería. Finalmente, en este sistema de creación va a ser posible añadir etiquetas escritas por el usuario, las cuales se van a utilizar para buscar la publicación.
Importancia	Esencial
Restricciones	

En la aplicación se van a poder crear publicaciones, para que los usuarios puedan interactuar con esta, y compartir todo tipo de contenido. En este proceso se establecen las etiquetas, el contenido y el fondo de la publicación. El desarrollo de este requisito no depende de ningún otro, además, el contenido principal de la red social está basado en estas publicaciones, por lo tanto, dentro de la aplicación este requisito es esencial, y obligatorio implementarlo.

Tabla 4.7: Requisito funcional de borrar publicaciones.

Código	RF-07
Nombre	Borrar publicaciones
Dependencias	RF-06 para realizar pruebas.
Descripción	El sistema va a permitir a cada usuario borrar sus propias publicaciones.
Importancia	Alta
Restricciones	

De la misma forma que un usuario puede crear publicaciones, este mismo usuario va a poder borrar sus propias publicaciones, aunque no sea una funcionalidad fundamental del sistema sí que tiene una importancia alta, porque los usuarios pueden llegar a necesitar borrar una publicación por cualquier motivo. Para realizar las pruebas de una forma sencilla tal vez sería interesante implementar antes el requisito de creación de publicaciones.

Tabla 4.8: Requisito funcional de compartir publicaciones.

Código	RF-08
Nombre	Compartir publicaciones
Dependencias	RF-06
Descripción	Se va a permitir que los usuarios puedan compartir las publicaciones que deseen, de forma interna o externa, es decir que se van a poder compartir dentro de la propia aplicación, o en aplicaciones de terceros.
Importancia	Solo es esencial compartir internamente, externamente se implementará en futuras versiones.
Restricciones	

Tabla 4.9: Requisito funcional de reaccionar a las publicaciones.

Código	RF-09
Nombre	Reaccionar a las publicaciones.
Dependencias	RF-06
Descripción	Se va a permitir a los usuarios reaccionar a una publicación mediante emojis, para ello se va a disponer de un botón, en el cual van a poder seleccionar un emoji que se va a mostrar en la propia publicación, junto con un contador de la gente que ha reaccionado con un mismo emoji.
Importancia	Esencial
Restricciones	

Tabla 4.10: Requisito funcional de comentar en las publicaciones.

Código	RF-10
Nombre	Comentar en las publicaciones
Dependencias	RF-06
Descripción	Se van a poder realizar comentarios de texto sobre una publicación determinada.
Importancia	Esencial
Restricciones	Sobre los comentarios de texto no se va a poder interactuar de ninguna forma, ni reacciones, ni comentarios, ni se van a poder compartir.

Tabla 4.11: Requisito funcional de responder a las publicaciones.

Código	RF-11
Nombre	Responder a las publicaciones
Dependencias	RF-06
Descripción	Se debe poder responder a las publicaciones mediante otra publicación, con todas sus características y contenido disponibles, lo que significa que como cualquier otra publicación también tendrán: • Contenido de audio. • Imagen de fondo. • La opción de compartir. • La posibilidad de reaccionar. • Comentarios y respuestas.
Importancia	Esencial
Restricciones	Las publicaciones respuesta tendrán exactamente los mismos elementos y funcionalidad que una publicación normal. Todas estas funcionalidades se especifican desde el RF-07 al RF-11

Los cuatro requisitos anteriores consisten en las diferentes maneras que los usuarios pueden interactuar con las publicaciones de otros usuarios, por lo tanto, para poder realizar las pruebas con cierta facilidad, lo mejor es terminar antes la funcionalidad de crear publicaciones. Además, estas cuatro funcionalidades tienen la prioridad más alta, lo que quiere decir que son realmente fundamentales para la aplicación, ya que se tratan de características importantes de las redes sociales. Sin embargo, el requisito de compartir una publicación es esencial solo al hacerlo de forma interna, lo que significa que la funcionalidad de compartir externamente en otras plataformas, no es necesario llevarla a cabo, porque no se considera que proporcione el valor suficiente para las primeras versiones de la aplicación. El único requisito funcional que tiene una restricción es el de realizar comentarios de texto sobre las publicaciones, ya que estos solamente pueden tener texto y no van a poder tener ningún tipo de interacción.

Tabla 4.12: Requisito funcional de seguir a otros usuarios.

Código	RF-12
Nombre	Seguir a otros usuarios
Dependencias	RF-1
Descripción	El sistema va a permitir seguir a otros usuarios, para poder visualizar sus publicaciones en el timeline.
Importancia	Esencial
Restricciones	Los usuarios no van a poder seguirse a sí mismos

Tabla 4.13: Requisito funcional de buscar publicaciones.

Código	RF-13
Nombre	Buscar publicaciones
Dependencias	RF-06
Descripción	Se va a disponer de un sistema de búsqueda en el cual se podrá escribir el nombre de una etiqueta para encontrar todas las publicaciones que tengan una etiqueta asociada con ese nombre.
Importancia	Alta
Restricciones	No puede haber búsquedas vacías. Solamente se puede buscar por una etiqueta a la vez Es necesario que las publicaciones tengan la etiqueta asociada para que aparezcan en la búsqueda

Habitualmente este tipo de aplicaciones tienen diferentes sistemas de búsqueda, en un principio se ha pensado que lo mejor sería implementar la búsqueda de publicaciones en función de sus etiquetas, sin embargo, en futuras versiones este sistema se ampliaría para dar la posibilidad de realizar diferentes búsquedas sobre otras entidades de la aplicación. Este requisito solamente depende de la funcionalidad de crear publicaciones ya que las búsquedas se realizan sobre estas.

Tabla 4.14: Requisito funcional de línea temporal de publicaciones.

Código	RF-14
Nombre	Línea temporal de publicaciones principal
Dependencias	RF-06
Descripción	Esta línea temporal principal es la que se va a visualizar en la página principal de la aplicación, y muestra las publicaciones aquellos usuarios a los que el usuario actual sigue, en formato de listado y ordenadas descendentemente por fecha de creación.
Importancia	Esencial
Restricciones	

La aplicación debe de tener un listado de las publicaciones ordenadas por fecha de creación, asimismo este listado es lo primero que se mostraría a los usuarios después de iniciar sesión, o acceder a la aplicación con la sesión iniciada, lo que quiere decir que es el componente más importante de la aplicación, ya que se trata del elemento fundamental de la pantalla principal de la aplicación. Esta funcionalidad solamente depende de la creación de publicaciones para poder realizar pruebas fácilmente.

Tabla 4.15: Requisito funcional de notificaciones personalizadas.

Código	RF-15
Nombre	Notificaciones personalizadas
Dependencias	Depende de las todas las funcionalidades que crean notificaciones para poder probarla correctamente, aunque se pueden realizar pruebas con notificaciones falsas simplemente para visualizar.
Descripción	Al usuario se le deben notificar acciones que realicen los usuarios a los que sigue, o las interacciones que se realicen sobre él o alguna de sus publicaciones, por lo tanto, en la aplicación tienen que existir diferentes tipos de notificaciones en función de la acción que la genera, de esta forma solamente se pueden generar notificaciones de los siguientes tipos: • Nuevo seguidor, se genera para un usuario concreto cuando otro le sigue. • Comentario en una publicación, que se crea cuando un usuario comenta en una publicación, en este caso la notificación va dirigida al usuario que ha creado la publicación. • Respuesta en una publicación, es una notificación que se dirige al creador de una publicación cuando otro usuario realiza una publicación en respuesta a la suya. • Reacción, el creador de esta recibe la notificación de este tipo cuando se reacciona a alguna de sus publicaciones. • Publicación compartida, cuando un usuario comparte una de las publicaciones se crea una notificación para el usuario que ha creado la publicación. • Seguimiento, cuando un usuario sigue a otro usuario se genera una notificación dirigida a todos sus seguidores informando de este hecho. • Creación de publicación, cuando un usuario realiza una publicación se envía una notificación de este tipo a todos sus seguidores.
Importancia	Alta
Restricciones	Ninguna

La aplicación va a notificar al usuario de diferentes hechos que ocurran dentro de la aplicación, y que estén relacionados de una forma u otra con este usuario. Por otra parte, la importancia de esta funcionalidad no es la más alta que se le puede asignar, pero sí que resulta interesante llevarla a cabo en las primeras versiones debido a que es de gran ayuda a la hora de retener a los usuarios, lo cual es realmente importante para cualquier red social.

Tabla 4.16: Requisito funcional de visualizar notificaciones.

Código	RF-16
Nombre	Visualizar notificaciones
Dependencias	Seguir a un usuario Notificaciones
Descripción	Se van a poder visualizar todas las notificaciones que se han recibido en un formato de lista ordenada por fecha de creación empezando por la más reciente.
Importancia	Media
Restricciones	

Este requisito consiste en mostrar un listado de las notificaciones ordenadas por fecha de creación. Se trata de un requerimiento con un nivel de importancia medio ya que ni fundamental, ni completamente necesaria en las primeras versiones de la aplicación.

Tabla 4.17: Requisito funcional dejar de seguir a un usuario

Código	RF-17
Nombre	Dejar de seguir a un usuario
Dependencias	
Descripción	Se debe poder seguir a cualquier usuario registrado en la aplicación.
Importancia	Esencial
Restricciones	Solamente se va a poder dejar de seguir a aquellos usuarios que previamente se estaban siguiendo.

De la misma manera que los usuarios van a poder seguir otros usuarios, también es necesario realizar la acción inversa, es decir, la posibilidad de dejar de seguir a aquellos usuarios que en

algún momento se siguieron. La importancia de este requisito es la más alta, lo que significa que es fundamental, y necesario para el funcionamiento completo de las primeras versiones de la aplicación.

Tabla 4.18: Requisito funcional de actualizar los listados.

Código	RF-18
Nombre	Actualizar listados
Dependencias	Funcionalidades basadas en listados
Descripción	Todos los listados de la aplicación se deben poder actualizar manualmente, y automáticamente cada vez que se navegue a la pantalla que contenga el listado.
Importancia	Alta
Restricciones	Ninguna

Varias funcionalidades de la aplicación se basan en listados, los cuales se deberían de poder actualizar manualmente de una manera sencilla. Además, también es interesante que estos listados se actualicen automáticamente cada vez que se muestren al usuario. Esta funcionalidad depende de aquellas que requieran realizar listados, y no tiene restricciones relevantes. El objetivo de este requisito es proporcionar una buena experiencia de usuario durante la navegación por los listados de la aplicación.

Tabla 4.19: Requisito funcional de cerrar sesión.

Código	RF-19
Nombre	Cerrar sesión
Dependencias	RF-01
Descripción	Los usuarios van a poder cerrar la sesión de su cuenta en la aplicación, para así poder volver a la pantalla de inicio de sesión, o de registro.
Importancia	Alta
Restricciones	Al cerrar sesión se revoca el acceso a la aplicación, y solamente se podrá registrarse en la aplicación, o iniciar sesión de nuevo.

En último lugar, se proporciona a los usuarios una forma de cerrar sesión de la aplicación para acceder al inicio de sesión o al registro, no tiene la importancia más alta, ya que se puede implementar en las versiones más avanzadas, sin suponer un inconveniente mayor para los usuarios, pero es muy interesante dar esta posibilidad a los usuarios, para que la red social resulte más interesante a los usuarios activos.

La única restricción es que, al realizar el cierre de sesión, se revoca la autorización de esta, y el usuario no va a poder acceder a ningún recurso de la aplicación, y por motivos de seguridad, tampoco se van a poder realizar operaciones en base de datos mediante la API. En estas circunstancias, los usuarios solamente van a poder iniciar sesión de nuevo, o registrarse en la aplicación. Como es natural, este requisito depende completamente de que se termine previamente la funcionalidad de inicio de sesión.

4.2 Requisitos no funcionales

Los requisitos no funcionales, o atributos de calidad se tratan de requisitos que especifican criterios que pueden usarse para juzgar la operación de un sistema, en lugar de sus comportamientos específicos como seria en el caso de los requisitos funcionales. Aunque, calidad del software o su eficiencia o capacidad de mantenimiento son más difíciles de evaluar y solo pueden medirse indirectamente [5].

En este sentido la red social debe tener ciertas características de calidad, para asegurar que, de forma interna se cumplen ciertos estándares del desarrollo de aplicaciones, también es importante para dar una buena imagen de cara al usuario final, ya que habitualmente hay ciertos aspectos que estos usuarios tienen muy en cuenta a la hora de valorar una aplicación. Para cumplir con todo esto, y con 3 de los principales objetivos de seguridad de [6], en los siguientes apartados se han especificado los requerimientos no funcionales de la red social.

Escalabilidad

Al tratarse de una red social, el software que se desarrolle debe de ser fácilmente escalable, ya que el crecimiento en el número de usuarios afectaría de forma negativa algunas de las características del sistema, y a la experiencia del propio usuario. Esto cobra más importancia si se tiene en cuenta que las redes sociales necesitan de usuarios para ser atractiva, y llamar la atención de nuevos usuarios, retroalimentándose constantemente. Por lo tanto, uno de los objetivos siempre va a ser captar el mayor número de usuarios, y por esta razón el sistema debe estar preparado ante el posible crecimiento de la plataforma.

Para cumplir con este requisito, lo que se debe de hacer en primer lugar, es optar por realizar el despliegue sobre servidores en los que sea posible, cambiar sus recursos en función de la demanda de la aplicación, de esta manera se descarta la construcción de un propio hardware servidor.

Por parte del software es necesario que se utilice una plataforma de desarrollo altamente escalable, y la cual sea fácilmente separable para desplegarse en diferentes instancias, ya que en un principio, habrá un número de usuarios lo suficientemente bajo como para poder desplegar

sobre la misma instancia, los diferentes entornos del servidor de la aplicación, y la base de datos: Pero a medida que el número de usuarios va aumentando, una técnica de escalabilidad es dividir los elementos del backend, y desplegarlos maquinas separadas, para mantener el rendimiento a pesar del aumento de peticiones.

Finalmente, como bien se ha dicho previamente, la implantación de la plataforma debería realizarse mediante servicios que permitan la escalabilidad vertical, es decir aumentar la potencia de la instancia añadiendo más núcleos de cómputo y más memoria, esto implica utilizar servicios en la nube que permitan cambiar la configuración en cualquier momento.

Rendimiento y eficiencia

El rendimiento y la eficiencia se trata de la capacidad de respuesta del sistema, para ejecutar una acción dentro de un intervalo de tiempo determinado, y en unas condiciones particulares de trabajo. Se trata de un requisito indispensable en este tipo de aplicación, ya que la mayoría de sus funcionalidades dependen del tiempo de respuesta de sistema. Por eso se realizó un análisis y se comparó el rendimiento de lenguajes del lado del servidor, como NodeJS, Python, y PHP [7] , donde se concluyó que NodeJS es una muy buena herramienta para realizar interacciones en tiempo real, donde destaca por realizar las comunicaciones, enviando y recibiendo datos entre cliente y servidor de una manera muy rápida.

Es fundamental para alcanzar un buen nivel de rendimiento de un nuevo sistema, que los esfuerzos se enfoquen por cumplir este requisito desde el inicio del desarrollo, y se amplíen durante su construcción, por lo tanto para asegurarse de que se cumple este requisito, después de finalizar cada tarea, además de realizar pruebas sobre la funcionalidad, también se tienen que llevar a cabo diferentes tipos de validaciones enfocadas al cumplimiento este requisito, lo cual permitiría mantener el rendimiento del sistema, y detectar rápidamente problemas de rendimiento, para solucionarlos durante el propio desarrollo de la plataforma.

Estas pruebas deben de tener presente que el rendimiento es importante en esta aplicación, y para ello se ha definido que el tiempo de respuesta máximo de cualquier ruta de la API sea de 300 milisegundos, sin tener en cuenta la latencia de los proveedores. Se ha definido así porque se trata de un tiempo que todavía resulta inapreciable para el usuario, y es algo que se puede conseguir de una manera sencilla si se implementa bien el software del servidor.

Por parte de la aplicación móvil, se debe comprobar que su uso del almacenamiento del dispositivo no es mayor a 100 Mb, y el uso de memoria por su parte no debe de superar los 300 Mb para poder cumplir con los requisitos de eficiencia.

En el caso de las pruebas de estrés, se debe validar que, con un número considerable de peticiones, en un corto periodo de tiempo, el incremento del tiempo de respuesta de cualquier petición no sea mayor al 50%, porque si se cumple el requisito previo. En el peor de los casos el tiempo de respuesta sería de 450 milisegundos, lo cual sigue siendo un tiempo difícil de apreciar por el usuario, y completamente asumible en la implementación del software. De esta manera, además, se validan los límites que el sistema puede soportar antes de que caiga el sistema, o que el tiempo de respuesta sea tan alto como para ralentizar cualquier funcionalidad de la aplicación.

Por otra parte, en las pruebas de estabilidad se comprobará que se pueden mantener constantes los tiempos de respuesta, el uso de la memoria RAM y de la CPU, durante más de 1 hora de

carga alta, simulando un uso de más de 5 peticiones de diferente naturaleza, por segundo. Estas pueden ser escogidas al azar de entre aquellas que sean más demandantes de recursos hardware.

Se ha escogido este número de peticiones por segundo porque está por encima de las proyecciones de uso de la red social a corto plazo.

Para concluir, hay que mencionar que el objetivo de esta prueba es comprobar, que el hardware puede aguantar un alto uso de procesador y memoria RAM, sin bajar de manera excesiva el número de peticiones que puede servir en un segundo, o su latencia. Por otra parte, también resulta interesante que la aplicación sea capaz de aguantar este escenario extremo, sin provocar errores o malos usos derivados de la carga a la que está sometido el sistema, o la posible lentitud de las comunicaciones e interacciones.

Disponibilidad

La disponibilidad es una de las características de las arquitecturas empresariales, que mide el grado con el que los recursos del sistema están disponibles para su uso, a lo largo de un tiempo dado. Disponibilidad se refiere a la habilidad del conjunto de usuarios para acceder al sistema, someter nuevos trabajos, actualizar o alterar trabajos existentes, o recoger los resultados de trabajos previos. Si un usuario no puede acceder al sistema, se dice que está no disponible. El término tiempo de inactividad, *(downtime)* es usado para definir cuándo el sistema no está disponible. [8]

La red social debe permitir a los usuarios acceder en cualquier momento, debido a la naturaleza de este tipo de aplicaciones. Para ello, se debería establecer un alto porcentaje de disponibilidad del servidor de la plataforma, para medir el tiempo de funcionamiento de la aplicación. Este porcentaje se obtiene de dividir el tiempo total que ha pasado, entre el tiempo de inactividad del sistema.

Como tiempo de inactividad, solamente se contarán aquellos provocados por el software que se desarrolla, ya que cualquier fallo provocado por la instancia en la que se aloja el servidor, queda fuera del alcance de este proyecto, debido a que se trata de un servidor contratado a un proveedor externo. Por otro lado, los tiempos de inactividad provocados por pasos programados sí que se van a contar, ya que en teoría se deben de utilizar tecnologías que permitan subidas de versión sin caídas.

En definitiva, el porcentaje de disponibilidad no debe de bajar del 99.90%, ya que significaría una disponibilidad del sistema lo suficientemente alta como para cumplir con el requisito, para hacerse una idea de lo que supone cumplir con el porcentaje de disponibilidad, si este se cumple se asegura que al mes hay menos de 44 minutos en los que el servicio no está disponible, lo cual es una cifra muy buena para la plataforma que se desea construir.

Confidencialidad y seguridad

La información que provee el sistema debe de estar protegida ante acceso no autorizado, este requisito es crítico porque a través de la API se accede a la información almacenada en las bases de datos, por lo tanto, es muy importante que esta API solamente proporcione cierta información a los usuarios que se han identificado en la aplicación.

Para ello, el acceso a la API de la aplicación debe de estar debidamente protegido mediante middleware de autenticación y autorización. Por otro lado, se requiere evitar la divulgación de información confidencial o sensible a las personas o sistemas no autorizados, teniendo esto en cuenta, la confidencialidad de estos datos se puede conseguir mediante el cifrado y descifrado de estos [9] , por ello se plantea la encriptación de las contraseñas, y correos electrónicos de los usuarios.

Finalmente, las comunicaciones con la API se tienen que hacer a través del protocolo HTTPS [10] , ya que de esta forma se logra que la transferencia de datos sea segura, debido a que realiza un cifrado mediante SSL/TLS [11], de esta manera, se consigue que la información sensible no pueda ser usada por un atacante, que haya conseguido interceptar la transferencia de datos de la conexión, ya que lo único que obtendría sería un flujo de datos cifrados que le resultará imposible de descifrar. Usar este protocolo implica que las rutas a la API deben de estar debidamente certificadas, mediante una autoridad certificadora.

Integridad

El sistema debe asegurar que los datos están a buen recaudo, lo que significa que la información de los usuarios será manejada con una cuidadosa protección frente a la corrupción e inconsistencia de estos datos.

Para cumplir este requisito es necesario hacer uso de copias de seguridad sobre la base de datos, y los archivos que haya subido el usuario, sin embargo, esto no protegería frente a la corrupción de los datos, ya que, realizando solamente copias de seguridad, en algún momento, estas también se verían afectadas por este problema. Para resolver esto, se propone que las copias de seguridad estén versionadas, de esta manera se podrían recuperar los datos previos a cualquier corrupción del sistema, simplemente buscando la versión que todavía funcionaba correctamente. Para que este proceso sea totalmente efectivo, se plantea que las copias de seguridad y su versionado sea periódico, cuanto más corto sea este periodo, mejor será la capacidad de recuperación, pero a su vez mayor será el uso de recursos del sistema, por eso, inicialmente se ha determinado que este periodo sea de un día, ya que debido a la fase y la naturaleza del proyecto, ante un fallo, perder los datos de un día no resultaría crítico, y se mantendría un uso moderado de los recursos del sistema.

4.3 Análisis del marco legal y ético

La aplicación móvil que se plantea desarrollar para este proyecto contará con acceso a la red y almacena información de los usuarios, que en algunos casos puede ser sensible para estos. Además, los usuarios tienen múltiples herramientas dentro de la aplicación que les permiten realizar acciones que en algunos casos se tienen que controlar o limitar por el bien de la comunidad y de la plataforma, o para cumplir y evitar la violación de ciertos aspectos legales muy importantes hoy en día en este tipo de entornos. Por lo tanto, es necesario implementar estos documentos, y dotar a la solución de un mecanismo para que los usuarios puedan aceptar o rechazar estos acuerdos legales.

Términos y condiciones de uso

Los términos y condiciones son un acuerdo importante que debe incluirse en su sitio web o aplicación. No es obligatorio por ley, pero es importante contar con ellos. Es el acuerdo en el que establece las normas y directrices para sus usuarios.

Con un acuerdo de términos y condiciones, retendrá sus derechos a finalizar el acceso al sitio web o aplicación a los usuarios que abusen del mismo. Y también conservará su derecho a eliminar de su sitio web o aplicación el contenido generado por los usuarios que pueda violar derechos de autor.

En la aplicación que se va a desarrollar el usuario tiene total libertad para publicar cualquier tipo de contenido, por eso es necesario este tipo de elementos reguladores, de esta manera nos cubrimos frente a los usuarios que realicen un uso inadecuado de la plataforma, ya sea en contra de las leyes o de los términos y condiciones de la propia tienda de aplicaciones.

Política de privacidad

La política de privacidad es un documento legal por medio del cual el desarrollador informa a sus usuarios sobre la forma en la que recabará los datos personales, describe cuál será el tratamiento de los datos y, en su caso, solicita el consentimiento para la cesión de dichos datos a terceros.

El objetivo de este documento legal es ofrecer garantía jurídica y confiabilidad de las visitas o usuarios hacia nuestra empresa, en lo que se refiere al tratamiento de sus datos de carácter personal.

La mayoría de los países disponen de unas leyes de privacidad que requieren que las aplicaciones que recogen datos personales tengan una política de privacidad en vigor. Los datos personales son un tipo de información a través de la cual se puede identificar a un individuo, ya sea directamente o cuando ésta se combina con otros datos, como nombres, direcciones de correo electrónico, localización, direcciones de IP, fotografías e información de la cuenta, son todos datos directamente identificativos.

La aplicación que se va a llevar a cabo en este proyecto recoge datos como el correo electrónico y el nombre de usuario. Por otra parte, también solicita acceso al sistema de archivos y la galería del usuario. Por lo tanto, va a ser necesario preparar e incluir una política de privacidad en la aplicación, teniendo en cuenta las leyes en materia de privacidad de datos a nivel europeo (GDPR) y español (LOPD)

4.4 Casos de uso

Para complementar, y ayudar a entender la especificación de los requisitos funcionales, resulta bastante útil realizar el modelado de un diagrama de casos de uso [12] para el sistema que se desea implementar, ya que en este diagrama se especifica el comportamiento dinámico del sistema, las entidades externas que utilizan el sistema, y las interacciones entre el sistema y estas entidades. Habitualmente los diagramas de casos de uso se componen de los siguientes elementos para realizar el modelado [13] [12] :

- **Los actores**, que representan todos los elementos que pueden interactuar con el sistema, estos elementos pueden ser los usuarios u otros sistemas externos. Los actores representan el rol que lo relaciona con el sistema, no se tratan de instancias o elementos específicos.
- **Los casos de uso**, que consisten en las funcionalidades completas que ofrece el sistema.
- **Relaciones**, que sirven para asociar los actores con los casos de uso o para especificar relaciones de generalización entre actores, para así crear actores que tienen su rol especifico y el rol de otro actor más generalizado.
- **El propio sistema software** del que se van a extraer los requisitos funcionales

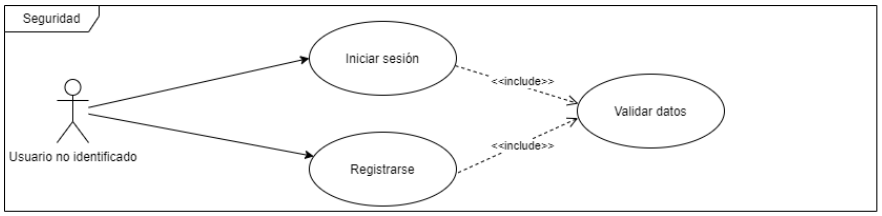


Figura 4.1: Casos de uso del sistema de seguridad.

En el diagrama anterior se puede observar que existen dos tipos de roles que interactúan con el sistema, los usuarios que han iniciado sesión, y los que no lo han hecho, esto se ha realizado así para poder dividir y visualizar fácilmente lo que se puede realizar cuando se ha iniciado sesión, y cuando no.

Otra aproximación seria utilizar un solo actor, e incluir en cada caso de uso el inicio de sesión, pero debido a la gran cantidad de casos de uso que tiene este sistema, el resultado sería un diagrama difícil de interpretar, con una confusión importante de relaciones.

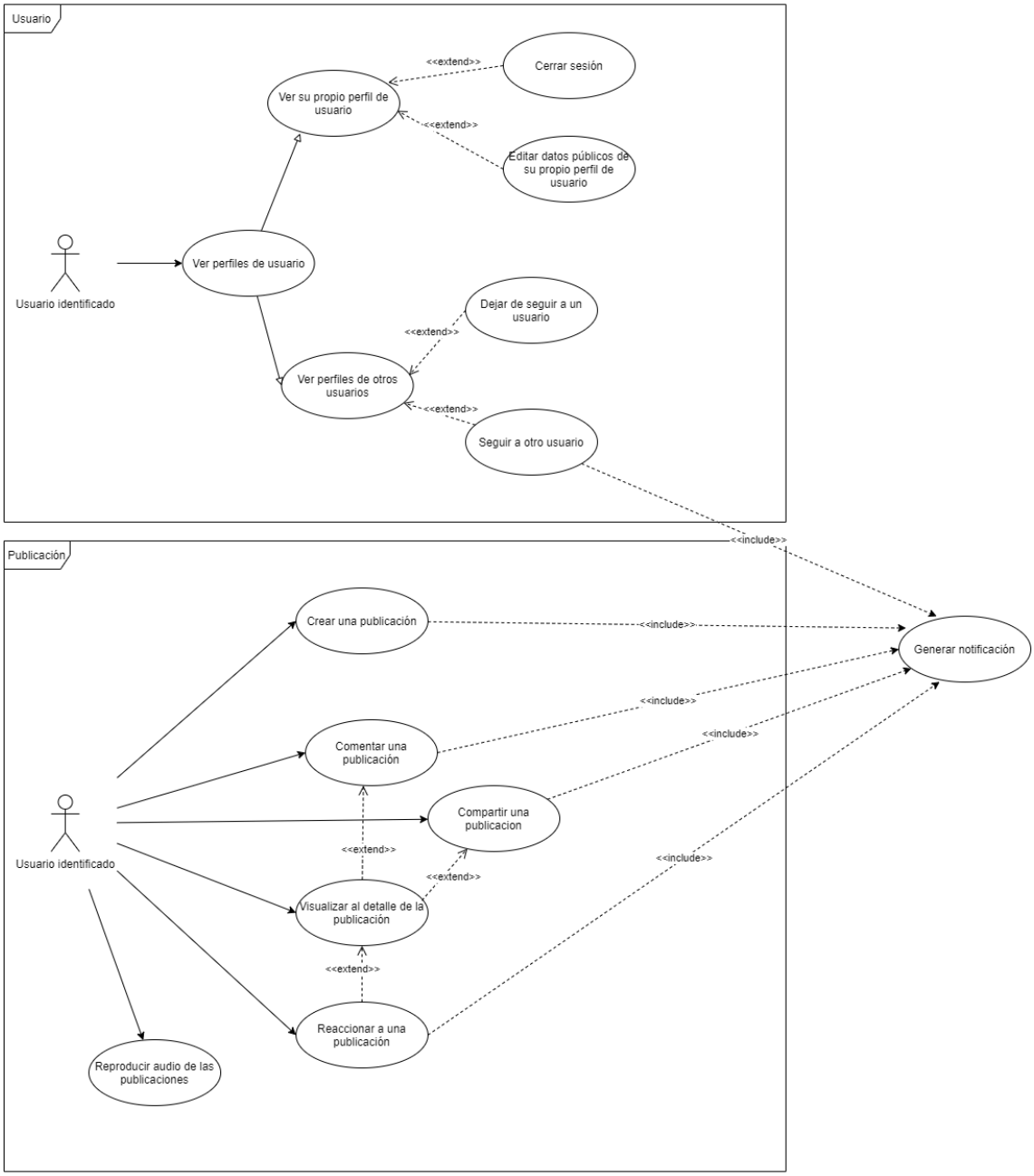


Figura 4.2: Casos de uso de los sistemas de usuarios y publicaciones.

Los casos de uso que se pueden ver en el diagrama han sido extraídos a partir de características, y comportamientos que se han acotado previamente para las primeras versiones de la red social. En primer lugar, como se ha adelantado anteriormente un usuario identificado es aquel que ha registrado en la aplicación, y/o ha iniciado sesión en esta. De esta manera se cumple con el requisito de seguridad, y gestión de cuentas de usuarios, ya que los usuarios no identificados solamente pueden iniciar sesión o registrarse en la plataforma, mientras que los usuarios identificados pueden realizar el resto de las acciones especificadas en los requisitos, y modeladas en el diagrama de la Figura 4.2.

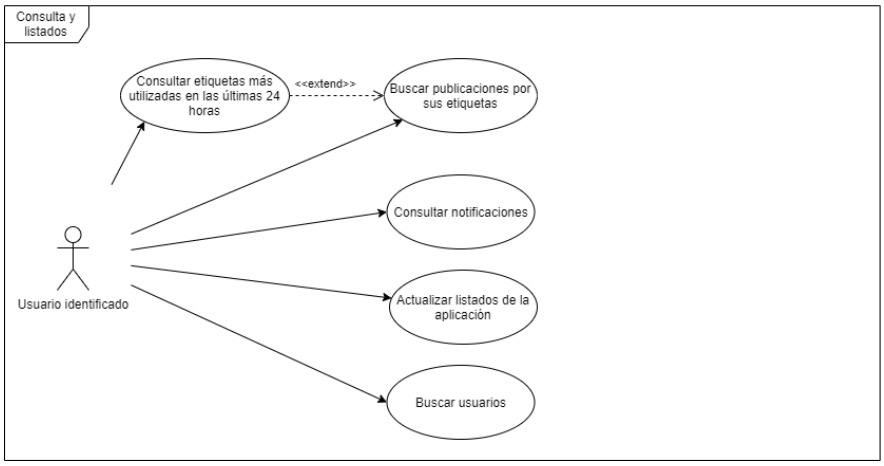


Figura 4.3: Casos de uso del sistema de consulta y listados.

En último lugar, el sistema de consulta y listados simplemente se compone de los casos de uso donde el usuario puede visualizar información o listados, y realizar acciones sobre estos listados. Este es el sistema más sencillo, pero también agrupa los casos de uso más habituales de la plataforma, ya que está basada en diferentes tipos de listados con información de las entidades requeridas, sobre los cuales se pueden hacer una gran cantidad de la funcionalidad propuesta.

En este caso el sistema cuenta con tantos casos de uso, que resulta muy complicado explicarlos en profundidad de una forma estructurada mediante una especificación textual, ya que la extensión del presente trabajo se alargaría demasiado, e impediría hablar de otros temas importantes del proyecto.

4.5 Modelado conceptual

Para empezar, se han diseñado los componentes de la aplicación teniendo en cuenta todo aquello que se había determinado previamente durante la etapa de análisis. Todos estos componentes son los estrictamente necesarios para poder llevar a cabo la funcionalidad requerida de la plataforma.

El primer objetivo es definir los componentes más básicos que van a formar el software a desarrollar, en este caso se tratan de las entidades y sus relaciones, que se pueden definir a partir del dominio de información que se ha determinado durante la fase de análisis. Para asistir y complementar este diseño, se ha realizado el diagrama de clases que se puede observar en la Figura 4.4.

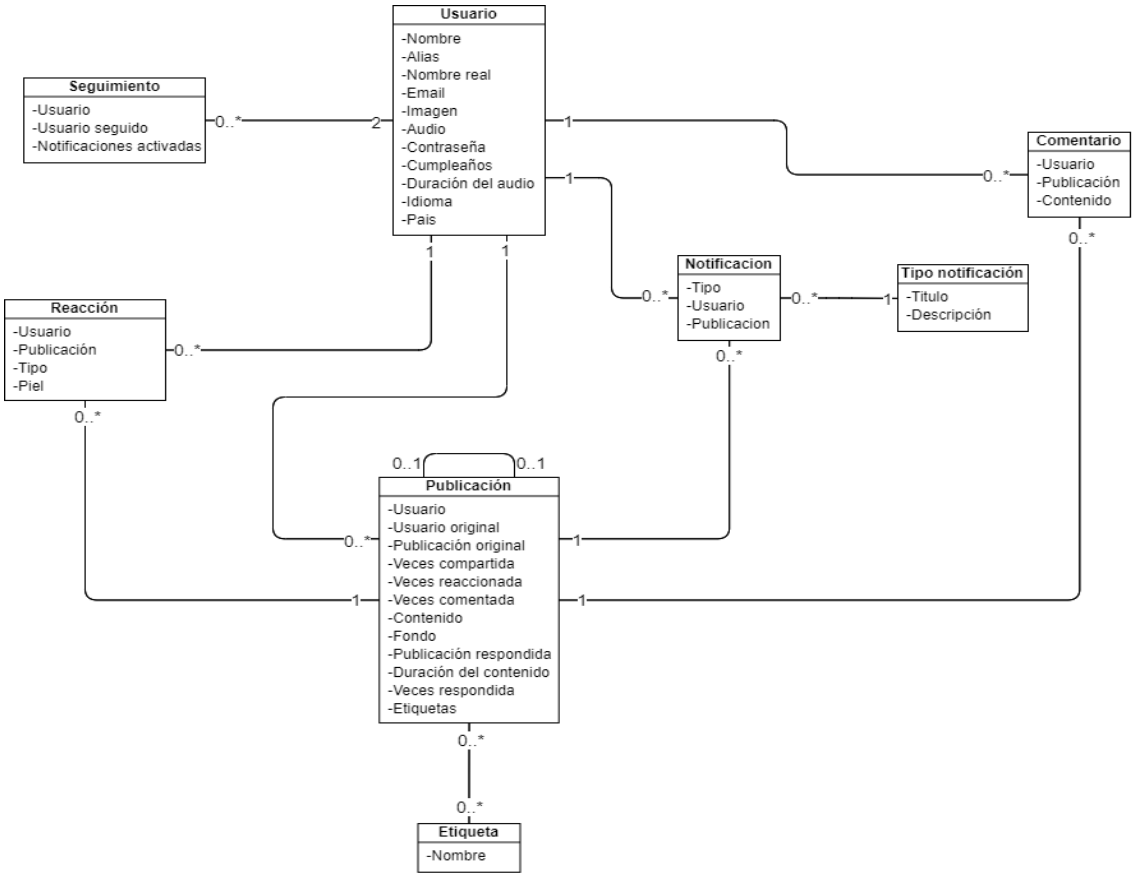


Figura 4.4: Diagrama de clases de la plataforma.

Tal y como se puede observar, las relaciones entre las entidades son básicas y definen muy claramente la estructura que va a tener la aplicación. De esta manera se puede afirmar que los dos componentes principales de la aplicación son los usuarios y sus publicaciones, ya que las relaciones del resto de componentes giran en torno a estos.

El usuario se trata del componente más importante del sistema, ya que el resto de las entidades están ligadas a esta, lo cual tiene bastante sentido ya que, desde el punto de vista funcional todas las acciones de la aplicación las realizarían sus usuarios. Además, esta entidad contiene la información necesaria del usuario, y cuenta con una entidad auxiliar que da sentido a la funcionalidad de seguimientos de la plataforma.

La publicación tiene toda la información necesaria a mostrar, y se relaciona con si misma para poder dar significado a acciones que se pueden realizar mediante una publicación, como es el compartir y el responder a otra publicación, ya que estas acciones básicamente crearían una publicación siguiendo el mismo esquema de datos que una publicación normal, pero dentro del mismo modelo hay que diferenciar estas publicaciones y relacionarlas como sea conveniente.

La relación entre los usuarios y las publicaciones expresa la funcionalidad de creación de publicaciones, y como cada usuario es capaz de crear varias de estas publicaciones, este diseño implica que cada publicación siempre tiene 1 usuario, mientras que cada usuario puede tener varias publicaciones.

Todas las funcionalidades especificadas se han modelado mediante los componentes secundarios que se relacionan con el usuario y con la publicación. En primer lugar, las notificaciones tienen información del usuario que ha realizado la acción que desencadena la notificación, e información de la publicación sobre la que se ha realizado la acción. Dependiendo de la acción existen diferentes tipos de notificaciones, lo cual se ve reflejado en la entidad de tipo de notificación relacionada exclusivamente con el componente de notificaciones, la cual contiene la descripción del tipo de notificación.

Sumado a esto, las entidades de comentario, y reacción están ligadas a un usuario, porque funcionalmente este realiza la acción, y también se relacionan a la publicación sobre la que se efectúa la acción. Además, estas entidades incluyen toda la información que se necesita para llevar a cabo estas acciones y persistirlas en la aplicación.

En la aplicación también existen varios subsistemas que se forman con estos componentes básicos, en primer lugar, se encuentra el subsistema de seguridad, que se encarga del alta de los usuarios en la aplicación, y de que solamente los usuarios autenticados puedan acceder a esta. Por otra parte, este subsistema asegura el acceso a los datos, prohibiendo el acceso a estos por parte de usuarios no identificados, y limitando el acceso de los datos privados a cada usuario. La construcción de este subsistema se basa completamente en el componente de usuarios, y abarca los puntos de entrada a cualquiera de los elementos de la plataforma.

Además de esto, se contempla el subsistema de usuarios, el cual se encarga de mostrar y gestionar toda la información de un usuario, y de sus elementos privados, pero no se tiene en cuenta la creación o autenticación del usuario, ya que de esto se encarga el subsistema de seguridad explicado previamente. Esto significa que el subsistema de usuarios estaría completamente formado por los componentes de usuario, notificación y seguimiento.

Por último, el subsistema publicaciones contiene toda la información de los componentes de publicaciones y las acciones que se pueden realizar sobre estas. Este subsistema está estrechamente relacionado con el de usuarios, ya que son estos los que realizan las acciones. Por lo tanto, los componentes que forman este subsistema son de publicación, reacción, comentario y etiqueta.

5. Diseño

En este capítulo se define el diseño del software del proyecto, el cual consiste en la etapa previa al desarrollo del software y en la que se realiza el visionado, y especificación de las soluciones software que se van a llevar a cabo durante el proyecto. Dicho de una manera abreviada, se trata de la fase en la que se transforma el dominio del problema, que se ha estudiado en la fase de análisis, en su representación del software.

En general, la actividad del diseño se refiere al establecimiento de las estructuras de datos, la arquitectura general del software, representaciones de interfaz y algoritmos. El objetivo del diseño es producir un modelo o representación de una entidad que se va a construir posteriormente.

Para facilitar el diseño del software se ha decidido que lo primero es especificar los componentes de la aplicación, ya que el resto del diseño depende profundamente del resultado de esta tarea. Después de esto se ha concretado la arquitectura de la aplicación a todos los niveles, esto quiere decir que en cada nivel se ha diseñado la arquitectura acorde a los requerimientos que se tiene de la aplicación. En último lugar, se ha procedido con el diseño de las interfaces de usuario y las comunicaciones del sistema.

5.1 Arquitectura y componentes del sistema

En primer lugar, en su nivel más alto el sistema se construye mediante la arquitectura cliente – servidor, que consiste en un sistema el que las tareas se reparten entre los proveedores de recursos o servicios, llamados servidores, y los demandantes, llamados clientes [14] .

En los últimos años el desarrollo de estas dos partes se ha ido separando y especializando hasta conformar dos bloques de desarrollo enfocados a cada parte de la arquitectura, la parte del cliente que hace referencia al interfaz de usuario que presenta y/o recibe la información, se denomina frontend, mientras que el backend consiste en la parte del servidor, de la lógica de negocio, y el almacenamiento de datos. Es decir, todas las funciones que requiere el cliente, pero estos servicios, se pueden implementar utilizando una gran variedad de enfoques, metodologías y tecnologías. Entre estos enfoques destaca el estilo arquitectónico *"REpresentationmal State Transfer"* (REST) [15], que intenta solucionar ciertos problemas de otros paradigmas como WSDL [16] o SOAP [17] . De esta manera, en el servidor del backend se implementaría una API RESTful [18] [19] para dar servicio al frontend siguiendo la especificación REST, esta API se conectaría a una base de datos, para permitir el intercambio de información entre ambas partes, y la gestión de esta por parte del servidor. Además, dispondría de capas de software que proporcionan interfaces de programación más cómodas y capaces de gestionar algunos de los recursos del sistema como son los middlewares [3].

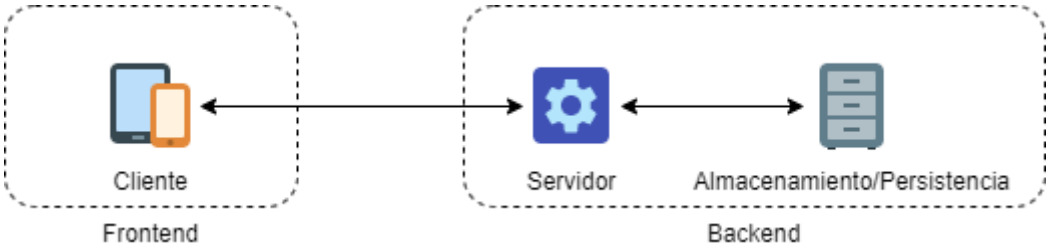


Figura 5.1: Arquitectura del sistema.

El cliente realiza las peticiones al servidor del backend solicitando los recursos que necesita para su funcionamiento, este servidor por su parte solamente aceptará peticiones de clientes autenticados mediante sus sistemas de seguridad. Las solicitudes podrán ser operaciones sobre la base de datos y sobre el sistema de archivos del backend, es decir que el servidor actuaría como puerta de entrada, e intermediario entre el cliente y la base de datos, o los archivos estáticos de la aplicación.

La principal ventaja de esto es que la estructura del sistema es modular, y cualquier cambio en un elemento no debería afectar al funcionamiento del resto.

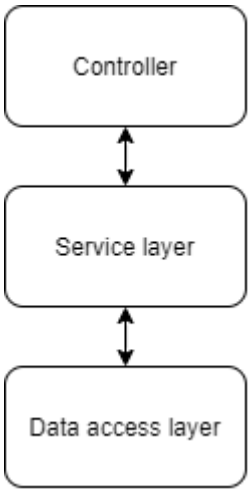


Figura 5.2: Arquitectura en tres capas de NodeJS.

La aplicación del servidor se ha desarrollado utilizando una arquitectura de tres capas de NodeJS, en la cual cada capa tiene un conjunto específico de responsabilidades, que están claramente definidas y son fáciles de comprender. Cada capa accede a la capa debajo de ella, pero nunca por encima de ella. La entrega de una solicitud toca cada capa comenzando desde la parte superior, viajando hacia abajo y luego resurgiendo de nuevo a la capa superior, tal y como se puede apreciar en la Figura 5.2.

La capa de controladores funciona mediante enrutadores denominados *routers* y los propios controladores. Un *route* es una sección que asocia un método HTTP (GET, POST, PUT, DELETE [20]), una ruta o patrón URL, con el controlador que se encarga de la entrada de la petición, realizando validaciones de entrada, o analizando el contenido de las peticiones, y enviando a la siguiente capa los resultados de todo esto para su procesamiento, el cual será enviado de vuelta a la capa *route* para enviar la respuesta a la petición realizada en un principio. Por ejemplo, cuando se realiza una petición GET del usuario actual, el *route* de esta petición se

encarga de direccionar al controlador adecuado, el cual establece el usuario actual de la aplicación mediante los datos de la petición, los resultados se envían a la siguiente capa para su procesamiento, y una vez realizado, se devuelven los resultados a la capa de controladores, la cual se encarga de enviarlos como respuesta a la petición.

Tal y como se puede apreciar en la Figura 5.3, hay tres niveles de enrutamiento, en primer lugar se tiene el enrutador principal que direcciona todas las peticiones de la ruta */api/v1* hacia el conjunto de enrutadores de segundo nivel, los cuales direccionan hacia el enrutador de tercer nivel de la entidad especificada en la ruta, finalmente este último nivel direcciona hacia el recurso de la entidad o el método HTTP seleccionado, y enlaza directamente con el controlador que corresponde tal y como se especifica en la Figura 5.4.

Esta manera de enrutar facilita la mantenibilidad de la API de la plataforma, ya que la ruta principal puede contener el cualquier número de rutas de segundo nivel, por lo tanto, se podrían enrutar varias versiones de forma sencilla manteniendo el soporte a las rutas antiguas. Por otro lado, también hace posible la extracción de conjuntos de rutas de segundo nivel para asociarlas de diferentes maneras a la misma u otra ruta de primer nivel.

Por ejemplo, si en un momento existe una ruta que consume demasiados recursos, simplemente con extraerla y enlazarla con otra ruta de primer nivel que se ejecute en un proceso o maquina aparte, se resolvería el problema. Y es que lo más interesante de todo esto es que con la arquitectura de enrutadores en varios niveles se puede realizar el escalado horizontal de varias maneras, y este último ejemplo sería una de ellas.

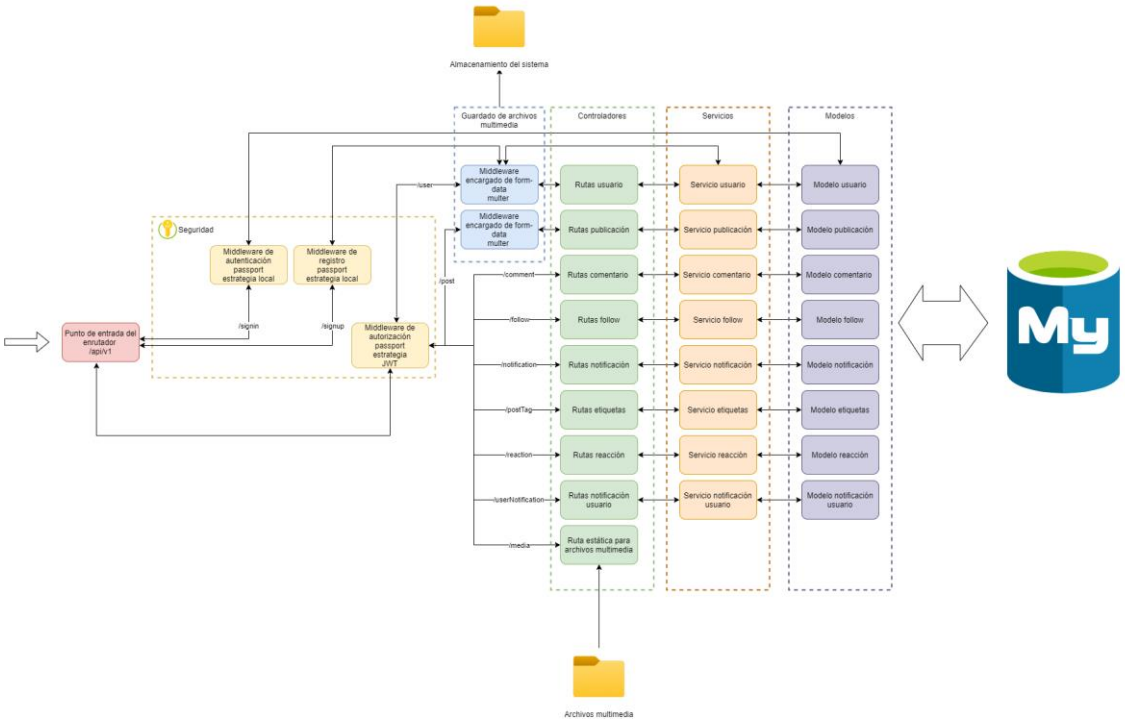


Figura 5.3: Arquitectura del backend de la aplicación.

Habitualmente en este tipo de arquitecturas en la primera capa de controladores, se suele organizar separando la lógica de los controladores de la lógica de enrutamiento y, por lo tanto, cada tabla tendría su controlador y su enrutador en esta primera capa.

Sin embargo, para la arquitectura de la aplicación servidora de la plataforma no se ha optado por esta solución, ya que en este caso lo mejor es tener toda esta lógica junta en el enrutador. Por lo general separar las lógicas proporciona pequeñas ventajas en la organización del proyecto, y encapsulación de los controladores para ser utilizados por diferentes enrutadores, cosa que no se puede aprovechar debido que ya existe una gran separación de la funcionalidad por entidad, que lo enrutadores no reutilizan lógica ni controladores, y tampoco utilizan aquellos controladores que corresponden a otras entidades.

La razón de esto último es que la aplicación se ha implementado siguiendo una arquitectura en tres capas para cada entidad, en la cual cada entidad de base de datos tiene un controlador/enrutador, un servicio y un modelo, porque se considera que la mejor alternativa es que las entidades tengan que estar separadas entre sí y no compartir responsabilidades ni afectar al resto, ya que de esta manera se facilita la futura mantenibilidad y escalabilidad de la aplicación.

Siguiendo con la arquitectura en tres capas, la siguiente que se presenta en la Figura 5.2 es la capa de servicios, que consiste en aquella en la que realiza toda la lógica de negocio de la aplicación, siguen la lógica que especifican las funcionalidades de la plataforma, validan los datos que reciben desde la capa anterior, pueden realizar llamadas a otros servicios de la capa y si fuese requerido, en esta capa se realizarían integraciones con sistemas externos.

Esta capa también debe seguir la arquitectura general, separando por entidades de bases de datos tal cual se ha justificado anteriormente, por lo tanto, cada entidad debe de tener su propio servicio vinculado a su controlador/enrutador, y a su modelo de acceso a datos, justo como se especifica en la Figura 5.3 y en explicaciones anteriores de la capa de controladores.

La lógica de la capa de servicios a menudo implica operaciones en la capa de acceso a datos, y por defecto, esta capa, se ha pensado para dar servicio a cada uno de los métodos CRUD de HTTP que proporcionan los controladores tal y como se puede ver en la Figura 5.4. Por esta razón cada servicio implementa las funciones find, save, update y delete, que se deben asociar con sus métodos HTTP correspondientes, realizando las llamadas desde el controlador del método que toca para cada función. Por ejemplo, el controlador/enrutador del método GET solamente puede llamar a la función find del servicio.

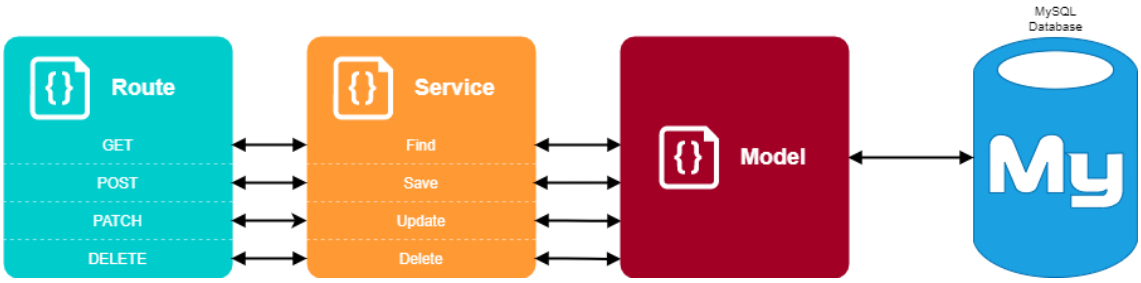


Figura 5.4: Arquitectura de cada ruta del backend.

En la figura anterior podemos ver la arquitectura, organización y convención interna en capas de cada una de las entidades o tablas que se han creado en base de datos, para almacenar la información de la aplicación. Por otra parte, se puede ver que la comunicación a lo largo de todas las capas está estrechamente ligada a las funciones básicas CRUD, que se solicitan mediante los métodos HTTP correspondientes, y se procesan mediante los servicios que se asocian a cada uno de estos métodos de manera interna, ya que estos servicios son los que

finalmente realizan procesamiento de los datos mediante el modelo. Todo esto también quiere decir que por cada tabla nueva que se cree en base de datos, es obligatorio crear los elementos de cada capa, para que esta nueva entidad cumpla con la arquitectura propuesta.

Para terminar con la arquitectura en tres capas es imprescindible comentar el papel de la capa de acceso a datos, en la cual se encuentran todos los objetos que realizan el mapeado a tablas de base de datos, y mediante los cuales se realizan las operaciones en base de datos. Estos objetos actúan como traductores de las operaciones que se quieren realizar y del propio modelo de datos sobre el cual se realizan estas operaciones. Es decir que estos modelos de la capa de acceso a datos sirven tanto para realizar las operaciones en su tabla de base de datos, como para mapear el resultado de las operaciones a un modelo que reconozca la aplicación servidora, la cual, a su vez, utilizaría estos modelos para propagarlos hacia el resto de las capas, que podrían proveer de datos a los clientes entre otras cosas.

Con la estructura especificada, cada servicio solamente puede acceder al modelo que se corresponde, y viceversa. Cualquier operación que implique varios modelos se debe de realizar entre los servicios del modelo, para no alterar la semántica de la arquitectura y para no empeorar ni la escalabilidad ni la mantenibilidad de la aplicación. En definitiva, esta capa se ocupa de toda la gestión con la base de datos, delegando la gestión de cada tabla en cada uno de los modelos que se implementan en la aplicación.

Hasta ahora se ha especificado rigurosamente la arquitectura que debe seguir la aplicación, y como se debe de implementar, sin embargo, esta estructura permite excepciones que de cierta manera no afectan a sus objetivos principales, y tampoco tienen efectos negativos sobre esta. En este aspecto se pueden encontrar varias excepciones que son necesarias para poder llevar a cabo la funcionalidad requerida para la plataforma.

Para empezar, en el servicio de comentarios la función *save* tiene que ser diferente, porque pensando en el rendimiento, la acción de comentar además de realizar una inserción en la tabla de comentarios, también se tiene que actualizar la tabla de publicaciones para incrementar un contador de comentarios para esa publicación, de esta manera para recoger el número de comentarios que tiene un publicación no se tienen que realizar consultas adicionales o relaciones entre tablas, lo cual mejoraría el rendimiento y escalabilidad considerablemente.

Este comportamiento también debería ser así cuando se elimina un comentario, ya que se trata de la misma acción, pero a la inversa, por lo tanto, cuando se elimina un comentario, además de eliminarlo en la tabla de comentarios también se debe decrementar el contador de comentarios de su publicación en la tabla de publicaciones.

En el servicio asociado a la tabla que almacena seguimientos también existen excepciones, la primera de estas se realiza en la función de obtención de datos para mejorar el rendimiento de la consulta a base de datos, y es que aparte de los datos de seguimiento también necesitaremos los datos de los usuarios asociados a estos, la solución sencilla es relacionar mediante las claves primarias con la tabla user para sacar sus datos de usuario.

El problema es que cuando buscamos los usuarios que siguen a otro usuario concreto lo que realmente estamos haciendo es una búsqueda en la tabla de seguimiento filtrando por el usuario seguido, todos los registros que nos devuelva la búsqueda van a tener diferentes usuarios seguidores pero el mismo usuario que se está siguiendo, por esta razón no resulta eficiente

realizar una asociación con este último usuario por cada registro, ya que los datos de la relación estarían repetidos y por otra parte, en los casos de uso, no sería del todo necesaria esta relación.

Esto mismo pasa cuando se realiza la misma relación, pero a la inversa, es decir cuando se buscan los usuarios a los que sigue un usuario determinado, ya que recuperamos todos los registros que tengan un usuario seguidor determinado, esto supone que, al relacionar con los datos de usuario, aquellos del usuario seguidor serían los mismos en todos los registros.

La solución a esto se debe realizar en el propio servicio, ya que se realizaría lógica de negocio específica para comprobar sobre qué campo se ha solicitado la búsqueda, y con esto realizar la asociación con los datos del usuario escogiendo programáticamente el campo contrario al que se ha solicitado. Por ejemplo, cuando se busquen los seguimientos que tiene un usuario concreto, mediante la lógica de negocio el resultado se relacionaría con los datos el usuario seguidor, ya que el usuario seguido siempre sería el mismo, esto mismo, en el caso contrario, se realizaría a la inversa. El principal motivo para realizar esta excepción es que puede haber usuarios con muchos millones de seguidores, o siguiendo a miles de personas, y optimizar este tipo de búsquedas mejoría la escalabilidad de la aplicación en caso de que crezca su uso.

Para terminar, esta arquitectura tal y como se ha especificado hasta el momento presenta un problema de replicación de código, ya que todas las entidades implementan la misma estructura y el código, en su mayoría, sería el mismo a excepción del servicio o modelo con el que enlazan a la siguiente capa. La solución a esto es variar la arquitectura interna para que exista un elemento en las capas de controladores y servicios con toda la funcionalidad común de la capa, es decir, que se requiere de un servicio y enrutador comunes, de los cuales heredarían los elementos de cada entidad de la misma capa, además, estos elementos en cada capa deberían ser lo completamente genéricos, para que la funcionalidad pueda enlazar con los elementos correspondientes de las otras capas.

Por poner un ejemplo, la funcionalidad del método GET es compartida por todas las entidades de la aplicación, en este caso el enrutador común tendría que procesar las peticiones y enlazar con la función *find* del servicio correspondiente, para ello el enrutador de cada entidad tiene que inyectar su servicio asociado al heredar las funcionalidades del enrutador común. En el caso de la capa de servicios ocurriría algo similar, ya que la función *find* del servicio de cada entidad debe realizar una búsqueda en base de datos mediante el modelo que corresponde a la entidad, entonces el servicio de la entidad heredaría del servicio común inyectando a este el respectivo modelo de la entidad.

Esta solución facilita la gestión de las excepciones a la arquitectura, ya que estas pueden sobrescribir la funcionalidad común, o implementarse aparte para extenderla. También simplifican la creación de nuevas rutas o *endpoints* de la API, ya que simplemente serían extensiones de los elementos comunes en caso de ser específicos. Y es que en caso de necesitar un nuevo *enpoint* para todas las entidades, solamente con implementarlo mediante los métodos comunes funcionaria para todas las entidades de la aplicación.

También se especifican los *endpoints* de autenticación, aunque quedan aislados de la arquitectura en tres capas, también siguen formando parte de la arquitectura general de la aplicación servidora. En la Figura 5.3 se puede apreciar que todas las peticiones que entran desde la ruta de primer nivel pasan por la capa de seguridad, y según esta arquitectura estas peticiones excepto *login* y *register* tienen que enrutarse a través middleware de autenticación y autorización mediante tokens, donde se comprueba si la petición tiene un token valido. Este

token se envía al cliente cada vez que un usuario se registra o inicia sesión, y lo devuelve en la cabecera de cada petición. Las peticiones direccionadas a login pasan por un middleware que comprueba los datos de autenticación de usuario, consultando la base de datos directamente mediante el modelo de la entidad de usuario, por otro lado, las peticiones a *register* pasan por otro middleware de registro que realiza esta acción almacenando los datos y archivos del usuario en el *backend*. Toda esta implementación determina los mecanismos de seguridad de la API, y sus funcionamientos dentro de la arquitectura general para cumplir con los requisitos de seguridad de la aplicación.

El ultimo componente del backend se trata de aquel que se encarga del almacenamiento y gestión de los datos de la aplicación. En este caso se ha optado por realizarlo en una base de datos relacional ya que utiliza modelo relacional para la estructuración y la gestión de la base de datos, debido a que se trata del modelo más utilizado y con más soporte actualmente.

Su idea fundamental es el uso de relaciones, y considera la base de datos como una colección de estas. Las características más destacadas de las bases de datos relacionales son:

- Cada relación representa una tabla que en realidad se trata de un conjunto de filas, y a su vez cada fila consiste un conjunto de campos los cuales representan un valor que interpretado se corresponden con atributos del mundo real.
- No se pueden tener dos tablas con el mismo nombre ni registro.
- Las relaciones entre tablas se realizan mediante las claves primarias y las claves ajenas de cada tabla.
- Las claves primarias son el identificador principal de un registro dentro de una tabla, y estos deben cumplir con la integridad de datos.
- Las claves ajenas se colocan en la tabla que se relaciona, y contienen el mismo valor que la clave primaria del registro de la tabla principal.

La arquitectura del cliente o frontend, en este caso se corresponde con la estructura que va a seguir internamente la aplicación móvil, en la cual se empaqueta la funcionalidad en componentes muy básicos que servirán para formar las pantallas de la aplicación, por lo tanto, esta arquitectura se ha ideado para aprovechar al máximo estos componentes, y beneficiarse de la estructura de módulos independientes, fácilmente escalables, mantenibles y reutilizables.

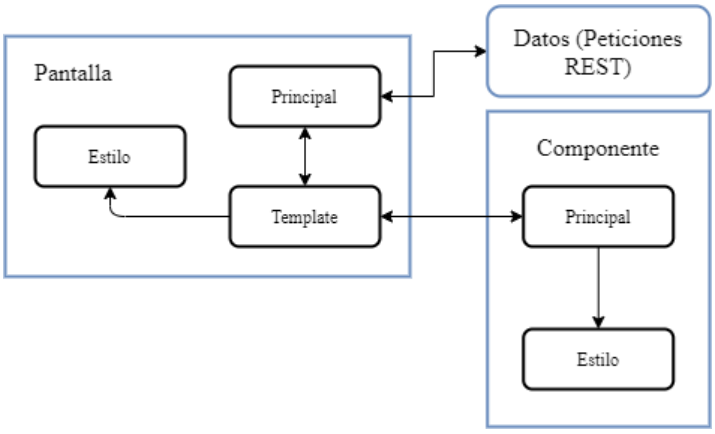


Figura 5.5: Arquitectura del frontend.

En la Figura 5.5 se puede observar visualmente la representación de la estructura interna de la aplicación, la cual implica que las pantallas de la aplicación deben estar formadas por un componente *template*, uno principal y una hoja de estilos. El primero consiste en la presentación de la pantalla, que a su vez suele estar formado por otros componentes más pequeños de la carpeta de componentes y que suelen ser reutilizados entre pantallas. El componente principal por su parte contiene lo necesario para el funcionamiento y renderización de la pantalla como su *template*, toda la lógica general y las peticiones para recuperar los datos necesarios. Finalmente, la hoja de estilos se ideó para implementar los estilos específicos que pudiese tener cada pantalla, para mantener la organización de los estilos de cada pantalla.

Además de esto, también se aprecia que los componentes siguen una pequeña arquitectura separando el propio componente de su estilo, para que en este último se implementen aquellos estilos que solamente se utilizan en el ámbito del componente, mientras que el propio componente se encarga de realizar toda la lógica, renderizado y procesamiento de los datos recibidos desde las pantallas.

En último lugar, el módulo de datos realizaría todas las peticiones al servidor para obtener los datos de las entidades que se han definido previamente en la arquitectura del servidor, y en el capítulo 4.5. Por lo tanto, este módulo contaría con las funciones para obtener datos desde las rutas de la API, y para mantener la coherencia con la estructura del sistema, el módulo de datos del frontend se divide en cada una de las entidades, es decir que cada división solamente cuenta con las funciones que realizan las peticiones a las rutas de la entidad del mismo grupo.

5.2 Comunicación

En el capítulo anterior se han determinado los componentes que realizan y gestionan las comunicaciones entre los dos grandes bloques que forman la aplicación. En la parte del frontend se trataría del componente de datos, mientras que en backend sería la capa de controladores. Es decir que las comunicaciones siempre se realizan entre estas dos capas mediante la API REST, utilizando los identificadores únicos de los recursos o URI (Uniform Resource Identifier) [21] , el protocolo HTTP para intercambiar mensajes, y la presencia de los verbos HTTP para especificar el tipo de operación que se desea ejecutar [15]. Además, estas especificaciones REST implican que las comunicaciones estarían organizadas por entidad, por lo tanto, el grupo de una entidad solamente podría realizar peticiones a la ruta de esa misma entidad. Como ya se

ha especificado previamente, las rutas implementan las funcionalidades *CRUD* mediante los métodos *GET*, *POST*, *DELETE* y *PUT* de HTTP. La parte del cliente podría implementar cada uno de estos métodos, para cada una de las entidades que forman el componente de datos, y de esta forma implementar la funcionalidad en torno a todas estas llamadas.

En este modelo existen dos formas de comunicarse con el servidor, mediante una solicitud de datos, y una solicitud para operar en base de datos mediante cierta información que se transmite. En el primer caso se corresponde con solicitudes de consulta mediante el método GET, en el cual las pantallas utilizan el módulo de datos para realizar una petición de este tipo a la API, la cual mediante los enrutadores y en última instancia la ruta procesan los datos de la petición y direccionan hacia el servicio correspondiente, en el cual se realiza la lógica de negocio y se recuperan los datos solicitados mediante la capa de acceso a datos, los cuales se vuelven a comunicar con la ruta para encargarse de devolver la respuesta al cliente.

En la otra forma de realizar las comunicaciones, las acciones se corresponden con los métodos POST, DELETE y PUT, ya que todas estas implican operaciones sobre registros en base de datos. Para ello, el cliente recoge los datos necesarios para realizar estas operaciones y realiza la petición correspondiente a la API, con estos datos. El servidor por su parte recibe la petición, y la direcciona hacia la ruta solicitada, que se encarga de procesar los datos de entrada y comunicarse con el servicio correspondiente, el cual ejecuta la lógica de negocio, y la operación en base de datos utilizando los datos transmitidos. Finalmente, el servidor recoge el resultado de las operaciones, y se lo envía como respuesta al cliente mediante la capa de controladores.

Se propone que estas comunicaciones entre cliente y servidor se realicen utilizando el formato de comunicaciones JSON, ya que se trata de uno de los formatos más extendidos de la actualidad, porque es un formato muy sencillo y fácil de tratar, y en este caso resulta muy conveniente, ya que las dos partes se implementarían mediante JavaScript.

5.3 Interfaz gráfica

En cuanto a las interfaces de la aplicación móvil, la aplicación se dividiría en dos grupos de interfaces diferenciados por su posición y acceso dentro del sistema, ya que existen interfaces que van a formar parte de la navegación principal de la aplicación, mientras que el resto solamente se van a poder acceder a través de otras interfaces.

Tanto las pantallas principales como las secundarías se agrupan en función del tipo de usuario que puede acceder a su funcionalidad, siguiendo la especificación de los casos de uso. Por lo tanto, hay un grupo de pantallas para usuarios no identificados, donde se realizan las funciones

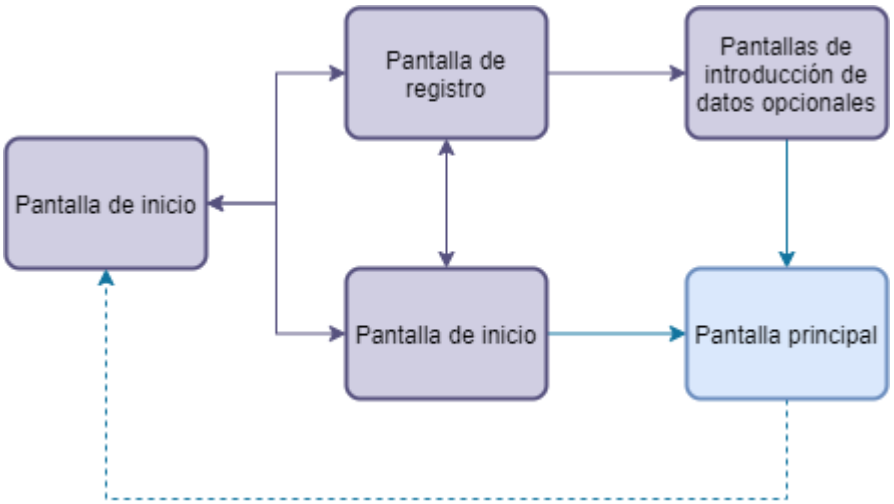


Figura 5.6: Flujo de las pantallas de inicio de sesión.

de registro e inicio de sesión en la aplicación. Mientras que el resto son pantallas para usuarios identificados. Esto es así porque a cada grupo de usuarios solamente se les muestran las pantallas del grupo al que pertenecen. Además, las pantallas de usuarios no identificados se estructuran en torno a un flujo de navegación definido para facilitar la experiencia del usuario y fomentar el registro de nuevos usuarios, que se puede observar a continuación en la Figura 5.6.

En último lugar, como se ha mencionado previamente, el diseño gráfico de la interfaz ha sido realizado por Luz Mélida García, como objeto del trabajo de fin del máster (TFM) en producción artística de la facultad de Bellas Artes de la Universidad Politécnica de Valencia, titulado “Diseño de interfaz de una red social de audios: Soundn y sus necesidades gráficas”. En principio no existían requisitos explícitos para este diseño, ya que los requisitos se centraban en la funcionalidad de la plataforma. Sin embargo, la estética de esta no se ha dejado al azar, ya que resulta uno de los componentes más importantes de este tipo de plataformas, debido a que uno de los objetivos principales las redes sociales es la captación de usuarios, donde el buen diseño de la interfaz y de la experiencia de usuario tienen un papel importante.

6. Implementación

Toda la implementación del sistema ha conllevado todos los trabajos necesarios para poner la aplicación móvil operativa y en producción. En primer lugar, se ha realizado la puesta en marcha de la infraestructura sobre la cual se desplegaría la plataforma, este trabajo comprende instalaciones y configuración de todo lo necesario de la máquina, despliegues automáticos, certificados SSL, y servidor de base de datos.

Como sistema gestor de base de datos se ha escogido MySQL ya que nos permite el almacenamiento, modificación y extracción de la información en la base de datos. Además, nos proporcionan métodos para mantener la integridad de los datos, para administrar el acceso de los usuarios a los datos y para leer de diferentes maneras, la información almacenada en la base de datos.

Después de esto se ha procedido con la implementación del software de la plataforma, empezando por el backend de la aplicación para respetar la metodología y planificación del trabajo. Además, se ha seguido la arquitectura cliente servidor que se ha diseñado previamente, y la cual se puede ver especificada con más detalle en la Figura 6.1.

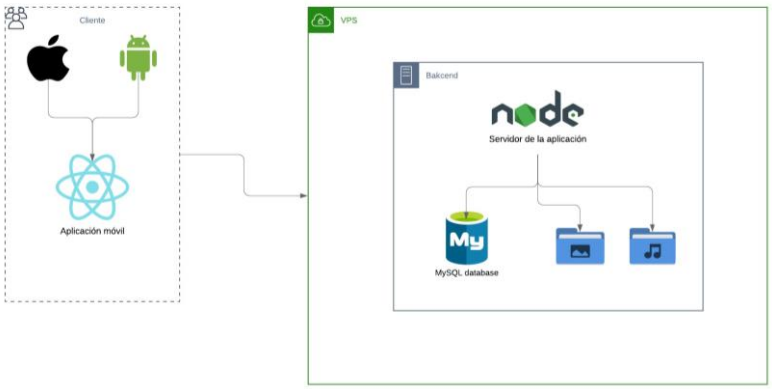


Figura 6.1: Implementación de la arquitectura cliente – servidor.

Se trata de la habitual arquitectura de las aplicaciones web formada por el cliente y el servidor, en este caso el cliente es el frontend de la red social, es decir sus dos aplicaciones React para móviles Android e iOS, y por otra parte, en el VPS tenemos alojados el backend de la aplicación, compuesto por las bases de datos y el servidor, que se trata de una aplicación NodeJS con toda la lógica de la aplicación, el acceso a base de datos, el guardado de archivos de la red social, y la comunicación con el frontend mediante una API privada.

Para terminar, se llevó a cabo la implementación de la aplicación móvil, y todas sus interfaces, siguiendo el diseño de pantallas y funcionalidades que se ha especificado previamente durante todo el capítulo 5, de esta manera se conseguiría cumplir con los requisitos funcionales establecidos, ya que estos diseños se han realizado para trasladar estos requisitos a la fase de implementación de la mejor manera posible. El desarrollo de la aplicación móvil se ha divido en tres fases tal y como se ha determinado en el plan de trabajo, en la primera fase se ha llevado a cabo la arquitectura que se ha diseñado, y con esto se ha establecido un esqueleto sobre el cual desarrollar el resto de la aplicación. En segundo y tercer lugar se han implementado todas las

funcionalidades y visualizaciones de las pantallas de la aplicación, y después de esto, se finaliza con la implementación de los estilos de las mismas pantallas.

6.1 Preparación de los entornos

Las primeras tareas realizadas de la fase de desarrollo fueron las instalaciones y configuraciones de las tecnologías utilizadas en el backend, lo primero en instalar fue NodeJS tanto en el VPS como en la maquina local donde se desarrollaría la aplicación backend, esta última Windows 10 instalado, lo cual facilitó la instalación porque se realizó mediante el asistente de instalación oficial de la plataforma.

En el ordenador de desarrollo también se instalaron las plataformas esenciales para la implementación, las cuales se tratad de SourceTree, Git, Postman, y Visual Studio Code. Este proceso se llevó a cabo mediante los asistentes de instalación para Windows que provee cada plataforma, y no hubo ningún problema debido a la facilidad de uso que tienen este tipo de instaladores.

En el caso del VPS se trata de un sistema Ubuntu, por lo tanto, también fue sencillo instalar esta tecnología, ya que simplemente hubo que ejecutar el comando de instalación del gestor de paquetes de Ubuntu. Este proceso también instaló el gestor de paquetes de node o NPM, por lo tanto, no se requirió de acciones adicionales para instalarlo o configurarlo.

En primer lugar, se debía de realizar la instalación y configuración del gestor de procesos PM2 en el VPS donde se va a desplegar el servidor, para llevar a cabo esto se siguió la guía de instalación[7] que nos facilita la web de PM2 en la cual solamente hay que ejecutar el comando npm para instalar PM2 de forma global. Por otra parte, para configurar los servicios que se ejecutarían mediante el gestor, se implementó un fichero llamado *ecosystem.config.js* que contendría varios despliegues, uno para cada entorno incluyendo su configuración específica, de esta manera mediante el parámetro *–only* se escoge que despliegue del fichero se realiza, por ejemplo, si se desea realizar un despliegue sobre producción, simplemente con ejecutar *pm2 ecosystem.config.js –only production* se ejecutaría un servidor de producción escuchando en el puerto definido, con el nombre establecido a *production* y los ficheros de logs correspondientes, entre otras posibles configuraciones.

Para que Gitlab sea capaz de ejecutar de forma remota los comandos de despliegue en el VPS donde se encuentra el backend de la aplicación, se requiere la instalación de un ejecutor de Gitlab en el VPS en la primera parte de la implementación de CI/CD, para ello simplemente se han seguido los pasos que se encuentran en las guías que proporciona Gitlab para este propósito[8].

[7] Guía de inicio de PM2 https://pm2.keymetrics.io/docs/usage/quick-start/
[8] Guía que se ha seguido para la instalación del ejecutor de Gitlab en el VPS https://docs.gitlab.com/runner/install/linux-manually.html

Lo siguiente que se realizó fue el registro del ejecutor para vincularlo con el proyecto del repositorio del servidor del backend en Gitlab, para ello se siguieron los pasos de la guía de registro de Gitlab[9], durante este proceso también se configuraron ciertas propiedades del proyecto y del ejecutor como establecer el ejecutor activo para el proyecto del servidor, o incluir la etiqueta *deploy* durante el registro para que el ejecutor solamente pueda aceptar cualquier trabajo que tenga esta etiqueta, esto es así porque este ejecutor solamente realizara la acción de desplegar, es decir mover los archivos a la carpeta correspondiente y ejecutar la aplicación. De esta forma el ejecutor aparece en el apartado de ejecutores activos para este proyecto tal y como se ve en la siguiente figura.

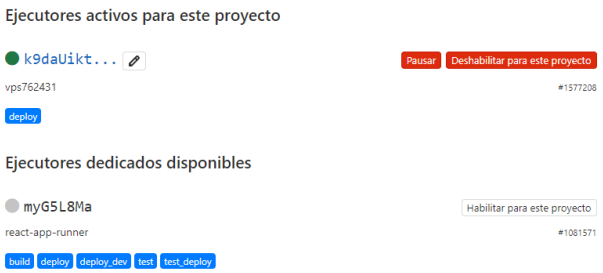


Figura 6.2 Ejecutor del proyecto.

Por otra parte, los trabajos que tengan otras etiquetas se realizaran mediante los ejecutores de Gitlab, ya que estos trabajos no necesitan de ningún recurso del VPS ni realizar tareas sobre este. De esta manera no se utilizan recursos de la maquina en la cual se encuentran desplegadas los *backends* de producción y pruebas, minimizando el riesgo de afectar al rendimiento de estos servidores.

Después de configurarlo todo correctamente ya es posible desplegar la aplicación servidora en el VPS mediante el ejecutor instalado, no obstante, las tareas que tengan asociadas a etiquetas que no sean *deploy* se ejecutaran en ejecutores compartidos que proporciona Gitlab, por lo tanto, las etapas de prueba y compilación se ejecutaran en los servidores de Gitlab ahorrando recursos del VPS donde se están ejecutando las aplicaciones productivas.

A continuación, en el VPS se instaló una instancia de MySQL para almacenar los datos de la aplicación productiva y la de pruebas. Este proceso se realizó ejecutando el comando de instalación de MySQL del gestor de paquetes de Ubuntu, y una vez realizado esto se procedió a configurarlo, esto se hizo siguiendo los pasos de la publicación del blog de Digital Ocean donde se explica meticulosamente como realizar este proceso[10], el cual consistía en cambiar la contraseña del usuario *root* para no comprometer la seguridad de la base de datos, y poder iniciar sesión sin problemas.

Para poder recibir las peticiones a dominio concreto, y direccionarlas al puerto en que se encontraría escuchando la API REST, se ha instalado Apache 2 en el VPS, que se trata de un servidor web HTTP. Este servidor es necesario configurarlo para poder direccionar las

[9] Guía para el registro de ejecutores en Linux https://docs.gitlab.com/runner/register/index.html#gnulinux
[10] Guía de instalación y configuración de MySQL en Ubuntu https://www.digitalocean.com/community/tutorials/como-instalar-mysql-en-ubuntu-18-04-es

peticiones a dominios concretos, para ello se ha preparado la característica de *sites-enabled*, que permite justamente esto.

Para terminar con la configuración del VPS, se instaló el certificado SSL para el dominio sobre el cual se iba a alojar la API REST, de esta manera se posibilitaba las peticiones vía HTTPS, y como consecuencia de esto se cumplía con el requisito de seguridad, ya que las conexiones estaban cifradas de punto a punto. Para realizar esto, se instaló *certbot* de *Let's Encrypt*, con el cual se ha realizado el proceso de certificación del servidor y dominio, previamente configurados mediante Apache. Este proceso es muy sencillo, ya que solamente se tiene que ejecutar el comando de certbot para certificar los sitios de Apache[11], el cual sirve para empezar la instalación interactiva del certificado para el sitio previamente configurado, durante esta instalación se solicita, el sitio o dominio sobre el que se quiere instalar el certificado, un email, y aceptar los términos de *Let's encrypt*. Este proceso internamente configura el servidor Apache para realizar las redirecciones a HTTPS, y para utilizar los certificados generados por la entidad certificadora. Sin embargo, todo esto no configura el certificado para el servidor de la API REST, lo cual se realiza más adelante durante la implantación.

Antes de empezar a desarrollar también se instaló una instancia de MySQL en el ordenador donde se realizaría el desarrollo del proyecto, ya que por motivos de seguridad no es recomendable conectarse a las bases de datos alojadas en el VPS directamente desde la aplicación que se está desarrollando en el equipo local. La instalación fue sencilla porque se realizó en un ordenador con Windows, y fue posible realizarla utilizando un asistente de instalación que la facilitó enormemente, en el cual se configuró el usuario, la contraseña y sus permisos para acceder a las bases de datos.

6.2 Verificación del proyecto y flujo de trabajo

Una vez instalado todo lo necesario, el siguiente paso fue verificar el clonado del repositorio del proyecto de Gitlab que se ha creado previamente, para ello se crea una carpeta completamente vacía en el equipo local de desarrollo, en la cual, se clona la rama *master* o *dev* del repositorio utilizando SourceTree. Después de esto la carpeta debería seguir vacía a excepción de una carpeta llamada .git que contiene toda la información del repositorio, en este punto ya es posible versionar cualquier cambio que se haga dentro de esa carpeta, pero como se ha explicado con anterioridad, no es una buena práctica realizar cambios directamente sobre las ramas *master* o *dev*, y para ello los cambios se van a hacer las ramas creadas para cada tarea.

En concreto, tanto en este proyecto como en el frontend se ha seguido el flujo de trabajo de gitflow, el cual se define un modelo estricto de ramificación diseñado alrededor de la publicación del proyecto. Proporciona un marco sólido para gestionar proyectos más grandes[12].

[11] El comando se trata de *certbot --apache*

[12] Explicación detallada: https://www.atlassian.com/es/git/tutorials/comparing-workflows/gitflow-workflow

La rama *master* almacena el historial de publicación oficial y la rama dev sirve como rama de integración para funciones. Por lo tanto, es conveniente etiquetar todas las confirmaciones a la rama *master* con la versión que se ha desplegado.

Cada nueva tarea tendría su propia rama, que, en vez de crearse a partir de *master*, se haría desde la rama dev y utilizarían esta como rama primaria. Cuando la tarea se completa, esta rama se vuelve a fusionar dentro de la rama dev, sin tocar en ningún momento la rama *master*.

Una vez el ciclo de desarrollo contiene todas las funcionalidades planificadas para esa versión validadas, se procedería a realizar la fusión de la rama *dev* dentro de *master*. En términos estrictos este último paso estaría mal hecho, ya que gitflow especifica que en el momento de desplegar se crea una nueva rama de desarrollo llamada reléase, sobre la cual se realizan las validaciones y correcciones en caso de que fuesen necesarias, y una vez terminado este proceso se fusionaría dentro de *master*. La principal ventaja de esto último es que la rama dev sería totalmente independiente del proceso de despliegue, lo que permitiría a equipos de trabajo mayor seguir trabajando sobre esta rama al margen de la reléase. En este caso, al contar con un solo desarrollador no aporta ninguna ventaja realizar este flujo de trabajo, pero en el caso de un crecimiento del equipo, el proyecto estaría preparado para seguir una metodología de trabajo más ágil y eficiente.

En este sentido, se tiene que crear la rama de la primera tarea que se va a realizar, esto se puede hacer desde SourceTree o desde la propia web de Gitlab. El problema de hacerlo desde SourceTree es que la rama no se asocia a la tarea que se ha creado y planificado mediante el sistema de tareas de Gitlab, así que la mejor opción siempre es crear la rama desde la página web cuando haya una tarea asociada.

En este punto se puede empezar con el desarrollo de la versión 0.0.0 de la aplicación, la cual se trata de una versión preliminar donde se realizan las tareas de creación las bases de datos e implementación de la estructura interna de la aplicación. Se realiza de esta manera porque es necesario llevar a cabo estas tareas antes de comenzar la implementación de las funcionalidades y las correcciones de la aplicación a partir de la versión 1.0.0.

6.3 Modelo de datos

Para implementar este modelo de datos lo mejor es utilizar un sistema gestor de bases de datos ya que permiten almacenar conjuntos de datos de una forma sistemática para poder utilizarlas en cualquier momento de forma rápida y estructurada.

Estos sistemas también proporcionan métodos para mantener la integridad de los datos, para administrar el acceso de usuarios a los datos, recuperar los datos tras un fallo del sistema y hacer copias de seguridad. Generalmente se accede a los datos mediante lenguajes de consulta, que se tratan de lenguajes de alto nivel que simplifican la tarea de construir las aplicaciones. Las bases de datos y los sistemas para su gestión son esenciales para cualquier área de negocio, y deben ser gestionados con esmero.

En específico se utiliza un tipo de sistema de gestión de base de datos que sigue el modelo relacional, que se fundamenta en el uso de relaciones pensadas como tablas compuestas por un conjunto de registros y columnas, se trata del modelo más utilizado para modelar datos reales y administrar los datos dinámicamente.

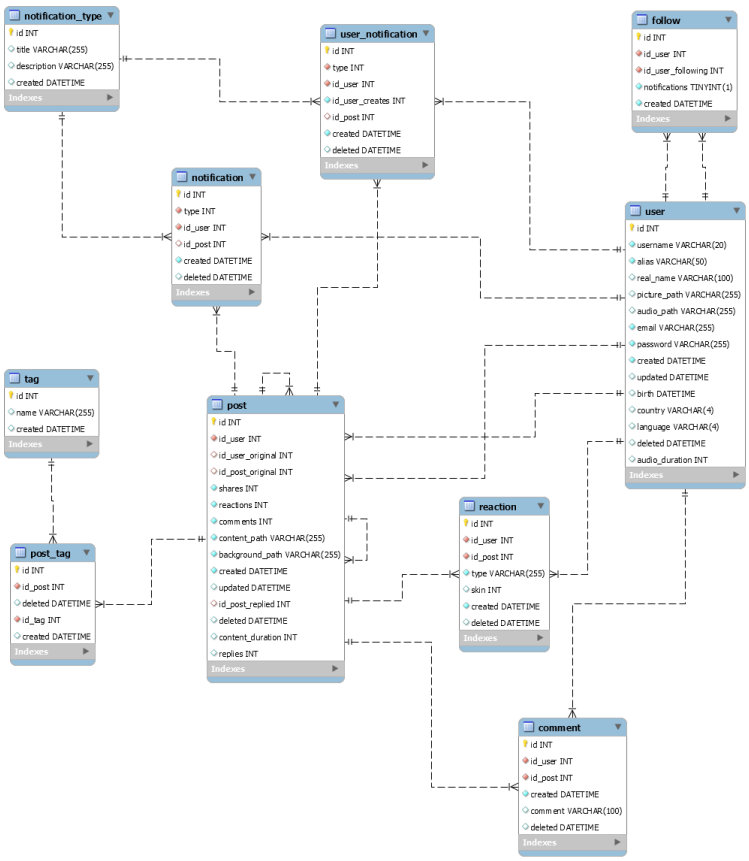


Figura 6.3: Diagrama entidad-relación de Soundn.

En la figura anterior del modelo de datos se muestra la estructura que van a tener los datos, el tipo de estos, la forma en que se relacionan y sus restricciones. Todo esto se ha definido así para representar los elementos de la realidad que intervienen en la aplicación.

Como ya se ha visto en el diagrama anterior, las tablas del modelo relacional se utilizan para modelar las entidades del mundo real que se quieren gestionar dentro de la aplicación, a su vez las características de estas entidades se implementan mediante las columnas que se especifican en el modelo relacional, y por supuesto, las filas de las tablas se corresponden a los registros de cada modelo que se crearan desde la aplicación, por ejemplo, en la tabla *User* cada fila se corresponde a cada usuario que se crea con diferentes características.

A continuación, se explica en profundidad todos los aspectos del modelo de datos que se ha diseñado para almacenar toda la información necesaria para cumplir los requisitos de la plataforma.

- **User**: En esta tabla se crea un registro cada vez que se crea un nuevo usuario en la aplicación y se almacena toda la información referente a los usuarios y sus perfiles mediante las siguientes columnas:
    - id: Clave primaria de la tabla, y campo a partir del cual se va a relacionar con el resto de las tablas.
    - username: Nombre del usuario, es único en la base de datos.
    - alias: Es el nombre del usuario que se va a mostrar en la aplicación, se hace así para que los usuarios puedan escoger su nombre sin ninguna limitación.
    - real_name: Nombre real del usuario que se almacena para futuras funcionalidades, pero de momento se trata de un dato opcional.
    - email: Correo electrónico del usuario, es único en la tabla.
    - picture_path: Ruta en la que se almacena la foto de perfil del usuario. Se almacena de esta forma para que la aplicación pueda acceder a ella y utilizar la foto de perfil cuando sea necesario.
    - audio_path: Ruta en la que almacena el servidor el audio de perfil, se utiliza para acceder a este siempre que la aplicación lo requiera
    - password: Contraseña del usuario, se encripta antes de almacenarse en la base de datos.
    - created: Fecha de creación del registro en la tabla
    - updated: Fecha de actualización de cualquier dato del registro en la tabla.
    - birth: Fecha de nacimiento del usuario.
    - country: Campo en formato ISO 3166-1 alfa-2 que indica el país que ha seleccionado el usuario en la aplicación.
    - language: Campo en formato ISO 3166-1 alfa-2 que indica el idioma que el usuario ha escogido como predeterminado de la aplicación.
    - deleted: Fecha en la que se desactiva el usuario.

Sin embargo, hay varias columnas que no se incluyen en las bases de datos, ya que se tratan de cálculos, a partir de otros datos, que realiza el servidor en tiempo de ejecución.

    - followers: Número de seguidores, se trata de un cálculo que se realiza sumando el número de registros de la tabla follow donde id_user_following de esta tabla, es igual al id del usuario actual.
    - following: Número de usuarios a los que sigue, se trata de la suma de registros de la tabla follow donde el id_user de la tabla es igual al id del, Sin embargo, estos cálculos solamente se realizan cuando se accede a la información de un solo usuario, en caso de pedirla para más de un usuario, por motivos de rendimiento no se realiza este cálculo, y por lo tanto no se adjunta a la información del usuario.
- **Follow**: Se trata de una tabla auxiliar en la que se guarda la información sobre el sistema de seguidores de la aplicación, para ello cuando un usuario sigue a otro se crea un registro en el cual se relaciona ese usuario con el que ha seguido mediante las columnas que se especifican a continuación:
    - Id: Clave primaria de la tabla.
    - id_user: identificador del usuario que realiza la acción de seguir a otro usuario. Se trata de una clave ajena que hace referencia a la clave primaria de la tabla user.
    - id_user_following: identificador del usuario al que sigue el usuario de la columna anterior, también es una clave ajena que relaciona con la tabla user a través de su clave primaria.

  - notifications: Se trata de un bit de control que indica si el usuario quiere recibir notificaciones relacionadas con el usuario al que acaba de seguir.
  - created: fecha en la que se crea el registro en la tabla.
- **Post**: Contiene toda la información acerca de una publicación de un usuario, así como la de las respuestas a las publicaciones, ya que en la aplicación existe un tipo de respuestas que también son una publicación. Las columnas que componen esta tabla son:
  - id: Clave primaria de la tabla.
  - id_user: id del usuario que ha creado la publicación, respuesta o una publicación para compartir la publicación de otro usuario. Además, esta columna es una clave ajena que se relaciona con la tabla user.
  - id_user_original: Usuario que crea la publicación, este campo es una clave ajena que hace referencia a la clave primaria de la tabla user. Este campo coincide con id_user cuando no se trata de una publicación para compartir otra debido a que este campo sirve para almacenar el identificador del usuario que ha creado la publicación que se ha compartido.
  - id_post_original: Post original que se ha compartido, se trata de una clave ajena que hace referencia a esta misma tabla. Por otro lado, este campo tiene el mismo valor que la clave primaria de la tabla id, si la publicación no comparte otra ya que su propósito es indicar el identificador de la publicación original.
  - shares: Número de veces que se ha compartido la publicación.
  - reactions: Número total de reacciones que ha recibido la publicación
  - comments: Número total de comentarios que se han hecho en la publicación.
  - content_path: Este campo almacena la ruta local en la que se guarda contenido de la publicación que siempre va a ser un audio. Esta ruta sirve para que la aplicación solicite el contenido al servidor a través de la API.
  - background_path: Ruta local en la que se guarda la imagen de fondo de la publicación, esta ruta sirve para que la aplicación se solicite la imagen al servidor cuando esta la necesite.
  - date: fecha en la que se crea la publicación.
  - update_date: Fecha de actualización de cualquiera de los campos de la tabla.
  - id_post_replied: identificador del post al que se responde, este campo está vacío si no se responde a nada.
  - deleted: Fecha en la que se elimina la publicación.
- **Comment**: Se trata de una tabla para guardar los datos de los comentarios de texto que se realizan a las publicaciones por medio de estos campos:
  - id: Clave primaria de la tabla
  - id_user: Identificador del usuario que crea el comentario, se trata de una clave ajena que hace referencia a la clave primaria de la tabla user.
  - id_post: identificador de la publicación en la que se comenta. Esta columna es una clave ajena que la relaciona con la clave primaria de la tabla post.
  - date: Fecha en la que se crea el comentario.
  - comment: Contenido del comentario.
  - deleted: Fecha de eliminación del comentario.

- **Tag**: Almacena la información necesaria de las etiquetas que se le pueden asociar a las publicaciones, y para ello cuenta con las siguientes de columnas:
    - id: Clave primaria de la tabla
    - tag: Nombre de la etiqueta
    - date: fecha de creación
- **Post_tag**: Se trata de la relación entre etiquetas y las publicaciones a las que han sido asociadas, para ello se utilizan estas columnas:
    - id: Clave primaria de la tabla
    - id_post: Identificador de la publicación. Es una clave ajena que hace referencia a la clave primaria de la tabla post.
    - id_tag: Identificador del tag y clave ajena que se refiere a la clave primaria de la tabla tag.
- **Notification**: En esta tabla se almacenan los datos de las notificaciones que se generan cuando un usuario realiza una acción determinada, para así mostrárselas a aquellos de sus seguidores que han aceptado recibir notificaciones relacionadas con este usuario. Se ha implementado de esta manera para que con el mismo registro en esta tabla se les notifique a todos los seguidores del usuario que ha generado la notificación, y así ahorramos la creación de un gran número de filas. Para ello se ha compuesto la tabla con las siguientes columnas:
    - id: Clave primaria de la tabla.
    - title: Titulo de la notificación. Está estrechamente relacionado con el tipo de notificación.
    - description: Descripción de la notificación. En gran medida dependerá del tipo de notificación.
    - type: Tipo de notificación, se refiere a la acción que se ha realizado para generar la notificación. Esta columna se trata de una clave ajena de la tabla *notification_type* que se refiere a su columna id.
    - id_user: identificador del usuario que ha realizado la acción que genera la notificación. Es una clave ajena que referencia a la tabla user.
    - id_post: En caso de ser una acción sobre alguna publicación, se indica en este campo el identificador de esa publicación. Este campo es una clave ajena que referencia a la tabla post.
    - created: Fecha en la que se crea el registro en la tabla.
    - deleted: Fecha en la que se elimina la notificación.
- **User_notification**: Consiste en una tabla muy similar a la anterior, con la diferencia de que sirve para almacenar notificaciones sobre las acciones que se han realizado a un usuario o sus publicaciones, por lo tanto, las columnas de esta tabla son:
    - id: Clave primaria de la tabla.
    - title: Titulo de la notificación.
    - description: Descripción de la notificación.
    - id_user: Identificador del usuario sobre el que se ha realizado la acción que se notifica, es decir al que va dirigida la notificación. Esta columna es una clave ajena que la relaciona con la clave primaria de la tabla user.
    - id_user_created: usuario que ha realizado la acción que ha generado la notificación.
    - id_post: id del post relacionado con la alerta.
    - type: Tipo de notificación, se refiere a la acción que se ha realizado para generar la notificación. Esta columna se trata de una clave ajena de la tabla *notification_type* que se refiere a su columna id.

    - created: Fecha en la que se crea el registro en la tabla.
    - deleted: fecha en la que se elimina la notificación.
- **Reaction**: En esta tabla se almacenan las reacciones de los usuarios a las publicaciones, para ello se ha construido con las siguientes columnas:
    - id: Clave primaria de la tabla.
    - id_post: Identificador de la publicación que ha recibido la reacción. Se trata de una clave ajena que referencia a la tabla post a través de su clave primaria.
    - id_user: Identificador del usuario que ha reaccionado a la publicación. Es una clave ajena que relaciona la columna con la clave primaria de la de la tabla user.
    - type: Tipo de reacción, estos tipos están predefinidos y estarán ligados a unos emoticonos que expresen la reacción.
    - created: Fecha en la que se crea el registro en la tabla.
    - delete: Fecha en la que se elimina la reacción.
- **Notification_type**: Se trata de una tabla auxiliar para almacenar los tipos de notificación que existen en la plataforma, de esta manera se puede almacenar su descripción en base de datos y compartirla entre las dos tablas de notificaciones.
    - id: Clave primaria de la tabla.
    - title: Titulo del tipo de notificación, es que finalmente se muestra en la aplicación.
    - description: Se trata de una descripción breve de la notificación que se genera con ese tipo.
    - created: Fecha en la que se crea un tipo de notificación.

Para implementar el modelo de base de datos que se ha diseñado para la red social se crea un script SQL con todas las sentencias necesarias para crear las tablas, columnas, relaciones y restricciones en el servidor MySQL, de esta manera, ejecutando manualmente este script en la instancia de MySQL server, se crea una base de datos con modelo de datos necesario para almacenar la información necesaria para que funcione la aplicación. Entonces simplemente ejecutando el script sobre la instancia que se ha instalado previamente, se crea una nueva base de datos para desarrollar la aplicación con el modelo de datos diseñado. De igual forma, también hay que ejecutar el script antes de realizar un despliegue con cambio de modelo de datos, primero se hace en el entorno de pruebas y después de comprobar que todo funciona adecuadamente, se procede a realizar esta tarea en la base de datos del entorno productivo, en este caso de creación de las bases de datos también se tiene que realizar estas tareas.

El flujo de trabajo de gestión de las bases de datos en este proyecto siempre consiste en la ejecución de un script SQL en la instancia local con las modificaciones necesarias para realizar el desarrollo de la tarea correspondiente. Luego, al terminar el desarrollo y antes de fusionar los cambios en la rama dev, se ejecuta ese mismo script sobre la base de datos de pruebas de la instancia de MySQL server del VPS. Al terminar esto se procede con la fusión de la rama en dev, lo cual provocaría la ejecución del pipeline de despliegue automático en el entorno de pruebas, de esta manera ya es posible realizar todo tipo de pruebas y validaciones de los cambios. Una vez se verifica que todo está correcto se procede a ejecutar el script en la base de datos de producción del VPS, y seguidamente se fusiona la rama dev en *master* para desencadenar la creación y ejecución del pipeline encargado del despliegue automático en el entorno de producción.

6.4 Inicialización y organización del proyecto backend

Este punto de la implementación se ha empezado con la tarea de estructuración del proyecto, donde se ha inicializado del proyecto, organizado las carpetas e implementado los modelos. Esto se ha llevado a cabo, en primer lugar, ejecutando el comando *npm init* en la carpeta del proyecto para crear los archivos *package.json* que incluyen información del proyecto y sus dependencias, seguidamente se ha creado la estructura de carpetas del proyecto de acorde a la arquitectura que se ha diseñado con anterioridad, el resultado de todo agrupa los enrutadores y controladores, y estos grupos a su vez se han agrupado dentro de la *API* de la aplicación. Luego toda aquella configuración necesaria en la aplicación se reúne en *config*, ya que, en esta carpeta, por ejemplo, se encontraría la configuración de conexión a la base de datos, o la de seguridad de la API.

Después de esto, en la carpeta *models* se han agrupado todos los modelos con los que se realiza el mapeo objeto – relación de base de datos. Por otro lado, los servicios que contienen la lógica de negocio se han juntado en *services*, y las utilidades que contienen funciones que se utilizan en todo el proyecto se han agrupado en *utils*. Para terminar, hace falta agrupar los ficheros de cambios en base datos, para ello, se ha creado una carpeta de *scripts* donde se almacenarán este tipo de archivos.

Seguidamente, para implementar los modelos se ha creado un fichero en la carpeta *models* por cada tabla de las bases de datos, estos ficheros contendrían toda la información de la estructura que tiene cada tabla. Un ejemplo de esta especificación se puede observar en la Figura 6.4 del modelo de *reaction*.

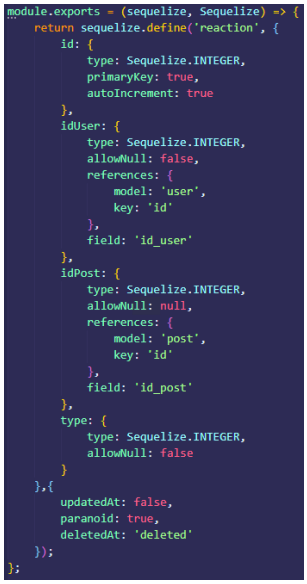


Figura 6.4: Implementación del modelo de follow mediante Sequelize.

Previamente se ha establecido que cada tarea que se termina se sube a la rama *dev*, pero en este caso todavía no se despliega debido a que en esta versión todavía no se ha implementado el sistema de despliegues automáticos en el repositorio del proyecto. De cualquier manera, al finalizar todas las tareas de la versión 0.0.0 y haber subido sus respectivas ramas a *dev*, ya se puede continuar con la fusión de la rama *dev* en *master*, y de esta manera subir su versión sin etiquetar debido a que se trata de una versión preliminar de inicialización del proyecto.

6.5 Integración y despliegues continuos

La siguiente versión que se ha desarrollado es la 1.0.0, que contenía tareas de gran parte la aplicación servidora del *backend* que se ha diseñado, y la implementación de los despliegues automáticos de la aplicación. De esta versión lo más destacable es la implementación de los despliegues automáticos, como previamente ya se han instalado y configurado el ejecutor de Gitlab y el gestor de procesos PM2.

Por lo tanto, para terminar de construir correctamente esta infraestructura de despliegue automático, solamente ha sido necesario especificar un fichero YML nombrado

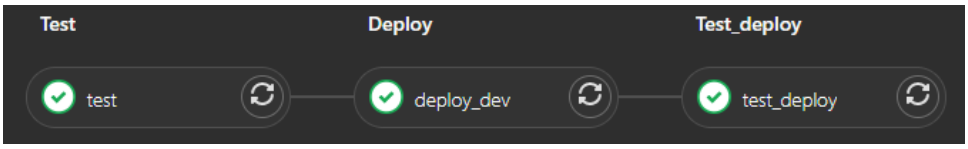


Figura 6.5: Etapas de un pipeline de la rama *dev*.

obligatoriamente como *.gitlab-ci.yml*, con todas las acciones necesarias para llevar a cabo la compilación, pruebas y despliegue del software. Para empezar, se ha organizado y se especificado cada una de las etapas, los trabajos de los que están compuestas, y los comandos que se ejecutan en cada trabajo. Además, también se ha tenido en cuenta que todos los trabajos deben de estar etiquetados para ser ejecutados.

Gitlab detecta automáticamente cuando hay un fichero *.gitlab-ci.yml* en el repositorio, y cada vez que se realizan confirmaciones sobre las ramas especificadas, se ejecuta un pipeline con aquello que se ha especificado en este fichero, y dividido por las etapas que también se especifican en este mismo fichero.

En este caso, se ha determinado una primera etapa llamada *test*, que constaría solamente de un trabajo con este mismo nombre, en el cual se han definido los comandos que realizarían las pruebas automáticas del sistema. Esta etapa se ha realizado antes que cualquier otra, ya que es necesario comprobar que todo funciona correctamente antes de llevar a cabo cualquier otra acción. El trabajo de esta etapa no es necesario llevarlo a cabo en el VPS, ya que solamente se ejecutan las pruebas del software, y no se mueven ficheros en el servidor, por lo tanto, a este trabajo se le asigna una etiqueta *test*, dado que el ejecutor instalado en el VPS se ha configurado para ejecutar solamente los trabajos que tengan la etiqueta *deploy*.

La siguiente etapa es la de despliegue, que tal y como se puede ver en la Figura 6.5, se ha nombrado como *deploy*. Esta etapa se ha compuesto de dos trabajos, debido a que se efectúan diferentes acciones para desplegar en los dos entornos que componen la infraestructura de la aplicación. Como consecuencia de esto, se ha determinado un trabajo llamado *deploy_dev* que

realizaría el despliegue del entorno de pruebas, solamente cuando se realicen confirmaciones en la rama *dev*. De esta misma manera, en esta etapa también se ha establecido un trabajo llamado *deploy*, que desplegaría la aplicación del entorno de producción cada vez que se hiciesen confirmaciones en la rama *master*. Cabe destacar que, debido a esta configuración de trabajos, esta etapa es exclusiva para las ramas dev y *master*, por lo tanto, el resto de las ramas no incluirían la etapa *deploy* en la ejecución de sus pipelines. Finalmente, estos trabajos se han etiquetado como *deploy*, para que puedan ejecutarse en el VPS donde se aloja el backend de la aplicación, mediante el ejecutor que se ha instalado y configurado previamente.

En la última parte de la implementación de los despliegues automáticos, se ha especificado una etapa denominada *test_deploy*, que prueba que el despliegue ha funcionado correctamente mediante un trabajo llamado *test_deploy*. Obviamente, esta etapa se ha configurado para ejecutarse siempre después de haber terminado la ejecución de la etapa *deploy*. De la misma forma que se ha hecho en la etapa anterior, en el trabajo de esta etapa se ha establecido la etiqueta *deploy*, ya que este trabajo requiere utilizar el ejecutor del VPS para realizar el comando que verifica que la aplicación está funcionando correctamente en la instancia.

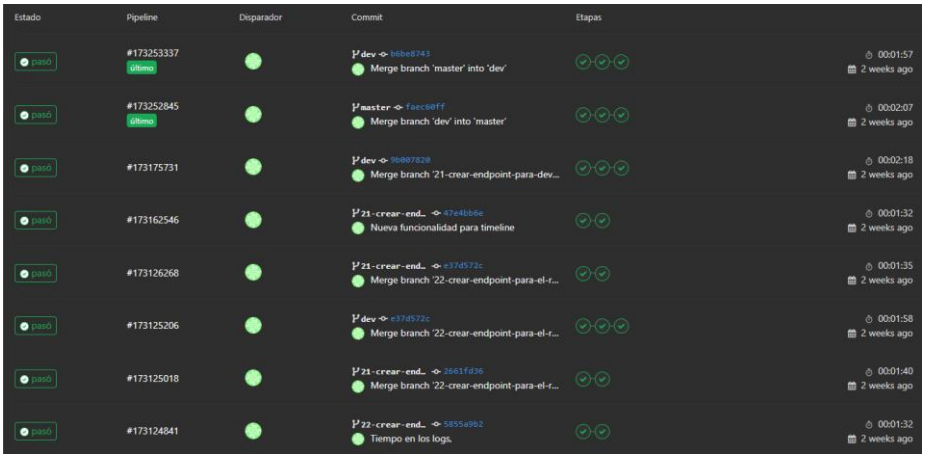


Figura 6.6: Pipelines ejecutados en las últimas semanas de desarrollo.

Tal y como se ha introducido previamente, realizar una confirmación incluyendo este fichero YML, Gitlab lo reconoce automáticamente y procede a ejecutar los pipelines de acorde a lo que se ha especificado en este fichero. Por eso, después de fusionar la rama de esta tarea en *dev*, cualquier otra rama que se fusione en *dev* a partir de ahora desencadenara el despliegue automático de la forma que se ha especificado, ya que realmente *dev* ahora también contiene el archivo YML. Además, si se desea ejecutar la etapa de pruebas para una rama específica durante el desarrollo de esta versión, simplemente se debería crear el archivo YML en la rama mediante la fusión de *dev* en esta rama, como resultado se conseguiría que cada vez que se confirmen cambios en la rama específica, se realicen las pruebas a través de un ejecutor compartido de los servidores de Gitlab.

En las siguientes versiones, después de que estos cambios se fusionen en la rama *master*, ya no sería necesario juntar *dev* en cada rama para desencadenar las pruebas automáticas, ya que las ramas que se creen a partir de ese momento ya tendrán el archivo YML debido a que estas suelen ser copias de la rama *master*.

6.6 Programación del backend

Avanzando con el desarrollo, se puede observar en la Figura 6.7, que las 12 tareas restantes de la versión 1.0.0 consisten íntegramente en la implementación de las características del servidor NodeJS. Al ser este el caso, solamente se va a entrar en detalle en aquellas tareas, funcionalidades o comunicaciones, que tengan particularidades en el desarrollo o sean de especial relevancia en el proyecto.

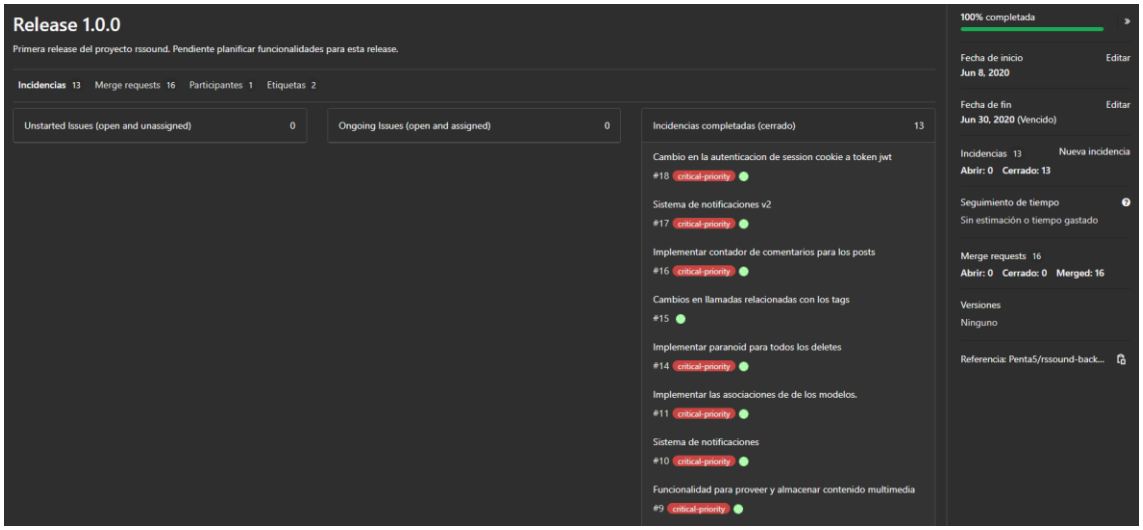


Figura 6.7: Tareas de la versión 1.0.0.

La ausencia de particularidades de los controladores y servicios de la mayoría de los modelos es muy relevante en el proyecto, ya que, para mejorar la reusabilidad, legibilidad, mantenibilidad y calidad del código, se ha seguido la especificación de la arquitectura, donde se especificaba que se debían crear elementos comunes en cada capa. Para ello, lo mejor ha sido generalizar las funciones similares entre sí para realizar una implementación común de estas, y que se puedan utilizar sin importar el servicio o modelo necesarios dentro de cada función. Sería algo similar a las funciones genéricas de Java, donde los tipos genéricos serian el servicio y el modelo. Para realizar esto, lo primero que se ha hecho es crear en la carpeta de rutas un fichero *common.js*, donde se implementan todas las funciones comunes de todas las rutas utilizando un servicio genérico que será del tipo que cada controlador implemente.

```
/**
 * Recupera el router con los metodos GET, POST y DELETE por defecto
 * @param {*} express
 * @param {*} asyncHandler
 * @param {*} service
 * @return {Router}
 */
exports.getCommonRouter = (express, asyncHandler, service, includeOwnUser) => {
    var router = express.Router();
    router.route('/')
        .get(asyncHandler(this.defaultGet(service)))
        .post(asyncHandler(this.defaultPost(service)))
        .delete(asyncHandler(this.defaultDelete(service)));
    router.get('/first', asyncHandler(this.defaultGetFirst(service)));
    if(includeOwnUser){
        router.get('/own', this.defaultGetOwn(service));
        router.get('first/own', this.defaultGetFirstOwn(service));
    }
    return router;
};
```

Figura 6.8: Función que crea y devuelve el router común.

En la Figura 6.8 se puede observar que el enrutador común se ha implementado mediante una función, al cual se le pasa el servicio por parámetro, para crear y devolver un enrutador enlazado a la raíz, con funciones por defecto que utilizan este servicio, para implementar las funciones que se enlazan a los métodos GET, POST, PATCH y DELETE. Además, a este enrutador habitualmente, a través del método GET, también se ha direccionado la ruta *first*, la cual se le enlaza una función por defecto, para obtener el primer registro de base de datos que cumpla ciertas características. En esta función generadora, finalmente se han implementado las rutas *own*, que se utilizan para devolver datos del usuario actual, sin necesidad de que la petición especifique el identificador del usuario entre su parámetro, o su cuerpo, ya que gracias a la estrategia JWT, se puede saber que usuario está realizando la petición sin especificarlo explícitamente. Pero esta ruta *own* es condicional, ya que hay ciertas entidades del modelo de datos que todavía no necesitan que se les direccione esta funcionalidad, por lo tanto, mediante la variable *includeOwnUser*, se ha dado la posibilidad elegir si incluir este direccionamiento en su enrutador.

```
/**
 * Metodo GET por defecto
 * @param {Service} service
 * @returns {Function}
 */
exports.defaultGet = service => {
    return async (req, res) => {
        Object.keys(req.query).length ? res.json( await service.getAllByCondition(req.query)) : res.status(400).send({
            err: 'No hay criterios de busqueda'
        });
    };
};
```

Figura 6.9: Función GET por defecto.

Profundizando en la implementación de las funciones por defecto, en la Figura 6.9 se puede observar una función a la que solamente se le pasa por parámetro el servicio, y devuelve la función asíncrona por defecto que habitualmente se enlaza con el método GET del enrutador. Como consecuencia, esta función realiza la llamada a la función del servicio, que se encarga de recuperar todos los registros de base de datos que cumplen las condiciones especificadas en los parámetros de la petición. Este servicio puede ser de cualquier modelo de datos, por lo tanto, sus funcionalidades son estándar y comunes para la mayoría de los modelos, sin embargo, para crear el enrutador se debe de pasar el servicio específico del modelo como se puede ver en la Figura 6.10.

Figura 6.10: Varias implementaciones a partir del enrutador común.

Cada una de las rutas puede utilizar la implementación del enrutador común, pero con sus propios servicios. Además, a este enrutador se le puede extender las rutas con funcionalidad especifica asociada, tal y como se puede visualizar en la Figura 6.10, en la ruta de las notificaciones que agrega su propio direccionamiento a *own* o también en la ruta de los tags del post que extiende una ruta a *hot* especifica de este modelo de datos.

Por otro lado, en la capa de servicios ocurre algo similar, por lo tanto, en la carpeta de servicios se ha implementado otro fichero *common.js*, con todas las funciones comunes de los servicios. Cada una con el acceso a base de datos a través de un modelo genérico, para que de esta forma cada servicio utilice el modelo que necesite.

Figura 6.11: Implementación de la función que devuelve el servicio común.

La Figura 6.11 muestra la función que devuelve un objeto servicio, con todas las funciones comunes de todos los servicios. Además, se puede observar como el modelo se pasa por parámetro para que utilizarlo en el objeto servicio, y que este a su vez pueda ser utilizado por cualquier ruta. En el fragmento de código de la Figura 6.11 se muestran a modo de ejemplo, dos de las funciones del servicio que utilizan las funciones *find* del modelo que proporciona Sequelize para realizar las consultas en base de datos.

```
const bcrypt = require('bcrypt');
var BCRYPT_SALT_ROUNDS = 12;

Object.assign(exports, commonServices(models.user));

/**@override */
exports.getByPk = async pk => {
    const user = await models.user.findOne({
        attributes: {
            exclude: ['password']
        },
        where: {
            id: pk
        },
        raw: true
    });
    await addFollowStats(user);
    return user;
};
```

Figura 6.12: Servicio del modelo user.

De una forma similar a las rutas, cada servicio puede elegir con que modelo se va a implementar la funcionalidad común de la que va a hacer uso. No obstante, existe una importante diferencia ya que las rutas a partir de su servicio obtenían un enrutador que se asociaba a su dirección, mientras que los servicios obtienen la funcionalidad común para su modelo, tal y como se pude observar en la Figura 6.12. Asimismo, se puede ver que esta forma de obtener la funcionalidad común permite sobrescribirla, tal y como se hace para la función *getByPk* que obtiene un registro en base de datos a partir de la clave primaria que se pasa por parámetro. Si se compara con la función común *getByPk* de la Figura 6.12 se puede ver que llaman a dos funciones *find* diferentes del modelo, ya que el servicio de usuario tiene que excluir la contraseña de los datos que devuelve la consulta, lo cual no se puede hacer mediante la función *findByPk* del modelo de Sequelize debido a que no tiene esta funcionalidad, mientras que *findOne* sí que la tiene.

La primera tarea de interés en mencionar es el desarrollo de la funcionalidad para almacenar y recuperar archivos multimedia, ya que ha sido necesario implementar cierta funcionalidad adicional en los controladores de las rutas de usuario y las publicaciones, para encargarse de las peticiones *multipart/form-data*, las cuales se han utilizado para incluir archivos multimedia. Esto ha sido así porque con Express no se puede implementar la funcionalidad necesaria para aceptar este tipo de peticiones, por eso se ha utilizado una tecnología middleware llamada Multer, que en este caso se ha utilizado para suplir esta necesidad.

Lo primero que se ha llevado a cabo es la forma de almacenar los archivos multimedia, para ello se va a seguir el diseño que se ha especificado en capítulos anteriores, y es que, como se ha explicado previamente, este diseño resuelve posibles problemas de rendimiento que pueden surgir al recuperar los archivos.

```
const multer = require('multer');
const fs = require('fs');
const { v4: uuidv4 } = require('uuid');
const storage = multer.diskStorage({
    destination: function (req, file, callback) {
        const now = new Date();
        const path = `media/user/${now.getUTCFullYear()}/${now.getUTCMonth() + 1}/${file.fieldname}`;
        fs.mkdirSync(path, {
            recursive: true
        });
        callback(null, path);
    },
    filename: function (req, file, callback) {
        callback(null, uuidv4() + '.' + file.originalname.split('.').pop());
    }
});

const upload = multer({ storage: storage });

exports.cpUpload = upload.fields([{
    name: 'picture',
    maxCount: 1
}, {
    name: 'audio',
    maxCount: 1
}]);
```

```
const fs = require('fs');
const { v4: uuidv4 } = require('uuid');
const storage = multer.diskStorage({
    destination: function(req, file, callback) {
        let now = new Date();
        let path = `media/post/${now.getUTCFullYear()}/${now.getUTCMonth() + 1}/${file.fieldname}`;
        fs.mkdirSync(path, {
            recursive: true
        });
        callback(null, path);
    },
    filename: function(req, file, callback) {
        callback(null, uuidv4() + '.' + file.originalname.split('.').pop());
    }
});

const upload = multer({ storage: storage });

exports.cpUpload = upload.fields([{
    name: 'background',
    maxCount: 1
}, {
    name: 'content',
    maxCount: 1
}]);
```

Figura 6.13: Implementación del almacenamiento de archivos.

Como se puede observar en la Figura 6.13, ambas implementaciones son muy similares, ya que se diferencian en que los archivos que se reciben mediante las peticiones de la ruta de usuario se almacenan en la ruta relativa "media/user/…", mientras las que se reciben por la ruta de publicaciones se almacenan en la ruta relativa de "media/post/…". En ambos casos, dentro de estas carpetas, su estructura se ha organizado en base al año y el mes en el que se recibe el archivo, y por el tipo de este. De esta manera, dentro de las carpetas de *post* y *user* la implementación crea una carpeta automáticamente para cada año, a su vez dentro de cada año, una para cada mes, y finalmente dentro de cada mes se crea una carpeta para cada tipo de archivo, ya que los archivos que se reciben a través *post* pueden ser *background* o *content*, mientras que los archivos de *user* pueden ser de tipo *picture* o *audio*. En la siguiente ilustración se puede ver un ejemplo de todo esto para archivos creados en Julio de 2020.

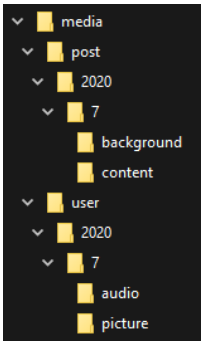


Figura 6.14: Ejemplo de árbol de carpetas.

Como se puede ver en la Figura 6.14, la creación de estas carpetas es diferida, es decir que se aplaza hasta el momento en el que se recibe un archivo para el cual todavía no se han creado las carpetas donde guardarlo, ya sea por la fecha en la que se recibe o el tipo de archivo. De esta manera se automatiza el trabajo de inicializar previamente un conjunto determinado de carpetas, y se evita cualquier error que pueda conllevar esta estrategia.

En último lugar, se ha tenido en cuenta que los archivos se sobrescriben cuando se guarda un archivo con el mismo nombre de otro que ya existe en la misma carpeta. Para resolver esto programáticamente hay dos métodos diferentes, el primero consistía en revisar todos los nombres de los archivos de la carpeta y cambiar el nombre para que no coincida con ningún nombre, el problema de esta solución es que resulta ineficiente a medida que va creciendo el número de archivos dentro de una misma carpeta, ya que requiere leer todos los archivos de forma secuencial, y cuantos más archivos haya más va a tardar esta lectura. Por otro lado, la solución que se ha implementado consiste en utilizar un generador de identificadores únicos, para establecer los nombres de los archivos durante su almacenamiento, de esta manera no se necesita leer los archivos de la carpeta, ya que el mediante el generador prácticamente se asegura que el nombre que se establece es único. Para la implementar esta solución se utiliza el generador de UUID v4 que se incluye en el módulo *uuid* para NodeJS, ya que, con este paquete, la probabilidad de encontrar un duplicado dentro de los 103 billones de UUID de la versión 4 es de uno en mil millones, lo que resulta más que suficiente para el sistema de guardado de archivos. Por lo tanto, en el motor de almacenamiento en disco de Multer se ha establecido que el nombre del fichero se compone partir del uuid generado, y la extensión del archivo original.

Hasta el momento solamente se ha implementado el guardado de archivos de las entidades *user* y *post*, por lo tanto, faltaría por programar la capa controladora, la capa de servicio, y la capa de acceso a datos de estas entidades. Pero debido a una limitación de Multer existen varias

particularidades, ya que esta tecnología solamente guarda los archivos al recibir la petición y siempre antes de ejecutar cualquier funcionalidad asociada a la ruta, es decir antes de ejecutar los controladores, la lógica de negocio del servicio, y la persistencia en base de datos. Esto significa que, si hubiese cualquier error durante la ejecución de la lógica de negocio, o durante la persistencia en base de datos se podría originar problemas de integridad de datos, y de espacio en el disco porque se crearían archivos innecesarios descontroladamente.

La solución que se ha implementado consiste, en eliminar los archivos creados previamente si hay un error durante la ejecución de las funciones asociadas a cada ruta. Para ello, las funciones *save*, *update* y *delete* de los servicios de user y post se construyen para que los cambios en base de datos se realicen al final de toda la lógica de negocio, ya que de esta manera nos aseguramos de que los errores se producirían antes o durante la modificación en base de datos, nunca después, ya que esto implicaría, que en caso de fallo, además de borrar el archivo, se debería buscar si el registro se ha creado o actualizado, y devolverlo a su valor anterior, o eliminarlo si no existía previamente.

A continuación, las rutas que utilizan Multer solamente realizan las llamadas a la función del servicio que les corresponda, porque el control de errores solamente se ha llevado a cabo para las funciones de los servicios correspondientes, mientras que este control no se ha implementado para las rutas, por motivos de simplificación de código. Teniendo esto en cuenta, para las promesas que devuelven las funciones de estos servicios se ha implementado el callback del método *then* de la promesa, para cuando esta se ha resuelto correctamente, y también el callback del método catch, en el cual se incluye la funcionalidad de borrado de los archivos que se han recibido en la petición, ya que este método solamente se ejecuta cuando la promesa se ha marcado como rechazada debido a un error durante su ejecución.

Como se ha venido explicando hasta ahora, el diseño de la forma de almacenar los archivos se enfoca principalmente en la eficiencia y rendimiento a la hora de recuperar estos archivos desde el exterior, ya que el tiempo de lectura aumenta cuantos más ficheros haya en una misma carpeta, no obstante, una buena solución a esto se ha conseguido con la implementación previamente explicada, dado que lo organiza todo en diferentes subcarpetas, y define un crecimiento moderado de estas.

Finalmente, para solicitar estos ficheros desde el exterior se ha implementado un servicio de archivos estáticos en Express que escucha las solicitudes en la ruta *media* tal y como se puede apreciar en la Figura 6.15

```
router.use('/media', passport.authenticate('jwt', {
    session: false
}), express.static('media'));
```

Figura 6.15: Implementación del acceso al directorio estático media.

En la Figura 6.15 también se puede observar que este *endpoint* utiliza la estrategia de autenticación mediante token JWT para poder acceder a los archivos, si tenemos en cuenta que solamente los usuarios que inician sesión con su usuario y contraseña pueden obtener el token JWT, se puede afirmar que solo los usuarios que se han autenticado correctamente pueden recuperar los archivos del directorio *media*.

Esta función del middleware permite acceder a un archivo determinado simplemente especificando la ubicación del archivo dentro de la propia ruta de la llamada. Por ejemplo, para acceder a un archivo de la carpeta background creado en Julio de 2020, la ruta de la llamada debería ser *api/v1/media/post/2020/7/background/1f448ba3(...)62f94.jpg* donde *api/v1* es el prefijo de la ruta principal, por su parte */media* es el prefijo donde se ha montado la vía de acceso al servicio del directorio estático, y finalmente, */post/2020/7/background/1f448ba3(...)62f94.jpg* es la ruta de acceso al fichero a través del servicio de archivos estáticos.

A pesar de que el acceso a los archivos se ha realizado mediante una implementación simple, provee de una forma fácil y eficiente de acceder a los archivos que necesita la aplicación de la red social. Su punto más fuerte viene dado por el funcionamiento en conjunto con todo el sistema, porque en base de datos se guardan las rutas de los archivos que se almacenan en el servidor, esto quiere decir que al solicitar los datos de un usuario o una publicación también se les transmite la ruta en la que se encuentran los archivos asociados a cada registro, por lo tanto, el frontend para recuperar los archivos simplemente tiene que realizar una concatenación de la URL de la API, con las rutas de los archivos que ha recibido como respuestas de peticiones anteriores.

Las siguientes tareas relevantes del proyecto son aquellas en las que se ha implementado la seguridad del sistema, debido a que se trata del punto de entrada a la API, y resulta fundamental en la arquitectura de la aplicación. Lo primero que se ha llevado a cabo es la configuración de las estrategias de autenticación de Passport, y para ello se ha creado un fichero *passport.js* en la carpeta *config*, y dentro de este fichero se han implementado por separado las configuraciones de las estrategias locales mediante usuario y contraseña, para el inicio de sesión, y para el registro de un usuario. Esto se ha hecho así porque realizan dos acciones diferentes que tienen que estar separadas, por una parte, la estrategia de inicio de sesión se ha configurado de manera que verifique que las credenciales son correctas, buscando en base de datos un usuario que cumpla con el nombre de usuario, y la contraseña que se han introducido al iniciar sesión, y en caso de cumplir con estas condiciones, también se generaría un token JWT para acceder al resto de rutas. Por otro lado, la estrategia local de registro se ha configurado para guardar los datos del usuario, y controlar que en caso de error se borran todos los archivos que se han recibido en la petición junto a los datos del usuario.

La última estrategia que se ha implementado en este fichero es la autenticación mediante JWT, mediante la cual se verifica que el usuario almacenado en el token JWT existe en base de datos, ya que estos tokens solamente se han generado para aquellos usuarios que se han autenticado mediante su usuario y contraseña.

Para terminar, se crean las rutas en la API para el inicio de sesión y el registro de usuarios, y se les asocian las estrategias correspondientes que se han configurado previamente en el fichero *passport.js*. Por otra parte, al resto de rutas de la API se le asocia la estrategia de autenticación mediante JWT, para controlar que a estas rutas solamente pueden acceder usuarios que han recibido el token JWT, al haberse autenticado previamente en la aplicación mediante su usuario y contraseña.

En el resto de las versiones que se han programado, se han implementado la gran mayoría de la funcionalidad requerida, lo cual ha consistido en completar cada una de las capas de la arquitectura para cada una de las entidades que existen. Además, durante este periodo también

se han creado todas las funcionalidades extra, y las excepciones que podrían tener algunas de las entidades para poder implementar la funcionalidad requerida. Aunque parezca bastante trabajo debido a la cantidad de entidades que se han diseñado, realmente una gran parte de tiempo se ha empleado en la corrección de todo lo que se iba haciendo, para que su funcionamiento sea el adecuado antes de realizar cada integración, y por supuesto antes de la versión final del servidor.

6.7 Preparación para el desarrollo del frontend

El frontend, como ya se ha venido comentando hasta ahora se ha realizado mediante la tecnología React Native, que sirve para el desarrollo de la aplicación móvil para las dos plataformas compartiendo el mismo código, además de esto, se ha utilizado un repositorio de Gitlab aparte para versionar este código, y finalmente para validar todo aquello que se va implementando se ha hecho uso del emulador de Android, que provee una emulación de un dispositivo con el sistema operativo Android, sobre el cual se instala la aplicación para poder visualizar los cambios en caliente, es decir mientras se está desarrollando.

De la misma manera que se ha hecho con el backend, lo primero es poner en marcha el sistema de versiones para el código, pero con la diferencia de que no se necesita realizar un sistema de integración y despliegue continuos, ya que en la aplicación móvil solamente se despliega al terminar el proyecto, es decir una única vez, esto implica que las automatizaciones son mucho menos rentables debido a que el coste temporal no se amortiza si solamente se va a desplegar una vez. Aunque es evidente que en el futuro todos los cambios que se realicen en la aplicación se van a tener que subir en nuevas versiones, lo que supone que las automatizaciones empezarían a ser más rentables debido, al alto gasto temporal en tareas de despliegue.

Después de inicializar el repositorio se han instalado y puesto en marcha todas las tecnologías para el desarrollo de la aplicación, lo primero ha sido la instalación del SDK de Android y el emulador para el desarrollo de la aplicación, para ello se ha hecho uso del SDK Manager de IntelliJ IDEA, esto es así porque este programa ya se había instalado previamente de anteriores proyectos, y al tener esta funcionalidad se ahorraba tiempo de instalación y configuración de Android Studio. En concreto el nivel de API del SDK utilizado en este proyecto es el 29, este nivel de API solamente permite instalar la versión 10 de Android en el emulador, esta decisión esta fomentada por el hecho de que se trata una de las versiones más recientes de esta plataforma, ya que su fecha de lanzamiento fue en 2019 y actualmente es la versión más utilizada a nivel mundial.

Después de esto, ya se puede realizar la configuración del emulador con el SDK que se acaba de instalar mediante el AVD Manager de IntelliJ IDEA, que se trata del gestor de dispositivos virtuales oficial de Android. Este programa facilita la creación y administración de los emuladores, por eso la creación y configuración del emulador se ha realizado siguiendo los pasos del asistente de creación, en los cuales se podía seleccionar el SDK, la resolución de la pantalla o el tamaño de las memorias.

A continuación, mediante Git se ha clonado en local el repositorio que se había inicializado previamente, ya que de esta manera se crea una carpeta con los contenidos que tiene el repositorio remoto en el momento del clonado, en este caso estaría vacía porque se acababa de

inicializar. Finalmente, se ha inicializado el proyecto de React Native mediante el comando *npx*[13] *react-native init <nombre del proyecto>* con el nombre de la carpeta creada previamente en el parámetro de nombre del proyecto, porque de esta manera Git detectaba como cambios todos los ficheros que generaba el comando dentro de la carpeta, y hacia posible la subida de estos al repositorio remoto, es decir que en remoto se subiría el proyecto de React Native inicializado, y ya se podría empezar la implementación de la plataforma a partir de este.

Siguiendo con el desarrollo del frontend, una vez se ha inicializado el repositorio en el que se va a versionar el código de la aplicación, se ha procedido de la misma manera que en el backend en su momento, es decir con el desarrollo de las tareas que se han especificado anteriormente en la fase de análisis.

La forma de trabajar sobre estas ha sido la misma que la que se especificó en la metodología de trabajo antes de comenzar el proyecto, y la cual también se ha seguido durante el desarrollo del backend. Esto quiere decir que el desarrollo de cada tarea se ha realizado en sus propias ramas del repositorio separadas, y cada vez que se terminaba una tarea, después de haberla validado y probado en su propia rama, se fusionaba dentro de la rama *dev* del repositorio de la aplicación, de esta forma todo aquello que se iba desarrollando se acumulaba en una rama sobre la cual se realizan todo tipo de pruebas de integración con el resto del código.

6.8 Programación del frontend

Una vez terminada la preparación, se comenzó con el trabajo de la propia aplicación móvil, y lo primero que se hizo fue la estructuración del proyecto y la implementación de la arquitectura del frontend. La organización del proyecto se hizo de acuerdo con los elementos que formarían la aplicación, es decir, los componentes de React, los elementos de acceso a datos mediante llamadas a la API, las pantallas, y los elementos gráficos de la aplicación, los cuales se agruparon en las carpetas *components*, *data*, *screens* y *assests* respectivamente.

En toda la arquitectura de la aplicación, las relaciones entre los elementos que derivan de componentes como la pantalla principal, su *template*, y los componentes personalizados, se han diseñado mediante un concepto principal de React llamado *props*, el cual es especifico de los componentes y se trata de una abreviación de propiedades. Cuando se tratan de componentes personalizados, estas *props* se pueden establecer de acuerdo con las necesidades del componente. Por otro lado, al declarar cualquiera de los componentes, también se les pueden pasar propiedades, para personalizarlos, inicializarlos con datos o configurar su lógica. Esto es similar a pasar elementos por parámetros a funciones, o constructores en algunos lenguajes de programación como JavaScript.

Los componentes personalizados se han implementado a partir de otros componentes nativos que proporciona React Native, o de componentes de terceros que se han importado mediante

[13] NPX se trata de una interfaz de línea de comandos para ejecutar paquetes binarios de NPM, que ya viene instalado con el gestor de paquetes por defecto, y el cual ya se había instalado previamente al realizar la puesta en marcha del backend.

NPM, y a su vez las pantallas se han formado con estos componentes o los nativos, las cuales deciden cual es el comportamiento de cada componente mediante sus *props*, estos componentes a pesar de ser reutilizables, se han diseñado para ser altamente personalizables en función de las propiedades que se les pasen, y por lo tanto se otorga mucha más versatilidad y flexibilidad a la hora de implementar las pantallas.

Un ejemplo notable de esto es el componente timeline que se ha diseñado para ser un listado que proporciona cierta funcionalidad como paginación, refresco y pantalla en espera para que no se tenga que implementar cada vez que se implementa un listado en la aplicación. Además, estos elementos de funcionalidad también son personalizables, ya que cada pantalla puede decidir mediante los *props* como hacer la paginación o el refresco del listado, por otra parte, es posible decidir cuál será el elemento del listado, y como va a ser mostrado, ya que estos elementos a su vez son componentes, en la Figura 6.16 que se muestra a continuación se puede la implementación del componente *timeline* mientras que en la Figura 6.17 se presenta la declaración un *timeline* en la pantalla de notificaciones para mostrar un listado de notificaciones que se pueda actualizar, o paginar.

```
class Timeline extends React.PureComponent {

  constructor(props) {
    super(props);
    this.state = {
      page: 0,
      loading: false,
      scrolling:false,
      refreshing: false
    };
  }

  resetPagination = () => this.setState({page:0});

  refreshData = () => {
    this.props.onRefresh(this.state.page);
  }
  scrollToTop = () => this.flatList && this.flatList.scrollToOffset({ animated: true, offset: 0 });

  /********************** PRIVATE **********************/
  renderFooter = () => this.state.loading ? <View style={timelineStyle.loadingFooter}><Loading /></View> : null;

  onEndReached = () => {
    if(this.state.scrolling && !this.state.loading){
      this.setState({ page: this.state.page + 1, loading: true }, () => this.props.onEndReached(this.state.page).then(() => this.setState({ loading: false, scrolling: false})));
    }
  }

  /********************** RENDER **********************/
  render() {
    return (
      <View style={[timelineStyle.viewContainer, { width: this.props.size || '100%'}]}>
        { this.props.data ?
          <FlatList
            ref={ref => this.flatList = ref}
            keyboardShouldPersistTaps={'handled'}
            extraData={this.props.data}
            data={this.props.data}
            renderItem={this.props.component}
            keyExtractor={item => item.id.toString()}
            ListHeaderComponent={this.props.header}
            ListFooterComponent={this.renderFooter}
            onEndReachedThreshold={0.8}
            onEndReached={this.onEndReached}
            onRefresh={() =>this.setState({ refreshing: true, page:0 }, () => this.props.onRefresh().then(() => this.setState({ refreshing: false })) )}
            refreshing={this.state.refreshing}
            onMomentumScrollBegin={()=> this.setState({scrolling: true})}
          /> :
          <View style={timelineStyle.loading}><Loading /></View>
        }
      </View>
    );
  }
}

export default Timeline;
```

Figura 6.16: Implementación del componente timeline.

```
const FollowingTabTemplate = props => {

  const { notifications, navigation, updateNotifications, template } = props;

  return (
    <View style={generalStyle.viewContainer}>
      <Timeline
        data={notifications}
        ref={ref => template.timeline = ref}
        component={({ item }) => (
          <Notification
            notification={item}
            navigation={navigation}
          />
        )}
        onRefresh={updateNotifications}
        onEndReached={updateNotifications}
      />
    </View>
```

Figura 6.17: Declaración de un componente timeline de notificaciones.

Se puede observar en la Figura 6.17 que las funcionalidades de paginación y refresco (*onEndReached* y *onRefresh*) ejecutan la función *updateNotificacions* que se ha pasado como una *prop* del *template* de la pantalla y consiste en una función que se implementa en la propia pantalla principal, y simplemente realiza una petición a la API de notificaciones para pedir todas las notificaciones del usuario de nuevo, y actualizar el estado de la pantalla con estas notificaciones, lo cual forzaría un renderizado del listado con los nuevos datos.

Tal y como se había especificado en la metodología de trabajo, después de estructurar el frontend se continuó con el desarrollo de las funcionalidades y componentes de las pantallas principales, las cuales se decidió que fuesen la pantalla de registro, inicio de sesión, principal, explorar, notificaciones, y perfil de usuario, debido a que estas se tratan de pantallas que se acceden directamente mediante la propia navegación diseñada, al contrario de las secundarias que siempre se acceden desde las principales.

En el capítulo 5.2 se especificó que la estructura de navegación de las pantallas para usuarios no identificados era diferente. En específico, se componían de 4 pantallas entre las que se podía navegar mediante botones o enlaces. Esta navegación se ha implementado poniendo en primer plano la pantalla a la que se accede, para ello se ha utilizado el contexto de navegación de React, el cual proporciona métodos que facilitan este trabajo. En esta versión, en los formularios de registro o de inicio de sesión no se ha implementado una validación de los datos en el cliente, por lo tanto, las únicas validaciones que se realizan son aquellas que se han implementado previamente en el servidor, sin embargo, para trabajos futuros no se descarta incluir este tipo de funcionalidad en la parte del cliente.

La estructura de las pantallas para usuarios identificados se ha realizado mediante una vista de pestañas con una barra de navegación inferior, ya que se trata de una estructura de navegación a la que la mayoría de los usuarios están acostumbrados porque es ampliamente utilizada en muchas de las aplicaciones utilizadas actualmente, debido a la buena experiencia de usuario y la gran facilidad de uso que proporciona.

Sin embargo, en las primeras versiones la estructura era diferente, ya que solamente se realizaba la funcionalidad y la visualización sin ningún tipo de maquetación de las pantallas, por lo tanto, se implementó una estructura de navegación temporal mediante botones, la cual cambiaría a la versión definitiva explicada anteriormente durante las fases de diseño y maquetación como se puede ver en la Figura 6.18.

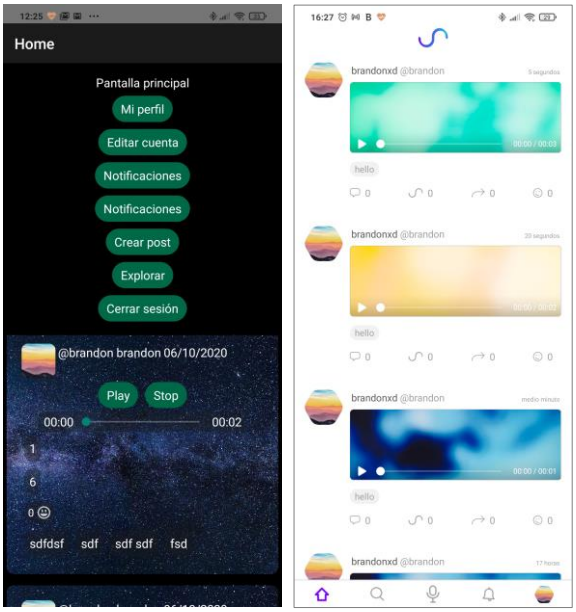


Figura 6.18: Diferencias en la estructura de la navegación.

Siguiendo con la implementación, una de las primeras cosas que se tuvo que realizar en esta fase fue la gestión del *token* de inicio de sesión, ya que era necesario para el resto de la implementación de la aplicación, debido a que se trata de una característica clave a la hora de realizar peticiones al backend. Para llevarla a cabo, se pensó en que la mejor manera de hacerlo era guardar el *token* JWT en un fichero del cliente cuando el backend lo devolviese al iniciar sesión o al registrarse. De esta manera el cliente siempre tendría este token con el que poder realizar las peticiones a la API de la aplicación, y cuando se cerrase sesión, este token se borraría.

Haciendo las cosas de esta forma se podría saber cuándo un usuario ha iniciado sesión, y así controlar que la aplicación solamente es usada por usuarios autenticados, comprobando en cada navegación o apertura de la aplicación el estado del token, para llevar al usuario a la pantalla de inicio si no tiene este token, y en caso contrario realizar la navegación a la pantalla principal, si se acabase de abrir la aplicación, o a la pantalla correspondiente si se estuviese navegando. Además, para identificar al usuario a la hora de realizar cualquiera de estas peticiones, se debía de establecer como *token* de autenticación en el encabezado de la petición, ya que la seguridad del servidor solamente acepta y procesa peticiones con tokens JWT válidos.

Todo esto lleva a un concepto importante, el cual se trata del contexto de la aplicación ya que es utilizado para almacenar información del usuario. En el caso anterior, el token JWT se consulta desde el contexto cada vez que se realizan las peticiones para establecerlo como cabecera de autenticación. El contexto sirve para almacenar información global del usuario de la aplicación, utilizada recurrentemente en varios lugares de esta, por ejemplo, a la hora de realizar peticiones que necesitan el identificador del usuario para devolver cierta información. El contexto siempre se establece cuando se inicia la aplicación leyendo el token almacenado en el fichero del cliente (Se considera usuario identificado) y realizando una petición a la ruta *user/own* del servidor, que devuelve los datos del propio usuario en función del token, después de esto los datos y el token se establecen como contexto de la aplicación. En caso de no tener el fichero del token, se considera que el usuario no está identificado y por lo tanto la navegación se establece en las pantallas de inicio de sesión o de registro, para poder establecer este contexto con el token y el usuario.

Otro concepto relevante que se desarrolló en esta fase fue el de paginación de las listas. Previamente ya se había introducido este término al comentar el componente *timeline*, pero profundizando más en este aspecto se trata de uno de los más importantes de la aplicación, ya que sirve principalmente para mejorar el rendimiento de esta de una manera significativa. La idea detrás esta característica es que los listados solamente carguen una porción indicada de los datos, y a medida que se va avanzando en el listado se van mostrando más porciones de los datos. La forma de implementarlo en la aplicación se ha realizado a través del componente *timeline* y las pantallas que hacen uso de este componente.

En primer lugar, el objetivo principal de realizar la paginación en este componente es el de poder utilizar la paginación en cualquier listado de la aplicación sin importar el modelo de datos, ni la API a la que se le realizan las peticiones. Esto último no depende tanto de la aplicación, sino que se trata de una característica de la propia API, y es por esto por lo que se realizaron cambios convenientes sobre los servicios para que fueran capaces de poder recibir peticiones con parámetros de paginación, por lo tanto, hubo que cambiar las APIs que enviasen listados de datos.

Como la mayoría de los servicios heredaban del mismo servicio común fue rápido cambiar todas las APIs, ya que solamente se tuvo que cambiar una vez en este servicio, y en aquellos que lo sobrescribían. La implementación de esta nueva característica se hizo mediante los parámetros *limit* y *offset* de la función de búsqueda en base de datos del ORM Sequelize, de esta manera la solución fue utilizar los parámetros de *query* de la URL, para recibir los datos de paginación junto a los de filtrado, e introducirlos directamente en la función de consulta de Sequelize. Los parámetros de paginación sirven para establecer el número de elementos que va a devolver la consulta en base de datos, y el número de elemento a partir del cual se devuelven los registros, por ejemplo, cuando el *limit* es 3 y el *offset* es 1, se van a devolver los tres primeros registros a partir del primero, es decir los elementos 2,3 y 4.

En la parte de la aplicación móvil simplemente se debían pasar estos parámetros utilizando la paginación del *timeline*, para ello la función de paginación, que se le pasa como *prop* al componente, se ejecuta incluyendo como parámetro el número de página en la que se encuentra el listado actualmente, de esta manera se realiza el cálculo con este número de página para establecer el *limit* y el *offset* en la petición. Estos dos parámetros se construyen en un objeto junto al resto de parámetros para la consulta, y se pasan a la función que realiza la petición a la API, en cambio, todavía se necesita serializar este objeto de parámetros, ya que las convenciones de los métodos HTTP y la arquitectura REST, especifican que no se le debería dar ningún significado al cuerpo en las peticiones GET aunque este permitido, si el método de solicitud no incluye la semántica definida para un cuerpo de entidad, entonces el cuerpo del mensaje debería ignorarse al manejar la solicitud [22].

Incluso en publicaciones más recientes de la especificación HTTP se comenta que en el método GET el cuerpo de mensaje no tiene ninguna semántica definida y por lo tanto enviarla en un método GET podría causar que en algunas implementaciones se rechazase la petición [20].

Por estos motivos se ha implementado una función auxiliar para serializar este objeto y concatenarlo a la URI de la petición, para que de esta manera las comunicaciones y el propio backend funcionen de acorde a las especificaciones de las arquitecturas REST y el protocolo HTTP,

En la siguiente fase se desarrollaron las pantallas secundarias, que se trata de todas aquellas que no se muestran directamente en la barra de navegación y que se acceden desde las pantallas principales, y por lo tanto era estrictamente necesario completar las principales para poder llevarlas a cabo. Respetando la especificación de requisitos, las pantallas secundarias que se implementaron fueron la de detalle de una publicación, comentarios, respuestas, edición de perfil, búsqueda de publicaciones, búsqueda de usuarios, seguidores y seguidos.

Durante el desarrollo de las pantallas secundarias hubo varios problemas que se pudieron resolver con relativa facilidad y rapidez, sin embargo, hubo otros problemas que requirieron más tiempo del esperado estas pantallas, el ejemplo más notable es el desarrollo de una vista de pestañas para mostrar los comentarios o las respuestas en la pantalla de detalle de una publicación. El problema consistía en que no se podía conseguir hacer un *scroll* para poder visualizar los comentarios o las respuestas a pantalla completa a medida que se iba bajando en el listado, ya que además de esta vista por pestañas, la primera mitad de la pantalla del detalle de publicación se compuso con los datos de la publicación.

Para resolver esto se introdujo la primera parte del detalle en la propia barra de navegación, tal y como se puede apreciar en la Figura 6.19. Por otra parte, para que suba o baje correctamente se implementaron cálculos y animaciones de React Native, en función del deslizamiento realizado sobre el listado de comentarios o respuestas tal y como se puede observar en los fragmentos de código de la Figura 6.20

```
const tabBar = (props) => {
  const translateY = position.interpolate({
    inputRange: [0, HEADER_HEIGHT],
    outputRange: [0, -HEADER_HEIGHT],
    extrapolate: 'clamp',
  });
  return (
    <Animated.View
      style={[
        postDetailScreenStyle.detailHeaderContainer,
        { transform: [{ translateY }] },
      ]}>
      <PostDetail
        onReactionPress={onReactionPress}
        height={HEADER_HEIGHT}
        post={post}
        reactions={reactions}
        addReaction={addReaction}
        onProfileImagePress={goToUserProfile}
        onTagPress={goToSearchTag}
      />
      <TabBar
        {...{ ...props, ...tabBarStyleOptions}}
        getLabelText={({ route }) => route.name}
        onTabPress={({ route, preventDefault }) => {
          if (isValidTabPress.current) {
            preventDefault();
          }
        }}
      />
    </Animated.View>
  );
};
```

Figura 6.19: Implementación de la barra de navegación con el detalle de la publicación.

```
const position = React.useRef(new Animated.Value(0)).current;
const isValidTabPress = React.useRef(false);

const firstRef = React.useRef();
const secondRef = React.useRef();

const syncOffset = (ref, y) => {
  console.log(y);
  ref?.current?.scrollToOffset({
    offset: y > HEADER_HEIGHT ? HEADER_HEIGHT : y,
    animated: false,
  });
  isValidTabPress.current = false;
};
```

```
<Tab.Screen name={'Comments'} >
  {() => <Comments
    comments={comments}
    template={template}
    setComment={setComment}
    onCommentButtonPress={addComment}
    scrollRef={firstRef}
    onMomentumScrollBegin={() =>
      isValidTabPress.current = true}
    onScroll={Animated.event(
      [{ nativeEvent: { contentOffset: { y: position } } }],
      { useNativeDriver: true },
    )}
    onMomentumScrollEnd={(e) => {
      syncOffset(secondRef, e.nativeEvent.contentOffset.y);
    }}
    onProfileImagePress={goToUserProfile}
    headerOffset={HEADER_HEIGHT + TAB_BAR_HEIGHT}
  />}
</Tab.Screen>
```

Figura 6.20: Cálculos y *scroll* de las pestañas.

El resultado de estas dos fases de desarrollo es una aplicación en la cual se pueden visualizar los datos y elementos gráficos, así como realizar las funcionalidades que se habían especificado en la fase de análisis. De esta forma ya se tenía una aplicación bastante funcional sobre la cual realizar las pruebas y cambios pertinentes de una forma temprana, ya que en caso de realizar estas pruebas después de realizar el diseño y maquetación de la aplicación probablemente hubiese sido mucho más complicado o hubiese requerido de más tiempo del esperado.

Es por esto por lo que en este punto la aplicación tenía un aspecto visual similar al que se puede observar en la primera captura de la Figura 6.18, el cual no encajaba con los requisitos de la plataforma. Esto se tuvo en cuenta en durante la planificación de la aplicación, y es por ello por lo que lo siguiente que se hizo fue el diseño visual de la aplicación.

6.8.1 Trabajo en colaboración

En este punto la colaboradora (Luz) comenzó con el diseño de la interfaz gráfica, pero para mejorar la agilidad y reducir tiempos del proyecto, en paralelo al trabajo que realizaba la diseñadora, a medida que ella terminaba ciertas tareas los resultados de estas se recibían como indicaciones y especificaciones sobre la maquetación, diseño de la interfaz y experiencia de usuario. Todo esto simplemente se implementaba en la aplicación mediante las hojas de estilo CSS de React Native, y cambios en la distribución de la pantalla para ajustarlas a los requerimientos del diseño.

Durante este proceso se ha estado en contacto con el diseñador para validar y recibir información de todo aquello que se iba realizando en la aplicación, de esta manera cuando surgieron cambios fueron solamente sobre el diseño, y además de pequeña escala, debido a que la manera de trabajar en paralelo implicaba pequeñas tareas que no suponían ningún problema, al estar todavía en la fase de desarrollo, incluso a veces la implementación de la misma tarea. Por lo tanto, los cambios nunca supusieron un riesgo durante esta fase del desarrollo, y se pudieron cumplir los objetivos de diseño de la aplicación.

7. Testing e implantación

Hasta el momento todo lo que se ha realizado ha sido sobre el entorno de desarrollo, ya que se trata de una buena práctica en el desarrollo profesional realizar las actividades del ciclo de vida del desarrollo software sobre diferentes entornos, para mantener modularizadas y aisladas cada una de estas actividades para evitar cualquier tipo de conflicto. En este caso se trata de realizar la implantación sobre el entorno de producción que será donde se realicen todas las actividades de la aplicación que se va a publicar en las tiendas de aplicaciones.

7.1 Pruebas

En la sección anterior se ha explicado la fase de implantación del software, sin embargo, tal y como se ha especificado con anterioridad, antes de la implantación se tienen que pasar las pruebas que se han diseñado para cumplir con los requisitos del proyecto y de calidad del propio software, y todo eso desarrolla en este capítulo que trata de las pruebas que se han realizado para verificar que el sistema que se ha construido cumple con los criterios que se establecieron durante el análisis. Además, se explican las pruebas unitarias y de integración que se hacían durante y al final de la implementación cada versión del proyecto, ya que durante la implementación de las versiones las pruebas se realizaban sobre cada tarea que se realizaba.

Siguiendo la metodología de trabajo propuesta, todo lo que se vaya a desplegar en producción debe de ser probado previamente en un entorno lo más parecido a este.

Durante el desarrollo de todas las tareas se han realizado las pruebas pertinentes para asegurarse de que las modificaciones que se hacen dentro de una tarea funcionan correctamente, de esta manera durante la implementación de cualquier tarea no se realizan pruebas sobre aquello que no se ha tocado. Esto es así porque las pruebas de todo el sistema se realizan justo después de finalizar la tarea y subirla a dev, ya que en este punto se ejecuta el despliegue automático en el entorno de pruebas con el software que contiene los cambios realizados, de esta forma se prueba que todo el sistema funciona correctamente sin importar si lo que se está probando está relacionado o no con las modificaciones que se han realizado durante el desarrollo de la tarea. Con esta metodología se realizan pruebas de integración completas por cada incremento de funcionalidad de las versiones, esto es importante porque cualquier cambio puede provocar efectos inesperados en partes del sistema que no se hayan modificado o que ni si quiera estén relacionadas con estos cambios.

El desarrollo del *backend* se lleva a cabo antes que el de la aplicación móvil, por lo tanto, para las pruebas completas que se realizan al finalizar cada tarea se utiliza postman, ya que permite realizar peticiones a la API de la misma forma que lo haría la aplicación. A medida que se van finalizando las tareas lo primero que hay que hacer es preparar el espacio de trabajo, si no se ha hecho previamente, para probar las llamadas que se hayan modificado, para ello organizan sus peticiones en función de la tabla de base de datos con la que operan, en segundo lugar, se especifican las cabeceras, los parámetros y los cuerpos de ejemplo de estas llamadas. Hay que tener en cuenta que postman permite guardarlo todo para reutilizarlo cuando sea necesario lo

que significa que todo lo anterior solamente es necesario hacerlo la primera vez. En último lugar se procede a realizar las pruebas utilizando las peticiones de postman que se han creado hasta el momento, la forma más adecuada de realizar las pruebas se explica en detalle en el capítulo 7.1, pero en resumen este proceso consiste en cambiar de diferentes maneras ciertas características de las peticiones y realizar un envío por cada cambio, para intentar encontrar fallos en la API.

Al final de cada tarea es esencial comprobar que aquello que se ha implementado funciona correctamente, de forma unitaria e integrada, para ello lo mejor es realizar pruebas manuales enfocadas a cada funcionalidad. De esta manera al terminar cada tarea además de probar su funcionamiento también se comprueba que los cambios no han afectado a cualquier funcionalidad relacionada, y de esta forma poder detectar problemas lo antes posible. En el siguiente nivel se prueban todas y cada una de las funcionalidades al terminar la versión o el bloque de funcionalidades y/o pantallas, con esto se consigue probar que todo funciona correctamente antes de desplegar la versión a producción, estas pruebas han ido creciendo a medida que se han ido añadiendo funcionalidades nuevas a la plataforma, en un principio se realizaban las mismas pruebas que se realizaba sobre cada tarea, ya que la funcionalidad era tan básica que con estas pruebas era más que suficiente para cubrir toda la aplicación, pero en las últimas versiones se realizaban pruebas al funcionamiento completo de la aplicación y el servidor en conjunto, estas pruebas se han ido añadiendo durante su construcción y se han consolidado como un estándar de validación para las versiones que se desean sacar adelante.

Debido al gran número de funcionalidades, el número de pruebas que se han realizado sobre el sistema también ha sido muy elevado lo que significa que analizarlas y explicarlas daría para otro trabajo aparte y se escaparía del alcance de este proyecto, sin embargo, sí que se van a explicar brevemente aquellas que se han realizado al final de las versiones más tardías, y al final del propio proyecto.

7.1.1 Diseño de las pruebas

Para comprobar que se cumplen con los requisitos de escalabilidad, disponibilidad, rendimiento y eficiencia, se han realizado pruebas de carga, estrés y rendimiento sobre el servidor de la plataforma que se ha desarrollado, es decir que las pruebas están enfocadas a medir las diferentes métricas sobre unos *endpoints* del servidor seleccionados debido al uso que hacen de los recursos.

Antes que nada, lo ideal es identificar el tiempo que se tarda simplemente en realizar una comunicación con la instancia en la que se aloja el servidor, ya que se trata de una instancia en la nube de OVH que se encuentra en servidores de otros países de Europa. Realmente la latencia total es la que importa de cara a los usuarios, sin embargo, también resulta interesante medir el tiempo de procesamiento de las peticiones, ya que se trata de un tiempo que se puede optimizar realizando cambios en cualquiera de los elementos del backend. El tiempo de latencia con el servidor es de 34 milisegundos, medidos mediante el comando *ping*.

Para realizar pruebas de estrés y rendimiento se ha utilizado la herramienta RESTful Stress, ya que proporciona diferentes funcionalidades para testear diferentes atributos de interés del sistema que se ha implementado.

En primer lugar, se tuvieron que decidir las rutas que se iban a probar, ya que inicialmente no se tienen los recursos suficientes para realizar las pruebas sobre todas las rutas que se han implementado, por lo tanto, era imprescindible elegir aquellas rutas que se sabía que iban a ser más utilizadas, que tienen una funcionalidad muy similar al mayor número posible de rutas, o que pueden consumir más recursos que el resto. De esta manera, los *endpoints* que se han escogido para someterse a pruebas han sido los siguientes:

- Inicio de sesión, que se trata de la ruta *signin* de la API, la cual se ha seleccionado porque es muy probable que utilice más recursos que la media, y porque esta ruta sería cada vez más utilizada a medida que crezca el número de usuarios de la plataforma.
- La llamada de recuperación de las publicaciones de la línea temporal, la cual se ubica en la ruta *post/timeline*, y se ha escogido porque se prevé que será una de las rutas más importantes de la aplicación, al estar en la pantalla de inicio de esta y como consecuencia de esto, se estima que sería la más utilizada.
- La ruta para solicitar los datos del usuario actual, nombrada como *user/own* se ha escogido porque también se estima que será una de las más utilizadas de toda la aplicación.
- El endpoint para recuperar el listado de las etiquetas más utilizadas, ubicado en tag/hot, se ha elegido porque aparte de que podría usarse bastante en la aplicación, también se implementa mediante lógica y consulta complejas que pueden ser potencialmente intensas para la base de datos.
- Por último, se han optado por probar las rutas */user* y */post*, ya que se tratan de las rutas implementadas mediante la funcionalidad común para el método GET de cada entidad, lo que supone que esos *endpoints* sean muy parecidos a la mayoría de las listas que hay en la aplicación.

El objetivo principal de cualquiera de estas pruebas es verificar si se cumplen los requisitos especificados, tanto en escenarios habituales, como en escenarios extremos de estrés. Para ello se ha planteado que la prueba de rendimiento consistiese en comprobar el tiempo medio de respuesta que tienen las peticiones sobre diferentes *endpoints*.

Para verificar los tiempos de respuesta, funcionamiento, y estabilidad durante escenarios extremos, se planteó una prueba de estrés que consistiría en realizar un gran número de peticiones a cada ruta, con un bajo retardo entre cada una de ellas, y con una duración total de entre 1 y 2 minutos.

Finalmente, para validar la resistencia, el correcto funcionamiento de la aplicación y, por otra parte, detectar posibles problemas de fugas de memoria, se decidió realizar una prueba combinada de estrés y estabilidad, por lo tanto, tendría una duración muy prolongada, y se realizarían peticiones a las rutas con una latencia entre ellas muy baja.

La prueba de estrés se ha diseñado mediante un proceso, en el que se realizan una cantidad determinada de peticiones a cada una de las rutas, para establecer y analizar la media y los máximos de las latencias, el uso de CPU, y el uso de memoria RAM. Las peticiones de este proceso deben tener un retardo de 10 milisegundos entre peticiones, ya que se tiene que simular un escenario extremo. Finalmente, como RESTful Stress solamente funciona con número de solicitudes, para cumplir con el objetivo de duración, se ha decidido utilizar los resultados de las pruebas de rendimiento para establecer el número de peticiones a realizar por cada ruta, de esta

manera en función de cada latencia media, se establecería cada número de peticiones necesario para alcanzar la duración especificada.

Para completar la prueba de carga de todo el sistema, se ha decidido probar mediante peticiones a todas las rutas especificadas a la vez. Para medir su rendimiento en un escenario extremo, también se ha determinado que el retardo entre grupo de peticiones sea de 10 milisegundos, y a su vez durante un largo periodo de tiempo. En específico, la duración sería de más de una hora, y para ello se han realizado ensayos previos, en los cuales se ha determinado que el proceso de realizar las peticiones de todas las rutas a la vez tiene una latencia media de 800 milisegundos, por lo tanto, para cumplir con el objetivo de duración se han establecido 5000 repeticiones de este proceso. Con todo esto se sacarían los máximos y medias de las latencias, el uso de la CPU, y del uso de memoria RAM. A diferencia del resto de pruebas, durante la prueba de carga, también se debería utilizar la funcionalidad de la aplicación móvil, para intentar detectar problemas de funcionamiento, o fugas de memoria durante periodos de alta carga.

En último lugar se ha diseñado una prueba para validar la disponibilidad del sistema, ya que este sistema debería estar disponible online las 24 horas del día, para que cualquier usuario pudiese entrar en cualquier momento. Para llevar a cabo esta prueba se ha preparado una ruta específica sobre la cual se realizarían las peticiones para comprobar el estado del servidor, se trata de una ruta llamada *status* sin seguridad que siempre devuelve un 1, ya que solamente se quiere comprobar que el servicio y las comunicaciones estas funcionando. De esta manera se ha configurado un monitor de Uptime Robot, para que realice peticiones HEAD a esta ruta periódicamente, en específico cada 5 minutos. El objetivo es determinar el tiempo de inactividad no planeado, y con esto sacar el porcentaje de disponibilidad del sistema en las últimas 24 horas, la última semana y el último mes.

7.1.2 Resultados

Después de ejecutar las pruebas diseñadas, se han medido los resultados tanto de las pruebas, como de la monitorización de todo el sistema ya desplegado en el entorno productivo, y subido a cada una de las tiendas de aplicaciones móviles.

En primer lugar, después de realizar las pruebas detalladas anteriormente, se han recogido los resultados y se han analizado para determinar que se cumple con los requisitos y objetivos del proyecto.

Tabla 7.1: Resultados de las pruebas de rendimiento y carga.

Endpoint	Número de peticiones	Retardo	Duración	Tiempo de respuesta promedio	Tiempo de respuesta máximo	Tiempo de respuesta mínimo	Uso de CPU	Uso máximo de CPU	Uso de memoria	Uso máximo de memoria
api/v1/signin	300	10 ms	127 s	425 ms	512 ms	402 ms	80%	90%	80 MB	90 MB
api/v1/user/own	1000	10 ms	61 s	61 ms	151 ms	47 ms	15%	20%	90 MB	95 MB
api/v1/user?limit=100&offset=0	1000	10 ms	66 s	67 ms	197 ms	52 ms	20%	30%	95 MB	95 MB
api/v1/post?offset=0&limit=2000	400	10 ms	117 s	293 ms	751 ms	233 ms	80%	85%	100 MB	100 MB
api/v1/tag/hot	1000	10 ms	57 s	58 ms	164 ms	46 ms	15%	20%	90 MB	95 MB
api/v1/post/timeline	1000	10 ms	4 min 16 s	257 ms	441 ms	203 ms	75%	80 %	105 MB	110 MB
Todos los anteriores a la vez	5000	10 ms	69 min	832 ms	1682 ms	114 ms	99%	100%	95 MB	100 MB

En la Tabla 7.1 de resultados de las pruebas de estrés y resistencia se puede observar que por lo general se tiene un buen rendimiento y eficiencia, ya que la utilización de los recursos es aceptable durante la ejecución de las pruebas sobre los *endpoints* que más se usarían en la aplicación, manteniéndose entre unos valores de 10 y 30 por ciento en la gran mayoría. Estas peticiones, además, tienen unos tiempos de respuesta muy bajos, de entre 58 y 67 milisegundos de media, lo cual cumple con un amplio margen los requisitos de latencia de la plataforma. Estas pruebas se hicieron con un retardo de 10 milisegundos entre petición y petición ya que se trataban de una prueba de estrés, pero con estos resultados se puede determinar que el número de peticiones de este tipo por segundo, que acepta el servidor, se encuentra ligeramente por debajo de 20. En cuanto a la aplicación móvil no se pudieron ver fallos ni un empeoramiento en la experiencia de usuario debido a las latencias.

Como se ha mencionado en el capítulo de Pruebas, el endpoint que más se usaría es *post/timeline* que ha dado unos resultados de uso promedio de CPU del 75 por ciento, a pesar de que la petición se realiza sin paginación de la búsqueda, y por lo tanto los resultados reflejan la utilización máxima de recursos a la que puede llegar esta llamada. Teniendo esto último en cuenta, se puede afirmar que los resultados del consumo de recursos son aceptables, mientras que los datos de procesamiento de solicitudes son realmente buenos, ya que de media la latencia media es inferior al máximo de 300 milisegundos especificado. La experiencia de usuario de la aplicación no fue perjudicada durante este proceso, se notaba una aplicación un poco más lenta de lo habitual, pero algo totalmente tolerable. Asimismo, tampoco hubo fallos ni problemas de funcionamiento debidos al incremento en los tiempos de respuesta.

El caso de la ruta *signin* que sirve para iniciar sesión, resultó ser la única que no cumplió con objetivo de rendimiento ya que su tiempo de respuesta medio fue de 425 milisegundos, la razón este tiempo de respuesta tan alto es la funcionalidad de comparación de contraseñas de *bcrypt*,

ya que estas se guardan encriptadas en base de datos, y aquellas que se envían en el cuerpo de la petición de inicio de sesión se tienen que encriptar para comprobar que coinciden. Esta función es muy costosa y utiliza una gran cantidad de recursos de la CPU, tal y como se han reflejado en los resultados, con picos de uso del 90 por ciento y un uso medio del 80 por ciento. Para poder ajustarse a los objetivos de la aplicación fue necesario bajar el coste de esta función, para ello se ha modificado el número de rondas de 12 a 8 para las que se procesan los datos, o *salt rounds* de *bcrypt* para alcanzar el tiempo de respuesta más que aceptable. La razón de esto es que se trata del número de rondas por las que pasa el módulo para proporcionar el hash, cuanto más alto sea el número más seguro será el hash generado, y más tiempo se necesita para desencriptarlo[14].

Mientras se afrontaba el problema anterior, se llegó a la conclusión de que la ruta *signup* también tendría tiempos de respuesta excesivos, y por lo tanto se procedió de la misma manera para mejorar los tiempos y cumplir con los objetivos especificados.

Finalmente, los resultados obtenidos de la prueba de carga y estrés fueron realmente positivos, ya que el sistema fue capaz de aguatar, sin caídas ni fugas de memoria, una carga muy por encima de la esperada durante un gran periodo de tiempo, esta carga a pesar de estar muy por encima de lo esperado no fue suficiente para romper el servidor, pero sí que introdujo un tiempo de respuesta elevado de 832 milisegundos de media, con picos de hasta 1682 milisegundos, sin embargo esto no se trata de un problema para el cumplimiento de los requisitos no funcionales, ya que se trata de un escenario extremo muy difícil que se de en el mundo real. Además, estos altos tiempos de respuesta no provocaron errores en la funcionalidad de la aplicación móvil, pero sí que empeoró la experiencia de usuario, lo cual confirma que las latencias bajas son realmente importantes para la plataforma.

Los resultados de las pruebas siempre fueron satisfactorios en su mayoría debido a la metodología de trabajo, ya que en la parte del backend cuando se terminaba cada tarea se hacían pruebas sobre los cambios de esta, y a su vez al integrar cada versión de código se volvían a realizar pruebas de integración sobre todos los cambios de la versión, esto minimizó los problemas detectados cada vez y los resultados habitualmente eran buenos tal y como se puede ver a continuación.

Por otra parte, las pruebas de la aplicación móvil daban resultados más moderados, que en parte tiene sentido al tratarse de software que tiene más características a probar, como la cantidad de funcionalidad y elementos visuales que se han implementado para cumplir con los requisitos de la plataforma, y es que cuantos más elementos se tienen, más combinaciones con fallos hay. Por ejemplo, al probar la pantalla de inicio, en la primera iteración se encontró que algunas de las imágenes del reproductor de las publicaciones no se cargaban, y era debido a un fallo de programación en el establecimiento del estado del componente. Este tipo de problemas iban surgiendo y corrigiéndose a medida que se avanzaban con las tareas del frontend, ya que no se definió un proceso o protocolo de pruebas riguroso, ya que no ha sido necesario en estas fases iniciales. Sin embargo, para realizar el mantenimiento y el desarrollo de nuevas funcionalidades en un futuro, será de mucha utilidad diseñar una batería de pruebas que se ejecuten sobre todas las confirmaciones de código.

[14] Información acerca sobre el número de rondas, se encuentra en el apartado “A Note on Rounds” del siguiente enlace https://www.npmjs.com/package/bcrypt

La última prueba realizada tenía como objetivo validar el requisito de disponibilidad del sistema, el cual especificaba que el porcentaje de disponibilidad en el último mes debería ser mayor al 99,9% ya que se trata de una cifra exigente pero alcanzable teniendo en cuenta los recursos del sistema, y las características del proyecto. Los resultados de esta prueba se pueden ver en la Figura 7.1

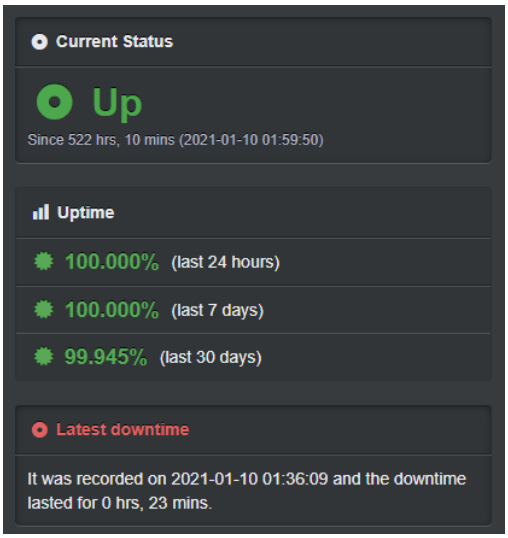


Figura 7.1: Tiempos de actividad de Soundn en Uptime Robot.

Como se puede ver en, tanto en las últimas 24 horas como en la última semana, del día que se hizo la captura, no ha sucedido ningún corte de servicio, sin embargo, en el último mes el servicio dejo de funcionar el 10 de enero durante 23 minutos, debido a un fallo de la instancia, ya que el resto de las aplicaciones alojadas en esa misma instancia reportaron el mismo tiempo de inactividad. A pesar de esto, el sistema ha podido cumplir con el requisito de disponibilidad que se demandaba, ya que su porcentaje de actividad en el último mes ha sido del 99,945%, algo por encima del 99,9% requerido. Este tiempo total de caída resulta bastante aceptable teniendo en cuenta el tipo de proyecto, que se trata de un servicio que todavía se está sometiendo a pruebas, y que en el momento de lanzamiento no se espera un amplio número de usuario, ni una exigencia de disponibilidad muy alta.

En conclusión, se puede decir que la aplicación que se ha desarrollado cumple con todos los requisitos especificados durante la fase de análisis, puesto que el resultado final cuenta con todas las funcionalidades que se especificaron al principio, y además las pruebas confirmaron el cumplimiento de los requisitos no funcionales y las especificaciones de calidad.

7.2 Implantación

La implantación del software consiste en el proceso de establecer el sistema en un entorno productivo, tanto la parte del servidor, como la propia aplicación móvil. Para ello, esta implantación se realiza en dos fases que se explican a continuación:

1. Después de realizar todas las pruebas y no encontrar ningún problema, se procede con la implantación del sistema en el entorno productivo, y el despliegue de la aplicación móvil en las tiendas de aplicaciones.
2. Se monitoriza el estado de la plataforma ya desplegada en producción y se miden los resultados de haber realizado la implantación para sacar conclusiones acerca de los objetivos del proyecto realizado.

Los despliegues de este entorno se hacen sobre el VPS utilizando los pipelines de Gitlab CI/CD de la misma manera que se haría con cualquier otro entorno, ya que se realizan las mismas acciones, pero con configuraciones diferentes, debido a que cada entorno tiene sus peculiaridades. Todas las acciones del despliegue se ejecutan de forma automática, lo que significa que no es necesaria la intervención de una persona en todo el proceso, estas acciones se desencadenan cuando se realiza una fusión con la rama del repositorio correspondiente con el entorno, en este caso cuando se ejecutan fusiones de ramas con *master* se desencadenan las siguientes acciones:

- Realizar una copia de seguridad de los archivos multimedia que se han subido desde la aplicación al servidor backend de producción.
- Eliminar los archivos desplegados en producción previamente si existiesen.
- Copiar los archivos fuente de la nueva versión de producción.
- Instalar las dependencias de node.
- Reiniciar el servidor PM2 de producción con la configuración adecuada para este entorno:
    - El puerto desde el que se escuchan las peticiones se establece a 21996
    - La variable de entorno NODE_ENV se establece a “production” lo que significa que el servidor apuntara a las bases de datos de producción, y la ejecución de este se realizara utilizando el protocolo https para proteger las peticiones mediante SSL/TLS, utilizando los certificados generados en la propia instancia utilizando certbot.
    - Se establecen los ficheros para escribir los logs de producción.

Si todo ha ido correctamente, después de la ejecución se debería haber desplegado la versión del código que se acaba de fusionar en la rama *master*, y por lo tanto ya estaría operativa la aplicación backend en producción y solamente quedaría por realizar el despliegue de la aplicación móvil, sin embargo, este proceso solamente requiere de una compilación del código que apunte al puerto correspondiente del servidor de producción y subirla a las tiendas de aplicaciones.

7.2.1 Resultados de la implantación

Entrando en detalle en la funcionalidad conseguida en cada pantalla, mediante la Figura 7.2 se puede comprobar que las pantallas de autenticación cumplen con los requisitos funcionales RF-01, RF-02 y RF-03. Así como se especificaron en estos requisitos, se puede apreciar que el inicio de sesión se realiza mediante el nombre de usuario y la contraseña, mientras que el registro se lleva a cabo introduciendo el nombre de usuario, alias, contraseña y correo electrónico, y aceptando los términos y condiciones de uso y la política de privacidad de la aplicación, que se pueden visualizar pulsando sobre el mismo texto, y aceptar marcando la casilla. Después de esto también se pueden observar 2 pantallas para introducir los datos opcionales, como la foto de perfil y el audio de descripción.

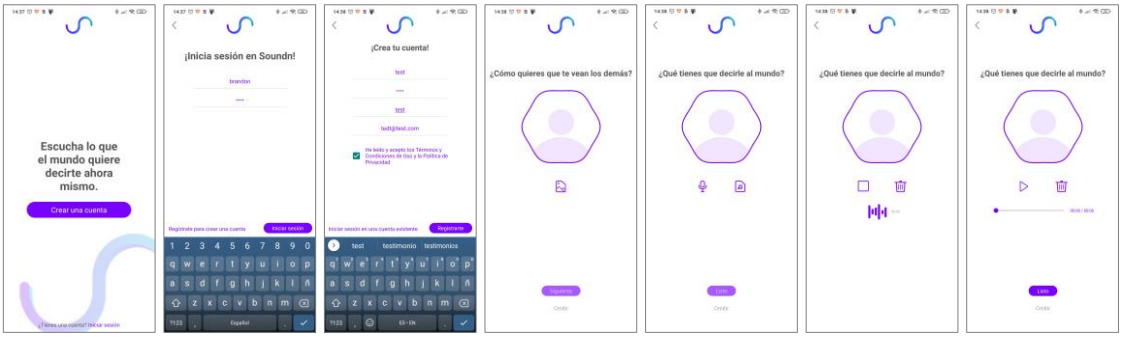


Figura 7.2: Grupo de pantallas de autenticación.

De la misma manera, las pantallas principales de la aplicación mostradas en la Figura 7.3, cumplen con otra gran parte de los requisitos funcionales, específicamente con RF-04, RF-06, RF-07, RF-08, RF-09, RF-14, RF-12, RF-15, RF-17, y RF-18. Esto se puede apreciar, en primer lugar, en la pantalla de inicio de la aplicación debido a que contiene la línea temporal de publicaciones, que se trata de una lista de publicaciones de los usuarios a los que se sigue, ordenadas por la fecha de creación, donde cada publicación tiene los botones necesarios para compartirla o reaccionar a esta, y el acceso al propio detalle de la publicación.

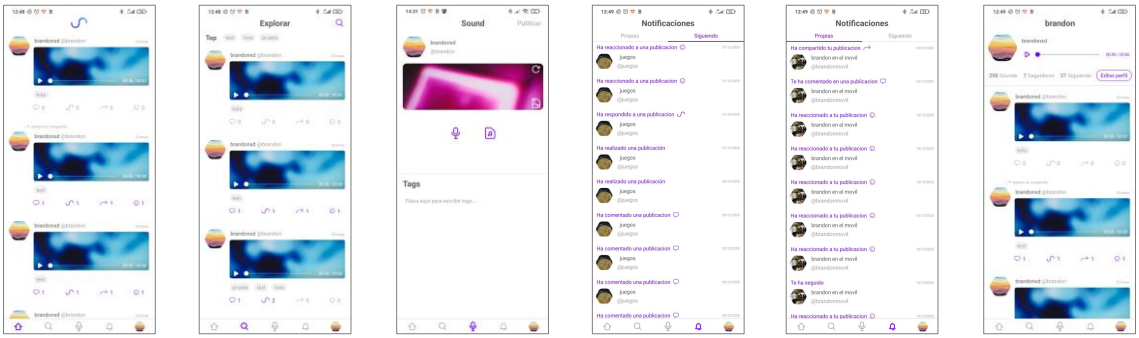


Figura 7.3: Resultado de las pantallas principales de la aplicación

Seguidamente, se puede comprobar que el resultado final de la pantalla de explora está compuesta del listado de las etiquetas más utilizadas en las últimas 48 horas, acompañado de una línea temporal de las publicaciones que tienen alguna de estas etiquetas, ordenadas por fecha de creación, con la misma funcionalidad descrita previamente. También se certifica que el resultado de la pantalla de creación de una publicación permite personalizar la publicación

mediante una foto seleccionada por el usuario, o una aleatoria seleccionada por la propia aplicación, es posible realizar una grabación de audio en el instante, o seleccionar un archivo de audio desde el almacenamiento del dispositivo, y en último lugar se pueden etiquetar las publicaciones mediante etiquetas escritas por el usuario.

La pantalla de notificaciones muestra las notificaciones organizadas por pestañas de propias y siguiendo, la primera muestra un listado de las notificaciones de acciones que otros usuarios han realizado sobre el usuario actual, o sobre sus publicaciones. Mientras que la pestaña de siguiendo muestra un listado de las acciones que han realizado los usuarios a los que el usuario actual está siguiendo.

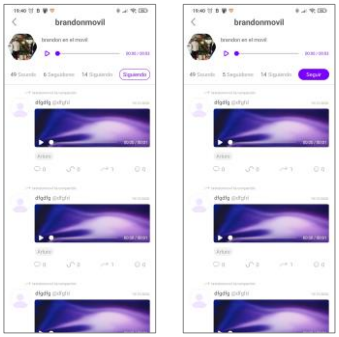


Figura 7.4: Perfil de usuario con botones de seguimiento.

Para terminar con el resultado de las pantallas principales, se puede apreciar que en la pantalla de usuario se muestran los datos del usuario: nombre, alias, foto de perfil, audio de perfil y datos de seguimiento. Además, se dispone de una línea temporal de todas las publicaciones del usuario, o de aquellas que ha compartido. Por último, existe un botón para seguir o dejar de seguir, en caso de tratarse de un perfil de un usuario diferente al actual, este mismo botón, en caso contrario, sirve para acceder a la edición de datos del usuario, tal y como se muestra en la Figura 7.3 y en la Figura 7.4.

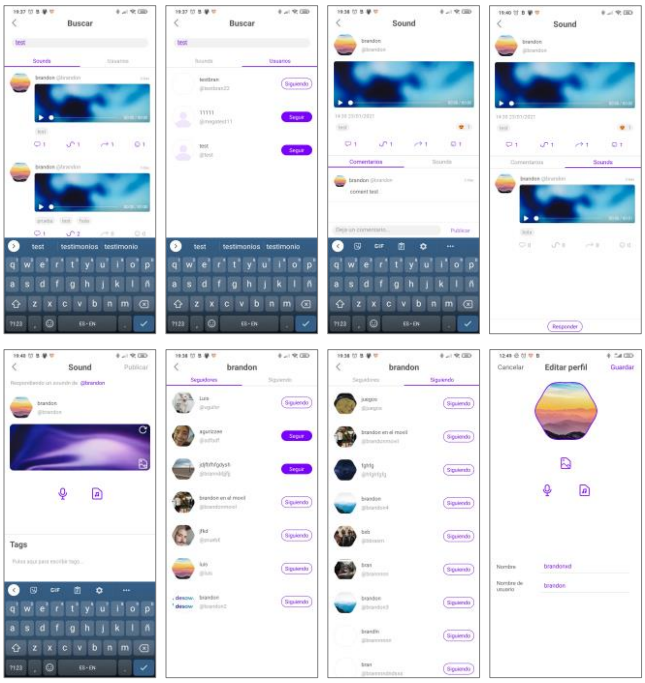


Figura 7.5: Resultado final de las pantallas secundarias.

Para cerrar este capítulo, solamente queda por analizar los resultados de las pantallas secundarias de la aplicación. Estas se muestran en la Figura 7.5 y como se puede ver cumplen con otra gran parte de los requisitos funcionales: RF-05, RF-08, RF-09, RF-10, RF-11, RF-12, RF-13, y RF-18.

En primer lugar, se presentan las pantallas de búsqueda de publicaciones o usuarios, en las cuales se puede realizar la búsqueda de publicaciones mediante sus etiquetas, o los usuarios mediante su nombre de usuario único. Después de esta pantalla se muestra la del detalle de una publicación, a la cual accede desde cualquier publicación. En esta pantalla se puede reaccionar a esta misma publicación, visualizar sus reacciones detalladamente, y compartirla. Además, es posible visualizar el listado de comentarios y respuestas, donde también se puede realizar un comentario de texto o una respuesta de audio, creada mediante la pantalla de creación de respuestas, que se aprecia a continuación de las del detalle en la Figura 7.5.

La siguiente pantalla se trata de los listados de seguimiento, que se compone de dos pestañas con las pantallas del listado de seguidores y seguimientos de un usuario concreto. En esta pantalla se pueden visualizar los listados de seguimiento, realizar la acción de seguir o dejar de seguir a los usuarios listados, y acceder a sus perfiles de usuario.

En último lugar se cuenta con la pantalla de edición de datos del usuario, donde este podría cambiar su foto y/o audio de perfil, alias, y nombre de usuario. A esta pantalla solamente puede acceder el propio usuario a través del perfil de usuario tal y como se ha visto previamente.

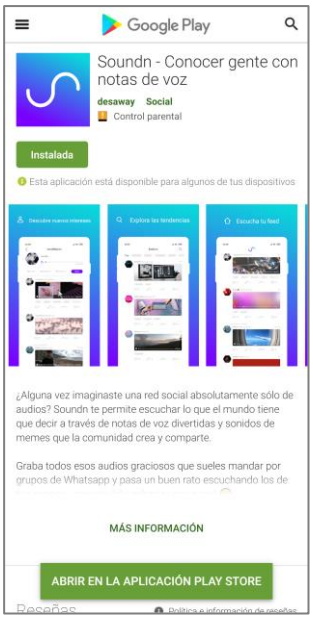


Figura 7.6: Pagina de Soundn en la tienda de aplicaciones.

Sumado a todo esto, finalmente la aplicación ha sido desplegada en la tienda de aplicaciones móviles más importante de la actualidad, es decir en la Play Store. El proceso de subida a la plataforma consiste en introducir los datos, descripción, e imágenes descriptivas de la aplicación, y después de esto se presenta la aplicación para que se someta a una validación, en la cual, primero se realiza una encuesta de clasificación del contenido, para establecer la clasificación por edad. Después de esto se verifica que la aplicación cumple con los requisitos de funcionamiento, y que su contenido es adecuado para la clasificación obtenida previamente. En

el caso de Soundn, la primera vez no se consiguió validar en la Play Store, porque se había clasificado como PEGI 3 después de la encuesta, sin embargo, el proceso de validación determinó que no era apta para este público, porque la aplicación podría contener palabras malsonantes, ya que los usuarios tienen cierta libertad, dentro de los límites legales y morales. Después de volver a realizar la encuesta, teniendo en cuenta este tipo de usos, la aplicación fue clasificada como PEGI 12, y con esto pudo ser validada sin problemas. Por último, el producto final de todo el proyecto se puede apreciar en la Figura 7.6[15].

[15] El enlace a la tienda Play Store: https://play.google.com/store/apps/details?id=es.desaway.soundn

8. Conclusiones

En conclusión, durante este proyecto se ha conseguido poner a punto una instancia que pudiese alojar correctamente todos los componentes del backend de la aplicación, para ello se ha instalado y configurado las diferentes tecnologías que se utilizan en el sistema, por otra parte, también se han creado los certificados para que las comunicaciones sean bajo HTTPS, y como resultado de estas tareas, el sistema completo que se ha implementado opera correctamente. Además este sistema cuenta con una infraestructura de integración, despliegue y entrega continuos, para facilitar y automatizar los procesos posteriores al desarrollo del software, como son las pruebas, el despliegue o la validación, de esta manera se ha mejorado la metodología de trabajo de los desarrolladores del proyecto, los cuales pueden estar enfocados exclusivamente en tareas de programación, también se propicia la entrega inmediata de código, ahorra costes y esfuerzos, aumenta la confianza a la hora de desplegar a producción, simplifica el trabajo en equipo, favorece la comunicación entre los departamentos y posibilita la detección temprana de errores.

Por otro lado, toda la plataforma se ha implementado siguiendo estándares y buenas prácticas del desarrollo de software empresarial. Además de esto, poner en marcha este proyecto ha supuesto el uso de tecnologías de desarrollo punteras como React Native o NodeJS, las cuales se han utilizado para construir la aplicación móvil, y el servidor API REST. Todo esto significa que se han cumplido todos los objetivos técnicos que se habían marcado al inicio del proyecto.

Del capítulo 7.2.1 se puede sacar la conclusión de que se han conseguido llevar a cabo todas las funcionalidades que se habían propuesto desde un principio, lo cual ha supuesto un cumplimiento de los objetivos funcionales, siguiendo las pautas marcadas por el análisis y diseño del software. Además, tras pasar diferentes tipos de validaciones externas, se ha conseguido publicar la aplicación en la Play Store de Google, que era uno de los objetivos más importantes para la red social, ya que estar en esta plataforma de distribución es fundamental para el crecimiento de la aplicación.

En general, se ha tratado un proyecto muy grande, ya que se han abarcado muchos campos de la producción de software, y es que durante el proyecto se han realizado tareas de gestión, análisis, diseño, investigación, desarrollo, sistemas, pruebas, etc. Esto quiere decir que incluso hay algunas cosas que se han omitido para no alargar demasiado este documento, porque o eran secundarias, o no afectaban directamente al cumplimiento de los objetivos del proyecto. A nivel personal, el desarrollo de este proyecto ha supuesto un gran desafío, que se ha afrontado con muchas ganas, motivación y determinación, por lo tanto, ha sido una experiencia muy positiva, ya que se ha desarrollado desde cero toda una red social con un enfoque diferente a las existentes.

8.1 Trabajos futuros

A pesar de haberse incorporado bastantes funcionalidades, sí que es cierto que puede haber una falta de características totalmente enfocadas al contenido de audio. Por lo tanto, los siguientes pasos que se pueden dar en la aplicación deberían dirigirse a implementar funcionalidad centrada en todo lo que tiene que ver con el audio.

De esta forma, se plantea llevar a cabo la creación de listas de reproducción personalizadas para las publicaciones que se han hecho en la aplicación, de esta manera, cada usuario podría componer diferentes listas con las publicaciones que quieran, para reproducirlas automáticamente. Además, esta funcionalidad podría mejorarse y proporcionar más valor al usuario si se implementa la reproducción de publicaciones en segundo plano, para escuchar publicaciones cuando el teléfono está bloqueado o cuando la aplicación esta minimizada.

Para proporcionar características que afecten directamente a la experiencia del usuario, en el futuro se añadiría la característica de modulación voz para poder cambiarla a gusto del usuario, de esta manera también se incluirían herramientas adicionales para poder editar el audio de las publicaciones. En las próximas versiones de la aplicación, también se podría incluir una característica para avisar de contenido explicito en las publicaciones, de esta manera, los usuarios estarían prevenidos antes de reproducir este tipo de contenido.

La inclusión de mensajes privados en la plataforma mejoraría el aspecto social de esta, ya que permite una conexión más personal entre los usuarios, además solamente se permitiría enviar mensajes privados en formato audio para mantener la coherencia con el resto de la aplicación. También resultaría interesante, crear un sistema de reportes para denunciar usuarios o publicaciones que no cumplan con las reglas de la plataforma, junto con esto también se llevaría a cabo una forma para realizar reclamaciones de derechos de autor sobre cualquier publicación, ya que en cualquier momento los usuarios podrían realizar publicaciones de contenido protegido con copyright.

Para mejorar el sistema de monetización de la aplicación, y atraer creadores de contenido de este tipo de formato, se plantea implementar la funcionalidad para que los usuarios puedan introducir anuncios en las publicaciones que ellos mismos realicen, de esta manera se le pagaría una parte de estos anuncios al creador, y otra parte se le abonaría a la plataforma. El dinero que se pagaría por estos anuncios sería proporcional al número de reproducciones que tenga la publicación, ya que, cuanto mayor sea el número de reproducciones a más gente le llega el anuncio, de esta manera la plataforma y los creadores de contenido podrían favorecer el crecimiento de la aplicación, y beneficiarse de esto.

Por otro lado, también se propone crear posteriormente un sistema de donaciones, para que los usuarios puedan realizar donaciones de dinero a cualquier otro usuario de la plataforma, ya sea a través de sus publicaciones, o directamente desde su perfil de usuario. La estrategia de estas donaciones sería que gran parte se le abonase al usuario al que va dirigida, y otra pequeña parte se pagaría a la plataforma en concepto de comisiones de servicio.

Otra funcionalidad enfocada a la monetización dentro de la aplicación, consistiría en dar la posibilidad de poder vender contenido, es decir que los usuarios podrían realizar ciertas publicaciones que solamente pueden reproducirse y descargarse si el usuario paga la cantidad

que establece el creador de la publicación, y de la misma forma que en las donaciones, una gran parte de este dinero se le pagaría al creador, mientras que el resto se le pagaría a la plataforma como costes de distribución.

Finalmente, buscando atraer a nuevos usuarios se propone realizar diferentes integraciones con plataformas de reproducción de música en *streaming* como Spotify o Tidal, de esta manera cualquier usuario de estas plataformas podría compartir canciones o listas de reproducción de estas plataformas externas en Soundn.

8.2 Relación con los estudios cursados

Algunos de los aspectos que se han tratado en este trabajo están estrechamente relacionados con los estudios que se han cursado en el Máster Universitario en Ingeniería Informática y en el Grado en Ingeniería Informática, ya que el proyecto ha implicado tareas técnicas, de gestión y de dirección.

Durante los estudios universitarios se ha tratado extensamente varias facetas de los proyectos de TI, estos estudios han otorgado una visión general de todas las tareas que se realizan en este tipo de proyectos, y de las competencias necesarias para llevarlas a cabo correctamente.

Como en casi cualquier proyecto de software, lo primero que tiene que concretar es la planificación del proyecto, ya que cualquiera de los trabajos que se tienen que llevar a cabo dependen del resultado de esta planificación. Esto relaciona el trabajo realizado al principio del proyecto con todos los estudios cursados sobre gestión y dirección de proyectos de TI, los cuales también han proporcionado cierto conocimiento analítico que, junto con la experiencia en el desarrollo de software y la formación recibida en este campo, sirven de apoyo para realizar las tareas de análisis de la plataforma de audios.

Profundizando en los aspectos técnicos del proyecto, lo primero a mencionar es el trabajo con servidores virtuales en la nube, sobre lo cual se ha formado y desarrollado desde el primer momento en el Máster en Ingeniería Informática, esto ha sido de gran ayuda a la hora de trabajar con el servidor virtual de la plataforma. Además, la preparación obtenida a través de las diferentes asignaturas enfocadas en la materia de redes y computación ha servido para la puesta en marcha y configuración del servidor y sus comunicaciones en red.

Por otra parte, el diseño e implementación del modelo de datos mediante herramientas de gestión de bases de datos, se trata de un aspecto sobre el cual también se ha obtenido formación durante los estudios cursados, y que están estrechamente relacionados con los trabajos de implementación del backend de la plataforma social. Además, el modelo de datos también se debe implementar dentro del propio servidor del backend con la API REST, la cual se ha introducido brevemente en algunas de las materias que se han estudiado.

Por otro lado, el testing y las pruebas que se han realizado sobre la aplicación están muy influidas por aquellas asignaturas enfocadas a estas tareas del ciclo de vida del software, lo que significa que en ciertas ocasiones se han aplicado los conocimientos adquiridos durante la etapa de formación en esta disciplina, y sobre todo se ha afrontado el diseño de las pruebas con el cambio de mentalidad explicado en estas asignaturas, el cual consistía en realizar este diseño

para buscar el fallo del software, teniendo en cuenta que es imposible abarcar todos los posibles casos de prueba, y que por lo tanto habría que enfocarse en aquellos donde sea más probable encontrar errores, ya que de esta manera se afronta de una manera más productiva esta fase.

Finalmente, poniendo en práctica las competencias adquiridas durante el máster, se ha realizado un esfuerzo en planificar adecuadamente el tiempo disponible y programar las tareas para cumplir con los objetivos que se han marcado para este proyecto. Además, una de las cosas más interesantes que se ha promovido, es la autogestión y el aprendizaje autónomo, los cuales se han cumplido durante todo este proyecto debido a la gran cantidad de conocimientos que se han adquirido de una forma autodidacta o autodirigida, lo cual es una de las competencias más interesantes para los profesionales dedicados a cualquiera de los campos de la ingeniería informática.

Agradecimientos

A mis padres, su enorme esfuerzo y paciencia han sido claves en mi formación, siempre han estado a mi lado, y han trabajado muy duro para darme un buen futuro.

A Joan, por su valiosa ayuda en la redacción de este trabajo.

Y a todos los compañeros con los que he trabajado a lo largo de estos años, durante esta etapa aprendí muchísimo con ellos, lo cual hoy me sigue sirviendo para afrontar diferentes tipos de proyectos software.

9. Bibliografía

[1] I. Sommerville, Ingeniería del software, Madrid [etc.]: Pearson Addison Wesley: ISBN 8478290745, 2005.

[2] R. S. Pressman, Ingeniería del software: un enfoque práctico, McGraw-Hill: ISBN 9789701054734, 2006.

[3] P. A. Bernstein, «Middleware: A Model for Distributed System Services,» *Commun. ACM*, vol. 39, nº 2, pp. 86-98, DOI 10.1145/230798.230809, 1996.

[4] S. Vinoski y S. Tilkov, «Node.js: Using JavaScript to Build High-Performance Network Programs,» *IEEE Internet Computing*, vol. 14, pp. 80 - 83, DOI 10.1109/MIC.2010.145, 2010.

[5] R. S. Pressman, A manager's guide to software engineering, McGraw-Hill: ISBN 9780070508200, 1993.

[6] B. W. Lampson, «Computer Security in the Real World,» *IEEE*, vol. 37, nº 6, pp. 37-46, DOI 10.1109/MC.2004.17, 2004.

[7] K. Lei, Y. Ma y Z. Tan, «Performance Comparison and Evaluation of Web Development Technologies in PHP, Python, and Node.js,» de *IEEE 17th International Conference on Computational Science and Engineering*, Chengdu, China, 2014.

[8] M. R. Mesbahi, A. M. Rahmani y M. Hosseinzadeh, «Reliability and high availability in cloud computing environments: a reference roadmap,» *Human-centric Computing and Information Science*, vol. 8, pp. 1-31, DOI 10.1186/s13673-018-0143-8, 19 Agosto 2018.

[9] N. Yashar, B. Behrouz, F. Xia, L. Yang y M. Obaidat, «Safety challenges and solutions in mobile social networks,» *IEEE*, vol. 9, nº 3, pp. 1-21, DOI 10.1109/JSYST.2013.2284696, 2013.

[10] R. Khare y S. Lawrence, «Upgrading to TLS Within HTTP/1.1,» IETF, RFC 2817, DOI 10.17487/RFC2817, 2000.

[11] T. Dierks y E. Rescorla, «The Transport Layer Security (TLS) Protocol Version 1.2,» IETF, RFC 5246, DOI 10.17487/RFC5246, 2008.

[12] M. Fowler, UML distilled : a brief guide to the standard object modeling language, Massachusetts : Addison-Wesley: ISBN 9780321193681, Addison-Wesley.

[13] H.-E. Eriksson, M. Penke, B. Lyons y D. Fado, UML 2 Toolkit, OMG: ISBN 978-0-764-55519-0, 2003.

[14] J. Sventek, «The distributed application architecture,» Informe técnico, Hewlett-Packard Company, 1992.

[15] R. Fielding, «Architectural Styles and the Design of Network-based Software Architectures,» Tesis doctoral, University of California, Irvine, 2000.

[16] D. Booth y C. K. Liu, «Web Services Description Language (WSDL) Version 2.0 Part 0: Primer,» W3C, 2007.

[17] M. Gudgin, M. Hadley, N. Mendelsohn, J. Moreau y H. Nielsen, «SOAP Version 1. 2 Part 1: Messaging Framework,» W3C, 2003.

[18] R. Taylor y R. Fielding, «Principled design of the modern Web architecture,» *Proceedings of the 2000 International Conference on Software Engineering. ICSE 2000 the New Millennium*, vol. 2, pp. 407-416, DOI 10.1109/ICSE.2000.870431, 2000.

[19] L. Richardson y S. Ruby, RESTful web services, O'Reilly Media Inc.: ISBN 978-0-596-52926-0, 2008.

[20] R. Fielding y J. Reschke, «Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content,» IETF, RFC 7231, DOI 10.17487/RFC7231, 2014.

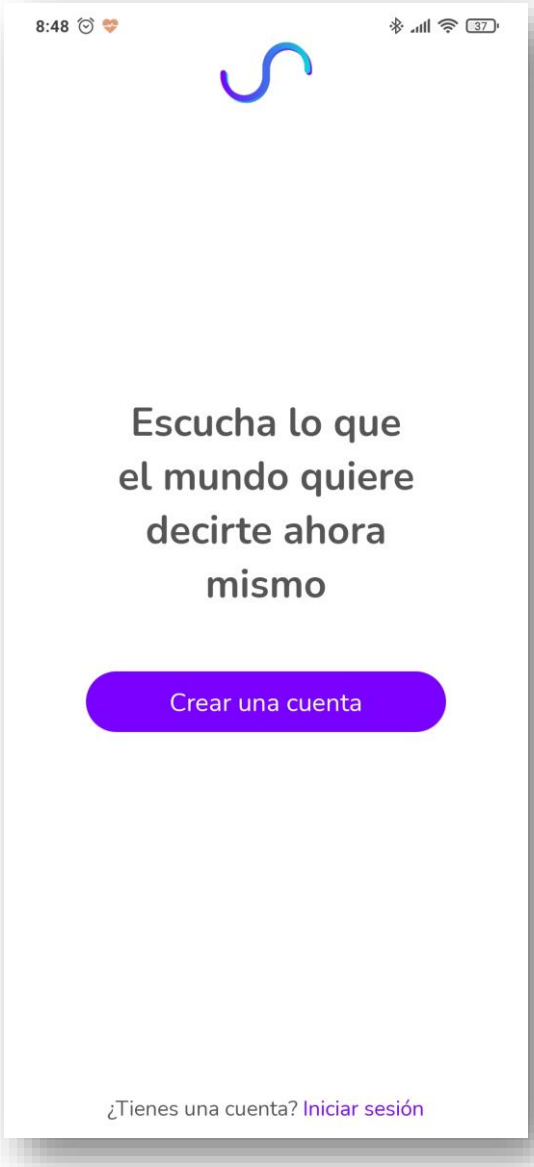
[21] T. Berners-Lee, R. Fielding y L. Masinter, «Uniform Resource Identifiers (URI): Generic Syntax,» IETF, RFC 2396, DOI 10.17487/RFC2396, 1998.

[22] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach y T. Berners-Lee, «Hypertext Transfer Protocol -- HTTP/1.1,» IETF, RFC 2616, DOI 10.17487/RFC2616, 1999.

ANEXO

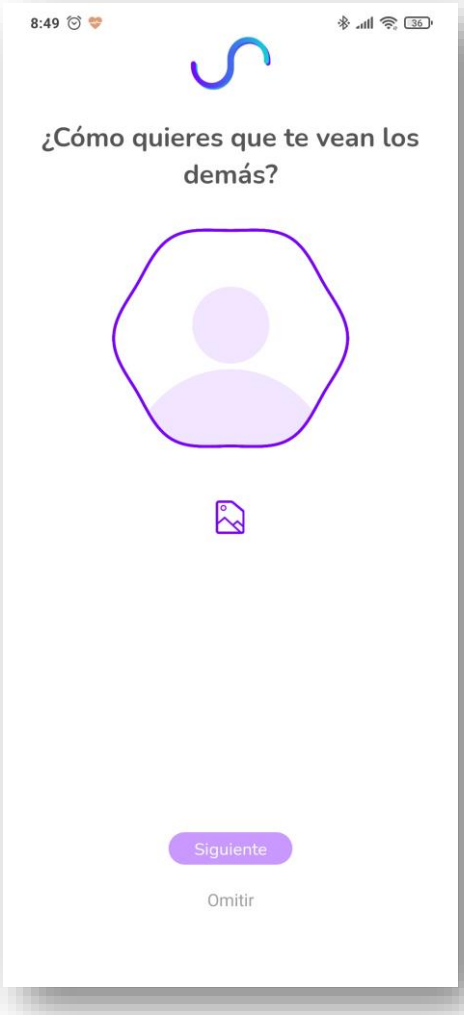
Manual de usuario

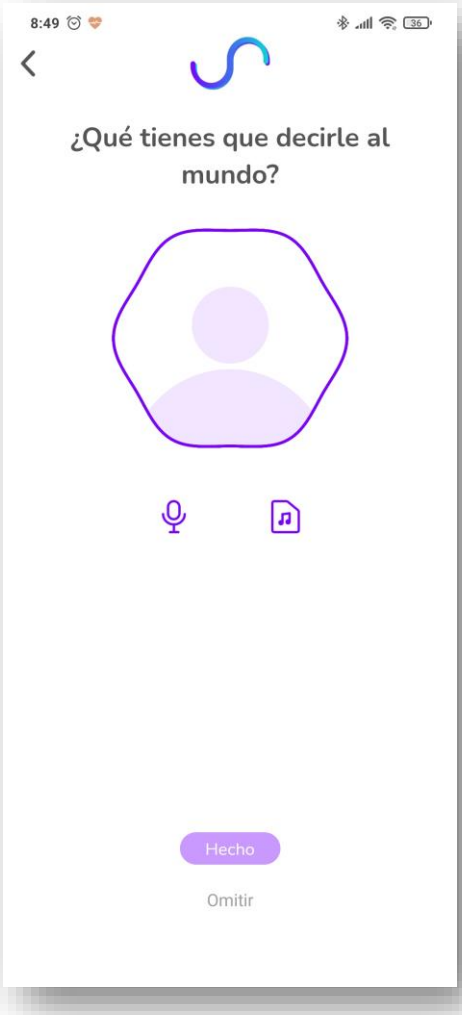
En el momento de redacción de este trabajo, la aplicación solamente se puede instalar en dispositivos Android, mediante la tienda de aplicaciones Google Play. Para ello simplemente se tiene que acceder al enlace https://play.google.com/store/apps/details?id=es.desaway.soundn donde se podrá descargar e instalar fácilmente. Una vez instalada ya se puede utilizar la aplicación, pero para poder acceder al contenido de esta, previamente hay que registrarse si no se tiene una cuenta.



El registro consiste en un pequeño formulario donde hay que rellenar todos los campos de forma obligatoria, y aceptar los términos y condiciones de uso, y la política de privacidad de la aplicación para poder utilizarla, estos términos se pueden visualizar pulsando sobre el propio texto, mientras que para aceptar simplemente hay que marcar la casilla predispuesta para ello.

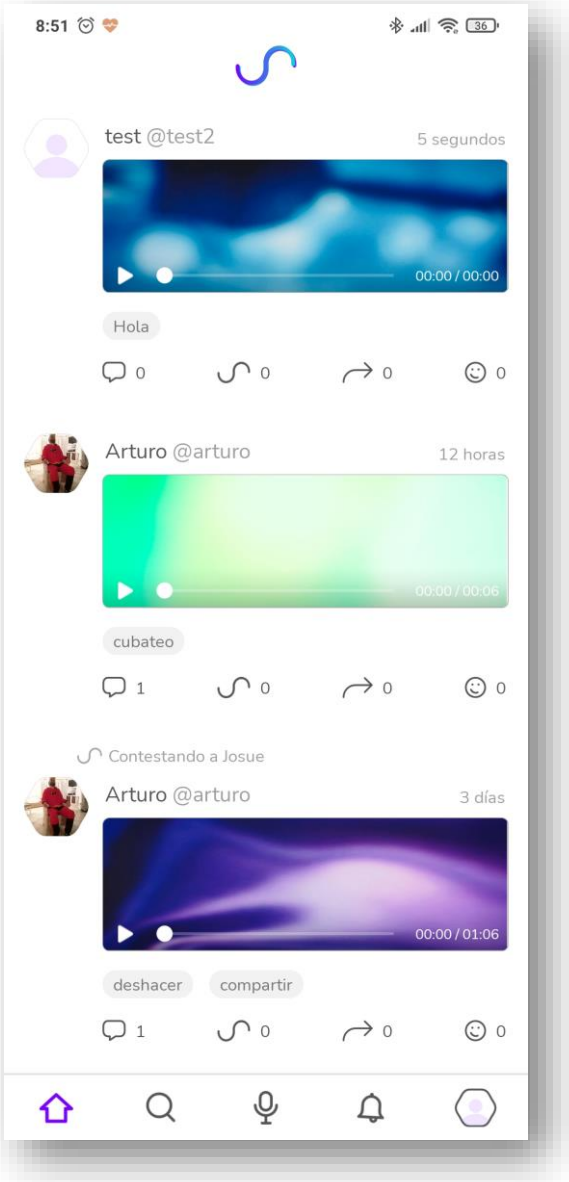
Después de esto, se navegará primero a la pantalla de subida de imagen de perfil, y luego a la de subida de audio de perfil, donde se podrá subir este contenido de manera opcional, ya que en estas dos pantallas existe la opción de omitir.



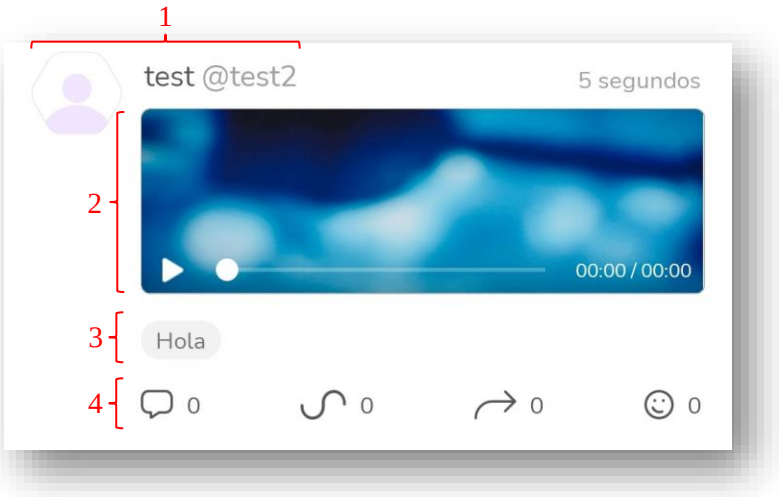


Por otra parte, el inicio de sesión es sencillo, simplemente hay que introducir el nombre de usuario y la contraseña que se han establecido en el momento del registro.

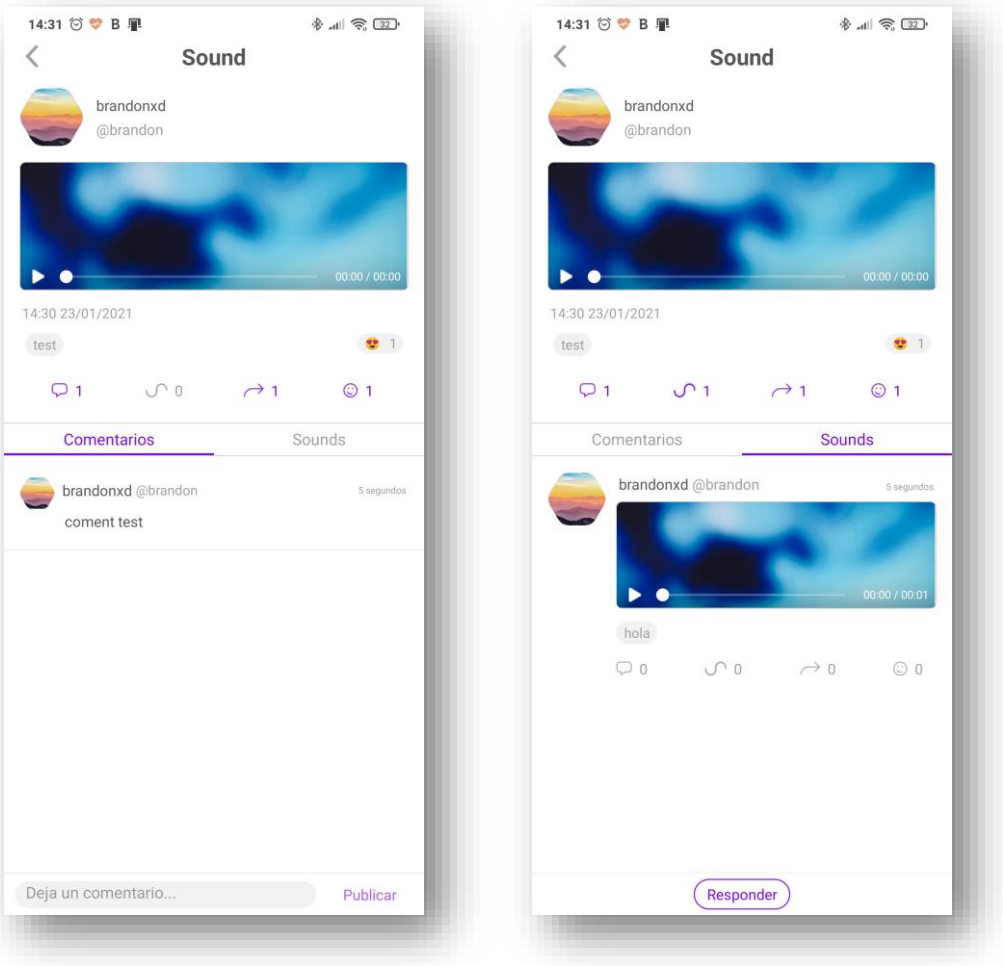
Después de esto ya se puede acceder a todo el contenido de la aplicación, empezando por la pantalla principal, donde se pueden visualizar todas las publicaciones de las personas a las que se sigue, se trata de un listado en el cual hay que desplazarse verticalmente para visualizar las publicaciones, por otra parte, para actualizar el listado hay que desplazarse hasta el principio del listado deslizando el dedo hacia abajo hasta que aparezca la acción de actualizar.



Cada publicación consta de 4 elementos, la información del usuario que la ha publicado, el propio contenido de la publicación, la barra de etiquetas, y la barra de interacción:



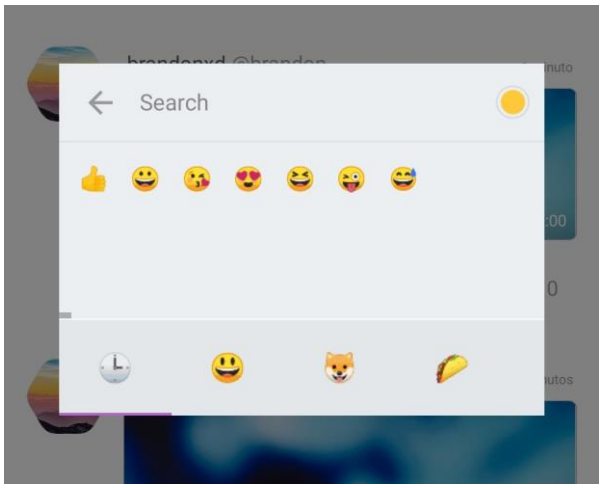
1. A través de la foto de perfil del elemento de información de usuario, se puede acceder a su perfil de usuario.
2. El contenido consiste en un reproductor de audio con una imagen de fondo, un botón, y un deslizador. El botón sirve para alternar la pausa, con el deslizador se puede establecer el momento de reproducción, mientras que si se toca en la imagen de fondo se navega al detalle de la publicación.



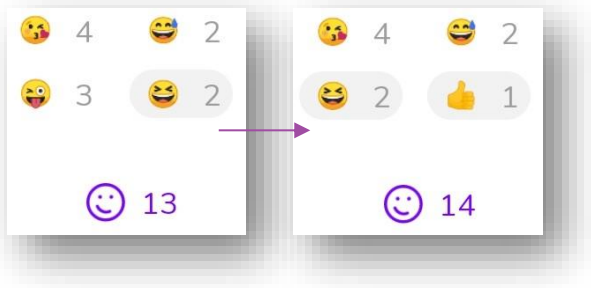
Como se puede ver, en esta pantalla se muestran mas detalles de la publicación, como la fecha completa de publicación, sus reacciones, comentarios, y sus respuestas.

3. La barra de etiquetas muestra las etiquetas de la publicación, además al pulsar sobre una etiqueta se navega a la pantalla de búsqueda para mostrar todas aquellas publicaciones que tienen esa misma etiqueta.
4. La barra de interacción está formada por varios botones con iconos descriptivos que muestran el número de comentarios, respuestas, veces que se ha compartido y reacciones de la publicación. Además, mediante estos mismos botones se puede comentar, responder con otro audio, compartir, y reaccionar, para ello simplemente hay que pulsar sobre el botón indicado para la acción deseada.

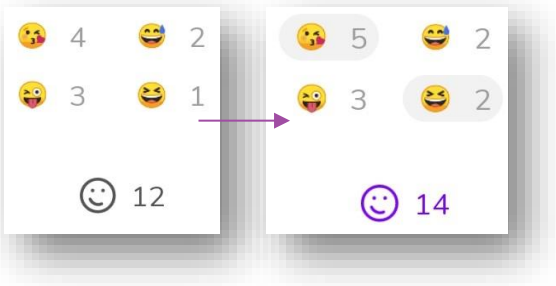
Es posible reaccionar desde una publicación, su detalle o desde la propia reacción al pulsar sobre esta. En esta acción se muestra una ventana modal con las reacciones disponibles.



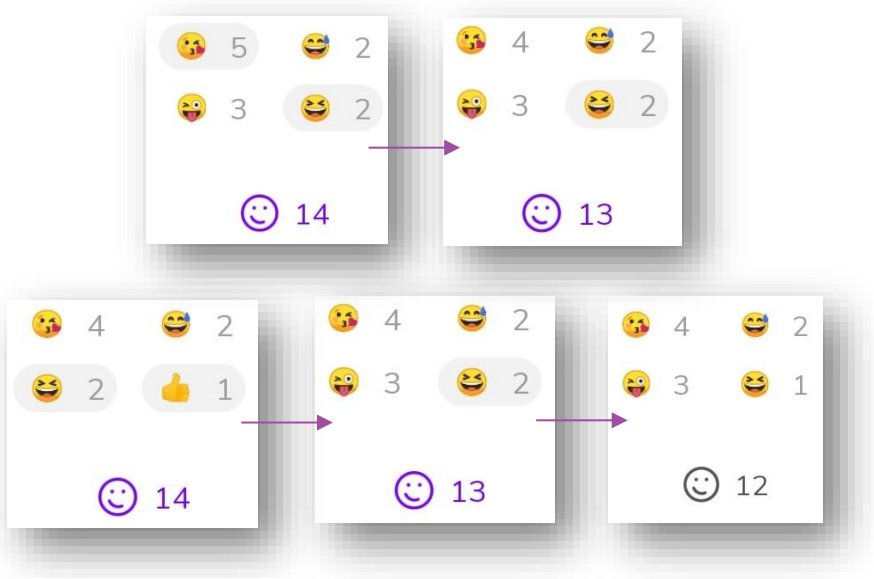
Al seleccionar una reacción aumentará el contador de reacciones de la publicación, y en el detalle de la publicación se podrá ver la nueva reacción, con su propio contador que habrá incrementado en 1.



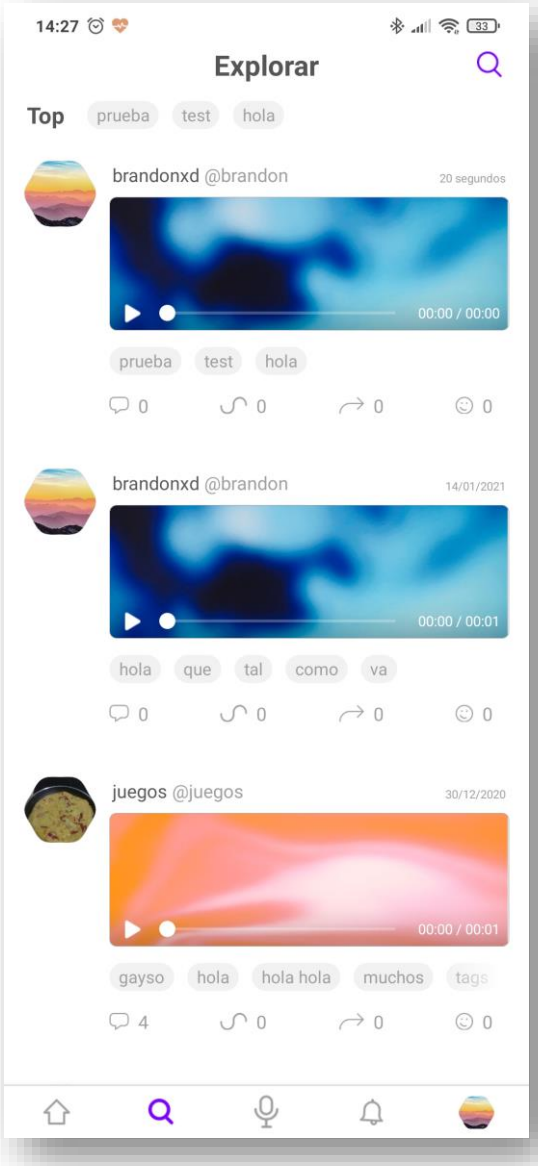
Además, si se pulsa sobre alguna reacción sin sombrear, se crea una nueva reacción del mismo tipo que se ha pulsado, aumenta el contador de esta reacción en 1, y se sombrea, indicando de esta manera que se ha reaccionado con esa reacción en específico.



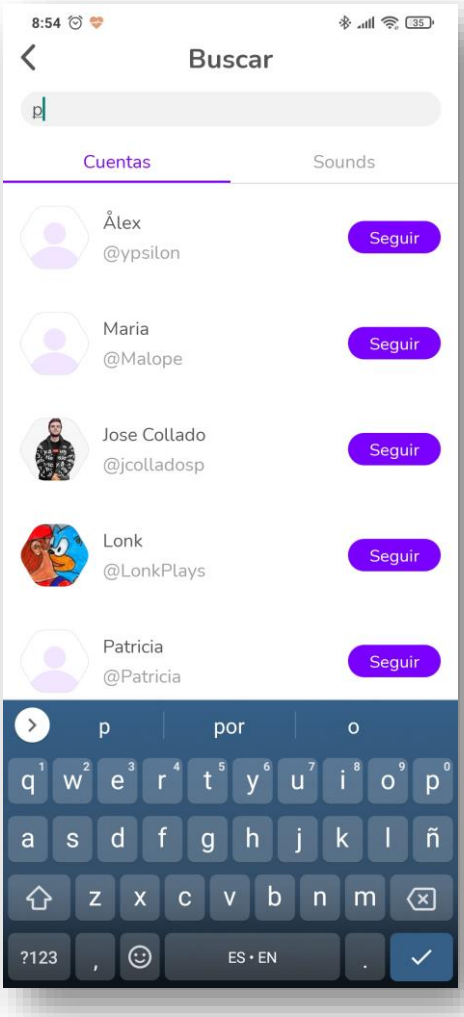
Para borrar reacciones, únicamente se puede hacer desde el detalle de la publicación, pulsando sobre la misma reacción, aquellas que aparecen sombreadas son a las que se ha reaccionado, y las que se pueden eliminar, al hacerlo desaparecerá el sombreado y el contador de esa reacción decrementará en 1, pero en caso de que el contador llegue a 0, desaparecerá también la reacción.

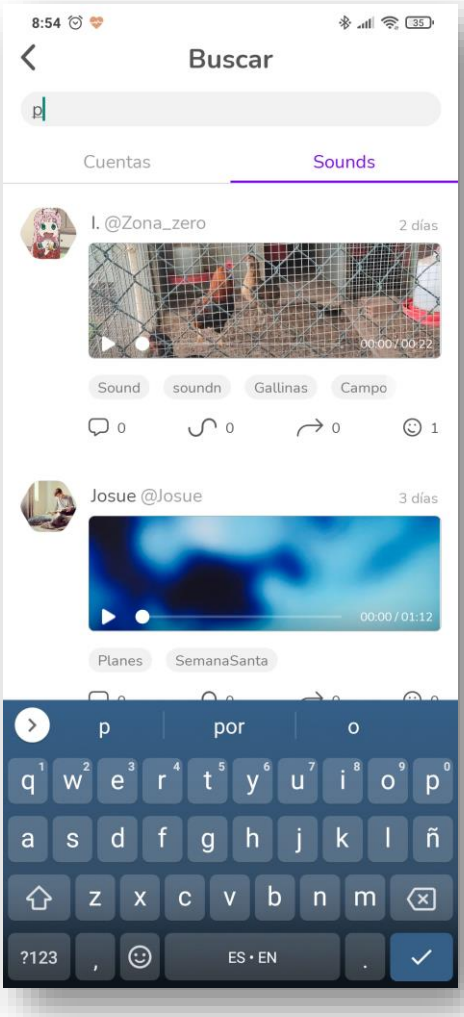


En la siguiente pantalla llamada explorar, se muestran las etiquetas más utilizadas en los últimos días, así como las publicaciones que las han utilizado. El funcionamiento es exactamente el mismo que en la pantalla principal, la única diferencia reside en las publicaciones que se muestran.

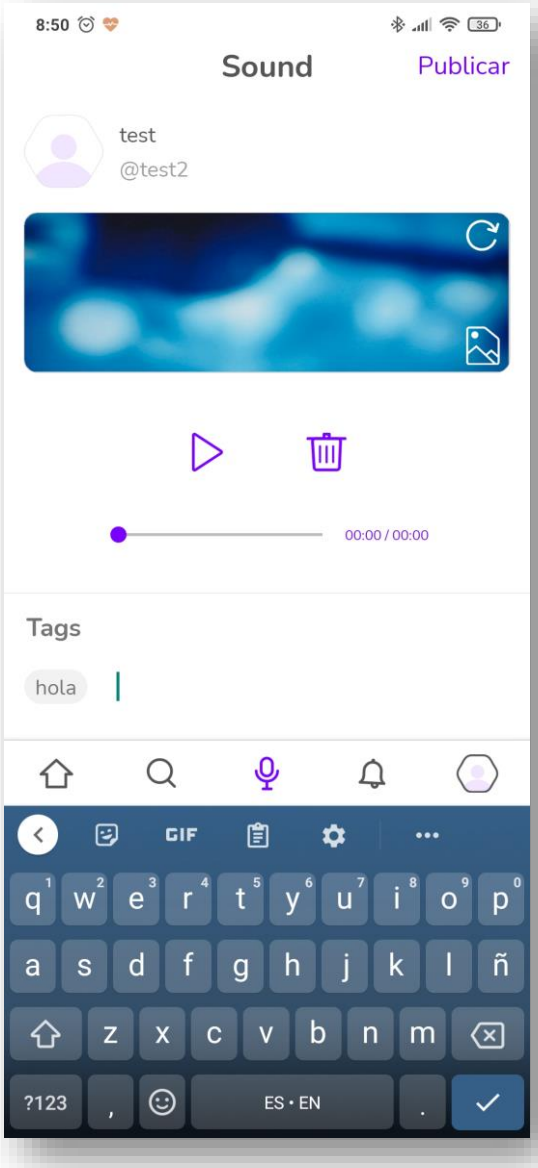


En esta misma pantalla se puede navegar a la pantalla de búsqueda mediante la lupa ubicada en la parte superior derecha. En esta pantalla se pueden buscar usuarios por su nombre de usuario, y publicaciones por sus tags, mediante el campo de texto ubicado en la parte superior donde a la vez que se escribe, se muestran los resultados organizados en pestañas de cuentas y sounds que tienen en el mismo formato de lista de publicaciones visto hasta ahora.



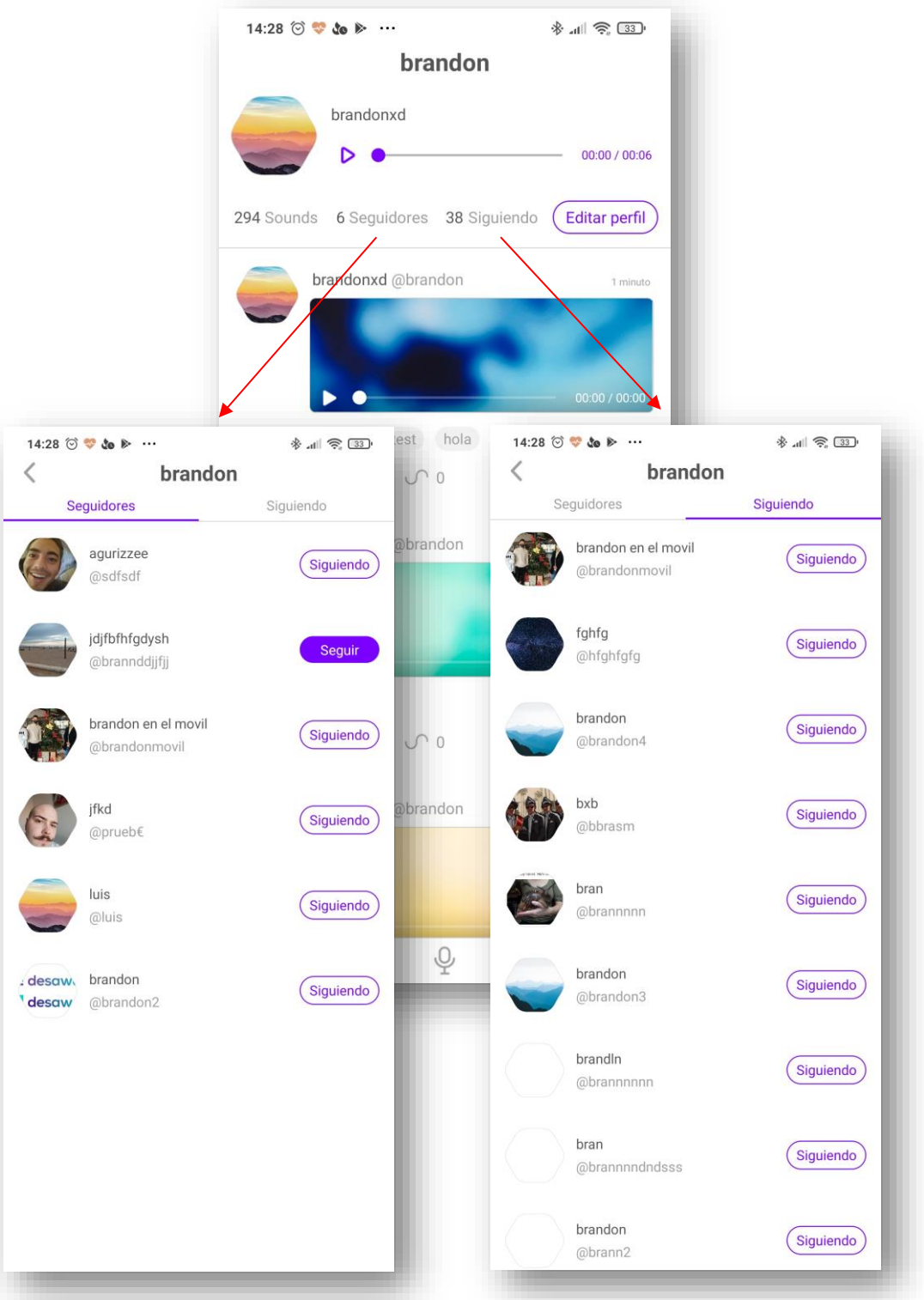


En la pantalla de creación de publicaciones, para crear una lo primero que hay que hacer establecer el fondo de la publicación, se puede seleccionar un fondo desde la galería, o uno por defecto aleatorio. Después de esto se puede grabar el contenido o subirlo desde el dispositivo. Por último, para añadir etiquetas se utiliza el campo de texto inferior, donde las etiquetas se separan automáticamente mediante el espacio, y a medida que se escriben se pueden previsualizar. En caso de querer eliminar una de estas etiquetas introducidas simplemente hay que pulsar sobre ella.



La pantalla de notificaciones, como su nombre indica, simplemente muestra las notificaciones que se han recibido, agrupándolas en pestañas de propias o siguiendo.

En la pantalla de perfil de usuario se muestra un listado con las publicaciones del usuario, y datos de seguimiento que si se pulsan navegan al listado de seguidores o personas que se están siguiendo.



Finalmente, si el perfil es el propio, aparece un botón que permite acceder a la pantalla de edición de datos del perfil, donde se puede cambiar la foto de perfil, el audio de perfil, el nombre de usuario, o el alias. Los cambios en esta pantalla solamente se hacen efectivos al pulsar sobre guardar de la parte superior, o se descarta, volviendo atrás, saliendo de la aplicación o pulsando en el cancelar de la parte superior.