Diseño y desarrollo de

una red social:

PureMusic

Grado en Ingeniería Informática

Trabajo F

in de Grado

Autor:

Antero Guarinos Caballero

Tutor/es:

Otto Colomina Pardo

Julio 2022



Índice de Contenidos

[1. Introducción 1](#_Toc107848062)

[1.1 Objetivos 1](#_Toc107848063)

[1.2 Estado de la cuestión 2](#_Toc107848064)

[Instagram 2](#_Toc107848065)

[Twitter 2](#_Toc107848066)

[Facebook 3](#_Toc107848067)

[Spotify 3](#_Toc107848068)

[Conclusión 4](#_Toc107848069)

[2. Tecnologías y herramientas 5](#_Toc107848070)

[2.1 Tecnologías y herramientas en el Frontend 6](#_Toc107848071)

[2.1.1 Angular 6](#_Toc107848072)

[2.1.2 Bootstrap 7](#_Toc107848073)

[2.2 Tecnologías y herramientas en el backend 7](#_Toc107848074)

[2.2.1 Nestjs 7](#_Toc107848075)

[2.1.2 phpMyAdmin 8](#_Toc107848076)

[2.3 Herramientas en el desarrollo 9](#_Toc107848077)

[2.3.1 Visual Studio Code 9](#_Toc107848078)

[2.3.2 Git/GitHub 9](#_Toc107848079)

[2.3.3 Postman 10](#_Toc107848080)

[3. Arquitectura 11](#_Toc107848081)

[3.1 Modelo Cliente-Servidor 11](#_Toc107848082)

[3.2 Arquitectura de la aplicación 11](#_Toc107848083)

[4. Diseño de la aplicación 13](#_Toc107848084)

[4.1 Requisitos funcionales 13](#_Toc107848085)

[4.2 Diagrama de clases 23](#_Toc107848086)

[4.3 Casos de uso 24](#_Toc107848087)

[4.3.1 Contexto teórico 24](#_Toc107848088)

[4.3.2 Casos de uso de la aplicación 25](#_Toc107848089)

[4.4 Diseño de la interfaz 34](#_Toc107848090)

[5. Metodología de desarrollo 38](#_Toc107848091)

[5.1 Modelos del software 38](#_Toc107848092)

[Desarrollo iterativo e incremental 38](#_Toc107848093)

[Metodologías ágiles 38](#_Toc107848094)

[5.2 Scrum 39](#_Toc107848095)

[5.3 flujo de trabajo 42](#_Toc107848096)

[6. Testing 43](#_Toc107848097)

[7. Conclusiones y líneas futuras 47](#_Toc107848098)

[8. Bibliografía 47](#_Toc107848099)

[Anexo 1: Qué es NGRX 49](#_Toc107848100)

[Anexo 2: Autenticación con JWT en NestJS 52](#_Toc107848101)

[Anexo 3: Interfaz gráfica final 55](#_Toc107848102)

**Índice de tablas**

[Tabla 1: Requisito funcional 1 - Registrar Usuario 13](#_Toc107848103)

[Tabla 2: Requisito funcional 2- Subir foto de perfil 14](#_Toc107848104)

[Tabla 3: Requisito funcional 3- Iniciar sesion 14](#_Toc107848105)

[Tabla 4: Requisito funcional 4 - Cerrar sesión 15](#_Toc107848106)

[Tabla 5: Requisito funcional 5 - Ver perfil personal 15](#_Toc107848107)

[Tabla 6: Requisito funcional 6 - Modificar datos del perfil 16](#_Toc107848108)

[Tabla 7: Requisito funcional 7 - Subir post 16](#_Toc107848109)

[Tabla 8: Requisito funcional - Eliminar post 17](#_Toc107848110)

[Tabla 9: Requisito funcional 9 - Ver posts 17](#_Toc107848111)

[Tabla 10: Requisito funcional 10 - Filtrar posts 18](#_Toc107848112)

[Tabla 11: Requisito funcional - Ver un post 18](#_Toc107848113)

[Tabla 12: Requisito funcional 12 - Buscar un usuario 19](#_Toc107848114)

[Tabla 13: Requisito funcional 13 - Ver usuario que sigues 19](#_Toc107848115)

[Tabla 14: Requisito funcional 14 - Seguir un usuario 20](#_Toc107848116)

[Tabla 15: Requisito funcional 15 - Reproducir canción 20](#_Toc107848117)

[Tabla 16 : Requisito funcional 16 - Añadir categoría 21](#_Toc107848118)

[Tabla 17: Requisito funcional 17 - Borrar categoría 21](#_Toc107848119)

[Tabla 18: Requisito funcional - Borrar usuario 22](#_Toc107848120)

[Tabla 19: Requisito funcional 19 - Listar usuarios 22](#_Toc107848121)

[Tabla 20: Caso de uso 1 - Registrar usuario 25](#_Toc107848122)

[Tabla 21: Caso de uso 2- Iniciar sesión 26](#_Toc107848123)

[Tabla 22: Caso de uso 3- Añadir nuevo post 27](#_Toc107848124)

[Tabla 23: Caso de uso 4 - Buscar un usuario 27](#_Toc107848125)

[Tabla 24: Caso de uso 5 - Ver perfil de otro usuario 28](#_Toc107848126)

[Tabla 25: Caso de uso 6 - Ver post 28](#_Toc107848127)

[Tabla 26: Caso de uso 7 - Borrar post 28](#_Toc107848128)

[Tabla 27: Caso de uso 8 - Comentar post 29](#_Toc107848129)

[Tabla 28: Caso de uso 9 - Listar post 30](#_Toc107848130)

[Tabla 29: Caso de uso 10 - Seguir a un usuario 30](#_Toc107848131)

[Tabla 30: Caso de uso 11 - Listar usuarios que sigues 30](#_Toc107848132)

[Tabla 31: Caso de uso 12 - Listar categorías 31](#_Toc107848133)

[Tabla 32: Caso de uso 13 - Filtrar por categorías 32](#_Toc107848134)

[Tabla 33: Caso de uso 14 - Listar usuarios 33](#_Toc107848135)

[Tabla 34: Caso de uso 15 - Borrar usuarios 33](#_Toc107848136)

[Tabla 35: Caso de uso 16 - Añadir categoría 33](#_Toc107848137)

[Tabla 36: Caso de uso 17 - Borrar categoría 34](#_Toc107848138)

**Índice de figuras**

[Figura 1: Modelo cliente-servidor 11](https://capgemini-my.sharepoint.com/personal/antero_guarinos-caballero_capgemini_com/Documents/Desktop/proyectos%20angular/TFG/memoria.docx#_Toc107848139)

[Figura 2: Diagrama de clases de la aplicación 23](#_Toc107848140)

[Figura 3: Casos de uso de sesión 25](#_Toc107848141)

[Figura 4: Casos de uso de post 26](#_Toc107848142)

[Figura 5: Casos de uso de usuarios 29](#_Toc107848143)

[Figura 6: Casos de uso de categoría 31](#_Toc107848144)

[Figura 7: Casos de uso de admin 32](#_Toc107848145)

[Figura 8: modificación de la plantilla del header 35](#_Toc107848146)

[Figura 9: Modificación del registro de usuario 35](#_Toc107848147)

[Figura 10: Modificación de ver perfil 36](#_Toc107848148)

[Figura 11; Modificación de listar categorías 36](#_Toc107848149)

[Figura 12: Plantilla de crear un post 37](#_Toc107848150)

[Figura 13: Peticiones HTTP desde postman 43](#_Toc107848151)

[Figura 14: Ejemplo de petición desde postman 44](https://capgemini-my.sharepoint.com/personal/antero_guarinos-caballero_capgemini_com/Documents/Desktop/proyectos%20angular/TFG/memoria.docx#_Toc107848152)

[Figura 15: Error de petición desde postman 45](#_Toc107848153)

[Figura 16: Prueba de JWT desde postman 45](#_Toc107848154)

[Figura 17: Registro desde postman 46](#_Toc107848155)

[Figura 18: Publicando un post desde postman 46](#_Toc107848156)

[Figura 19: Ejemplo de intercambio de datos entre componentes 51](#_Toc107848157)

[Figura 20: Implementando JWT 52](#_Toc107848158)

[Figura 21: Implementando JWT - 2 53](#_Toc107848159)

[Figura 22: Implementando JWT - 3 53](#_Toc107848160)

[Figura 23: Comprobando cookies 54](#_Toc107848161)

[Figura 24: Implementando JWT - 4 54](#_Toc107848162)

[Figura 25: Implementando JWT – 5 54](#_Toc107848163)

[Figura 26: Interfaz final de registro 55](#_Toc107848164)

[Figura 27: Interfaz final de publicar post 55](#_Toc107848165)

[Figura 28: Interfaz final de ver un post 56](#_Toc107848166)

[Figura 29: Interfaz final de gestionar perfil 57](#_Toc107848167)

[Figura 30: Interfaz final de modificar post 57](#_Toc107848168)

[Figura 31: Interfaz final de ver categorías 58](#_Toc107848169)

[Figura 32: Interfaz final de página de inicio 58](#_Toc107848170)

[Figura 33: Interfaz final de ver perfil 59](#_Toc107848171)

[Figura 34: Interfaz final de login 60](#_Toc107848172)

# 1. Introducción

Hace más de una década, con el gran desarrollo de las redes sociales, el intercambio de información y las relaciones interpersonales en Internet se volvieron indispensables, y algunas plataformas asumieron el monopolio de las redes sociales, a veces incluso uniéndose bajo una misma marca. Estas grandes plataformas reúnen a la gran mayoría de consumidores de estos productos, dejando pocas oportunidades para nuevos competidores que para tener éxito deben buscar características y nichos demográficos específicos que brinden las mejores y más especializadas soluciones a las necesidades que buscan atender.

En este proyecto se ha llevado a cabo el diseño y desarrollo de una nueva red social llamada PureMusic. Esta es una aplicación que busca poner en contacto a personas que comparten mismos gustos musicales, y a su vez, poder escuchar su música favorita sin necesidad de abrir varias aplicaciones simultáneamente.

Este documento se centrará en el proceso de ingeniería de software asociado con el desarrollo del proyecto propuesto. Profundizaremos en el análisis y especificación del proyecto a desarrollar, el diseño de la interfaz gráfica y la experiencia de usuario, y también destacaremos las claves para implementar una solución ordenada y fácilmente escalable en el tiempo.

## 1.1 Objetivos

1. Realizar un estudio de la situación actual de las redes sociales pioneras

2. Ofrecer a los usuarios la capacidad de que, con una sola aplicación, estar al tanto de noticias, eventos, nuevas canciones y publicaciones de sus artistas favoritos

3. seleccionar las tecnologías y herramientas más apropiadas para el desarrollo.

4. Implementar una solución incrementable y escalable con el tiempo.

## 1.2 Estado de la cuestión

Si tenemos que clasificar PureMusic en otras redes sociales esta irá en el ámbito de componente musical, es decir, la intención de esta aplicación es que el usuario solo acceda si de verdad necesita saber alguna novedad de algún artista, quedando así las otras redes sociales para consultar otra información que desee.

Por lo tanto, nos centraremos en redes sociales que sean similares o que suplen esta necesidad de cierta manera. Nos centraremos en Instagram, Facebook y Twitter.

### Instagram

Instagram es una red social muy popularizada entre jóvenes que ofrece la posibilidad de compartir fotografías con otros usuarios y poder recibir comentarios o “me gustas” (likes) de tus seguidores. Dentro de estas fotografías que podemos subir, podemos añadir etiquetas o hashtags para etiquetar según el tipo de fotografía o que se puede ver en ella, así será más fácil su clasificación a la hora de realizar búsquedas de una temática en concreto (geeknetic, 2020).

Esto está muy bien para ver lo que hacen tus artistas día a día y tener así un contacto más estrecho con ellos, pero Instagram no se diseñó para anunciar por ejemplo eventos, conciertos o quedadas (y mucho menos apuntarte a estos).

A diferencia de Instagram, PureMusic se centrará en proporcionar información de las novedades de nuestros artistas y músicos que sigamos, dejando así de lado lo que sería la vida social de cada uno de estos artistas.

### Twitter

Twitter es un servicio que permite que los grupos de amigos, familiares y compañeros de trabajo se comuniquen y estén en contacto a través de mensajes rápidos y frecuentes. Las personas publican Tweets, que pueden contener fotos, videos, enlaces y texto. Estos mensajes se publican en tu perfil, se envían a tus [seguidores](https://help.twitter.com/es/using-twitter/following-faqs.html) y también se pueden encontrar a través de la búsqueda de Twitter (Help.twitter, 2022).

Aunque en Twitter puede contener más variedad de contenido multimedia, sigue sin estar enfocado a un sector en concreto, ya que en esta red social podemos encontrar cualquier tipo de tema.

### Facebook

Facebook es una red social que fue creada para poder mantener en contacto a personas, y que éstos pudieran compartir información, noticias y contenidos audiovisuales con sus propios amigos y familiares (webEscuela, 2018).

Facebook es la red social preeminente con una amplia gama de usuarios y funciones, no fue diseñada para personas que buscan contenido exclusivamente relacionado con la música, pero debido a su gran cantidad de funciones, de alguna manera permite que se use con los "grupos" de Facebook. Los grupos son comunidades de usuarios en una red social reunidos por temas de grupo.

La diferencia que PureMusic pretende marcar es que, como ya hemos señalado, el propósito de esta red social es crear comunidades arraigadas para la música, en lugar de unir a las personas de una manera más casual.

### Spotify

Spotify es un servicio de música, pódcasts y vídeos digitales que te da acceso a millones de canciones y a otro contenido de creadores de todo el mundo (support.spotify, 2022).

Esta red social puede ser la que sea nuestra competencia directa, pero a diferencia de PureMusic vamos a poder interactuar con los artistas de una manera más cercana, ya que la principal funcionalidad de Spotify es reproducir música.

En Spotify para ser la red social pionera en streaming de música, tiene muy pocas funcionalidades sociales y han optado por otro tipo de funcionalidad que principalmente es la reproducción de música. Nuestra aplicación aparte de ofrecerte este sistema de escuchar música en streaming, el usuario podrá realizar cualquier otra acción dentro de la aplicación.

### Conclusión

En definitiva, lo que quiero transmitir y proporcionar con PureMusic es la capacidad de hacer las funciones más importantes de cada red social que he mencionado en una sola, es decir, esta red social busca rapidez y sencillez.

Como vamos a observar, esta red social no tiene nada innovador respecto a las que ya hay, pero te permite realizar cualquier acción con una sola aplicación.

Un ejemplo claro sería un usuario el que es aficionado a un artista en concreto. Para estar al tanto de las novedades del artista tiene que entrar a Twitter y buscar sobre su artista. Mientras tanto si quiere escuchar su música tendrá que abrir Spotify y poner la lista de reproducción. Si el usuario desea asistir a un evento o concierto de ese artista tendrá que buscar una página que proporcione la venta de entradas del mismo.

Como hemos visto, tendríamos 3 aplicaciones simultáneas para satisfacer las necesidades del usuario sobre un mismo tema (podríamos haber incluido más aplicaciones para satisfacer más necesidades).

# 2. Tecnologías y herramientas

Las aplicaciones web son programas informáticos que se ejecutan en servidores web a los que los usuarios pueden acceder a través de Internet mediante un navegador. Todos los datos que usa se almacenan en la red y se procesan allí, por lo que, a diferencia de las aplicaciones de escritorio, no es necesario instalarlas en su computadora, pero sí requieren una conexión a la red.

Las aplicaciones web se basan en una arquitectura cliente-servidor. Combinan secuencias de comandos del lado del servidor con secuencias de comandos del lado del cliente (JavaScript y HTML) para administrar el almacenamiento y la recuperación de información para proporcionar información a los usuarios. Esto hace posible interactuar con ellos a través de formularios en línea, sistemas de gestión de contenido, carritos de compras, etc. (Profile, 2020).

Los distintos tipos de aplicaciones web que podemos encontrar en la actualidad son los siguientes (yeeply, 2020):

* [Aplicación web estática](https://www.yeeply.com/blog/6-tipos-desarrollo-de-aplicaciones-web/#estatica)
* [Aplicación web dinámica](https://www.yeeply.com/blog/6-tipos-desarrollo-de-aplicaciones-web/#dinamica)
* [Tienda virtual](https://www.yeeply.com/blog/6-tipos-desarrollo-de-aplicaciones-web/#tienda)
* [Portal web app](https://www.yeeply.com/blog/6-tipos-desarrollo-de-aplicaciones-web/#portal)
* [Aplicaciones web con gestor de contenido](https://www.yeeply.com/blog/6-tipos-desarrollo-de-aplicaciones-web/#gestor)

No obstante, todas las aplicaciones web tienen una serie de características en común (Profile, 2020):

* No necesitan descargarse e instalarse en el dispositivo, se alojan en el servidor y se accede a ellas desde cualquier navegador, evitando cualquier limitación de espacio.
* Son muy compatibles y accesibles. Se requiere un sólo desarrollo para que puedan ejecutarse en distintos sistemas operativos y sean accesibles desde cualquier dispositivo con conexión a internet.
* Su mantenimiento y actualización es sencilla. No hay incompatibilidad entre versiones porque todos los usuarios utilizan la misma.

## 2.1 Tecnologías y herramientas en el Frontend



### 2.1.1 Angular

Angular es un Framework de JavaScript de código abierto escrito en TypeScript. Su objetivo principal es desarrollar aplicaciones de una sola página. Google se encarga del mantenimiento y constantes actualizaciones de mejoras para este framework. (hiberus, 2021).

TypeScript define un conjunto de tipos de JavaScript para ayudar a los usuarios a escribir código JavaScript más comprensible. Todo el código TypeScript se compila en JavaScript y se ejecuta sin problemas en cualquier plataforma.

Específicamente, en el desarrollo de software, un framework es una estructura de soporte técnica y conceptual definida, generalmente con artefactos o módulos de software específicos, que pueden servir como base para organizar y desarrollar software.

En otras palabras, un framework es una plantilla, esquema o estructura conceptual basada en tecnología que le permite trabajar de una manera más simple. De esta forma se evitan posibles errores de programación.

Por lo tanto, un framework es un conjunto de herramientas y módulos que se pueden utilizar para diferentes proyectos. Facilita el desarrollo multifacético, mejorando el tiempo, la energía y la organización.

He escogido utilizar Angular como base para mi frontend ya que, incluyendo todas las facilidades que he nombrado anteriormente que nos aporta, no he tenido oportunidad de estudiarlo en la carrera y pienso que es una de las herramientas pioneras en el ámbito del desarrollo sorfware.

### 2.1.2 Bootstrap

[Bootstrap](https://getbootstrap.com/) es un framework de desarrollo web gratuito y de código abierto. Está diseñado para facilitar el proceso de desarrollo de los sitios web responsivos y orientados a los dispositivos móviles, proporcionando una colección de sintaxis para diseños de plantillas. En otras palabras, Bootstrap ayuda a los desarrolladores a [construir sitios web](https://www.hostinger.es/tutoriales/como-crear-una-pagina-web/) más rápidamente, ya que no tienen que preocuparse por los comandos y funciones básicos. Consta de scripts basados en [HTML](https://www.hostinger.es/tutoriales/que-es-html), [CSS](https://www.hostinger.es/tutoriales/que-es-css) y JS para diversas funciones y componentes relacionados con el diseño web. (Hostinger, 2022).

Utilizar una de las plantillas que nos ofrece Bootstrap nos será de gran utilidad, ya que podemos utilizar un diseño bonito, fácil e intuitivo de una manera rápida y sencilla.

No hay que olvidar que tenemos que modificar la plantilla conforme a nuestros datos, y proyectar toda la lógica de nuestros ficheros javascript en el html, por lo que esto tan solo nos ayudará en el diseño visual de nuestra aplicación.

## 2.2 Tecnologías y herramientas en el backend

### 2.2.1 Nestjs

****NestJS es un framework progresivo de Node.js para la creación de aplicaciones eficientes, confiables y escalables del lado del servidor, el cual está construido y es completamente compatible con[TypeScript](https://www.typescriptlang.org/) (no obstante aún nos permite la codificación en JS puro), combinando elementos de la programación orientada a objetos (POO en español; OOP, según sus siglas en inglés, Object-oriented programming), programación funcional (FP, según sus siglas en inglés, functional programming) y programación reactiva funcional (FRP, según sus siglas en inglés, functional reactive programming). (gfourmis, 2021)

Uno de los motivos por lo que he escogido este framework es porque está inspirado en angular, por lo que la gran mayoría de los conocimientos adquiridos aquí, los podemos explotar en angular y viceversa.

También es compatible con TypeScript, por lo que también nos será más fácil tanto la corrección y detección de errores, como su escalabilidad del desarrollo.

Gracias a Nestjs, podemos hacer uso de las librerías de Express, que nos serán de gran ayuda para las peticiones HTTP para nuestra API REST. También dispondremos de la librería de TypeORM para la persistencia de datos de nuestra aplicación, la cual hablaremos más adelante.

Gracias a este ORM, podremos manipular nuestra base de datos, pero a través de los objetos que hemos creado en nuestro código. Esto se consigue gracias al mapeo de datos de TypeORM.

### 2.1.2 phpMyAdmin

PhpMyAdmin es una aplicación web que sirve para administrar bases de datos MySQL de forma sencilla y con una interfaz amistosa. Se trata de un software muy popular basado en PHP. La ventaja de usar una aplicación web es que nos permite conectarnos con servidores remotos, a los cuales no siempre se puede acceder usando programas de interfaz gráfica.

Para usar phpMyAdmin simplemente necesitas subir el conjunto de archivos PHP que componen la aplicación a un servidor web, configurar con los datos de acceso a MySQL y empezar a administrar las bases de datos. Con phpMyAdmin puedes hacer todo tipo de operaciones, desde la creación y borrado de bases de datos a la administración de las tablas (crear, modificar y eliminar) y, por supuesto, de sus propios datos. (arsys, 2021).

Normalmente, una aplicación que su código crece y no es estática, necesita poder gestionar y almacenar los datos de esta, por lo que he optado por usar MySQL, un gestor de BBDD que nos permite conectarnos con el framework de nestjs para hacer consultas.

Hay diferentes formas de acceder al gestor de base de datos (como por ejemplo en terminal), pero yo he decidido usar phpMyAdmin ya que este dispone de interfaz gráfica y es muy intuitivo trabajar con él, además que ya nos proporciona muchas configuraciones e instalaciones listas con tan solo descargar el programa.

## 2.3 Herramientas en el desarrollo

### 2.3.1 Visual Studio Code

Como editor de código se escogió Visual Studio Code ya que permite la instalación de extensiones creadas por terceros para una mejor experiencia con la sintaxis de ciertos lenguajes de programación o nuevas características de estos.

### 2.3.2 Git/GitHub

Se escogió Git como sistema de control de versiones y la plataforma web Github para alojar el repositorio remoto. En esta plataforma también se utilizó una funcionalidad para la gestión de proyectos siguiendo una metodología de desarrollo iterativo y ágil, ofreciendo la posibilidad de crear issues, tableros, etc. del programa, entre otras cosas.

El repositorio se encuentra en el siguiente enlace:

### 2.3.3 Postman



Esta aplicación nos permite realizar pruebas a nuestra API, ya que es un cliente que hace peticiones HTTP para poder comprobar nuestras respuestas y hacer testeo de nuestro código.

# 3. Arquitectura

## 3.1 Modelo Cliente-Servidor

La arquitectura de la aplicación se basa en el patrón cliente-servidor Este es un modelo de comunicación que vincula a varios dispositivos a través de una red donde, el cliente, realiza peticiones de servicios al servidor, que se encarga de satisfacer dichos requerimientos (Pérez Porto & Gardey, 2018).

Con esta arquitectura, las tareas se distribuyen entre los servidores (que proveen los servicios) y los clientes (que demandan dichos servicios). Este tipo de modelos permite repartir de la capacidad de procesamiento (Pérez Porto & Gardey, 2018).

De acuerdo con Sommerville (2005, p. 226) los clientes pueden conocer los nombres de los servidores disponibles y los servicios que proporcionan, mientras que los servidores no necesitan conocer la identidad de los clientes. Estos acceden a los servicios proporcionados a través de llamadas utilizando un protocolo de petición-respuesta como puede ser el protocolo HTTP (Sommerville 2005, p. 226).

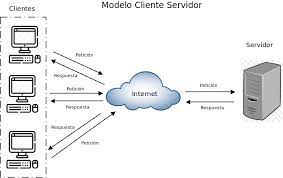


Figura 1: Modelo cliente-servidor

Una ventaja importante de este modelo es que es una arquitectura distribuida, se puede hacer uso de los sistemas de red con muchos procesadores distribuidos y es fácil añadir un nuevo servidor e integrarlo con el resto del sistema o actualizar los servidores de forma transparente sin afectar a este (Sommerville 2005, p. 227).

## 3.2 Arquitectura de la aplicación

Como hemos comentado en el punto anterior, disponemos de un servidor escrito en NodeJS (utilizando NestJS) que este se conecta a un servidor de base de datos de MySQL el cual es el que utilizaremos para nuestra persistencia de datos.

Por otro lado, en el lado del cliente está hecho con Angular, que mediante peticiones HTTP nos comunicaremos con el servidor para obtener los datos que solicitemos.

Al ser una aplicación relativamente pequeña y desarrollada por una única persona, se ha decidido no implementar un gestor de estados para el lado del cliente, ya que la cantidad de datos a gestionar es todavía relativamente pequeña. Estamos hablando de Utilizar REDUX que se traduce en “gestor de estados” del lado del cliente.

Este gestor de estados nos permitiría separar los servicios que se encargan de hacer peticiones HTTP de los demás componentes de angular, ya que gracias a REDUX podemos reutilizar diferentes formar de llamar al servicio con distintos parámetros o que nos devuelva los datos que únicamente necesitemos en ese componente (Ver más en Anexo 1)

# 4. Diseño de la aplicación

En el proceso de especificación y diseño de la aplicación primero se definieron los requisitos o requerimientos funcionales de la aplicación. Seguidamente veremos el diagrama de clases. Por último, repasaremos el diseño inicial de la interfaz de la aplicación.

## 4.1 Requisitos funcionales

De acuerdo con Sommerville (2005, p. 108), los requerimientos para un sistema son la descripción de los servicios proporcionados por el sistema y sus restricciones. Estos requerimientos reflejan las necesidades de los clientes de un sistema que ayuda a resolver algún problema como el control de un dispositivo, hacer un pedido o encontrar información (Sommerville 2005, p. 108). El proceso de descubrir, analizar, documentar y verificar estos servicios o restricciones se denomina ingeniería de requerimientos (Sommerville 2005, p. 108).

Según Sommerville (2005, p. 109) los requerimientos funcionales, en concreto, son declaraciones de los servicios que debe proporcionar el sistema, de la manera en que éste debe reaccionar a entradas particulares y de cómo se debe comportar en situaciones particulares. En algunos casos, los requerimientos funcionales de los sistemas también pueden declarar explícitamente lo que el sistema no debe hacer (Sommerville 2005, p. 109).

Los requerimientos funcionales son:

|  |
| --- |
| **Registrar usuario** |
| **Justificación:** Un usuario puede tener una cuenta en la aplicación |
| **Precondiciones:** El usuario puede iniciar sesión cuando desee |
| **Datos de entrada:** Nombre de usuario, email, contraseña, rol, foto de perfil |
| **Descripción:** El usuario introduce sus datos |
| **Pruebas de aceptación:** el email no repetido en la BBDD |
| **Postcondiciones:** Los datos del usuario se guardarán en la base de datos, la foto en el sistema de archivos del servidor y la sesión se iniciará automáticamente accediendo a la pantalla principal. |

Tabla 1: Requisito funcional 1 - Registrar Usuario

|  |
| --- |
| **Subir foto de perfil** |
| **Justificación:** Un usuario puede subir una foto de perfil al registrarse |
| **Precondiciones:** El usuario necesita tener una foto de perfil en su ordenador |
| **Datos de entrada:** La foto en cuestion |
| **Descripción:** Selecciona una foto de la carpeta que desee |
| **Pruebas de aceptación:** La foto debe tener uno de los formatos de imagen |
| **Postcondiciones:** La imagen se sube al servidor y se modifica el registro del usuario en la BD, añadiendo la ruta a su foto de perfil. |

Tabla 2: Requisito funcional 2- Subir foto de perfil

|  |
| --- |
| **Iniciar sesión** |
| **Justificación:** Un usuario quiere iniciar su sesión en su dispositivo. |
| **Precondiciones:** El usuario debe estar registrado en el sistema. |
| **Datos de entrada:** Email y contraseña |
| **Descripción:** El usuario introduce su correo y contraseña para generar un token de sesión. |
| **Pruebas de aceptación:** El correo electrónico y la contraseña son válidos. |
| **Postcondiciones:** La sesión se iniciará accediendo a la pantalla principal, el token de sesión se almacenará. |

Tabla 3: Requisito funcional 3- Iniciar sesion

|  |
| --- |
| **Cerrar sesión** |
| **Justificación:** Un usuario quiere cerrar su sesión en su cierto dispositivo. |
| **Precondiciones:** El usuario debe estar registrado en el sistema. |
| **Datos de entrada:** Token de sesión. |
| **Descripción:** El usuario introduce su correo y contraseña para generar un token de sesión. |
| **Pruebas de aceptación:** El correo electrónico y la contraseña son válidos. |
| **Postcondiciones:** La sesión se iniciará accediendo a la pantalla principal, el token de sesión se almacenará. |

Tabla 4: Requisito funcional 4 - Cerrar sesión

|  |
| --- |
| **Ver perfil personal** |
| **Justificación:** Un usuario quiere ver información de su perfil o de otro usuario |
| **Precondiciones:** El usuario debe estar registrado en el sistema. |
| **Datos de entrada:** Token de sesión. |
| **Descripción:** El usuario introduce su correo y contraseña para generar un token de sesión. |
| **Pruebas de aceptación:** pulsar el botón de ver perfil |
| **Postcondiciones:** se muestra la información del usuario seleccionado |

Tabla 5: Requisito funcional 5 - Ver perfil personal

|  |
| --- |
| **Modificar datos del perfil** |
| **Justificación:** Un usuario quiere modificar algún dato de su perfil |
| **Precondiciones:** El usuario debe estar registrado en el sistema. |
| **Datos de entrada:** Token de sesión. |
| **Descripción:** El usuario rellena todos los campos que desea modificar, incluyendo si desea una nueva foto de perfil |
| **Pruebas de aceptación:** pulsar el botón de confirmar |
| **Postcondiciones:** se muestra la información actualizada del usuario |

Tabla 6: Requisito funcional 6 - Modificar datos del perfil

|  |
| --- |
| **Subir post** |
| **Justificación:** Un usuario quiere subir contenido |
| **Precondiciones:** El usuario debe estar registrado en el sistema. |
| **Datos de entrada:** Titulo del post, comentario, foto, canción, categoría, token de sesión. |
| **Descripción:** El usuario rellena todos los campos que desea |
| **Pruebas de aceptación:** pulsar el botón de confirmar |
| **Postcondiciones:** se muestra la información actualizada del usuario |

Tabla 7: Requisito funcional 7 - Subir post

|  |
| --- |
| **Eliminar post** |
| **Justificación:** Un usuario quiere borrar algo de su contenido |
| **Precondiciones:** El usuario debe estar registrado en el sistema. |
| **Datos de entrada:** El post a borrar, token de sesión. |
| **Descripción:** El usuario selecciona el post y pulsa el botón de borrar |
| **Pruebas de aceptación:** pulsar el botón de confirmar |
| **Postcondiciones:** se muestra la información actualizada del usuario |

Tabla 8: Requisito funcional - Eliminar post

|  |
| --- |
| **Ver Posts** |
| **Justificación:** Un usuario quiere ver el contenido de los demás usuarios |
| **Precondiciones:** El usuario debe estar registrado en el sistema y seguir a 1 usuario o más. |
| **Datos de entrada:** token de sesión. |
| **Descripción:** El usuario se posiciona en el home |
| **Pruebas de aceptación:** pulsar el botón de home |
| **Postcondiciones:** se muestra los posts en orden más reciente |

Tabla 9: Requisito funcional 9 - Ver posts

|  |
| --- |
| **Filtrar Post** |
| **Justificación:** Un usuario quiere ver el contenido de los demás usuarios por categorías |
| **Precondiciones:** El usuario debe estar registrado en el sistema y seguir a 1 usuario o más. |
| **Datos de entrada:** token de sesión. |
| **Descripción:** El usuario se posiciona en el home y selecciona una de las categorías |
| **Pruebas de aceptación:** pulsar una de las secciones de categoría |
| **Postcondiciones:** se muestra los posts en orden más reciente filtrados por esa categoría |

Tabla 10: Requisito funcional 10 - Filtrar posts

|  |
| --- |
| **Ver un post** |
| **Justificación:** Un usuario quiere ver el contenido de un solo post |
| **Precondiciones:** El usuario debe estar registrado en el sistema |
| **Datos de entrada:** el post a ver,token de sesión. |
| **Descripción:** El usuario se posiciona en el home y selecciona un post que está visualizando |
| **Pruebas de aceptación:** pulsar en la foto del post |
| **Postcondiciones:** se muestra el contenido total del post |

Tabla 11: Requisito funcional - Ver un post

|  |
| --- |
| **Buscar un usuario** |
| **Justificación:** Un usuario quiere buscar un usuario en concreto |
| **Precondiciones:** El usuario debe estar registrado en el sistema |
| **Datos de entrada:** el campo de entrada del buscador,token de sesión. |
| **Descripción:** El usuario se posiciona en el home y selecciona el buscador |
| **Pruebas de aceptación:** pulsar el botón de buscar |
| **Postcondiciones:** se muestra el perfil del usuario |

Tabla 12: Requisito funcional 12 - Buscar un usuario

|  |
| --- |
| **Ver usuarios que sigues** |
| **Justificación:** Un usuario quiere ver a qué usuarios sigue |
| **Precondiciones:** El usuario debe estar registrado en el sistema |
| **Datos de entrada:** token de sesión. |
| **Descripción:** El usuario entra a su perfil y pulsa el botón de seguidos |
| **Pruebas de aceptación:** pulsar el botón de seguidores |
| **Postcondiciones:** se muestra la lista de usuarios a los que sigue |

Tabla 13: Requisito funcional 13 - Ver usuario que sigues

|  |
| --- |
| **Seguir un usuario** |
| **Justificación:** Un usuario quiere seguir a otro usuario |
| **Precondiciones:** El usuario debe estar registrado en el sistema |
| **Datos de entrada:** id del usuario a seguir,token de sesión. |
| **Descripción:** El usuario entra al perfil del usuario a seguir y pulsa el botón de seguir |
| **Pruebas de aceptación:** pulsar el botón de seguir |
| **Postcondiciones:** el botón de seguir cambia a “siguiendo” |

Tabla 14: Requisito funcional 14 - Seguir un usuario

|  |
| --- |
| **Reproducir canción** |
| **Justificación:** Un usuario quiere reproducir una canción de un post |
| **Precondiciones:** El usuario debe estar registrado en el sistema |
| **Datos de entrada:** post con canción,token de sesión. |
| **Descripción:** El usuario encuentra un post que tiene publicada una canción y desea reproducirla |
| **Pruebas de aceptación:** pulsar el botón de reproducir |
| **Postcondiciones:** Se abrirá un reproductor de contenido mp3 que reproduce la canción |

Tabla 15: Requisito funcional 15 - Reproducir canción

|  |
| --- |
| **Añadir categoría (rol admin)** |
| **Justificación:** El administrador quiere añadir una categoría |
| **Precondiciones:** El usuario debe estar registrado en el sistema y ser administrador |
| **Datos de entrada:** token de sesión. |
| **Descripción:** El usuario entra al perfil del administrador y crea una categoría nueva |
| **Pruebas de aceptación:** rellenar los campos |
| **Postcondiciones:** Se mostrará el listado de las categorías |

Tabla 16 : Requisito funcional 16 - Añadir categoría

|  |
| --- |
| **Borrar categoría (rol admin)** |
| **Justificación:** El administrador quiere borrar una categoría |
| **Precondiciones:** El usuario debe estar registrado en el sistema y ser administrador |
| **Datos de entrada:** la categoría a borrar, token de sesión. |
| **Descripción:** El usuario entra al perfil del administrador y borra una categoría |
| **Pruebas de aceptación:** ninguna |
| **Postcondiciones:** Se mostrará el listado de las categorías |

Tabla 17: Requisito funcional 17 - Borrar categoría

|  |
| --- |
| **Borrar usuario (rol admin)** |
| **Justificación:** El administrador quiere borrar un usuario |
| **Precondiciones:** El usuario debe estar registrado en el sistema y ser administrador |
| **Datos de entrada:** elusuarioa borrar, token de sesión. |
| **Descripción:** El usuario entra al perfil del administrador y borra un usuario |
| **Pruebas de aceptación:** ninguna |
| **Postcondiciones:** Se mostrará el listado de los usuarios |

Tabla 18: Requisito funcional - Borrar usuario

|  |
| --- |
| **Listar usuarios (rol admin)** |
| **Justificación:** El administrador quiere borrar una categoría |
| **Precondiciones:** El usuario debe estar registrado en el sistema y ser administrador |
| **Datos de entrada:** la categoría a borrar, token de sesión. |
| **Descripción:** El usuario entra al perfil del administrador y borra una categoría |
| **Pruebas de aceptación:** ninguna |
| **Postcondiciones:** Se mostrará el listado de las categorías |

Tabla 19: Requisito funcional 19 - Listar usuarios

## 4.2 Diagrama de clases

Según Kimmel (2008, p.101), el diagrama de clases constituye la vista más común e importante del diseño. Se les llama estáticos porque no describen acción, sino cosas y sus relaciones. Se diseñan para mostrar todas las piezas de la solución y deben transmitir un sentido del sistema que se estructurará en reposo (Kimmel 2008, p. 101).

La aplicación dispone de 4 entidades:

* Usuario
* Categoría
* Post
* Comentario

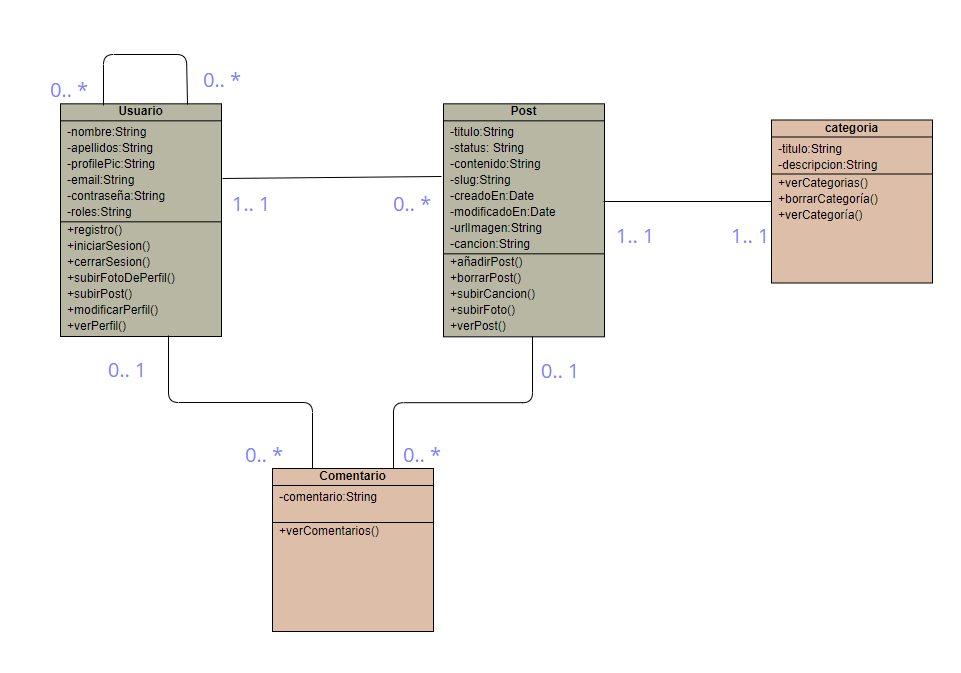


Figura 2: Diagrama de clases de la aplicación

## 4.3 Casos de uso

### 4.3.1 Contexto teórico

Según Cevallos (2019), un caso de uso es una descripción del funcionamiento de un sistema desde la perspectiva del usuario. Los diagramas de casos de uso utilizan actores y casos de uso para modelar la funcionalidad de un sistema.

**Elementos**

Un diagrama de caso de uso básico contiene solo unos pocos símbolos: actores, casos de uso y conectores. El personaje representa al actor, la persona que realiza la acción. La notación de casos de uso se usa para representar la funcionalidad del sistema, y ​​los conectores se usan para representar la relación entre los actores y los casos de uso. Los conectores varían en estilo para transmitir información más específica sobre la relación entre los actores y los casos de uso. Los conectores de un solo hilo llamados asociaciones conectan a los actores con los casos de uso que pueden ejecutar. Los conectores punteados se denominan dependencias.

**Estereotipado de los conectores**

Como dijo Kimmel (2008, pp. 24-25), los estereotipos agregan detalles a las relaciones entre elementos en un diagrama de casos de uso. Una dependencia entre dos casos de uso significa que, hasta cierto punto, un caso de uso dependiente requiere un caso de uso dependiente. Dos estereotipos que refinan las dependencias en un caso de uso son incluir y extender.

Una dependencia marcada con un estereotipo “incluir” significa que el caso de uso dependiente necesita implementar la implementación de la que depende, pero no al revés. Un caso de uso dependiente es una entidad completa y distinta y no debe depender de un caso de uso dependiente (Kimmel 2008, p. 25).

Los estereotipos de extensión se usan para agregar más detalles a las dependencias, lo que significa que estamos agregando más funcionalidad, opcionalmente (Kimmel 2008, p. 26).

### 4.3.2 Casos de uso de la aplicación

Dividiremos los casos de uso por secciones para una mejor visibilidad y entendimiento de los diagramas.

Primero mostramos los casos de uso de un usuario que entra en contacto con la aplicación por primera vez:

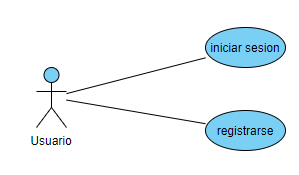


Figura 3: Casos de uso de sesión

|  |  |
| --- | --- |
| **Caso de uso: Registrar Usuario** | |
| **Actor: Usuario** | |
| **Curso Normal** | **Alternativas** |
| 1. Pulsar el botón de registro desde home. |  |
| 1. El usuario sube una foto de perfil y rellena los datos del formulario. |  |
| 1. El sistema comprueba que los datos con correctos. | Si no lo son muestra un mensaje de error. |
| 1. Registra el usuario en la BBDD. |  |
| 1. Se guarda la imagen en la carpeta. |  |

Tabla 20: Caso de uso 1 - Registrar usuario

|  |  |
| --- | --- |
| **Caso de uso: Iniciar sesión** | |
| **Actor: Usuario** | |
| **Curso Normal** | **Alternativas** |
| 1. Pulsar el botón de iniciar sesión desde home. |  |
| 1. Rellena los campos del formulario de inicio de sesión. |  |
| 1. El sistema comprueba que los datos son correctos. | Si no lo son muestra un mensaje de error. |
| 1. Se inicia sesión y te redirige a home. |  |

Tabla 21: Caso de uso 2- Iniciar sesión

En todos los demás casos de uso a continuación, damos por hecho que todo incluyen el caso de uso “Iniciar sesión”, puesto que todos son funcionalidades para usuarios registrados en la aplicación.

A continuación, mostramos los casos de uso relacionados con los posts del usuario y los perfiles de usuarios:

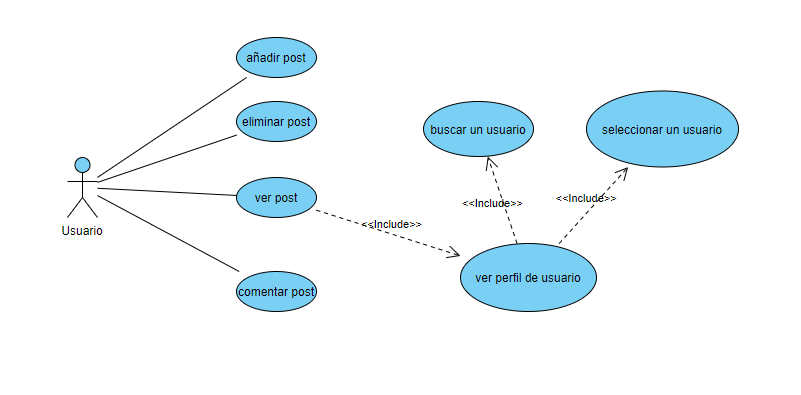


Figura 4: Casos de uso de post

|  |  |
| --- | --- |
| **Caso de uso:** Añadir nuevo post | |
| **Actor:** Usuario | |
| **Curso Normal** | **Alternativas** |
| 1. El usuario selecciona el botón de añadir un nuevo post desde su perfil. |  |
| 1. El usuario rellena los datos del formulario. |  |
| 1. Guarda el post en la BBDD. |  |
| 1. Se redirige a su perfil. |  |

Tabla 22: Caso de uso 3- Añadir nuevo post

|  |  |
| --- | --- |
| **Caso de uso:** Buscar un usuario | |
| **Actor:** Usuario | |
| **Curso Normal** | **Alternativas** |
| 1. En la página home, el usuario se sitúa en el buscador e introduce el nombre a buscar. |  |
| 1. El sistema hace una petición a la BBDD para buscar el nombre del usuario. |  |
| 1. El sistema muestra por pantalla una lista de todos los usuarios que contengan en su nombre lo que ha buscado el usuario. | Si no hay ninguno, no se mostraría nada. |
| 1. El usuario selecciona el usuario que desee. |  |

Tabla 23: Caso de uso 4 - Buscar un usuario

|  |  |
| --- | --- |
| **Caso de uso:** Ver perfil de otro usuario | |
| **Actor:** Usuario | |
| **Curso Normal** | **Alternativas** |
| 1. En la página home, el usuario selecciona el nombre de otro usuario que este sigue. | Hacer el proceso de “buscar un usuario”. |
| 1. El sistema hace una petición a la BBDD para buscar el nombre del usuario. |  |
| 1. El sistema muestra el perfil del usuario seleccionado. |  |

Tabla 24: Caso de uso 5 - Ver perfil de otro usuario

|  |  |
| --- | --- |
| **Caso de uso:** Ver post | |
| **Actor:** Usuario | |
| **Curso Normal** | **Alternativas** |
| 1. En la página home, el usuario selecciona el post a ver. | En el perfil de usuario, también puede seleccionar el post a ver. |
| 1. El sistema hace una petición a la BBDD para buscar el post. |  |
| 1. El sistema muestra el post seleccionado. |  |

Tabla 25: Caso de uso 6 - Ver post

|  |  |
| --- | --- |
| **Caso de uso:** Borrar post | |
| **Actor:** Usuario | |
| **Curso Normal** | **Alternativas** |
| 1. El usuario se dirige a su perfil. |  |
| 1. El usuario selecciona el post a borrar. |  |
| 1. El sistema hace una petición a la BBDD para borrar el post. |  |
| 1. Se redirige a su perfil de nuevo. |  |

Tabla 26: Caso de uso 7 - Borrar post

|  |  |
| --- | --- |
| **Caso de uso:** Comentar post | |
| **Actor:** Usuario | |
| **Curso Normal** | **Alternativas** |
| 1. El usuario selecciona un post. |  |
| 1. El usuario introduce un comentario en el post seleccionado. |  |
| 1. El sistema hace una petición a la BBDD y guarda el comentario. |  |
| 1. Se visualiza el post seleccionado anteriormente. |  |

Tabla 27: Caso de uso 8 - Comentar post

Ahora pasaremos a los casos de uso relacionados con los perfiles de otros usuarios (básicamente para poder listar los posts de los usuarios que sigues).



Figura 5: Casos de uso de usuarios

|  |  |
| --- | --- |
| **Caso de uso:** Listar post | |
| **Actor:** Usuario | |
| **Curso Normal** | **Alternativas** |
| 1. El usuario se dirige al home. |  |
| 1. El sistema hace una petición a la BBDD para buscar los posts. |  |
| 1. El usuario visualiza todos los posts de los usuarios a los que sigue. |  |

Tabla 28: Caso de uso 9 - Listar post

|  |  |
| --- | --- |
| **Caso de uso:** Seguir a un usuario | |
| **Actor:** Usuario | |
| **Curso Normal** | **Alternativas** |
| 1. El usuario selecciona un usuario mediante los casos de uso anteriores |  |
| 1. El usuario pulsa el botón de seguir | En caso de que le siga, aparecerá dejar de seguir |
| 1. El sistema hace una petición a la BBDD para guardar la información |  |
| 1. El usuario visualiza el perfil del usuario seleccionado |  |

Tabla 29: Caso de uso 10 - Seguir a un usuario

|  |  |
| --- | --- |
| **Caso de uso:** listar usuarios que sigues | |
| **Actor:** Usuario | |
| **Curso Normal** | **Alternativas** |
| 1. El usuario se dirige al su perfil. |  |
| 1. El usuario pulsa sobre el botón de seguidos. |  |
| 1. El sistema hace una petición a la BBDD para buscar los usuarios que sigues. |  |
| 1. Se muestra la lista de los usuarios que sigues. |  |

Tabla 30: Caso de uso 11 - Listar usuarios que sigues

los casos de uso de categorías.

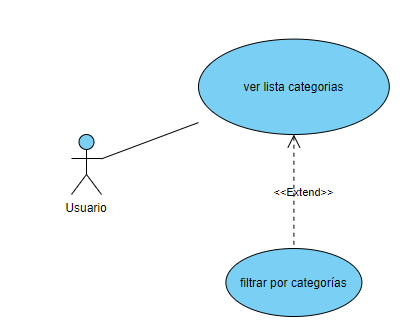


Figura 6: Casos de uso de categoría

|  |  |
| --- | --- |
| **Caso de uso:** Listar categorías | |
| **Actor:** Usuario | |
| **Curso Normal** | **Alternativas** |
| 1. El usuario se sitúa en el home. |  |
| 1. El sistema hace una petición a la BBDD para traer las categorías. |  |
| 1. El home muestra a un lateral todas las categorías disponibles. |  |

Tabla 31: Caso de uso 12 - Listar categorías

|  |  |
| --- | --- |
| **Caso de uso:** Filtrar por categorías | |
| **Actor:** Usuario | |
| **Curso Normal** | **Alternativas** |
| 1. El usuario se sitúa en el home. |  |
| 1. El sistema hace una petición a la BBDD para traer las categorías. |  |
| 1. El usuario selecciona una de las categorías. |  |
| 1. El sistema te redirige al home y muestra los post de los usuarios que sigues con el filtro de las categoría que has seleccionado. |  |

Tabla 32: Caso de uso 13 - Filtrar por categorías

Y, por último, los casos de uso del usuario administrador (este también tiene los casos de uso mencionado anteriormente).

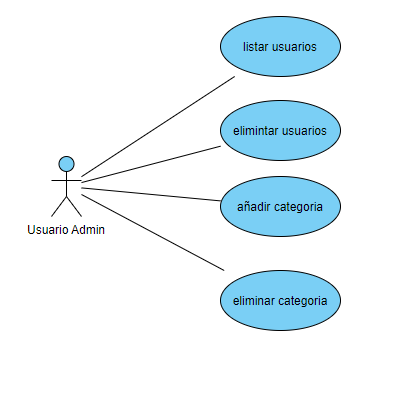


Figura 7: Casos de uso de admin

|  |  |
| --- | --- |
| **Caso de uso:** Listar usuarios | |
| **Actor:** Usuario | |
| **Curso Normal** | **Alternativas** |
| 1. El usuario se sitúa en la página de administrador. |  |
| 1. El sistema hace una petición a la BBDD para traer todos los usuarios. |  |
| 1. Se muestra el listado de los usuarios |  |

Tabla 33: Caso de uso 14 - Listar usuarios

|  |  |
| --- | --- |
| **Caso de uso:** Borrar usuarios | |
| **Actor:** Usuario | |
| **Curso Normal** | **Alternativas** |
| 1. El usuario se sitúa en la página de administrador. |  |
| 1. El sistema hace una petición a la BBDD para traer todos los usuarios. |  |
| 1. Se muestra el listado de los usuarios |  |
| 1. El usuario pulsa el botón de borrar sobre un usuario |  |

Tabla 34: Caso de uso 15 - Borrar usuarios

|  |  |
| --- | --- |
| **Caso de uso:** añadir categoría | |
| **Actor:** Usuario | |
| **Curso Normal** | **Alternativas** |
| 1. El usuario se sitúa en la página de administrador. |  |
| 1. El sistema hace una petición a la BBDD para traer todas las categorías. |  |
| 1. El usuario pulsa el botón de añadir categoría |  |
| 1. El usuario rellena el formulario | Si algún campo es erróneo saltará un error. |
| 1. El sistema registra en la BBDD la nueva categoría |  |

Tabla 35: Caso de uso 16 - Añadir categoría

|  |  |
| --- | --- |
| **Caso de uso:** borrar categoría | |
| **Actor:** Usuario | |
| **Curso Normal** | **Alternativas** |
| 1. El usuario se sitúa en la página de administrador. |  |
| 1. El sistema hace una petición a la BBDD para traer todas las categorías. |  |
| 1. El usuario pulsa el botón de borrar la categoría. |  |
| 1. El usuario visualiza el listado de categorías. |  |

Tabla 36: Caso de uso 17 - Borrar categoría

## 4.4 Diseño de la interfaz

Como se ha comentado en el punto 2, se ha utilizado las plantillas de Bootstrap para el diseño de la aplicación. Uno de los motivos principales por los que he decidido ayudarme de esta herramienta de diseño es su por su desarrollo responsive.

El desarrollo responsive nos permite visualizar nuestra aplicación de forma adaptable en cualquier tipo de dispositivo como por ejemplo una pantalla, teléfono, Tablet, etc. Esto es muy importante, ya que nuestro usuario podrá acceder a nuestra aplicación desde cualquier parte, sin necesidad de disponer de un ordenador.

Bootstrap me ha facilitado mucho la implementación y el desarrollo de mi proyecto ya que, al ser solo un desarrollador, implementar un modelo responsive se vuelve algo tedioso junto con las demás funcionalidades de la aplicación.

Los enlaces para previsualizar nuestro proyecto:

Página principal: <https://startbootstrap.com/previews/blog-home>

Página de ver un solo post: <https://startbootstrap.com/previews/blog-post>

Página de gestionar cuenta: <https://startbootstrap.com/previews/sb-admin-angular>

Aun así, se ha tenido que añadir cambios, adaptaciones o nuevas páginas por necesidades de implementación:

Primero, se ha añadido un reproductor para la funcionalidad de reproducir una canción

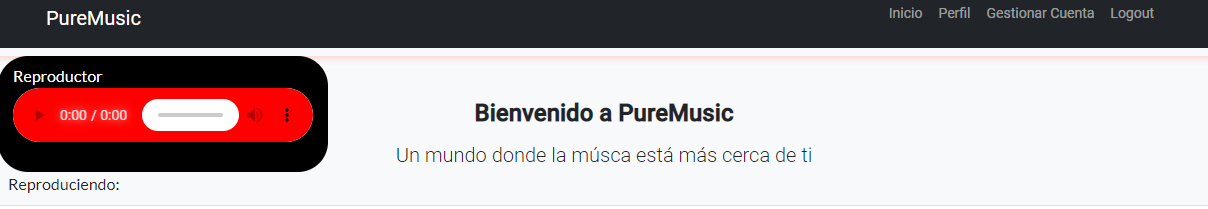


Figura 8: modificación de la plantilla del header

En el apartado de registro, hemos tenido que añadirle la posibilidad de subir una foto para así almacenarla en nuestro servidor.

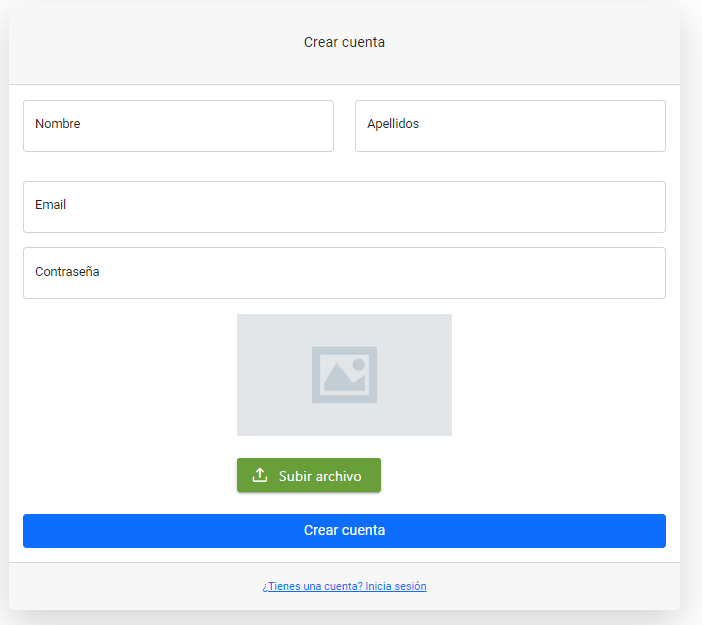


Figura 9: Modificación del registro de usuario

Para ver los seguidores y los seguidos, se ha implementado un botón con scroll para listar los mismos:

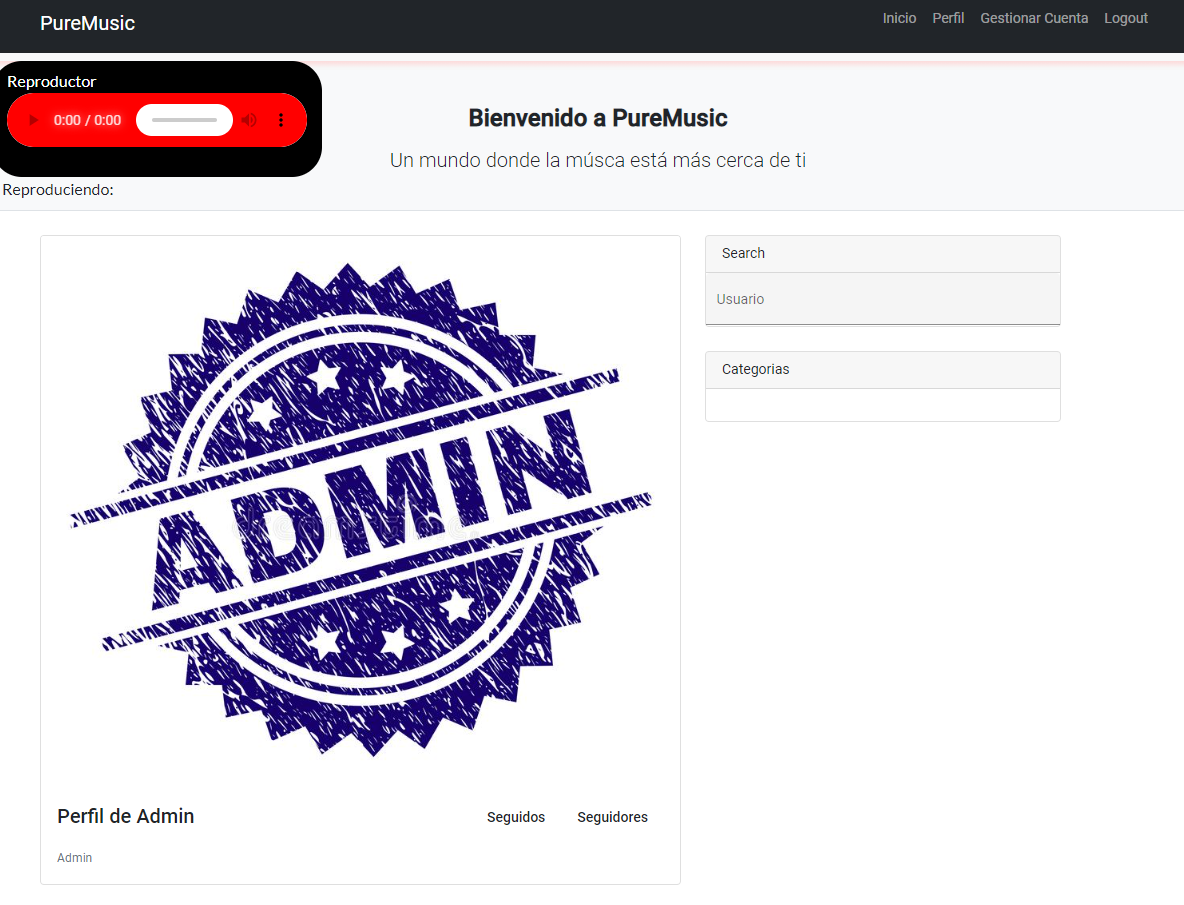


Figura 10: Modificación de ver perfil

Y, por último, listar categorías e usuario para la parte de administrador.

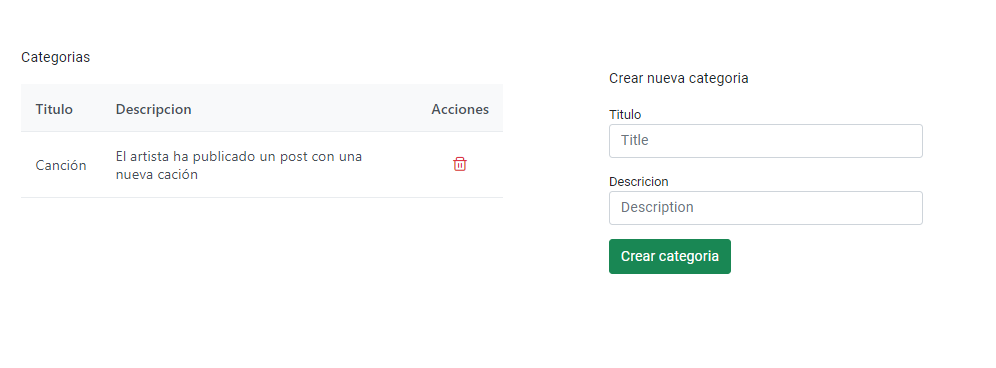


Figura 11; Modificación de listar categorías

Para crear un post o actualizarlo, hemos realizado una pestaña específica para él, en el que hacemos uso de la librería tiny para que el usuario pueda escribir cómodamente, al igual que 2 botones para subir tanto una canción, como una foto.

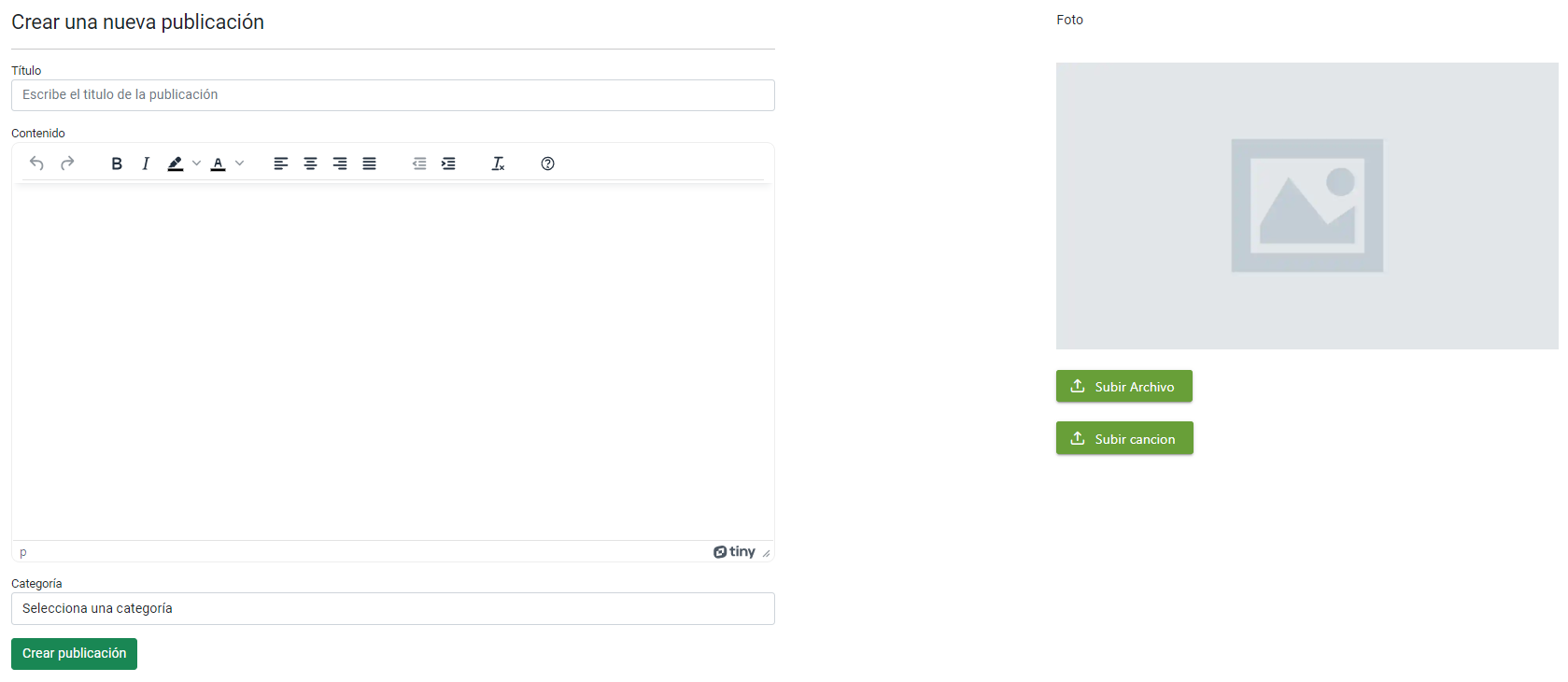


Figura 12: Plantilla de crear un post

Hay muchísimos más cambios, ya que como se ha nombrado en el punto 2, se ha ajustado la plantilla a nuestras necesidades, pero solo he querido nombrar las más importantes. Para una vista detalla de toda la aplicación, ver el Anexo 3 al final de este documento.

# 5. Metodología de desarrollo

Como explica Sommerville (2005, pp. 60-61), un modelo de software es una representación abstracta de un proceso de software, una abstracción de un proceso que se puede utilizar para explicar diferentes enfoques del desarrollo de software.

## 5.1 Modelos del software

### Desarrollo iterativo e incremental

En todos los grandes proyectos de software, el cambio es inevitable. Según Somerville (2005, p. 66), eso significa que el software no es un proceso único. En cambio, la actividad del proceso se repite periódicamente a medida que el sistema se reconstruye en función de las solicitudes de cambio. La esencia de un proceso iterativo es que la especificación se desarrolla junto con el software (Sommerville 2005, p. 66).

Según Sommerville (2005, p. 66), en la entrega incremental, la especificación, el diseño y la implementación del software se dividen en una serie de incrementos y se desarrollan secuencialmente. Los clientes determinan a priori los servicios que proporcionará el sistema, priorizan algunos servicios sobre otros y definen incrementos en los que cada servicio proporciona un subconjunto de la funcionalidad del sistema (Sommerville 2005, p. 66). De esta manera, los clientes no tienen que esperar hasta que el sistema esté completo para usarlo, también pueden usar el incremento inicial como prototipo y ganar experiencia (Sommerville 2005, p. 67).

### Metodologías ágiles

Como dice Sommerville (2005, p. 362), los métodos ágiles se basan en un enfoque iterativo para la especificación, el desarrollo y la entrega de software, principalmente para respaldar el desarrollo de aplicaciones donde los requisitos del sistema a menudo cambian rápidamente durante el desarrollo. Están diseñados para entregar rápidamente software funcional a los clientes, quienes luego pueden sugerir nuevos requisitos o cambios para su inclusión en iteraciones posteriores del sistema (Sommerville 2005, p. 362).

## 5.2 Scrum

Scrum es un marco de trabajo en equipo para el desarrollo ágil de soluciones (no solo software) que establece diferentes roles y responsabilidades para los miembros y proporciona un conjunto de conceptos y utilidades para ayudar en el proceso.

Dado que el desarrollo de este proyecto es un proceso individual, el marco se ha adaptado para usar los elementos y las utilidades que proporciona, como trabajos pendientes, sprints, historias de usuarios, que veremos a continuación.

Una cartera de pedidos (Backlog) es una lista del trabajo que se debe realizar en un producto, por lo que primero se especifica la cartera de pedidos inicial del producto que contiene las historias de los usuarios.

1. El usuario podrá tener una cuenta en el sistema.

2. El usuario podrá modificar su perfil.

3. El usuario podrá subir posts

4. El usuario podrá seguir a otros usuarios.

5. El usuario podrá ver las categorías.

6. El usuario podrá filtrar por categoría los posts.

7. El usuario podrá visualizar los posts de los usuarios a los que sigue

8. El usuario podrá borrar sus posts.

9. El usuario podrá visualizar contenido detallado de un post

10. El usuario podrá comentar un post.

11. El usuario podrá reproducir una canción de un post.

12. El usuario podrá subir una imagen para su perfil.

13. El usuario podrá subir una imagen para su post.

14. El usuario podrá subir un título y una descripción detallada para sus posts

15. El usuario administrador podrá eliminar un usuario.

16. El usuario administrador podrá listar todos los usuarios.

17. El usuario administrador podrá eliminar una categoría.

18. El usuario administrador podrá añadir una categoría.

Estas historias de usuario se refinan y se dividen en sprints. Antes de comenzar cada sprint, las historias de usuario se refinan y se dividen en tareas específicas. Este conjunto de tareas forma el sprint backlog, como se llama en Scrum. Con 6 sprints especificados, la estructura final es la siguiente:

Sprint 1: En este sprint empezaremos con nuestra estructura del backend y creando las entidades principales de nuestra aplicación.

Tareas:

* Crear estructura básica del backend.
* Crear modelos de Post, Usuario y Categoría.
* Crear controlador para Usuario y Categoría.
* Crear controlador para Post.
* Crear controlador para comentario

Sprint 2: En este segundo sprint nos centraremos en hacer las relaciones entre nuestras entidades y algunos servicios.

Tareas:

* Añadir relación entre usuario y post.
* Añadir relación entre categoría y post.
* Añadir relación entre comentario, post y usuario.
* Crear servicio de usuario.
* Crear servicio de categoría.
* Crear servicio de post.

Sprint 3: Como usuario quiero poder registrarme en la aplicación e iniciar sesión en ella para acceder a mis datos guardados.

Tareas:

* Implementar creación de token JWT.
* Implementar registro y login en el backend.
* Crear sistema de subida de archivos en el backend.
* Crear encriptación de contraseñas.

Sprint 4: Aquí nos encargaremos principalmente de proteger las rutas, añadir middleware y utilizar cookies.

Tareas:

* Añadir cookies.
* Utilizar guards para proteger rutas.
* Implementación de middleware.
* Añadir capacidad de recoger el usuario logueado mediante el token.

Sprint 5: En este sprint ya empezamos a implementar nuestro frontend y sus funcionalidades.

Tareas:

* Crear estructura básica del frontend.
* Crear componente de Usuario
* Crear componente de Header
* Crear componente de Post
* Crear componente de categorías

Sprint 6: en este sprint final, remataremos todo el trabajo que nos queda añadiendo el componente de comentarios, las funcionalidades de seguir a otro usuario, etc.

Tareas:

* Crear componente de comentarios.
* Crear funcionalidades de administrador.
* Finalizar funcionalidades finales del resto de componentes.

## 5.3 flujo de trabajo

Al ser un trabajo unipersonal, he decido usar un flujo de trabajo que sea cómodo para mí, por lo que he usado el flujo básico que ofrece GitHub.

El flujo básico se utiliza para desarrollo de una sola persona donde puede aprobar versiones de software localmente y luego enviarlas a la rama remota Máster. Durante este proceso, generalmente una persona realiza cambios de código y prueba localmente. Luego agrega los cambios al área de preparación, confirma y hace push para agregar los cambios al repositorio remoto.

Hago hincapié en que este método no sería el correcto si estuviéramos hablando de un equipo de desarrollo, ya que no tendríamos un seguimiento de todos los cambios individuales de cada colaborador y al contribuir a evitar que el trabajo concurrente entre en conflicto.

# 6. Testing

En esta sección, mostraremos las pruebas realizadas en la API de la aplicación. Para realizar estas pruebas se utiliza la herramienta Postman, descrita en la sección Técnicas y Herramientas de este documento.

Esta herramienta nos permite llamar a nuestra dirección deseada utilizando diferentes métodos HTTP para probar todos los recursos proporcionados por la API.

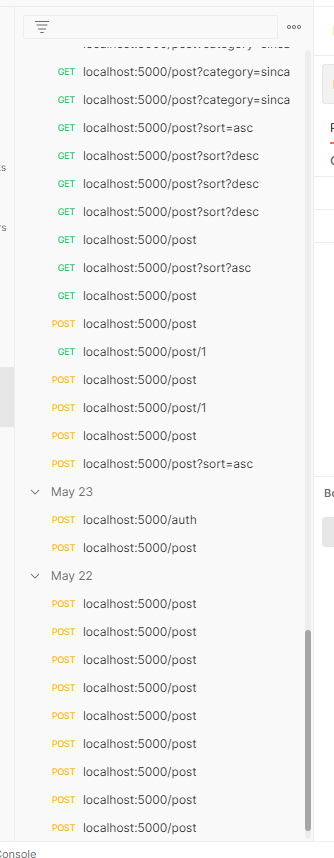


Figura 13: Peticiones HTTP desde postman

A medida que avanzaba el desarrollo y los sprints, aumentaba el catálogo de recursos disponibles en la API y, a su vez, aumentaba el número de pruebas en Postman, las cuales se iban almacenando en una misma colección.

Estas pruebas se pueden clasificar como pruebas funcionales, las cuales verifican que el código ejecutado tenga el comportamiento esperado.

Cada vez que se realiza un incremento en la aplicación, se agregan las pruebas necesarias para probar el nuevo código e integrarlo con el código anterior, por lo que todas las pruebas se vuelven a ejecutar para verificar que el comportamiento del código anterior no se vio afectado por el nuevo. incremento. Este tipo de pruebas se clasifican como pruebas de regresión.

A continuación, se muestran los diferentes tipos de respuestas que espera la API. Al acceder a un recurso válido, se devuelve un código 200 junto con el cuerpo de respuesta correspondiente.

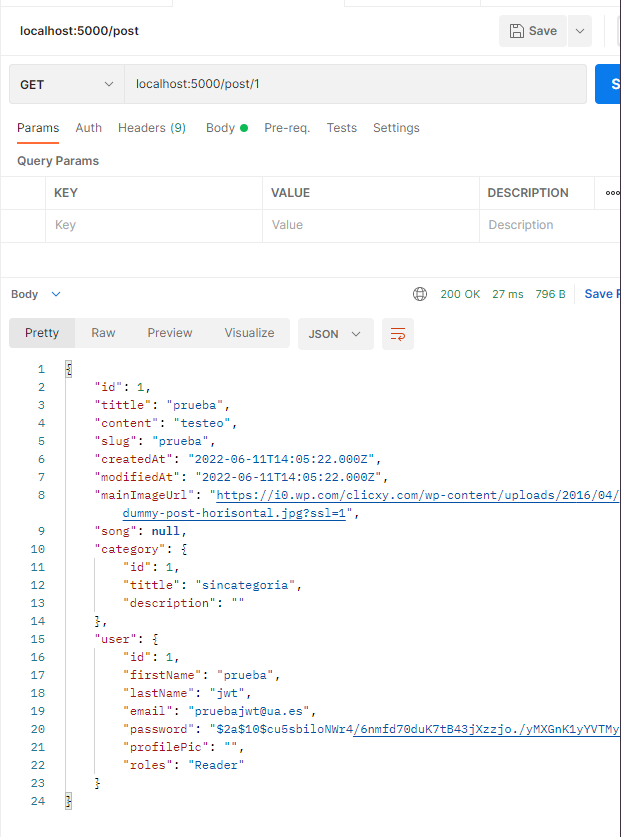


Figura 14: Ejemplo de petición desde postman

Sin embargo, si intentamos acceder a un recurso no disponible, como a la información de un usuario inexistente, se devuelve un 404:

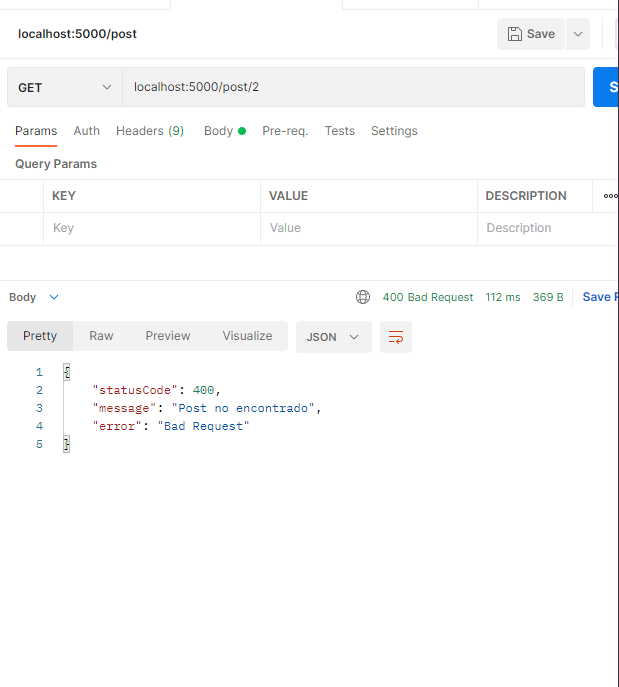


Figura 15: Error de petición desde postman

Otro ejemplo de las pruebas que he realizado es con el usuario registrado, ya que para publicar un post es necesario estar logueado en el sistema.

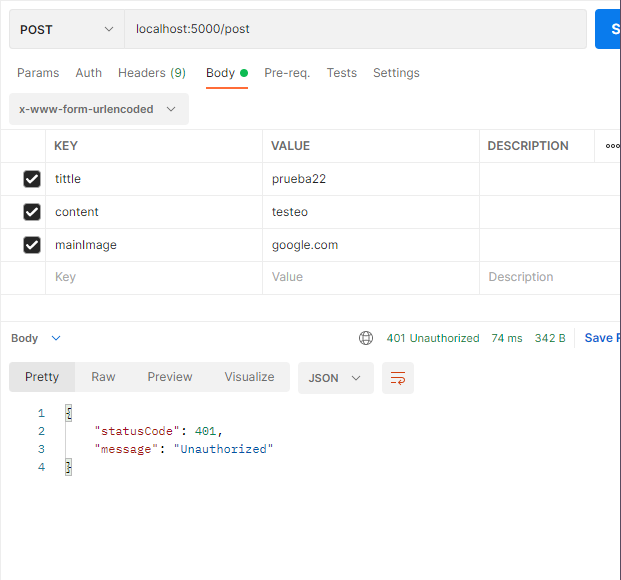


Figura 16: Prueba de JWT desde postman

Para ello nos registramos y por medio de las cookies podremos publicar un post.

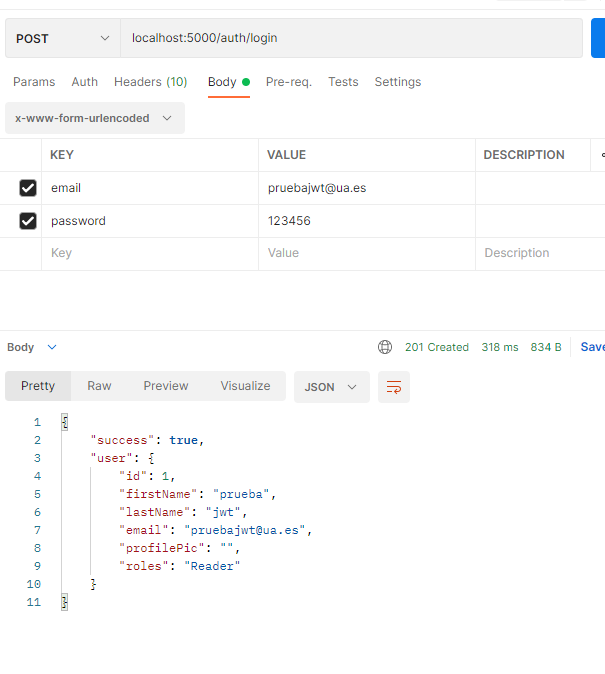


Figura 17: Registro desde postman

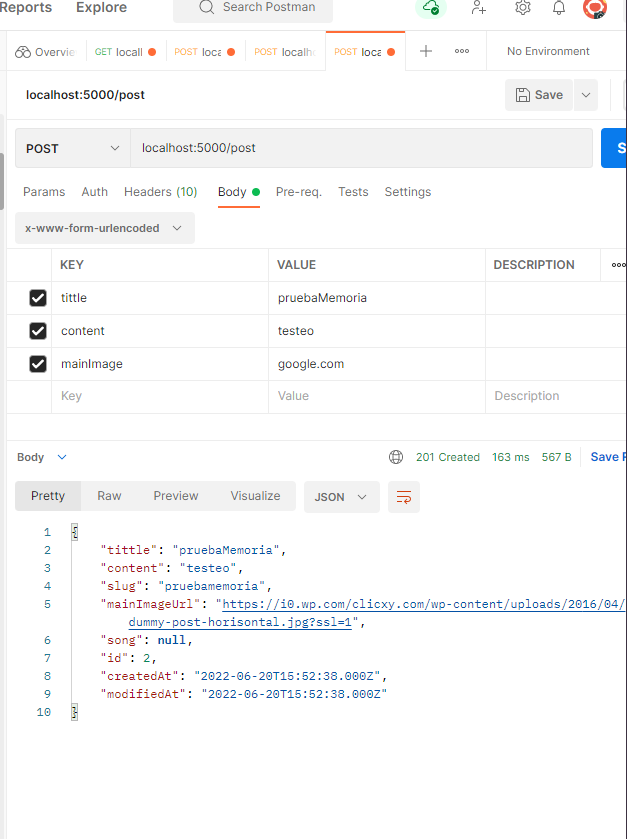


Figura 18: Publicando un post desde postman

# 7. Conclusiones y líneas futuras

En esta sección, veremos cómo se toman las decisiones sobre el desarrollo y la implementación del proyecto para lograr sus objetivos principales.

Estos objetivos son proponer una solución ideal a un problema real, crear un producto que se pueda alcanzar en la mayor cantidad y que el código pueda escalar y mejorar con el tiempo.

El uso específico de la tecnología híbrida facilita un mayor acceso y futuras incorporaciones a la aplicación que lleguen a los usuarios lo antes posible. Esto se debe a que se utiliza un conjunto específico de tecnologías independientemente de la plataforma en la que esté ejecutando la aplicación, lo que hace que el proceso de desarrollo sea único.

Con respecto a la escalabilidad del código, nuestra aplicación está pensada para que pueda seguir mejorando e implementando nuevas funcionalidades constantemente. Llegará un punto que tendremos demasiada información o demasiados archivos para trabajar, por lo que tendremos que optar por un sistema de estados, en concreto para angular está REDUX (véase Anexo 1).

Como líneas futuras se plantean posibles ampliaciones como:

* Chat entre dos o más usuarios de la aplicación.
* Poder inscribirse y pagar eventos de los artistas (como conciertos) desde la propia aplicación.
* Crear grupos de usuarios con los mismos intereses.
* Dar “me gusta” a un post.

# 8. Bibliografía

Geeknetic. (2020) ¿Qué es Instagram? Obtenido de:

<https://www.geeknetic.es/Instagram/que-es-y-para-que-sirve>

Help.twitter. (2022) ¿Qué es twitter? Obtenido de:

<https://help.twitter.com/es/resources/new-user-faq>

WebEscuela. (2018) ¿Qué es Facebook? Obtenido de:

<https://webescuela.com/facebook-que-es-como-funciona/#%c2%bfque-es-facebook>

Support.spotify. (2022). ¿Qué es Spotify? Obtenido de:

<https://support.spotify.com/es/article/what-is-spotify/>

Yeeply. (2020). Tipos de desarrollo de aplicaciones. Obtenido de:

<https://www.yeeply.com/blog/6-tipos-desarrollo-de-aplicaciones-web/>

Profile. (2020). Tipos de desarrollo de aplicaciones. Obtenido de:

<https://profile.es/blog/desarrollo-aplicaciones-web/>

Hostinger. (2022). ¿Qué es bootstrap? Obtenido de:

<https://www.hostinger.es/tutoriales/que-es-bootstrap>

Hiberus. (2022). ¿Qué es Angular? Obtenido de:

<https://www.hiberus.com/crecemos-contigo/que-es-angular-y-para-que-sirve/>

Gfourmis. (2021). ¿Qué es nestjs? Obtenido de:

https://gfourmis.co/nestjs-que-es-y-por-que-empezar-a-usarlo/

Arsys. (2021). ¿Qué es phpMyAdmin? Obtenido de:

<https://www.arsys.es/blog/phpmyadmin>

Sommerville, I. (2005). Ingeniería del Software 7ma. Ed. Madrid: Pearson Educación S.A.

<http://zeus.inf.ucv.cl/~bcrawford/AULA_ICI_3242/Ingenieria%20del%20Software%207ma.%20Ed.%20-%20Ian%20Sommerville.pdf>

Kimmel, P. (2008). Manual de UML. México D.F.: McGraw-Hill Interamericana Editores

<http://www.uenicmlk.edu.ni/img/biblioteca/Ing%20en%20sistema%20Manual%20de%20UML%20Kimmel.pdf>

Cevallos, K. (2015). Ingeniería del Software. Obtenido de:

<https://ingsotfwarekarlacevallos.wordpress.com/2015/06/04/uml-casos-de-uso/>

# Anexo 1: Qué es NGRX

Es un manejador de estados que usa RXJS diseñado para Angular que usa el patrón de arquitectura de datos llamado Redux. Estos conceptos nacen con la aparición de los SPA (Single Page Applications) porque antes la interacción del usuario con la web era más limitada, ya que cada acción que se realizaba era gestionada directamente por el backend.

RxJS es una librería de JavaScript para programación reactiva que facilita el trabajo con flujos de datos asíncronos bajo el uso de observables.

Podemos llamar a un objeto observador un sujeto con estado. Si el estado cambia, se notifica a sus suscriptores de los cambios, para que los suscriptores no tengan que preocuparse cuando ocurre un cambio de estado, porque el observador ya es responsable de notificar a los suscritos.

Con esto ya entramos a explicar qué es Redux, y es que el “estado” de la aplicación que en un principio, lo manejaba el backend, siendo él quien procesaba toda la lógica que el frontend le mandaba, y una vez procesada la información se la devolvía al front para que este la mostrase de nuevo.

Redux es un contenedor de estado predictivo que nos permite la gestión centralizada del estado y la lógica de nuestra aplicación. Nos brinda una arquitectura escalable con mayor control en el flujo de datos, esto es gracias al manejo del estado de la aplicación por parte de Redux. Al usar Redux, podemos hacer que nuestros cambios de estado sean predecibles en nuestra aplicación ya que nos facilita el manejo de errores con herramientas de depuración.

Por lo general, en una aplicación pequeña no es necesario poder manejar estos estados de forma diferente. Pero si tu aplicación es grande, es decir, está expuesta a continuos cambios es muy recomendable utilizar algún manejador de estados.

Gracias a este manejador de estados podemos separar de forma individual cada componente de la aplicación, y a su vez poder comunicarlo con otros componentes. Esto nos da mucha versatilidad, ya que nuestra aplicación carecerá de tantas dependencias (minimizar el acoplamiento)

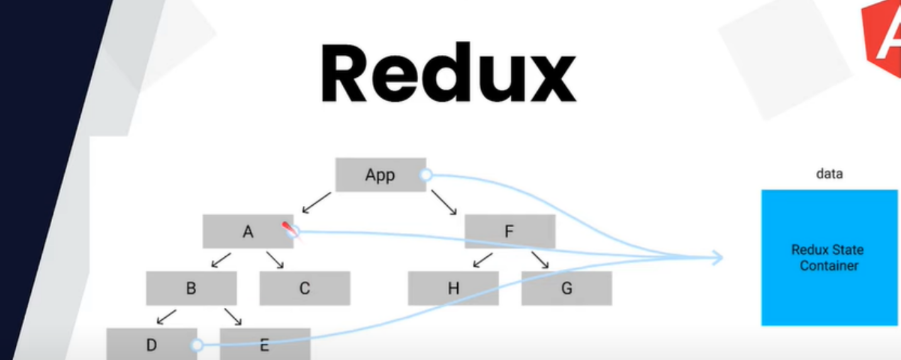


Figura 19: Ejemplo de intercambio de datos entre componentes

Aquí podemos ver como por ejemplo el componente D en vez de pedir información a su padre B, directamente se desentiende de él y le pide información al store de Redux.

Las partes que forman NGRX son:

* **Actions:** expresan un evento único que sucede a través de nuestra aplicación, es decir, son las encargadas de comunicar toda nuestra aplicación o de ejecutar otras acciones.
* **Reducers:** Estos son los responsables de cambiar de un estado actual a un NUEVO estado y están determinados por las acciones (ya que los reducers son los que escuchan las acciones).
* **Store:** Como concepto general, el store en Redux hace de base de datos del lado del frontend (en este caso, nuestro navegador web). Aquí será donde nuestra aplicación consulte los datos sin necesidad de estar llamando al backend con cada iteración que hagamos.
* **Selector:** A rasgos generales, son funciones puras que se encargan de obtener la información de nuestro store (puede ser toda la información o pequeñas propiedas/variables del store).
* **Effects:** Los efectos son los encargados de disparar las acciones que hacen que consuman servicios. Esto se hace para que los componentes queden aislados de los servicios, y no dependan de ellos.

Como conclusión hemos podido comprobar que es muy útil para usar la práctica de integración continua, ya que nos permite hacer integraciones casi automáticas (ya que el código de NGRX no varía mucho, aunque tiene distintas funcionalidades) para que así podamos detectar los fallos.

# Anexo 2: Autenticación con JWT en NestJS

Nuestra aplicación requiere una forma de saber si un usuario tiene la sesión iniciada o no, y a su vez no queremos que el usuario tenga que autenticarse cada vez que realiza alguna acción en la aplicación. Además, queremos restringir el acceso a algunos endpoints de forma que solo se pueda acceder cuando el usuario está registrado.

Para solventar todo lo que hemos nombrado vamos a hacer uso de la autenticación con json web tokens (JWT) ya que estos generan tokens “temporales” a partir de un usuario y una contraseña de nuestra BBDD.

La forma de implementar el JWT es algo compleja, pero gracias a la documentación oficial de Nestjs nos facilitará el trabajo.

Una vez descargadas todas las dependencias y paquetes necesarios debemos importarlos en nuestro componente de autentificación (generalmente tiene el nombre de auth).



Figura 20: Implementando JWT

Una vez hecho esto, pasaremos al servicio de nuestro auth component y tan solo buscaremos en la base de datos si nuestro usuario está registrado, y en caso de que lo esté, le asignaremos un token nuevo con la función jwt.signAsync



Figura 21: Implementando JWT - 2

Nuestro controlador, en caso de que el servicio no devuelva una excepción, creará dos cookies que servirá para comprobar que el usuario esté logueado.

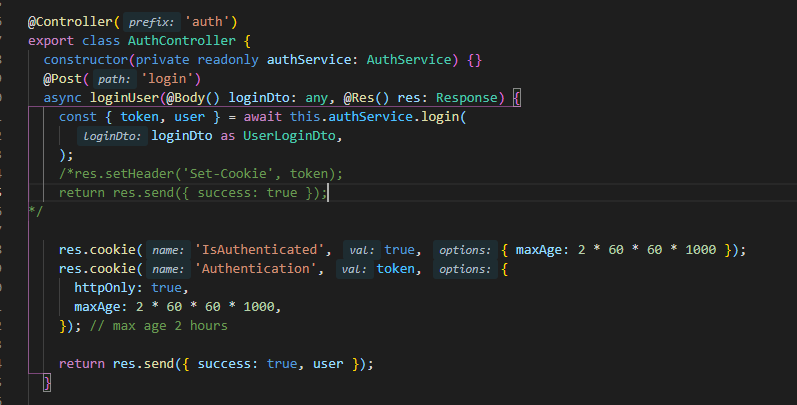


Figura 22: Implementando JWT - 3

Para comprobarlo, podemos iniciar sesión rápidamente desde Postman y veremos las 2 cookies que hemos devuelto en el controlador

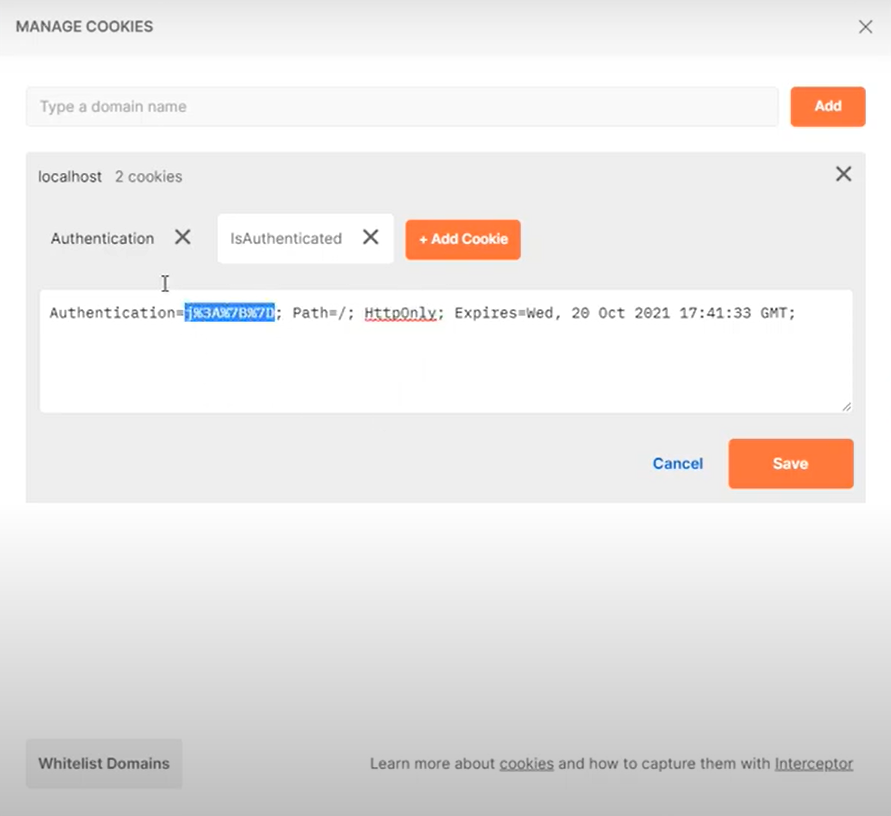


Figura 23: Comprobando cookies

Para una mayor seguridad frente amenazas, se ha implementado un hash para la contraseña del usuario que se registre en nuestro sistema. Siempre que un usuario se cree una cuenta, haremos uso del decorador @BeforeInsert para realizar el hasheo

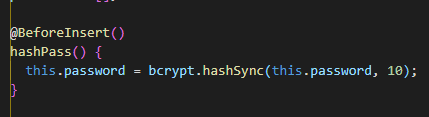


Figura 24: Implementando JWT - 4

Y gracias a bcrypt.compare, podemos comparar la contraseña que introduce el usuario para loguearse y la que tenemos almacenada en nuestra BBDD (podemos ver la función de this.verifyPassword en el servicio de auth)

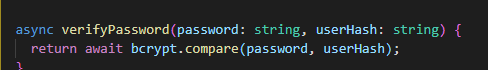


Figura 25: Implementando JWT – 5

# Anexo 3: Interfaz gráfica final

En esta sección mostraremos capturas del resultado final de la aplicación. Para que se vean todas las funcionalidades implementadas de cada pestaña he tenido que aumentar el campo de visión de la página, por lo que algunos componentes aparecerán movidos.

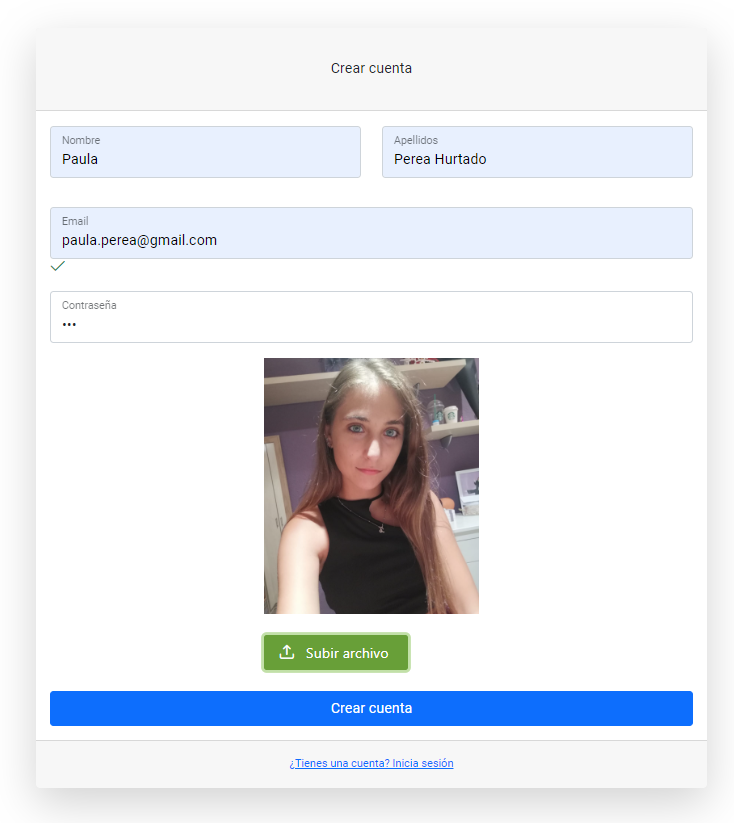


Figura 26: Interfaz final de registro

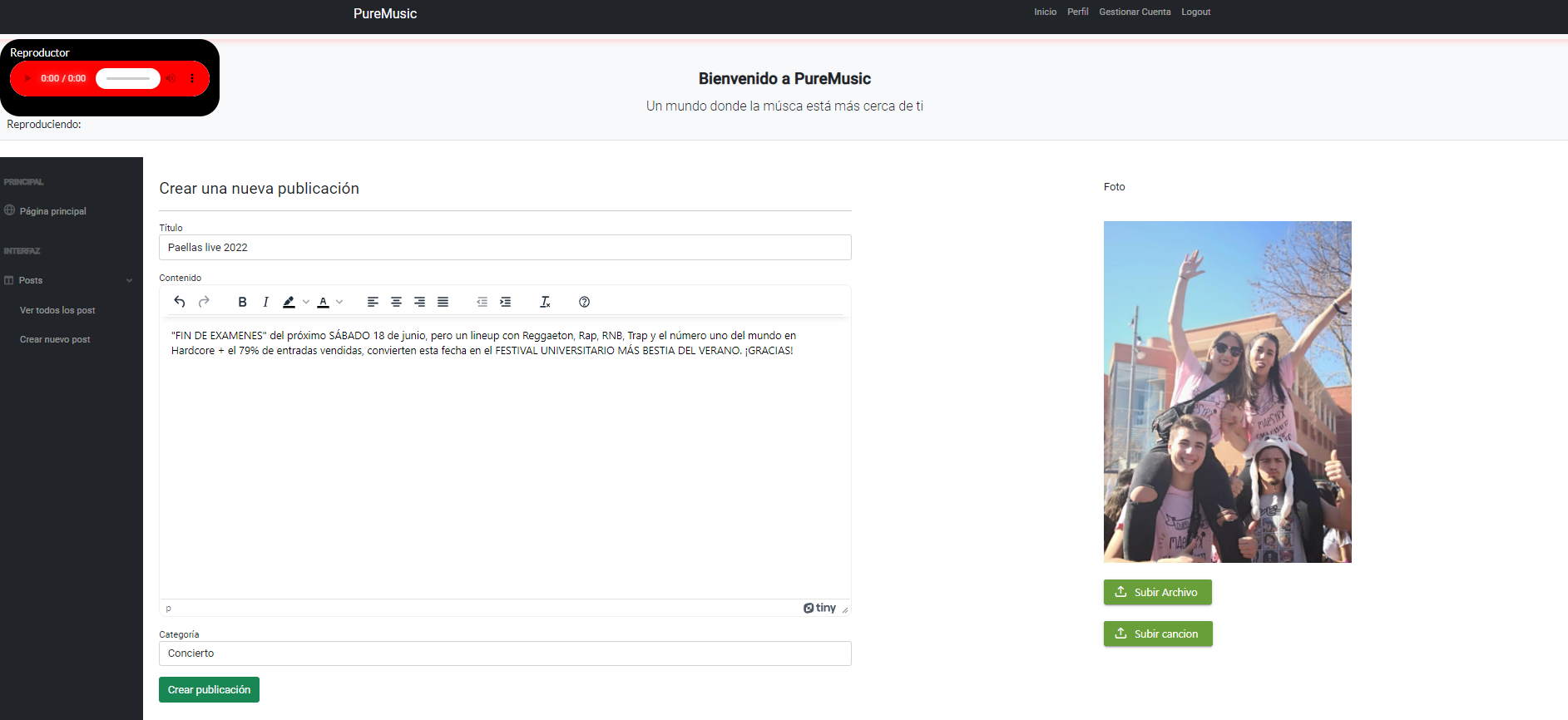


Figura 27: Interfaz final de publicar post

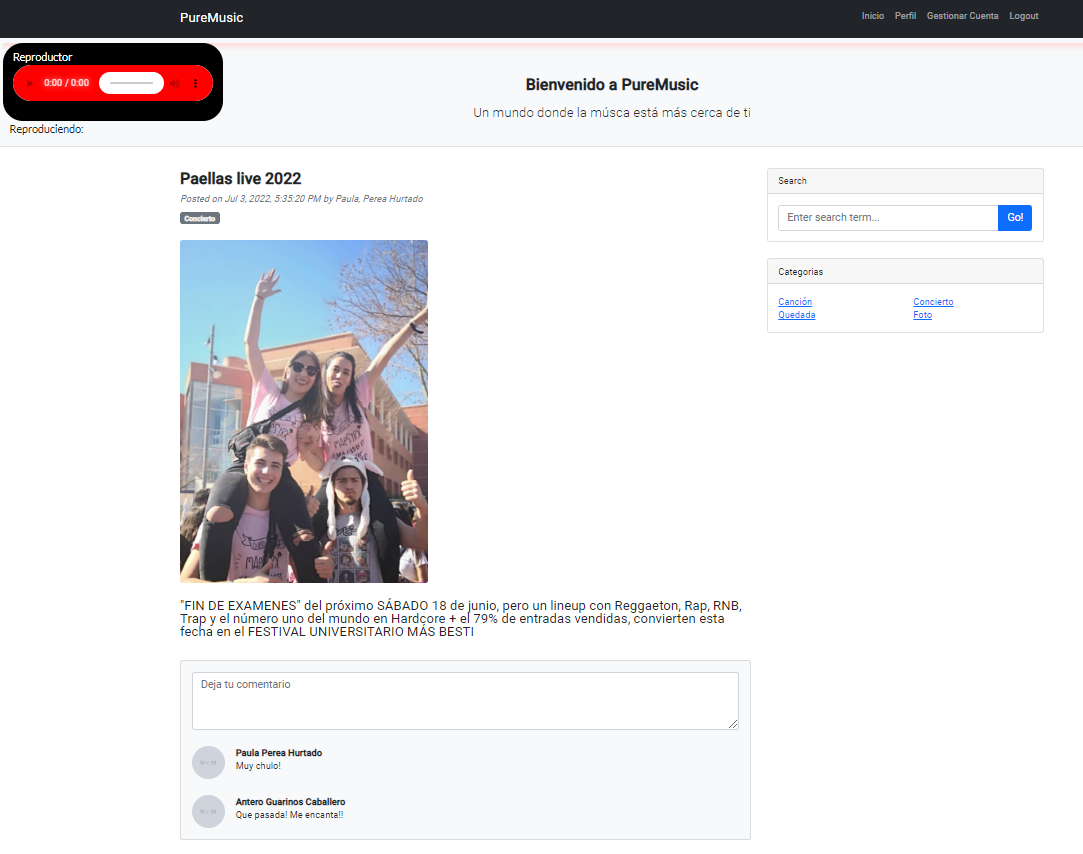


Figura 28: Interfaz final de ver un post

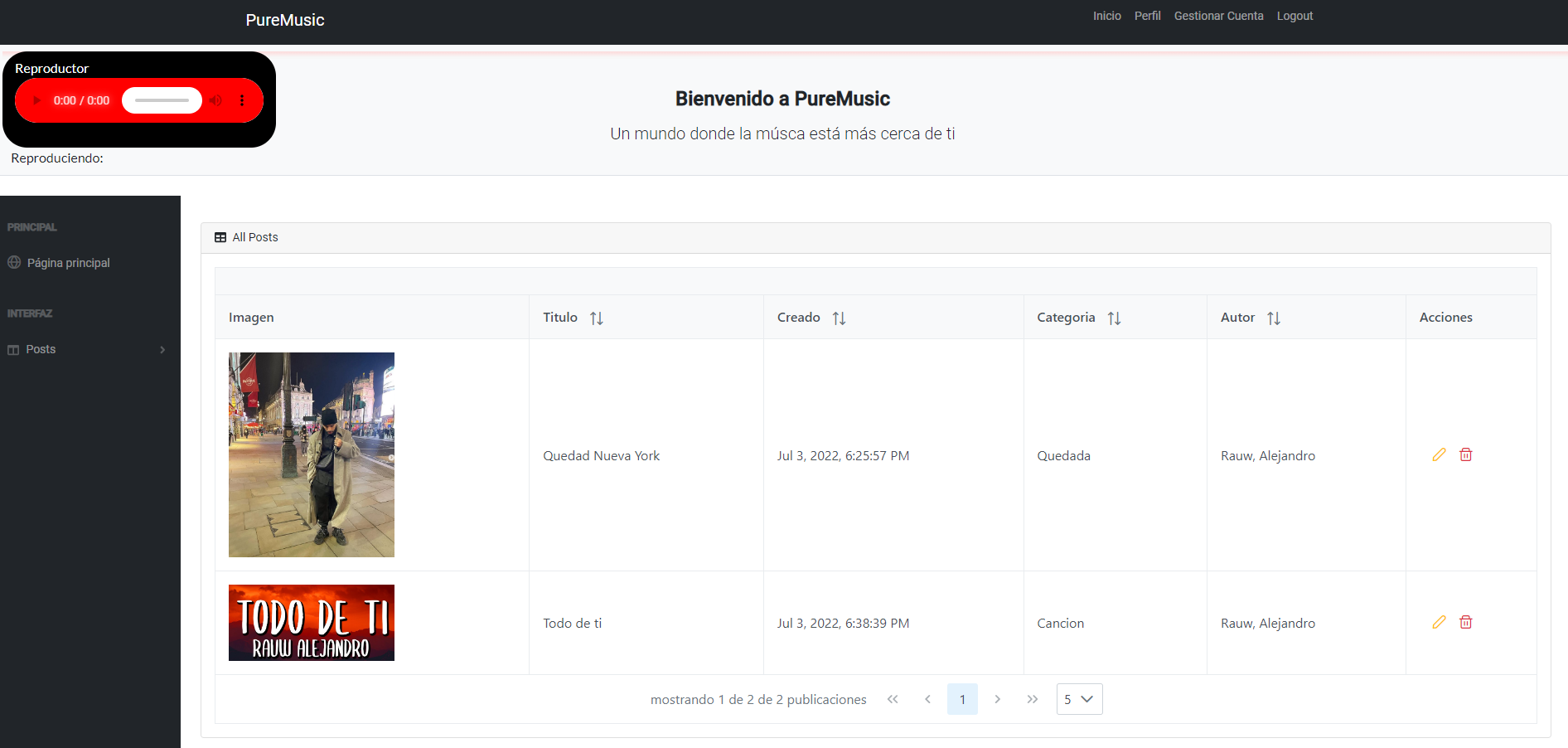


Figura 29: Interfaz final de gestionar perfil

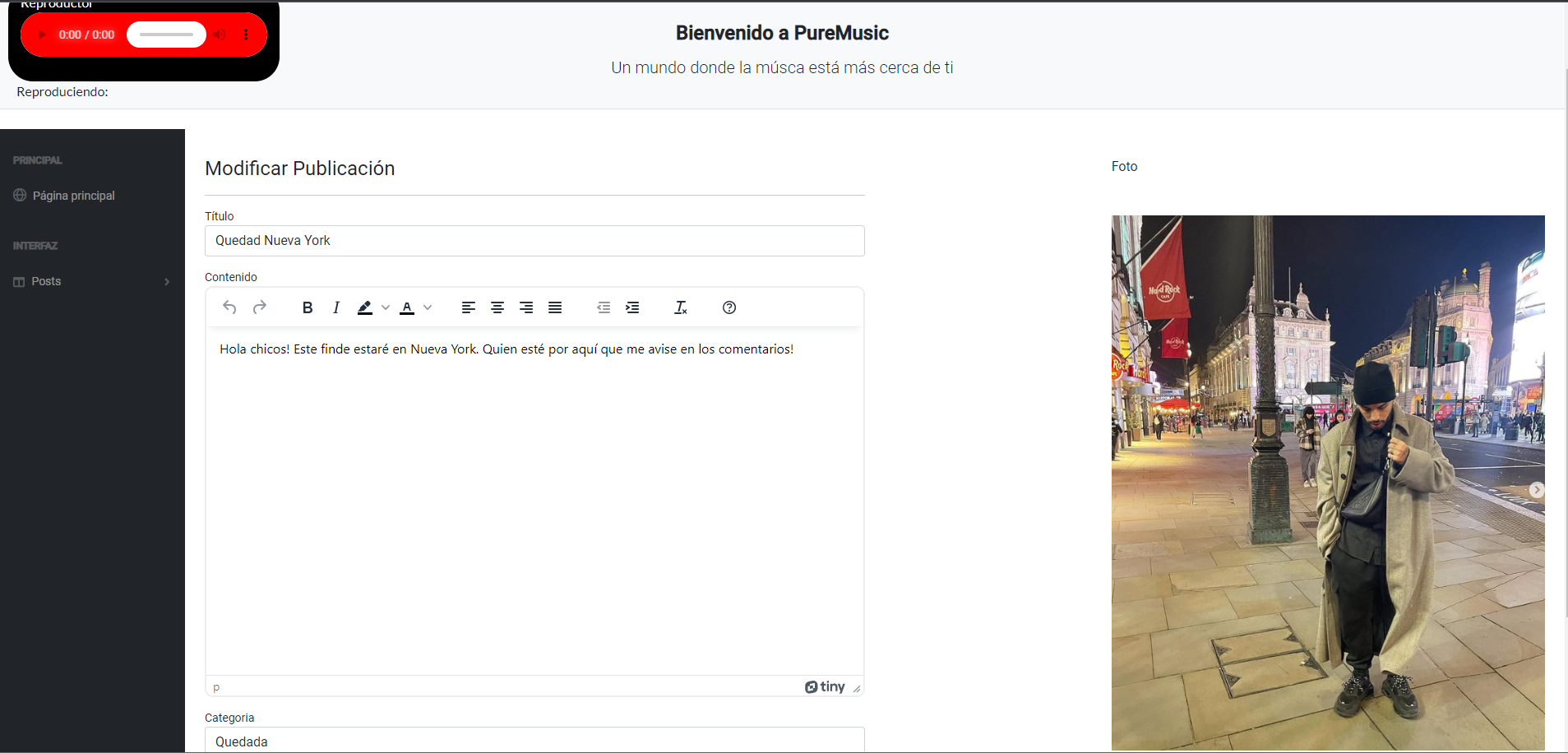


Figura 30: Interfaz final de modificar post

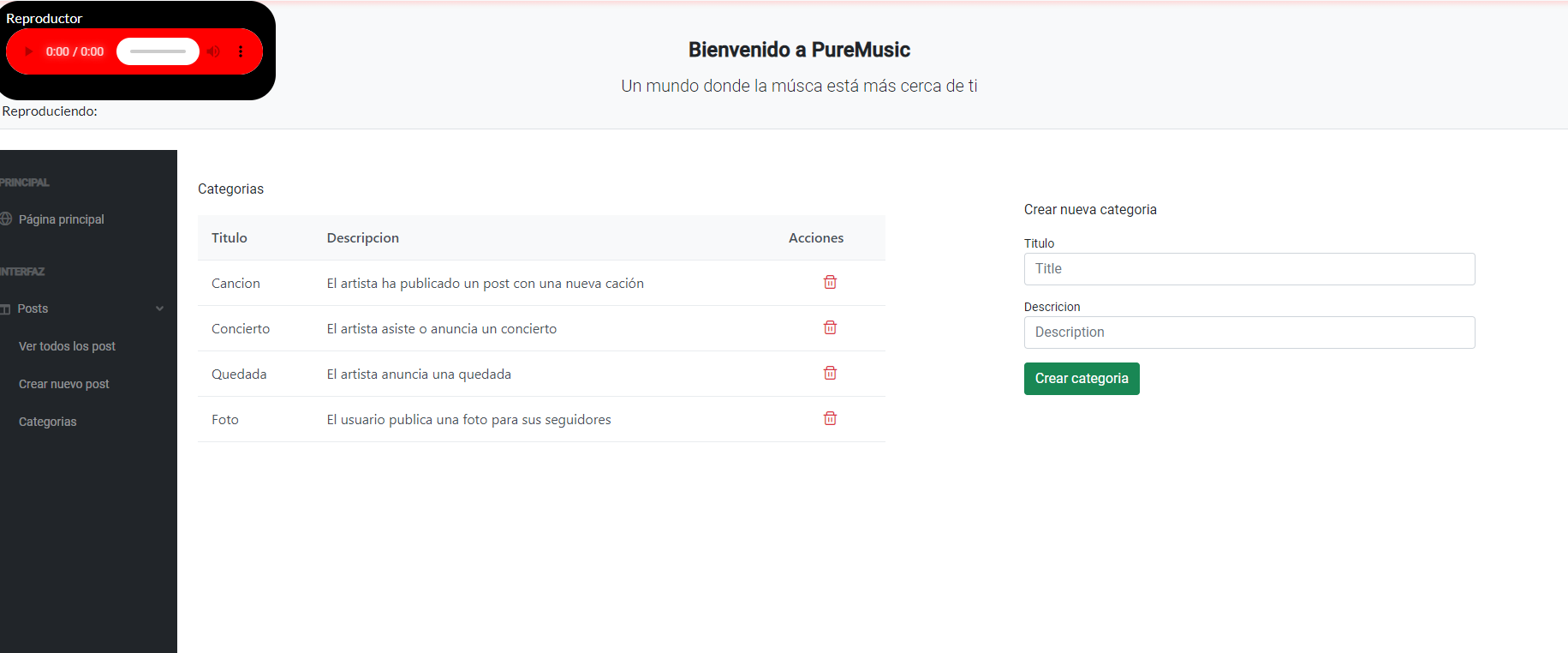


Figura 31: Interfaz final de ver categorías

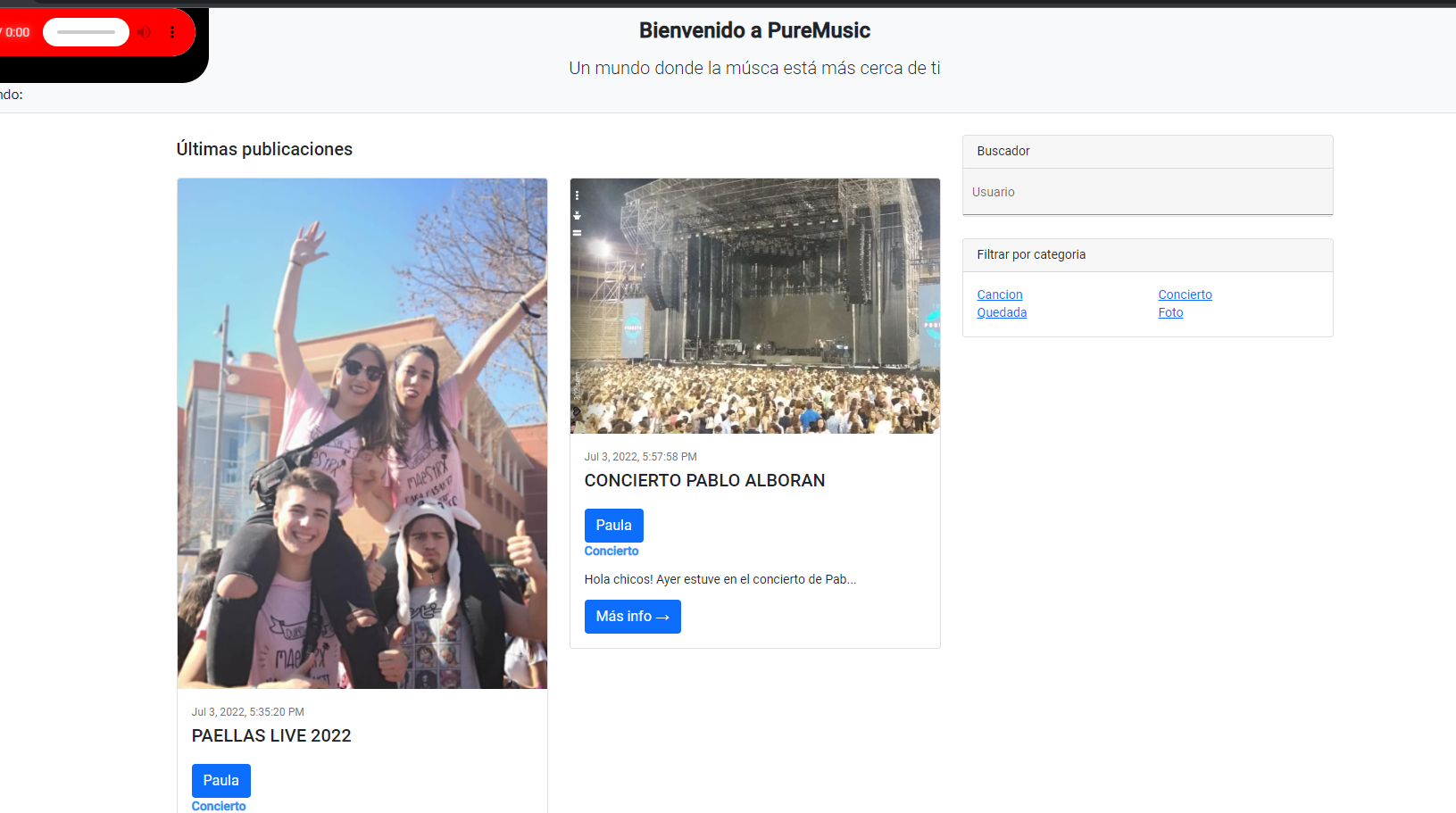


Figura 32: Interfaz final de página de inicio

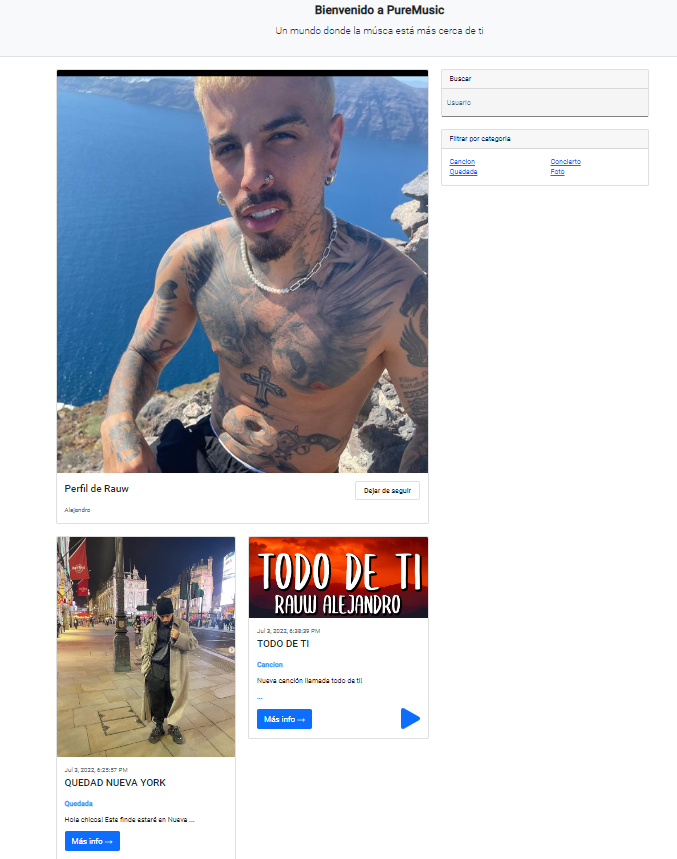


Figura 33: Interfaz final de ver perfil



Figura 34: Interfaz final de login