UNIVERSIDAD DE MÁLAGA

E.T.S. INGENIERÍA INFORMÁTICA
UNIVERSIDAD DE MÁLAGA

GRADO EN INGENIERÍA INFORMÁTICA

DESARROLLO DE UNA APLICACIÓN WEB PARA LA BÚSQUEDA DE MÚSICOS Y BANDAS

DEVELOPMENT ABOUT A WEB APPLICATION TO SEARCH MUSICIANS AND BANDS

Realizado por
ADRIÁN FERNÁNDEZ CLAVERÍAS

Tutorizado por
EDUARDO GUZMÁN DE LOS RISCOS

Departamento
LENGUAJES Y CIENCIAS DE LA COMPUTACIÓN
UNIVERSIDAD DE MÁLAGA

MÁLAGA, Junio 2020

Resumen:

En la actualidad si un músico está buscando una banda, siempre tiene que terminar recurriendo a anuncios en páginas no especializadas para este sector (como Facebook, Milanuncios...), anuncios en la vía pública o que el contacto se establezca a través de conocidos. Igualmente, a la inversa, cuando una banda necesita algún componente para completar el grupo, es difícil localizarlo. Estas razones ponen de manifiesto, lo tedioso que puede llegar a ser poner en contacto a músicos y a bandas entre sí.

Por eso surge la necesidad de crear esta aplicación web, una plataforma donde se puede buscar a un músico que toque un instrumento específico para completar la banda o a la inversa, que el músico que toca ese instrumento encuentre una banda donde pueda desempeñar su función. Por ejemplo, que una banda pueda encontrar a una persona que toque la guitarra, o viceversa, que un guitarrista pueda encontrar a una banda.

De esta forma una persona tendrá un lugar totalmente especializado para la búsqueda de una banda, y una banda que esté buscando a un músico para completar su grupo podrá utilizar esta aplicación para satisfacer su necesidad. Gracias a un chat de mensajes privados entre usuarios se podrá establecer un contacto de manera directa y sencilla entre las partes interesadas.

Palabras claves:

Angular, NodeJS, MongoDB, MySQL, red social para músicos, JWT.

Abstract:

Nowadays, if a musician is searching for a band, he always has to public an advertisement on web pages not specialized for this section (as Facebook, Milanuncios...), advertisement on the street or contact through acquaintances. Equally, the other way, when a band needs a musician to complete the band, it is difficult to find it. These reasons manifest, the difficulty of putting musicians and bands in contact.

For this reason, it is necessary to create this web application, a platform where you can search a musician who plays a specific instrument to complete the band or vice versa, the musician who plays that instrument to find a band where they can perform their function. For example, a band can find a person who plays the guitar or vice versa, a guitarist can find a band.

In this way, a person will have a totally specialized site to search for a band, and a band that is looking for a musician to complete his band can use this application to satisfy his need. Thanks to private messages chat between users, it will be possible to establish direct and simple contact between interested parties.

Keywords:

Angular, NodeJS, MongoDB, MySQL, social network for musicians, JWT.

Índice de contenidos

CAPÍTULO 1

Introducción

1.1. Motivación

A lo largo de la historia cuando un músico se ha encontrado con otro músico, han intentado armonizar sus sonidos conjuntamente. Esto ha hecho que los músicos se uniesen para formar una banda. Las bandas han permitido la composición de canciones, actuaciones y poner la banda sonora de una juventud.

Pero formar una banda es una labor tediosa ya que todos los componentes deben de estar de acuerdo en el estilo de música, ensayos, conciertos... es como llevar una segunda familia, por eso hace que mantener el grupo pueda ser complicado, lo que hace que un músico pueda rotar de un grupo a otro como si de un fichaje se tratara.

Para formar una banda lo primero que se debe hacer es encontrar a los músicos necesarios y que todos quieran hacer un estilo de música común, ya que por ejemplo una banda con un estilo de música flamenco es raro que necesite a un músico que toque la guitarra eléctrica. Por ello el trabajo que tiene un músico para formar una banda es compleja y desde siempre este asunto ha sido cuestión de suerte o simplemente tener buenos contactos.

Al conocer este problema cíclico entre músicos y bandas a lo largo de los años, ha hecho brillar la idea de desarrollar una aplicación que ponga fin a este problema o al menos pueda paliar un poco esa labor tan tediosa.

Hoy día cualquier acción o trámite gira en torno a Internet, por lo que sería interesante poder encontrar una aplicación que solvente este tema. Gracias a las nuevas tecnologías se podría desarrollar una aplicación donde todos los músicos y bandas de España pudiesen encontrarse y conocerse entre sí, en la que además puedan contactar entre ellos a modo de red social.

Recoger en una sola aplicación a todos los músicos y bandas de España, para que se conozcan entre sí, y puedan desarrollar sus actividades como músicos es algo que motiva comenzar a diseñarla.

1.2. Músicos y bandas

La aplicación que se va ha desarrollar (que de ahora en adelante será llamada JoinBand) será una aplicación que permita a los músicos y bandas de España conocerse entre sí, reduciendo el tiempo de búsqueda de los músicos y bandas, con un buscador exhaustivo y así puedan formar una banda fácilmente. Como se ha podido intuir hasta el momento, existirán dos tipos de usuarios/roles, los cuales se describirán a continuación con más detalle:

- Por una lado tenemos a los **músicos**. El problema principal que tiene un músico a la hora de encontrar una banda es lo complicado que es saber o conocer de una banda que necesite a una persona para completarla, y además, que toque él como músico, el instrumento que la banda demanda. Esto llega a ser un proceso demasiado lento, y si por algún caso la persona se muda de una ciudad a otra, la cual no conoce realmente bien, le será aún más complicado poder encontrar a una banda.
- Por otro lado tenemos a las **bandas**. El problema es similar al de un músico, pero está vez de manera contraria. Por ejemplo, una banda a la que le falta un cantante, ¿cómo puede encontrarlo? Para ello JoinBand permitirá buscar a un cantante en la provincia que seleccione.

Aunque si bien una banda está formada por varios músicos, dentro de la aplicación no se contemplará esto, puesto que la idea es que un usuario con el rol de músico pueda encontrar a un usuario con el rol banda o por lo contrario, que un usuario con el rol de banda pueda encontrar a ese usuario con el rol de músico que le falta. No tendría ningún sentido que un usuario con el rol de músico se registrase en la aplicación para informar que pertenece a una banda. Esto es un detalle importante que se ha de tener en cuenta a la hora de diseñar la base de datos, y la relación que existirá entre ellos.

1.3. Objetivos

A grosso modo, el objetivo principal de este proyecto es desarrollar una aplicación web junto con una API, la cual proveerá todo lo necesario para buscar a un músico o una banda en cada una de las ciudades españolas. La API debe ser capaz de servir a más de un cliente, por lo que se ha elegido una API RESTful.

La API contiene datos sensibles, como información personal de cada uno de los usuarios. Es por eso, que la seguridad y protección son cruciales. Al tener dos tipos de roles dentro de los usuarios (músico y banda), los recursos serán protegidos para que cada uno pueda utilizar lo que le corresponde con su rol de usuario.

Un objetivo importante es dar la posibilidad de poder contactar con cada usuario que esté registrado en la aplicación. Es por eso, que la aplicación debe proveer de un chat de mensajes privados.

Cada cliente de la aplicación debe poder crear una cuenta de usuario con el rol que desee. Debe ser capaz modificar algunos datos personales, usar el buscador de la aplicación e incluso eliminar su cuenta.

Por tanto, los objetivos más destacados para el desarrollo completo de la aplicación son:

- Desarrollar una API capaz de ofrecer una arquitectura orientada a servicios, para que se pueda resolver el problema de dificultad que tienen los músicos y bandas a la hora de encontrarse entre sí.
- Operaciones CRUD[1] para los dos roles de usuario que conforman la aplicación.
- Investigar y utilizar una forma de proteger la API.
- Investigar y utilizar una forma de integrar datos personales y los mensajes privados de cada usuario.
- Explorar una forma de realizar un chat de mensajes privados.
- Estudiar e implementar la manera de almacenar las imágenes de perfil de un usuario en la nube, de tal forma que se obtenga la dirección url para almacenarla en la base de datos.
- Desarrollar una aplicación web que se sirva de la API.

1.4. Estado del arte

En este apartado se verán algunas plataformas y herramientas que pueden tener algo de similitud con JoinBand. Es cierto que existen ciertas herramientas que pueden llegar a solventar un poco este problema, pero algunas de ellas tienen una gran problema, y es que no están especializadas en este tema, entonces hace que no tengan unas funcionalidades bien definidas y orientadas a los músicos y bandas.

- **Páginas de Facebook:** Debido a que no hay ninguna buena aplicación que solvente este problema, en Facebook se han creado a lo largo de los años páginas en las que se inscriben músicos y bandas. A través de comentarios se ofrecen para poder formar una banda, pero, ¿y si quieres que se muestren solo los músicos de tu ciudad? Tendrías que inscribirte en otro grupo de Facebook de tu ciudad, si tienes la suerte de que exista. Este es el principal problema de este tipo de soluciones, al no estar especializada en este problema, no puede responder de forma correcta las peticiones de un músico o una banda.
- **Milanuncios:** La mayoría de las personas ya conocen Milanuncios. Este es el portal más famoso para publicar un anuncio a nivel nacional, ya sea para vender algún artículo, comprarlo, buscar un servicio u ofertarlo. Por ello, Milanuncios se puede usar para publicar anuncios y ofrecerse como músico que busca una banda. El problema que ocurre en esta plataforma es similar a lo que ocurre en Facebook, al no ser una herramienta totalmente enfocada a la búsqueda de músicos y bandas, no tiene facilidad de uso, ni está orientada a los músicos. Es complicado que alguien vaya a la web de Milanuncios a buscar al músico que le falta para completar su banda.
- **BandMix:** Esta es una herramienta similar a Joinband que lleva funcionando varios años, pero es de pago. Se puede usar el buscador de la aplicación para encontrar a un músico, o incluso a una banda, pero el problema viene a la hora de poder contactar

con esa persona, ya que exigen pagar una cuota mensual para poder usar el chat de la aplicación y poder contactar con una persona. Por tanto este motivo hace muy poco funcional esta aplicación, ya que la mayoría de la gente no está dispuesta a pagar por utilizar un chat.

Aunque pueda haber algunas herramientas que realizan operaciones y funcionalidades parecidas, este proyecto intenta mejorar todas las funcionalidades que sean parte de estas herramientas y dar una mejor experiencia de usuario apoyándose en una interfaz de usuario sencilla pero efeciente al mismo tiempo.

1.5. Estructura de la memoria

Esta memoria se ha estructurado en un total de 5 capítulos que recogerán toda la información necesaria para la realización de este trabajo fin de grado.

- En primer lugar existe un capítulo de introducción. En este capítulo se redacta la motivación y puesta en situación de en qué consiste este proyecto. También se hace una descripción sobre el problema que trata de resolver esta aplicación para los músicos y bandas. Un punto muy importante en este capítulo son los objetivos, en el que se redactan cada uno los objetivos principales que debe cumplir JoinBand. Por último, en este capítulo se hace una revisión sobre el estado del arte y la herramientas existentes, similares a JoinBand.
- El segundo capítulo especifica los requisitos del proyecto. En este capítulo se hace constancia de todas las actividades que un usuario podrá realizar en la aplicación, y al tratarse de una arquitectura orientada a servicios se ha decidido dividir los requisitos en dos partes. Por un lado, los requisitos que debe cumplir la API, y por otro lado, se encuentran los requisitos que se deben contemplar en el desarrollo de la aplicación web.
- El tercer capítulo de esta memoria se centra en el diseño e implementación del proyecto. Por ello, en este capítulo aparece la arquitectura que sigue este proyecto, y se detalla cada uno de los componentes que lo conforman. En este capítulo también se especifican las tecnologías usadas en el proyecto, y una comparación con posibles tecnologías que se pensaron usar. En último lugar se dejan constatados los diseños de la API y de la aplicación web.
- La planificación que se ha llevado a lo largo del desarrollo del proyecto se incluye en el capítulo 4. En este se especifican las metodologías de trabajo elegidas y cómo se han adaptado estas en relación a la forma y características del proyecto.
- En el último capítulo de la memoria se encuentran las conclusiones finales del proyecto, se hace una visión global sobre el desarrollo y se detallan los impedimentos que se encontraron durante la fase de desarrollo. A modo de conclusión también se dejan anotadas algunas ampliaciones futuras que podría tener este proyecto, ya que una de las características de JoinBand es que puede ser muy escalable.

CAPÍTULO 2

Requisitos

Al tener que confrontar una arquitectura orientada a servicios, en la que se puede diferenciar dos grandes aplicaciones (como se especificará en el capítulo 3), los requisitos de la aplicación se van a ver diferenciados en dos grupos. Por un lado, los requisitos de la API, y por otro lado, los requisitos de la aplicación web (front-end).

Pero antes de entrar en detalle de cada uno de los requisitos, tanto de la API como de la aplicación web, a continuación se mostrarán las actividades que puede realizar un usuario, ya sea con el rol de banda o con el de músico.

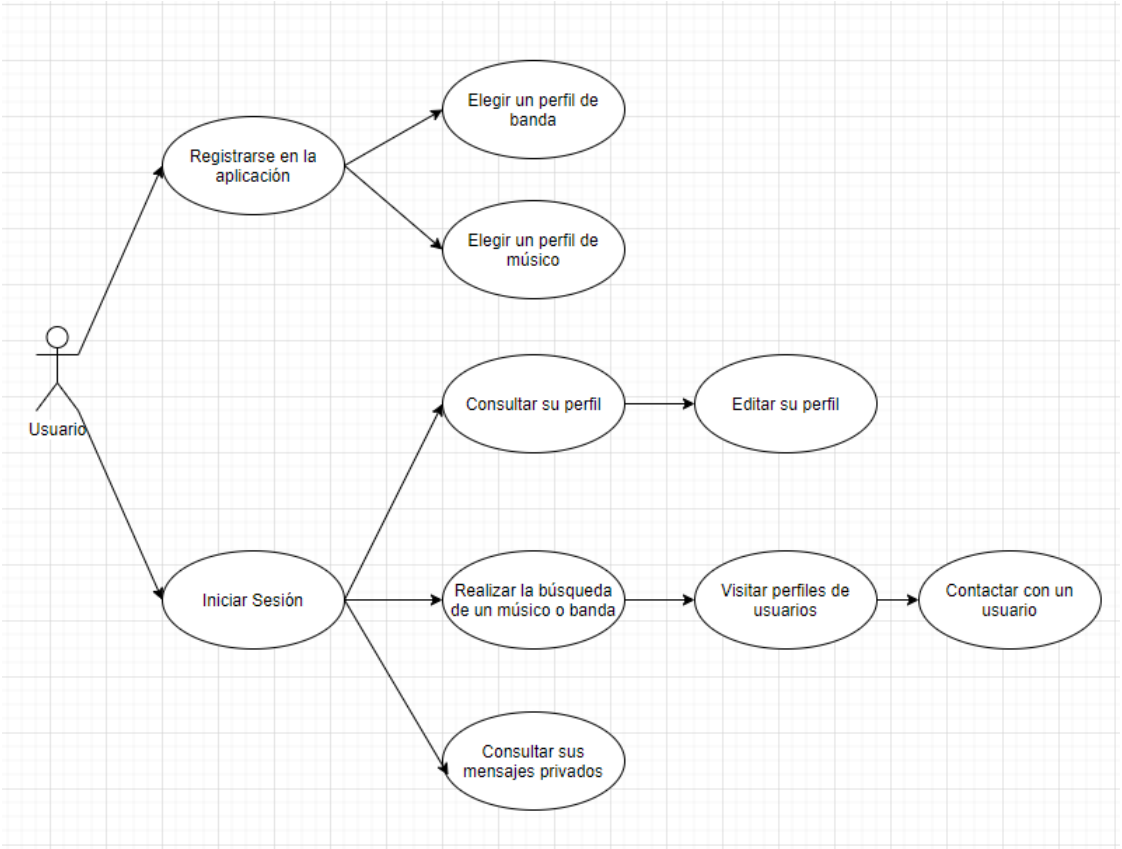

Figura 1. Casos de uso posibles para un usuario.

Como se muestra en la Figura 1, cuando un usuario entra en la aplicación puede realizar distintas acciones, que se pueden dividir en diferentes casos de uso:

Caso de uso: Registro

Un usuario siempre tendrá la opción de poder registrarse en la aplicación. A la hora de intentar realizar esta acción tendrá que elegir en este momento el tipo de registro que quiere, es decir, registrarse con el rol de músico o de banda.

Caso de uso: Inicio de Sesión

El usuario tiene otra actividad importante, se trata de iniciar sesión en la aplicación una vez que se ha registrado. A partir de aquí el usuario podrá hacer uso de todas las funcionalidades restantes de la aplicación, ya que si no se tiene una sesión iniciada no se podrá usar la aplicación, ni hacer uso de ninguna de las funcionalidades.

Caso de uso: Consultar perfil

Cuando inicia sesión, el usuario puede consultar su propio perfil, donde se encontrará toda su información y su foto de perfil. Además tiene la opción de modificar algunos de sus datos principales y actualizar la foto de perfil.

Caso de uso: Realizar búsqueda

Si el usuario tiene una sesión iniciada en la aplicación, podrá realizar la búsqueda de un músico o una banda en una provincia en concreto. A partir de este punto podrá visitar los perfiles de los usuarios que se han encontrado. Cuando se está visitando el perfil de un usuario, por ejemplo el de una banda, siempre se dará la opción de poder ponerse en contacto.

Caso de uso: Consultar mensajes

Como última actividad, un usuario siempre podrá consultar sus mensajes privados, es decir, se dará la opción de poder consultar sus mensajes en un chat y también poder responder a los mensajes que ha recibido dicho usuario.

Una vez vistas las principales actividades que puede realizar un usuario dentro de la aplicación, en los siguientes apartados se detallarán más específicamente los requisitos para la API y para la aplicación web.

2.1. Requisitos de la API y microservicio chat

La API es la encargada de contener toda la lógica de negocio de la aplicación, y es la encargada de interactúar con cada una de las bases de datos y los servicios externos, así como también proveer toda esta información a cada uno de los usuarios clientes.

Aunque la aplicación tiene dos roles de usuarios diferenciados, las funcionalidades de ambos son prácticamente iguales. Por lo tanto, las **funcionalidades iniciales** que todo usuario de la aplicación debe poder hacer son **unas operaciones CRUD, usar el buscador y usar el chat de mensajes privados**.

2.1.1. Usuarios

Bandas

- Una banda debe ser capaz de registrarse en la aplicación.
- Una vez que se ha registrado, una banda debe poder iniciar sesión en el sistema.
- Debe poder consultar su propio perfil de usuario.
- Editar su perfil, así como la contraseña, imagen de perfil o los años que llevan tocando juntos como banda.
- Debe poder listar a un usuario en concreto (esta opción servirá para que se pueda visitar el perfil de un usuario).
- Eliminar su cuenta de la aplicación.

Músicos

- Podrá registrarse en la aplicación con el rol de músico.
- Un usuario con el rol de músico podrá iniciar sesión una vez ha registrado en la aplicación.
- Debe poder consultar su propio perfil de usuario.
- Editar su propio perfil, así como la contraseña, imagen de perfil, el nombre de usuario...
- Debe poder listar a un usuario en concreto (esta opción servirá para que se pueda visitar el perfil de un usuario).
- Eliminar su cuenta de la aplicación.

Una vez que estos requisitos se cumplen, hay unas funcionalidades concretas que todo usuario de la aplicación debe poder realizar, y que por tanto la API debe implementarlos como lógica funcional.

Un aspecto a destacar es la actualización de la imagen de perfil. A la hora de almacenarla, la API recogerá esa nueva imagen y la almacenará en un cloud externo, pero debe ser capaz de obtener y almacenar la URL de la imagen en la base de datos.

2.1.2. Buscador de músicos y bandas

El requisito clave por excelencia que hace que la aplicación cobre sentido es el buscador de músicos y bandas.

En este modulo de la API se debe desarrollar una forma de realizar un buscador que permita encontrar a un músico o una banda (según se especifique en el buscador) en una provincia española. La API debe ser capaz de responder con una lista de los usuarios encontrados. Para ello existirán una serie de campos que debe gestionar la API para poder hacer funcionar el buscador:

- **"Qué está buscando"**. La API debe ser capaz de recoger este campo, en el que se especifica qué se está buscando, es decir, el tipo de músico (guitarrista, baterista, cantante...) o una banda.
- **"Dónde lo está buscando"**. Es el campo en el que se recogerá la información sobre la provincia en la que se quiere realizar la búsqueda, es decir, por ejemplo 'Málaga'.

2.1.3. Chat de mensajes privados

Otro requisito importante es desarrollar un microservicio que sea capaz de cubrir el chat de mensajes privados entre usuarios de la aplicación.

- El microservicio debe ser capaz de interconectar a los usuarios de la aplicación para que se pueda usar un chat de mensajes privados.
- Se deben poder listar los mensajes del chat, así como las distintas conversaciones que se tienen activas.

2.2. Requisitos de la aplicación web

La aplicación web es la encargada de mostrar los datos que la API le provee y pone en práctica toda la funcionalidad de la aplicación. Por tanto esta aplicación debe cubrir unos requisitos clave, que se van a estructurar en diferentes secciones en este apartado.

2.2.1. Usuarios y perfiles

- La aplicación web tendrá una primera página a modo de *landing page* con información general y detalles sobre cómo funciona la aplicación.
- La aplicación debe suministrar un formulario de registro en el que se pueda seleccionar fácilmente el tipo de usuario que eres (músico o banda), esto hará cargar de forma reactiva el formulario de registro para el tipo de rol seleccionado.
- Un usuario debe poder iniciar sesión con su cuenta, a partir de su email y contraseña.
- Todas las rutas funcionales de la aplicación deberán estar desactivadas si el usuario no tiene una sesión iniciada, excepto el registro de usuarios.
- Cada usuario tendrá un apartado dentro de su perfil para poder realizar modificaciones o actualizaciones, como por ejemplo, cambiar la contraseña o cambiar la imagen de perfil.

- Las vistas de los perfiles deben ser dinámicas, es decir, la información que se muestra para un perfil con el rol de músico no es la misma información que se debe mostrar para el rol de banda.
- Cuando se visita el perfil de un usuario debe aparecer una opción que indique 'Contactar', la cual conducirá al chat privado para que puedan conversar entre ellos.
- Un usuario siempre tendrá en su perfil una opción que le permita eliminar su cuenta.
- Cada usuario de la aplicación tendrá su propio perfil, que podrá ser visitado por cualquier usuario de la aplicación a través de la 'tarjeta' de usuario que aparecen en los resultados de búsqueda.

2.2.2. Buscador y resultados

- Una vez que un usuario ha iniciado sesión podrá usar el buscador de músicos/bandas, que constará de dos selectores. Uno para elegir 'qué busca', y otro selector para indicar la provincia donde quiere realizar la búsqueda.
- Si alguien entra en la aplicación e intenta usar el buscador de músicos/bandas, este debe ser bloqueado. Es decir, para que una persona pueda usar el buscador de músicos/bandas, previamente tiene que tener una sesión iniciada en la aplicación o si no dispone de una cuenta deberá de registrarse primero.
- El buscador de músicos/bandas deberá ofrecer una página de resultados de búsqueda, donde a modo de 'tarjetas' aparecerán cada uno de los usuarios encontrados que satisfacen la condición del buscador.
- Desde la página de resultado de la búsqueda, siempre se debe dar la opción de que el usuario pueda realizar una nueva búsqueda.

2.2.3. Chat de mensajes privados

La aplicación web debe ser capaz de conectarse con el microservicio del chat. Una vez que tiene una conexión establecida deberá tener las siguientes características:

- Se debe poder listas todas las conversiones que se tienen activas hasta el momento.
- Se podrá intercambiar todos los mensajes que los usuarios estén ejecutando en tiempo real.
- El usuario tendrá en todo momento la opción de consultar sus mensajes.

CAPÍTULO 3

Diseño e implementación

3.1. Arquitectura

JoinBand sigue una arquitectura orientada a servicios. Esto permite que al hacer algún cambio en la API los clientes tengan la última versión. A continuación se muestra una imagen de la arquitectura completa del proyecto.

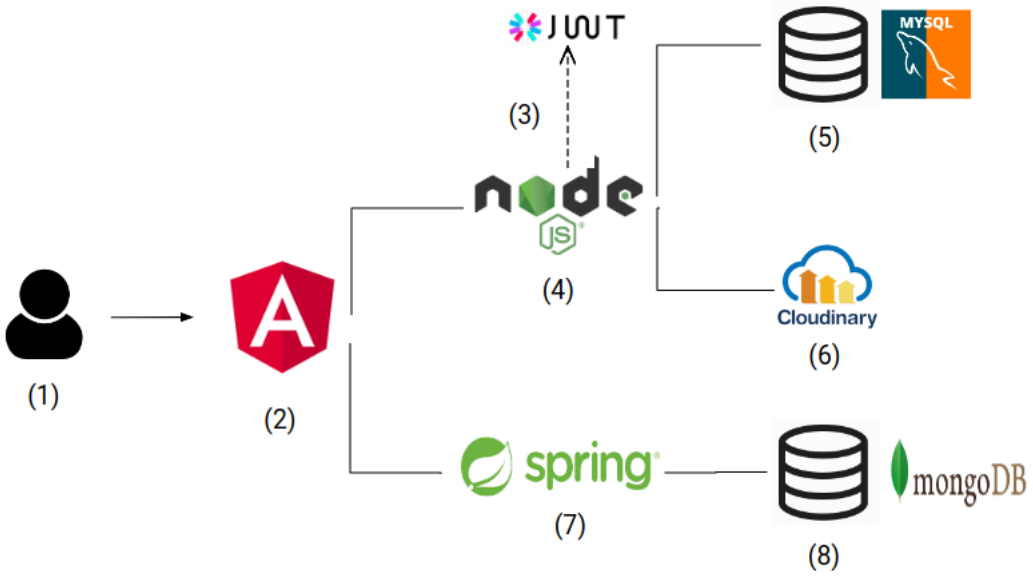

Figura 2. Arquitectura de la aplicación web

Como se muestra en la Figura 2, la aplicación se puede dividir en tres partes. Por un lado hay una aplicación front-end implementada en Angular, funcionando como cliente que se conecta tanto a la API en NodeJS, como al microservicio en SpringBoot que realiza la función del chat. La API en NodeJS se conecta con la base de datos MySQL y un servicio externo. Mientras que por otro lado, el microservicio de SpringBoot se conecta con la base de datos Mongo que almacenará los mensajes del chat.

El flujo de funcionamiento de la arquitectura sería el siguiente. Si un cliente entra en la aplicación (punto número 1 de la Figura 2), se encontrará con una aplicación web desarrollada en Angular. Una vez que se autentique el usuario en la aplicación, recibirá un token (generado a través de JSON Web Token, punto 3 de la Figura 2). En este punto el usuario estará listo para recibir toda la información que le sea necesaria desde la API (en NodeJS, punto número 4 de la Figura 2). La API conecta con una base de datos MySQL[2] (punto 5 de la Figura 2) para almacenar y obtener toda la información principal de un usuario. La API de JoinBand conecta con Cloudinary[3], (punto 6 de la Figura 2) una API que permitirá almacenar y descargar las imágenes de perfil de cada usuario de la aplicación. Cuando un usuario decida contactar con otro usuario y use el chat, entrará en juego la parte inferior de la figura, dónde se encuentra el microservicio de SpringBoot (punto 7 de la Figura 2) que conecta con la base de datos con MongoDB[4], la cual contendrá todo lo referente a los mensajes entre usuarios (punto 8 de la Figura 2).

3.1.1. Bases de datos

Una vez bien definidos los requisitos que debe cumplir el proyecto, es más fácil comenzar a diseñar la base de datos. En cuanto a las bases de datos que conforman JoinBand, son un total de dos bases de datos, que se decribirán en esta sección. Puesto que la aplicación web no tiene ninguna manera de gestionar las bases de datos, quien se encargará de gestionarlas será la API.

La razón por la que se ha elegido esta estructura y tener dos bases de datos es para poder tener lo mejor de ambos tipos de bases de datos, SQL y NoSQL.

Por un lado, existe la **base de datos en MySQL (SQL)** , que almacenará toda la información personal de los usuarios de la aplicación. Entre sus principales ventajas se encuentran:

- Madurez: Las bases de datos SQL tiene ya muchos años de madurez y aceptación por los desarrolladores por lo que existe un gran variedad de información para el desarrollo de la base de datos y extración de información, por lo que hace reducir el tiempo de desarrollo cuando se realiza un proyecto software.
- Atomicidad: La atomicidad es una característica clave de las bases de datos SQL, la cual permite que en cualquier operación realizada se garantizará que si cuando se realizar una operación y esta tiene algún tipo de problema, la información no se completará, es decir, o la operación se realiza por completo o no se realiza nada.

Por otro lado la **base de datos en MongoDB (NoSQL)**, que almacenará todo lo referente a los mensajes privados de la aplicación, dará grandes ventajas como pueden ser:

- Versatilidad: Dado que se basa en una notación ligera como es la notación JSON, permite una gran versatilidad a la hora de agregar, si fuera necesario, un nuevo campo sobre la 'colección'.
- Recursos y optimización: Con una base de datos NoSQL no se requieren servidores que tengan una gran cantidad de recursos para operar. NoSQL da una gran optimización gracias a su algoritmo interno para reescribir las consultas escritas por los usuarios con el fin de no sobrecargar el rendimiento de los servidores.

MySQL

El motivo por el que se ha elegido MySQL, y no otro sistema gestor de bases de datos como Oracle, ha sido por la gran aceptación que tiene MySQL dentro de la comunidad. Lleva muchos años funcionando y además, MySQL tiene licencia gratuita, lo cual se hace ideal para la realización de este trabajo fin de grado.

En esta base de datos se almacenarán todos los datos personales referentes a todos los usuarios de la aplicación, es decir, de los músicos y bandas. A continuación se mostrarán el modelo de la base de datos.

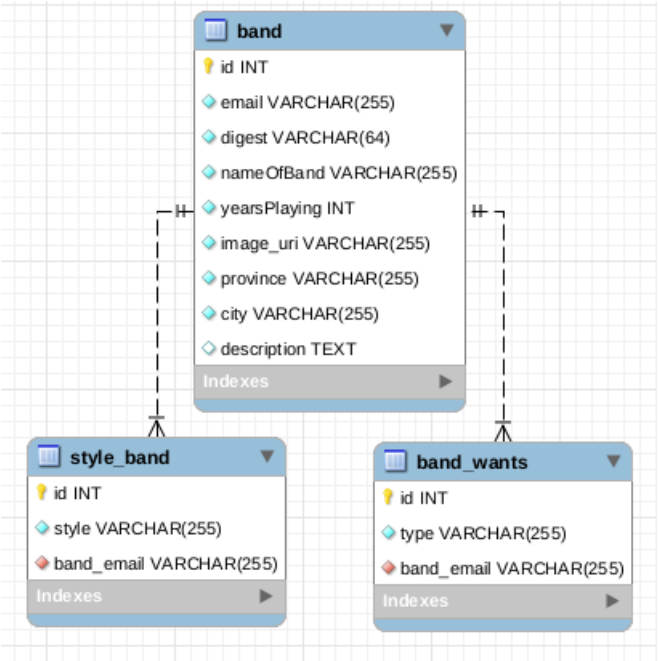

Figura 3.1: Base de datos MySQL (Parte de las *bandas*)

Como se pude ver en la Figura 2, existen tres tablas que hacen referencia a un usuario con el rol de banda. En primer lugar hay una tabla llamada *band*, la cual incluye los datos principales de una banda como: email, nombre de la banda, contraseña, años tocando juntos...

En la parte izquierda de la figura se puede ver la tabla *styles band*, esta tabla es una entidad débil de la tabla *band*, que tiene como clave foránea el email con el que la banda se ha registrado en la aplicación. Esta tabla se utiliza para almacenar en el atributo *type* el estilo de música que hace la banda, como por ejemplo puede ser 'Rock', 'Flamenco', 'Pop'... Esta tabla es necesaria debido a que una banda puede tener varios estilos de música, lo cual es una lista de estilos de música asociada a una banda.

La tabla *band wants*, que aparece en la parte derecha de la figura, al igual que la tabla *styles band*, es una entidad débil de la tabla *band* que tiene como foreign key el email de la banda. En esta tabla se guarda en el atributo 'type' el tipo de músico que está buscando la banda, es decir, por ejemplo este atributo puede ser 'Guitarra', 'Batería', 'Piano'... Esta tabla es necesaria ya que una banda puede estar buscando a varios músicos al mismo tiempo.

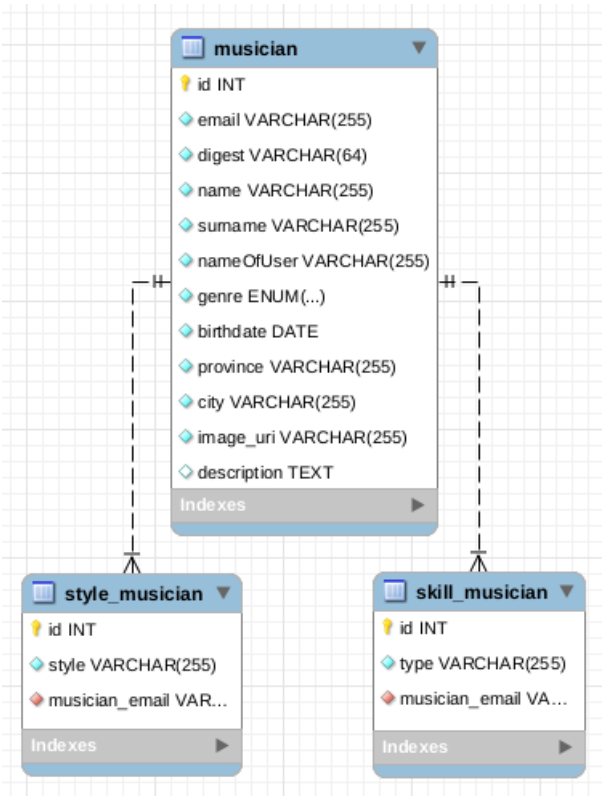

Figura 3.2: Base de datos MySQL (Parte de los *músicos*)

Como se puede ver en la Figura 3.2, existen tres tablas que son destinadas exclusivamente para un usuario con el rol de músico. En primer lugar hay una tabla llamada *musician*. En esta tabla se utiliza para almacenar aquellos datos personales y de principal importancia en un músico, como pueden ser: el email, nombre, apellidos, fecha de nacimiento...

Como se ve en la figura, en la parte izquierda, la tabla *styles musician* es una entidad débil de la tabla *musician*, que tiene como clave foránea el email con el que el músico se ha registrado en la aplicación. Esta tabla es útil para almacenar en el atributo *style* el estilo de música que esta relacionado con el músico, como por ejemplo puede ser 'Rock', 'Heavy Metal', 'Pop'... Esta tabla será necesaria debido a que un músico puede tener varios estilos de música a modo de lista.

La tabla *skills musician*, que aparece en la parte derecha de la figura, al igual que la tabla *styles musician*, es una entidad débil de la tabla *musician* que tiene como *foreign key* el email del músico. En esta tabla se guarda en el atributo *type* el tipo de hablididades que tiene el músico que esta relacionado con esta tabla, es decir, por ejemplo puede ser 'Guitarra', 'Batería', 'Piano'... Esta tabla es necesaria ya que un músico puede tener varias de las habilidades que se contemplan.

Se observa en las figuras sobre la base de datos MySQL, que no existe una relación directa entre la tabla *musician* y *band*. Este es el motivo por lo que anteriormente, en el capítulo 1.2 de esta memoria, se objetó que no existirá relación directa entre un músico y una banda, puesto que la idea del proyecto es que un usuario con el rol de músico pueda encontrar a un usuario con el rol de banda, o viceversa, que un usuario con el rol de banda

pueda encontrar a un usuario con el rol de músico.

MongoDB

En segundo lugar existe una base de datos no relacional, NoSQL, para la que se ha usado MongoDB gracias a su gran aceptación por la comunidad y previos conocimientos en este tipo de bases de datos. En esta base de datos se almacenarán los mensajes de cada usuario de la aplicación dentro del chat.

Existe una colección utilizada para almacenar los datos de los mensajes con los siguientes campos:

- Id: Con el id del usuario al que le llegan los mensajes.
- Messages: Incluye los mensajes que le han enviado a este usuario. Donde se distinguen los siguientes campos:

  from: Con el id del usuario del que proviene el mensaje.

  fromRole: Con el rol del usuario del que proviene el mensaje.

  text: Con el mensaje que se le ha enviado.

  date: Con la fecha en la que se ha enviado el mensaje.

De esta forma quedaría la base de datos de Mongo con los mensajes almacenados, así estarían separados los datos principales de la aplicación en la base de datos MySQL y los mensajes en esta base de datos.

3.1.2. Aplicación web

La aplicación web, al estar desarrollada en Angular, tiene una arquitectura muy consolidada que ya el propio framework la define. Este framework permite que la aplicación web tenga una integración con la API muy sencilla, es decir, cuando la aplicación web necesita realizar alguna operación de base de datos o realizar cualquier función solo tiene que utilizar un servicio de HTTP, que incluye el propio framework de Angular, la cual permite hacer una llamada a la API a partir de verbos GET, POST, PUT o DELETE y recibirá una respuesta en formato JSON por parte de la API.

Una vez la aplicación web ha recibido los datos pertinentes desde la API en formato JSON, solo hay que declarar un objeto que almacene estos datos y la aplicación web pueda utilizarlos para darle funcionalidad.

3.1.3. API RESTful

Una vez que se tiene diseñada la base de datos se pasa a pensar en la arquitectura de la API, de esta forma se hace más sencillo realizar el diseño. Cabe destacar que la arquitectura que se usa en la API será una arquitectura RESTful.

La API ofrecerá los servicios que se han definido en los requisitos, por lo que será la que se encargue de gestionar las peticiones del cliente, gestionará los datos, los guardará en las bases de datos (dependiendo de si es una información principal o el uso del chat, hará

uso de una base de datos u otra) y se encargará también de servir a los clientes a través de los end-points que se definirán en la API, respondiendo con mensajes en formato JSON.

Por tanto, la API seguirá una arquitectura de una aplicación normal MVC[5], pero en este caso solo tendrá la parte del Modelo y el Controlador, ya que las vistas son gestionadas por la aplicación web. Con esto se deja lista la arquitectura de la API.

3.1.4. Microservicio Chat

Una vez que se desarrolló la gran parte de la aplicación web y de la API, la última parte de desarrollo de este Trabajo Fin de Grado se centró en la implementación de un chat de mensajes privados entre usuarios de JoinBand.

Desde primera hora se pensó desarrollar el chat dentro de la API NodeJS utilizando WebSocket[6], pero iba a dejar la API con funcionalidades muy diferentes incluidas en un mismo lugar, y el chat en realidad se podría considerar como un servicio externo a esta API; lo cual mantener el chat dentro de la API de NodeJS haría que el proyecto no sea tan escalable como tener microservicios con la funcionalidades y requisitos en cada uno de ellos. Entonces, debido al conocimiento previo con SpringBoot se pensó realizar un pequeño microservicio que realizará la función del chat utilizando Server Sent Events. De esta forma la aplicación quedaría totalmente orientada a servicios, ya que si se va a realizar alguna función básica sobre los usuarios, operaciones CRUD o realización de una búsqueda de músicos o bandas, se haría uso de la API en NodeJS; pero si se quisiera hacer uso del chat y el envío de mensajes se haría uso de este microservicio.

Es importante destacar que para la base de datos de este microservicio se ha usado una imagen de Mongo para poder lanzarla a través de un docker, por lo tanto este microservicio se hace totalmente portable hacia cualquier entorno de desarrollo. Es una idea que se pensó a la hora de diseñar este microservicio, porque la clave es que pueda ser totalmente dinámico y portable.

Así pues, la arquitectura que implementa el módulo del chat se compone de SpringBoot junto con Server Sent Events, y realizará todo el desarrollo funcional, conectándose con la base de datos Mongo para el almacenamiento de mensajes.

3.2. Tecnologías

Durante el periodo de elección de tecnologías no hubo apenas cambios, y desde primera hora la elección de cada una de ellas fue prácticamente directa. Esto es un proyecto muy escalable, en cuanto a funcionalidades se refiere, por lo que la idea desde el primer momento ha sido elegir unas tecnologías que permitan escalar el proyecto facilmente y la integración sea eficaz.

Para el desarrollo de la API se ha elegido NodeJS. Este framework de Javascript está basado en el motor V8 de Chrome, uno de los más avanzados. NodeJS otorga muy buena integración con módulos como Express, que hace el desarrollo de la API más liviano. Una ventaja de NodeJs es que con NPM se pueden instalar multitud de módulos compatibles con NodeJS para un desarrollo más amigable. Otro motivo por el que se eligió NodeJS es por ser Javascript, esto hacía tener un mejor manejo de la aplicación puesto que en la aplicación front-end está el framework Javascript de Google, Angular.

A la hora de pensar en tecnologías para desarrollar la aplicación web se debatió entre dos frameworks. Por un lado React era una buena opción dada su facilidad de iniciación al framework, y este proyecto era una buena opción para conocer este gran framework. Pero como ya se ha mencionado, para el desarrollo de la aplicación front-end se ha elegido Angular. Angular brinda el poder de una web SPA (Single Page Application), por lo que hará una navegación bastante fluida. Este framework permite un desarrollo de aplicaciones mucho más ágil, pasando de un MVC a un juego de puzzles con los componentes. Angular es genial para el desarrollo de una aplicación escalable, y Joinband cumple satisfactoriamente esta característica ya que se pueden ir añadiendo multitud de funcionalidades. Un punto muy a favor de Angular es que usa Typescript, por lo que otorga un código limpio, escalable, consistente y fácil de realizar depuraciones.

Otro punto muy valioso por el cual se ha elegido Angular como framework para el desarrollo del front-end, es su fácil comunicación e integración que tiene con otros frameworks como por ejemplo, Ionic[7]. Ionic proporciona un SDK completo, que permite el desarollo híbrido de aplicaciones móviles, por lo que sería un punto a favor por si en un futuro se decide sacar alguna aplicación móvil para Android o IOS como otra versión a la aplicación web.

En referencia al framework de CSS que se ha elegido para realizar el diseño gráfico de la aplicación, ha sido Bootstrap. En un principio se pensó usar Material, que ya es de fácil integración con Angular, pero la gran ventaja que tiene Bootstrap es la aceptación de la comunidad y es más familiar su sistema de *grid*. Además, el diseño estético que proporciona Material está bastante usado y no era lo bastante apropiado para esta aplicación.

En cuanto a la seguridad de la API desde el primer momento se pensó en JSON Web Token[8]. JWT permite la creación de tokens de acceso que permitirán la identificación en la API y así poder otorgar al usuario los privilegios de realizar consultas a los end-points de la API. Una de las razones por las que se eligió JWT fue por querer conocer y entender este estándar, ya que de una forma u otra es bastante conocido y muy bien valorado por la comunidad.

Para almacenar las imágenes de perfil de cada uno de los usuarios de la aplicación se barajaron varias opciones como recursos externos, por ejemplo Google Drive fue una opción,

a partir de la API que ofrece Google. Pero esta opción era utilizar un recurso demasiado amplio para tan solo almacenar una simple imagen. Por ello se encontró Cloudinary, una compañía que ofrece servicios de gestión de imagen y video basados en la nube. La API que proporciona Cloudinary (la cual gestiona NodeJS para almacenar las imágenes y obtener la URL) es muy fácil de usar, ya que un token de usuario y la función pertinente permite subir las imágenes a este cloud.

Para el desarrollo del chat se ha elegido usar Java 14 con el framework SpringBoot, que es ideal para crear microservicios. La elección de esta tecnología se debe al previo conocimiento de este framework por haber trabajado con él, y saber la gran versatilidad que tiene. Con Lombok[9] se va a permitir la opción de reducir código duplicado a través de las anotaciones que permite esta librería de Java. Dentro de la tecnología usada para la implementación del chat en tiempo real se ha elegido la opción de usar Server Sent Events. Esto es necesario para que el microservicio pueda lanzar eventos cuando recibe un mensaje, y sean entregados al destinatario, si no se implementase una tecnología de este tipo habría que estar haciendo peticiones constantes desde la aplicación web para obtener los mensajes, lo cual no es una opción óptima.

3.3. Dependencias

Las dependencias del proyecto es un apartado importante que hay que remarcar. Sin estas dependencias el proyecto no funcionará, ya que necesita de estas para tener un funcionamiento completo y correcto de la aplicación.

El proyecto tiene diferentes dependencias que en los dos siguientes apartados quedarán categorizadas al apartado que corresponda, ya sea una dependencia para la aplicación web con Angular o para la API con NodeJS.

3.3.1. Aplicación web

En cuanto a las dependencias de la aplicación web, se han utilizado diferentes framework y librerías para la gestión de la interfaz y funcional en el desarrollo.

Bootstrap es el framework CSS utilizado, y es la dependencia más llamativa integrada con Angular. Con este framework se ha desarrollado toda la interfaz visual de la aplicación. Para que la versión de Bootstrap quede más simplificada en cuanto a ficheros JavaScript, Bootstrap se ha integrado con una librería especialmente diseñada para Angular, llamada Ngx-Bootstrap. Esta librería gestionará de una manera más óptima los componentes dinámicos que requieran de una función JavaScript, como puede ser un 'modal'.

Ng-Select es un módulo importante que se ha integrado con Angular, para poder incorporar aquellos componentes de selección que requieren de una funcionalidad extra, como puede ser por ejemplo, selectores múltiples y reecarga de otros selectores a partir de la opción escogida en el anterior selector, de forma reactiva.

Para el uso de los iconos que aparecen por toda la interfaz web, se ha utilizado el conjunto de herramientas que ofrece FontAwesome. Es una de las fuentes de iconos más conocida a nivel mundial, que funciona muy bien y es muy recomendada por la comunidad.

3.3.2. API

La dependencia más importante de la API es Express, una infraestructura web rápida, minimalista y flexible para NodeJS. Toda la API esta desarrollada a partir de Express, ya que alberga muchos métodos de programación de utilidad HTTP y middleware, y hace la creación de una API sólida con una estructura sencilla.

Otra dependecia relativa tratada es con Cloudinary. La API de JoinBand se conecta con la API de Cloudinary, que permitirá almacenar y gestionar las imágenes de perfil de cada uno de los usuarios de la aplicación.

La librería de JSON Web Token para Node es un elemento dependiente más, que se ha de incluir para que la generación y gestión de tokens y autenticación en la aplicación sea correcta y factible.

Por último, las dependencias de la API que hay que incluir son las bases de datos MYSQL y MongoDB. Ambos módulos se incluyen para que NodeJS pueda gestionar cada una de las bases de datos y pueda operar con ellas.

3.4. Diseño de la API

A la hora de hacer el diseño de la API hay que tener en cuenta algunas cuestiones importantes, como son la seguridad, el protocolo de intercambio de mensajes que se va a usar y el sistema de rutas que se va a montar. En este apartado se hablará de cada una de estas cuestiones y la decisión que se ha tomado.

3.4.1. REST

REST es una interfaz que permite trabajar con mensajes JSON, por lo que no hay que definir un estándar de intercambio de mensajes. Además, como el front-end de la aplicación existe Angular, es muy sencillo poder recibir mensajes en formato JSON. Eso permite que se puedan usar los verbos HTTP, ya conocidos y que son familiares, como son GET, POST, PUT y DELETE.

3.4.2. Rutas

Las rutas de la API, que servirán como endpoints para el cliente front-end, tienen que ser del estilo:

http://localhost:3000/api/request

Cuando se va a realizar una peticion con el verbo GET, la ruta tiene que llevar en la URL los parámetros que sean necesarios para poder realizar la función. Cuando se trata de una petición del tipo POST, esta debe llevar en el cuerpo el objeto en formato JSON, para que la API pueda recoger esos datos y realizar la función. De igual forma ocurre cuando se quiere realizar una operación de actualización, la cual usará el verbo PUT, que aparte de llevar en el cuerpo el objeto en formato JSON, llevará en la URL el *id* del usuario al que se le va a realizar la actualización. Y por último, para realizar una operación DELETE, se deberá especificar en la ruta el *id* del usuario al que hará referencia.

3.4.3. Seguridad

Cuando un usuario inicia sesión en la aplicación se le proporciona un *token*, generado a partir de JWT. Ese *token* tiene un tiempo de expiración, pero mientras el cliente tenga ese *token* activo podrá realizar todas las peticiones que necesite.

De esta forma se le proporciona a la API una seguridad robusta, que hace que no cualquier cliente pueda realizar peticiones a la API, sino que necesitará de un *token* para poder utilizar las funciones importante de la aplicación.

3.5. Diseño del microservicio de chat

Cuando empezó a desarrollarse el servicio de chat, se barajaron dos tecnologías: WebSocket o Server Sent Events. Ambos son capaces de enviar datos al navegador y responder automáticamente al cliente. Server Sent Events presenta la característica de facilidad de uso y un control más exhaustivo de los mensajes; ya que WebSocket está pensado para enviar y recibir datos desde el navegador. La elección fue Server Sent Event, dada también su gran facilidad de integración con el framework SpringBoot.

3.5.1. Server Sent Events

El microservicio recogerá los eventos POST de los mensajes que le llegan desde la aplicación web, y acto seguido los almacena en base de datos (ya que si el usuario no está conectado, los mensajes deben permanecer guardados para ser mostrados en cuanto el usuario vuelva a entrar al chat) y lanzará un evento (si el usuario destinatario está conectado) con el mensaje para que lo reciba el destinatario, a través de una conexión HTTP que se ha establecido. La aplicación web recoge de forma automática el evento lanzado por el servidor. Este evento incluye el mensaje con el destinatario, que será necesario para poder mostrárselo al usuario destinatario del mensaje.

3.5.2. Rutas

Las rutas del microservicio, que servirán como endpoints para el cliente front-end, son necesarias para que la aplicación pueda desempeñar su función. En este apartado se detallarán las rutas que tiene el microservicio del chat.

Primero, cuando un usuario entre al apartado del chat se abrirá una conexión HTTP desde el cliente con Angular al microservicio. Será a través de esta conexión por la que se enviarán los eventos que producirá el microservicio cuando reciba mensajes, y se recogerá a partir de un objeto EventSource desde la aplicación web. Para abrir la conexión basta con utilizar la siguiente ruta con un *'new EventSource'*:

http://localhost:8095/user/'userId'/my/messages

Una vez que se ha establecido la conexión HTTP, lo siguiente que se realiza es cargar los mensajes que tienen los usuarios. De esta forma el microservicio extrae los mensajes de la base de datos y se los ofrece a la aplicación web a través de la siguiente ruta:

http://localhost:8095/user/'userId'/role/'role'/load/messages

Para enviar un mensaje a un usuario hay que usar la siguiente ruta a través del método POST que ofrece el microservicio:

http://localhost:8095/toUser/'toUserId'/toRoleUser/'toRoleUser'/fromUser/'fromUserId'/fromRoleUser/'fromRoleUser'/send

Con esta ruta se puede hacer llegar el mensaje al microservicio, se encargará de guardarlo en base de datos, y acto seguido le ofrecerá el mensaje al destinatario si está conectado a través de la emisión de un evento.

3.6. Diseño de la aplicación web

Ya se ha hablado sobre la aplicación web, en sus distintas fases de desarrollo, pero en este apartado se va a ver detalladamente el contenido y funcionamiento de la aplicación web, así como su interfaz de usuario.

La aplicación web, construida a partir del framework Javascript Angular, está estructurada en componentes, que a su vez se encapsulan en módulos para tener una mejor estructura empaquetada de la aplicación. La aplicación interactúa con la API en todo momento para poder realizar las acciones funcionales a partir de peticiones HTTP, en las que se obtiene como respuesta de la API e formato de intercambio de datos JSON, que es ideal a la hora de desarrollar aplicaciones web con un framework de este estilo.

3.6.1. Interfaz de usuario

La interfaz de usuario consta de cinco páginas, que conformarán toda la aplicación web: una página principal, una página de registro, una página para mostrar el perfil de usuario, la página de resultados de búsqueda y el chat.

Página principal

Figura 4. Página principal de JoinBand.

Al entrar en la aplicación, lo primero que verá el usuario será la página principal. En esta página se muestrá información básica sobre cómo funciona la aplicación, enlaces para iniciar sesión, registrarse y un buscador de músicos y bandas.

El buscador que aparece en la página principal consta de dos campos:

- **¿Qué buscas?** Es un selector en el que se puede elegir el tipo de músico que se está buscando, o también se puede seleccionar que lo que está buscando es una banda.
- **¿Dónde lo buscas?** Es un selector donde se muestran todas las provincias a nivel nacional, que será útil para hacer el filtrado de la búsqueda exclusivamente para la provincia seleccionada.

Cabe destacar que alguna de las imágenes e ilustraciones que se encuentran en la página principal, se han descargado de Google imágenes, pero que previamente se ha consultado la web de la que proviene y tienen licencia gratuita para su uso.

En la parte superior de la interfaz siempre aparecerá una barra de navegación (*navbar*), pero tendrá unas opciones distintas dependiendo de si el usuario ha iniciado sesión o no.

- Si el usuario no ha iniciado sesión aparecen las opciones:

  Iniciar sesión. Esta opción mostrará una ventana emergente con dos campos, uno para introducir el email y otro para la contraseña, que permitirá iniciar sesión en la aplicación. Si consigue iniciar sesión correctamente recibirá un *token* desde la API, que se usará para realizar todas las peticiones funcionales posteriores. Si no consigue iniciar sesión, porque haya fallado el email o contraseña, se muestra una alerta mostrando el error.

  Resgistrarse. Es una opción que permitirá ir a la página de registro de usuarios.

- Si el usuario ha conseguido iniciar sesión se mostrarán las siguiente opciones:

  Mensajes privados. Con esta opción se podrá dirigir al usuario al chat de mensajes privados para que así pueda consultarlos y gestionarlos.

  Mi perfil. Es una opción que permitirá ir al perfil propio del usuario, donde podrá consultar sus datos y editarlos.

  Cerrar sesión. Se elimina el token del usuario, y por tanto queda cerrada la sesión. El *navbar* volverá a tener las opciones para un usuario que no ha iniciado sesión.

Página de registro

La página de registro tiene una interfaz sencilla que permitirá al usuario registrarse fácilmente en el sistema. Para ello existen dos formularios reactivos, es decir, serán cargados de forma reactiva dependiendo de la opción que se escoja a lo hora de registrarse. Como se muestra en la Figura 5, en la parte superior existe una opción en la que se puede elegir entre dos formularios: 'Soy un músico' o 'Soy una banda'. Elegida una opción se cargará de forma reactiva el formulario seleccionado con el rol de usuario con el que se quiere registrar en la aplicación.

Todos los campos de registro tienen validaciones, es decir, deben cumplir una serie de directices que se han establecido para que al completar el formulario la información que se envía a la API para dar de alta al usuario sea correcta. Por ejemplo, se valida que en el campo email, cumpla con las directrices de un email verdadero. A la hora de introducir la contraseña y repetir contraseña se comprueba que ambas coincidan. Y por supuesto

todos los campos deben tener la información pertinente introducida para que se active el botón de resgistrase.

Cuando el usuario cliquea en el botón de registro, se envía la información del formulario a la API, y dependiendo de la respuesta que dé la API se mostrará una alerta de error o de registro completado con éxito.

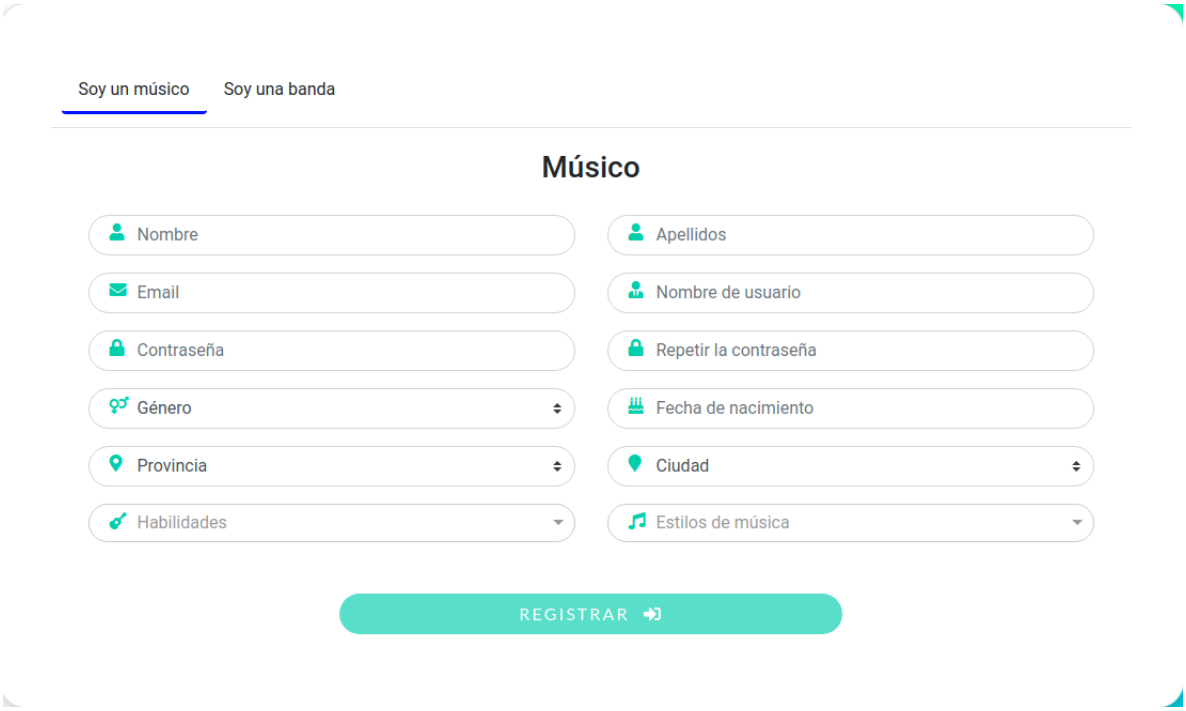

Figura 5. Página de registro.

Página de resultados

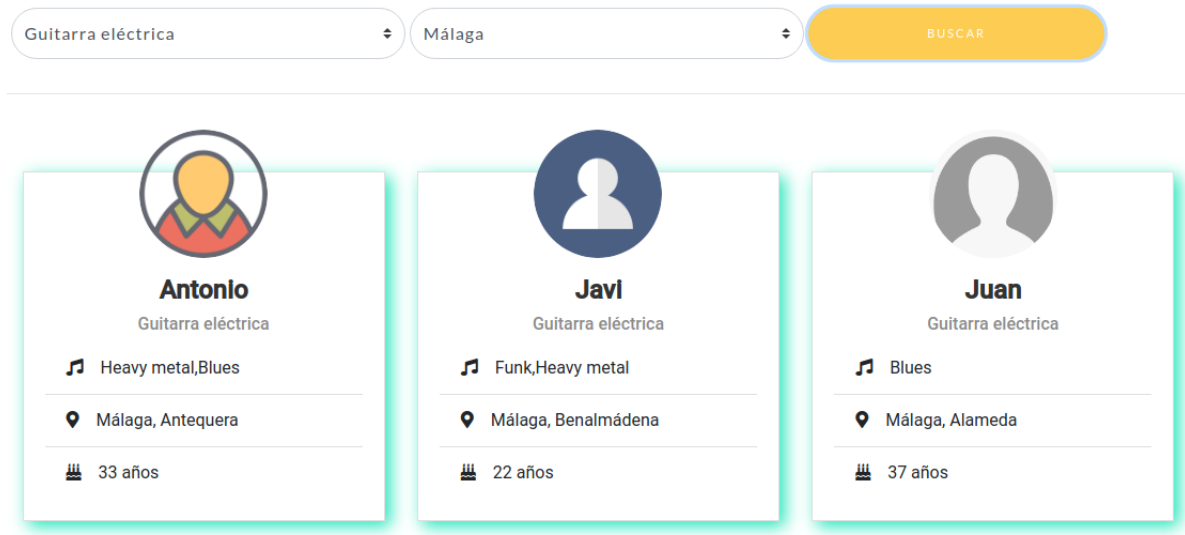

Figura 6. Página de resultados de búsqueda.

La Figura 6 muestra un ejemplo de búsqueda, en este caso se ha realizado una búsqueda sobre un músico que toque la guitarra eléctrica en Málaga. Como se observa en la Figura 6, en esta página aparecen 3 resultados, que corresponderán a 3 músicos que cumplen la condición de filtrado de la búsqueda. Por cada usuario se muestra una especie de tarjeta con la siguiente información:

- Una imagen, que corresponderá a su imagen de perfil.
- El nombre.
- La habilidad que desempeña.
- Datos importantes como por ejemplo, el estilo de música, su localización y la edad.

Si se hace *click* en alguna de las tarjetas se visitará el perfil de ese usuario. En la parte superior de la página de resultados aparecerá el buscador por si se quiere realizar una nueva búsqueda, que refrescará la página con los resultados asociados a la nueva búsqueda realizada.

Página de perfil

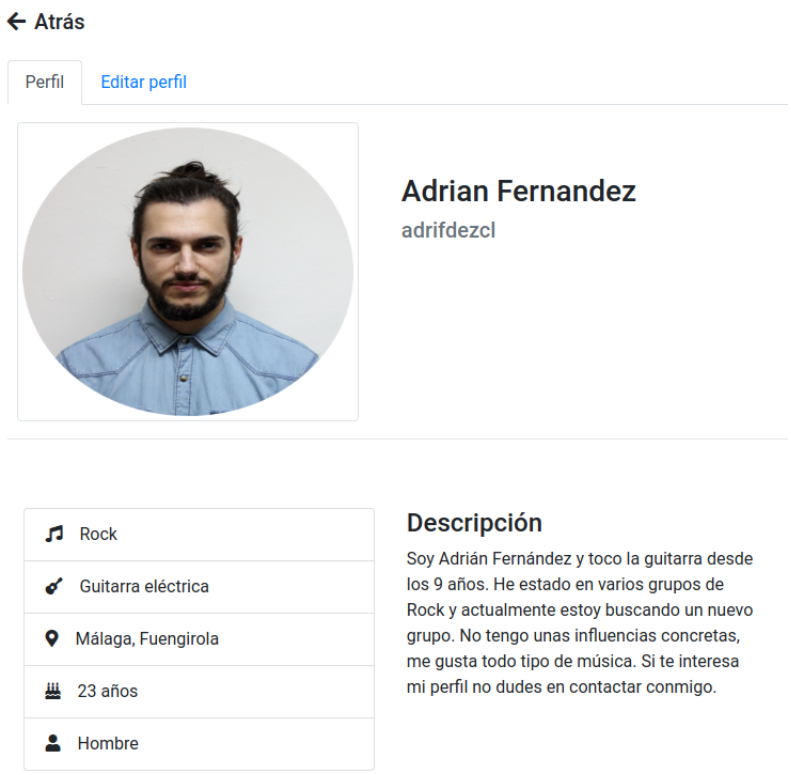

Figura 7. Página de perfil de usuario.

Una vez seleccionado un usuario en la página de resultados, la página que aparecerá será el perfil de usuario, tal y como se muestra en la Figura 7. En esta página se da una información más detallada sobre el músico o banda que se está visitando; como por

ejemplo, una descripción detallada que haya introducido previamente el usuario que se está visitando o el nombre de usuario.

Además, en esta página aparece un botón que indica 'Contactar', este es un botón que permitirá desplegar un formulario para el envío del mensaje con este usuario.

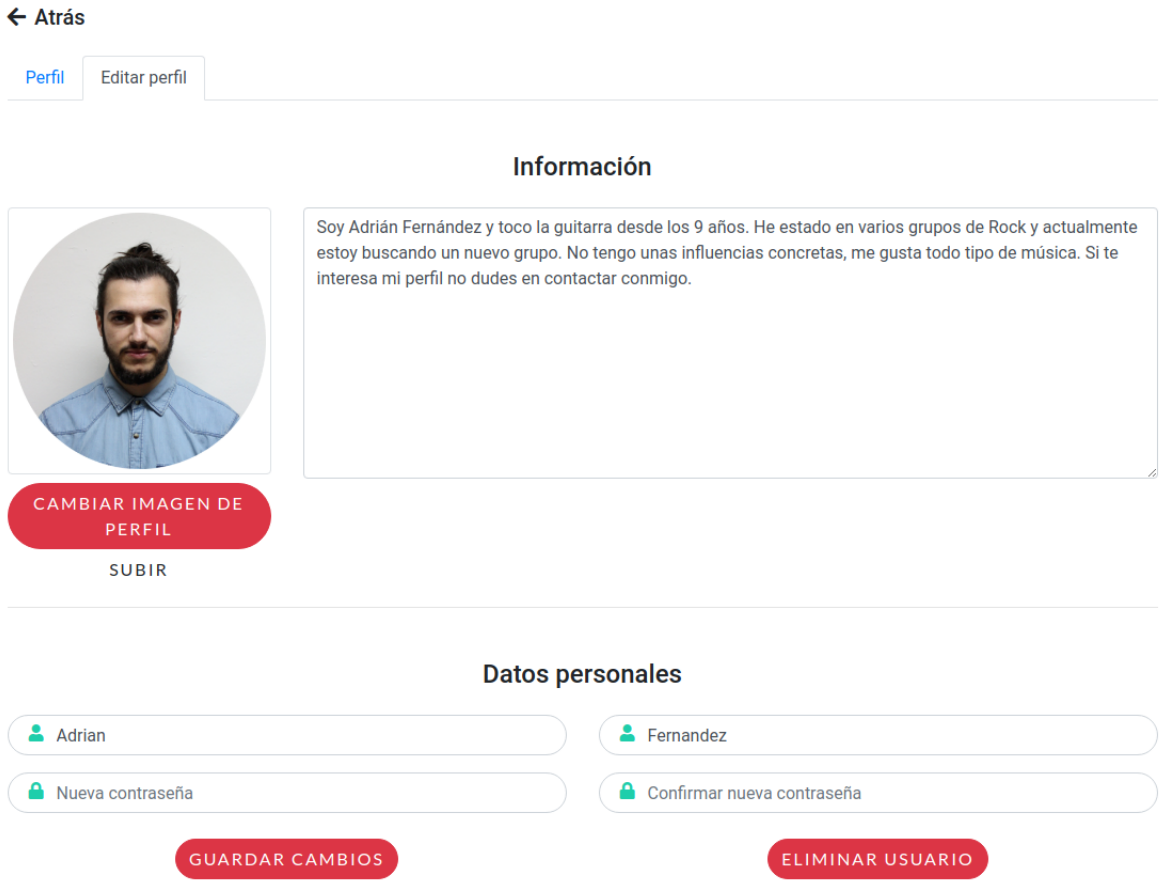

Figura 8. Página para editar el perfil de usuario.

La página de perfil es reutilizada para cuando el usuario que tiene la sesión iniciada navega a la página de 'Mi perfil', que siempre estará presente en la parte superior de la interfaz; obviamente en este caso no aparecerá el botón de contacto con este usuario, ya que es él mismo. En esta página aparecerá la información que corresponde con el usuario, y además, como se muestra en la Figura 8, aparecerá una opción activable que corresponde a la opción de editar su perfil.

En está pestaña se le permitirá al usuario poder actualizar su foto de perfil y algunos de los datos personales considerados más relevantes. Una opción importante que se le ofrece al usuario en esta ventana es poder eliminar su cuenta del sistema.

Página de mensajes privados

Si se pulsa en la opción del *navbar* que indica 'Mensajes Privados' se redirigirá a la página del chat. Tal como se muestra en la Figura 9, aparece un chat que permite hablar a través de mensajes privados con aquellas personas con las que se quiere contactar.

En la parte izquierda de la Figura 9, se puede observar una lista de usuarios, que corresponde a los chats que se tienen abiertos en el momento, y que por tanto existe una conversación con ese usuarios de la aplicación en ese momento.

En la parte derecha de la Figura 9, se pueden los mensajes con el usuario que se ha seleccionado para mantener una conversación, teniendo en la parte derecha aquellos mensajes que ha escrito el usuario y a la izquierda aquellos mensajes que proceden del usuario con el que se está contactando.

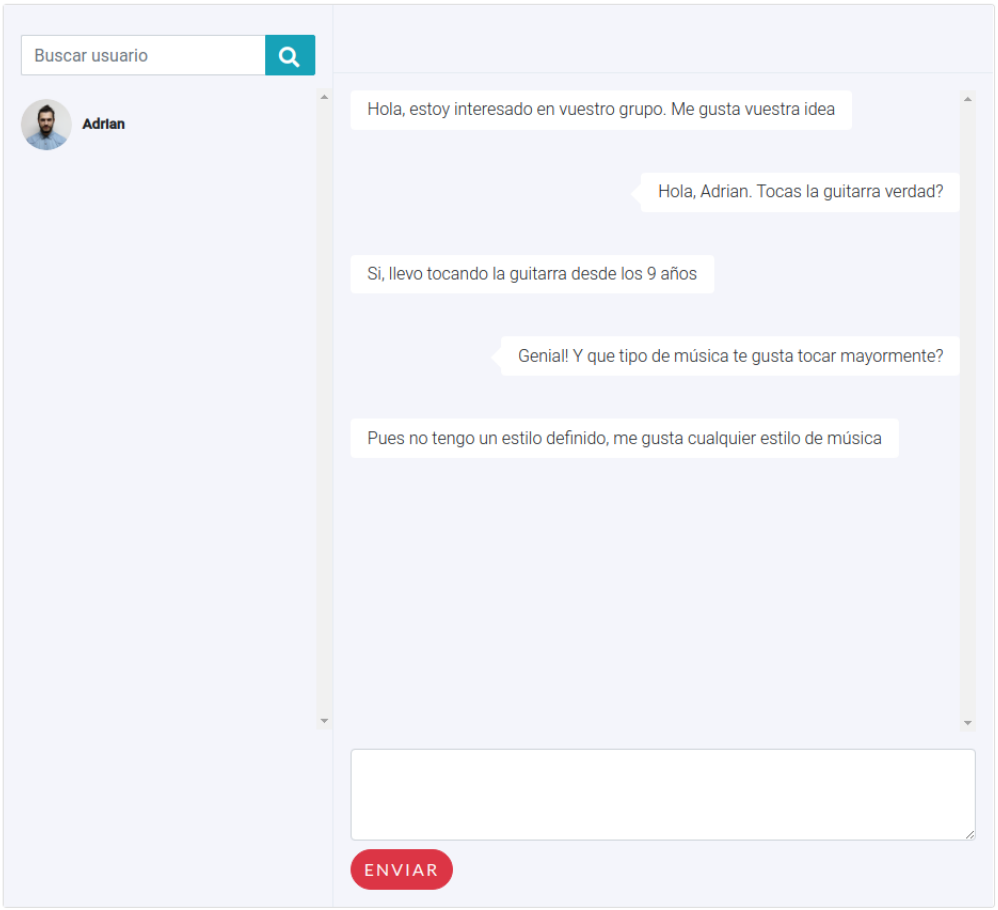

Figura 9. Chat de mensajes privados (Un músico conversando con una banda)

Responsive

Un punto muy positivo que no hay que dejar atrás es que gracias al framework de CSS Bootstrap, la aplicación web es totalmente responsive, lo que permite que la aplicación se pueda adaptar visualmente a cualquier dispositivo.

Es una parte muy importante que se ha querido tener en cuenta desde el minuto cero en que se empezó a desarrollar toda la parte gráfica de la aplicación, ya que hoy en día la mayoría de la gente usaría la aplicación web a través de un móvil o tablet y es necesario que la visualización de cada uno de los componentes se vean de forma óptima para poder hacer un uso correcto de la aplicación.

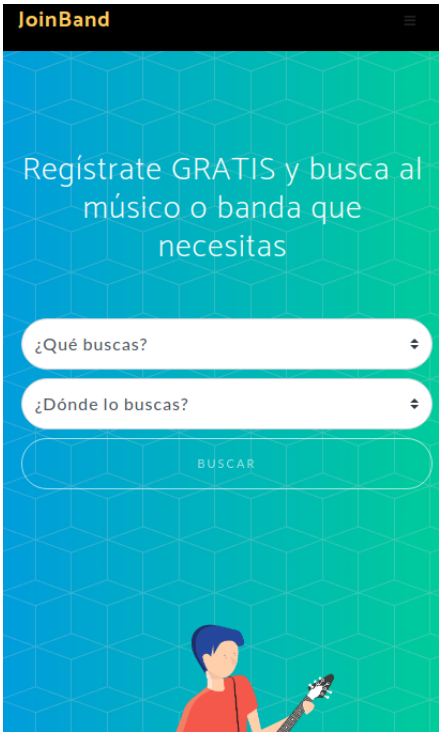

Figura 10. Ejemplo responsive para un móvil estándar.

CAPÍTULO 4

Metodologías

4.1. Planificación

El desarrollo del proyecto se ha estructurado en 4 fases o etapas.

Figura 11. Etapas del desarrollo del proyecto.

- La primera fase del proyecto se dedica a realizar un análisis exhaustivo de los requisitos funcionales y no funcionales, y desarrollar una pequeña memoria introductoria para tener un control, que posteriormente ha dado lugar a esta memoria actual.
- Una vez conocidos los requisitos, en la segunda etapa se comienzan a diseñar y desarrollar las bases de datos.
- Cuando la base de datos está diseñada, el desarollo de la API será el que conforme la fase 3 del desarrollo del proyecto.
- En la etapa 4 se desarrolla la aplicación web, que se servirá de la información que le provee la API.

En cuanto a la planificación se ha utilizado la metodología ágil Scrum. Al inicio de cada semana se hace una planificación de lo que se tiene que tener hecho para la próxima semana. Para ello se realizan unos tickets para la aplicación web y tickets para el desarrollo de la API. Cada ticket es una tarea que hay que realizar, y cada uno de ellos contiene una información detallada sobre la tarea. Esto permite desarrollar de una manera más eficiente, ya que si el desarrollo de una ticket durá más de un día, siempre se puede consultar la información que tiene escrita el ticket por si hay alguna duda.

En la aplicación web de Trello[10] se encuentran las pizarras que conforman la planificación del proyecto.

- Pizarra para la API (back-end).

- Pizarra para la aplicación web (front-end).

En cada una de las pizarras existen tres listas. En primer lugar hay una lista llamada 'To Do', donde se listan todas las tareas pendientes de desarrollar en la aplicación. Una segundo lista intermedia 'In progress', la cual contiene las tareas que se están realizando en ese momento. Por último, existe una tercera lista 'Done' que contiene aquellas tareas que han completado el proceso de desarrollo. Cabe destacar que si una de las tareas necesita una información más detallada, se incluye un apartado de descripción donde se especifica de forma precisa lo que requiere esa tarea.

Para llevar un control exhaustivo de la aplicación se ha usado el control de versiones más conocido, Git[11]. Esta ha sido una herramienta clave, ya que el proyecto se ha estado desarrollando con un portátil y con un ordenador de sobremesa, que gracias a Git y GitHub[12] se ha podido tener un control de versiones cuando se intercambiaba el desarrollo entre un ordenador y otro.

CAPÍTULO 5

Resultados

5.1. Conclusiones

Gracias a este proyecto se han podido investigar y aprender tecnologías novedosas, y sobre todo se han podido poner a prueba muchos de los conocimientos adquiridos a lo largo del grado. Ha sido todo un reto personal, pero que con la dedicación necesaria y el esfuerzo se ha conseguido.

Se ha desarrollado una aplicación que puede ayudar a muchos músicos y bandas de España, que antes tenían que pasar días, incluso meses en encontrar a los componentes para formar una banda completa y que gracias a JoinBand ese tiempo se puede reducir a simplemente unos minutos; además de dar la posibilidad de conocer a muchísima gente para futuros proyectos de cada uno de los músicos y bandas de este país.

En general el desarrollo del proyecto ha sido fluido, ha habido pocos momentos en los que se ha tenido que parar para hacer modificaciones que no se pensaban que serían necesarias, solo hubo pequeños cambios sin gran importancia. Por ejemplo, en la base de datos no tenía sentido pedir la fecha de nacimiento a una banda, por lo que ese atributo se cambió por 'Años tocando juntos' que tiene más sentido para este rol de usuario.

En cuanto a los objetivos que se tenían estipulados cumplir, es cierto que se han cumplido todos tal y como se habían pensado desde el primer concepto de JoinBand. Es verdad que el desarrollo del chat de mensajes privados fue un poco más costoso. Esto se debió a que no se tenía conocimiento sobre cómo funcionaba realmente un chat en tiempo real. Para ello se tuvo que investigar sobre librerías y diferentes opciones para que funcionaran los mensajes que se escribían en la aplicación web, se comunicaran con el microservicio, se almacenaran en la base de datos y se lanzara el evento que recogería la aplicación web para mostrarle el mensaje al destinatario en tiempo real.

En definitiva ha sido un proyecto adecuado para poder desarrollar una aplicación completa, pasando por cada una de las fases que caracterizan el desarrollo de un producto software, en la que se emplean distintas tecnologías actuales con mucho potencial.

Líneas futuras

En cuanto a ampliaciones futuras, JoinBand es una aplicación que va a permitir infinidad de ampliciones, ya que es un tipo de aplicación que puede variar mucho y dirigirse a múltiples caminos.

Una ampliación bastante viable sería implementar Docker para toda la aplicación. Con Docker se puede automatizar el despliegue de aplicaciones dentro de contenedores de software, lo cual da mayor facilidad en fases de desarrollo y despliegues de la aplicación a producción. Para despliegues de la aplicación sería interesante incluir algunas herramientas con Jenkins, que permitirá la conocida integración continua. La cuestión por la que no se han incluido estas mejoras en este proyecto es principalmente por cuestiones de tiempo y que añadirlas sobrepasaría con creces el tiempo estipulado para este trabajo fin de grado.

Otra posible ampliación muy llamativa es la opción de desarrollar la aplicación móvil como versión de la aplicación web. El desarrollo nativo en Android e IOS sería muy costoso, por lo que sería buena opción desarrollar una aplicación híbrida con frameworks como IONIC. Gracias a que la aplicación web está desarrollada con Angular, sería un desarrollo mucho más liviano a la hora de incorporar IONIC.

Bibliografía

[1] *CRUD*. URL: https://es.wikipedia.org/wiki/CRUD.

[2] *MySQL*. URL: https://www.mysql.com/.

[3] *Cloudinary*. URL: https://cloudinary.com/.

[4] *MongoDB*. URL: https://www.mongodb.com/es.

[5] *MVC*. URL: https://desarrolloweb.com/articulos/que-es-mvc.html.

[6] *WebSocket*. URL: https://tools.ietf.org/html/rfc6455.

[7] *Ionic*. URL: https://ionicframework.com/.

[8] *JSON Web Tokens*. URL: https://jwt.io/.

[9] *Lombok*. URL: https://projectlombok.org/.

[10] *Trello*. URL: https://trello.com/.

[11] *Git*. URL: https://git-scm.com/.

[12] *GitHub*. URL: https://github.com/.

Apéndices

APÉNDICE A

Manual de instalación

En este apéndice se detallarán los pasos a seguir que serán necesarios para poder ejecutar la aplicación en local, es decir, en su propio ordenador. Para poder ejecutar la aplicación se proporcionan 3 conjuntos de ficheros importantes, se trata de:

- Aplicación web. Para ello se proporciona la aplicación Angular, que será necesario arrancar.
- API. Se proporciona la API en NodeJS, lista para arrancar.
- Base de datos. Se ofrece el esquema de la base de datos MySQL que será necesario importar en el servidor MySQL.

A.1. Aplicación web

Para poder arrancar el proyecto front-end (se encuentra en el proyecto 'joinband-frontend') será necesario tener Angular en su versión 9, esto es lo único que se necesitará, ya que las dependencias de los paquetes se instalarán automáticamente cuando se compile Angular, por lo que no será necesario instalar nada más. Acto seguido será necesario dirigirse a una terminal y lanzar el siguiente comando para arrancar la aplicación web:

A.1.1 *Introducir en una terminal:* ***ng serve -o***

Una vez que se ejecute el comando anterior desde una terminal, se compilará todo el proyecto front-end, y con la opción **-O** se dice que se abra automáticamente en el navegador predeterminado que se tenga seleccionado en el sistema operativo. La página que se abrirá estará en *localhost:4200*

En cuanto a la aplicación web esto es todo lo necesario para hacerla funcionar, en este punto la aplicación web debe funcionar correctamente, excepto la funcionalidad que requiera de la API, que debe estar iniciada también para que pueda hacer las peticiones y obtener las respuestas.

A.2. API

Para poder poner en marcha la API en local (se encuentra en el proyecto 'joinband-backend'), primero de todo se tiene que tener en cuenta que esté iniciada la base de datos MySQL. También es necesario tener instalado NodeJS en su versión 12, puesto que es la versión con la que se ha desarrollado esta API.

Para tener el mismo esquema de la base de datos relacional, hay que importar en el servicio de MySQL, el archivo del esquema de base de datos que se proporciona.

Una vez que las bases de datos están iniciadas, se puede proceder a ejecutar la API. Para ello es necesario dirigirse a una terminal e introducir el siguiente comando:

A.2.1 *Introducir en una terminal:* ***npm run dev***

En este punto, con la API (junto con las bases de datos) y la aplicación web ejecutándose, está listo todo para poder usar la aplicación en todas sus facetas .

A.3. Microservicio del Chat

Para ejecutar el microservicio sería ideal tener un IDE como puede ser IntelliJ, ya que ha sido el que se ha usado para realizar el desarrollo de esta parte. De esta forma solo habría que importar el proyecto de 'joinband-chat' a IntelliJ y ejecutarlo.

Los requisitos para ejecutarlo sería tener una versión de Maven instalada, como por ejemplo la versión 3.6.3, que se encargará de la gestión y construcción de paquetes en este proyecto Java. Por otro lado, es necesario tener Docker instalado y levantar el docker que se encuentra dentro del proyecto 'joinband-chat', que iniciará la base de datos Mongo.

Para la colección de base de datos en Mongo no hay que preocuparse ya que se tiene definida como una clase dentro del código del microservicio y se lanza automaticamente al docker de mongo.

Con todos estos pasos realizados, la aplicación funcionará en su totalidad.

UNIVERSIDAD DE MÁLAGA | uma.es

E.T.S. DE INGENIERÍA INFORMÁTICA

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