Escuela Politécnica Superior de Jaén

Grado en Ingeniería Informática

UNIVERSIDAD DE JAÉN

Escuela Politécnica Superior de Jaén

Trabajo Fin de Grado

BAND & MUSIC APP: PLATAFORMA WEB PARA DIRECTORES, MÚSICOS Y USUARIOS APASIONADOS DE LA MÚSICA

Alumno: **Antonio Javier Ocaña García**

Tutor: Prof. D. José Ramón Balsas Almagro

Dpto: Informática

Septiembre, 2020

Universidad de Jaén

Escuela Politécnica Superior de Jaén

Departamento de Informática

Don José Ramón Balsas Almagro , tutor del Trabajo Fin de Grado titulado: Band & Music App, que presenta Antonio Javier Ocaña García, autoriza su presentación para defensa y evaluación en la Escuela Politécnica Superior de Jaén.

Jaén, Septiembre de 2020

El alumno: Los tutores:

Antonio Javier Ocaña García José Ramón Balsas Almagro

Índice

Índice de imágenes:

Índice de tablas:

Índice de diagramas:

Índice de paquetes:

Abreviaturas y siglas:

BBDD: Base de Datos

C: Could

CSS: Cascading Style Sheets

DAO: Data Access Object

DNI: Documento Nacional de Identidad

DTO: Data Transfer Object

e2e: End To End

HTML: HyperText Markup Language

HTTP: Hypertext Transfer Protocol

JPA: Java Persistence API

LXC: Linux Containers

M: Must

MVC: Modelo Vista Controlador

ORM: Object-Relational Mapping

POM: Project Objects Models

S: Should

SO: Sistema Operativo

SQL: Structured Query Language

SRC: Source

TFG: Trabajo Fin de Grado

TS: TypeScript

UML: Unified Modeling Language

URI: Uniform Resource Identifier

URL: Uniform Resource Locator

W: Would not have

WAR: Web Application Archive

1. Introducción

En la primera parte del Trabajo Fin de Grado (TFG), se presenta una breve introducción de lo que se hablará a lo largo del documento. En dicha sección se discutirá la motivación para realizar un trabajo como este, los diferentes objetivos a cumplir para obtener nuestro sistema, las metodologías que se utilizarán en el desarrollo y, para concluir, cómo está estructurado el documento.

1.1. Motivación

"Programar sin una arquitectura o diseño en mente es como explorar una gruta solo con una linterna: no sabes dónde estás, dónde has estado ni hacia dónde vas" (Danny Thorpe).

Gracias a una buena planificación de desarrollo, se pueden cubrir ciertas necesidades que quedan carentes en la sociedad de hoy día, de una manera eficaz.

Para programar es fundamental tener una buena base de análisis y diseño, elementos necesarios para llevar a cabo esta acción. Esto es de vital importancia ya que hay que tener muy claro lo que se quiere desarrollar y que nuestra aplicación cumpla las necesidades que se deben cubrir.

Por ello, surge la propuesta tras una idea propia que brotó para poder proporcionar a la banda de música a la que pertenezco, un paso más hacia la gestión informática de algunos procesos habituales realizados actualmente de forma manual. Por ejemplo, esta banda aún tiene libros de asistencias tanto a ensayos como a actos y nos proporcionan las partituras a cada uno/a de los miembros en mano. De esta manera, surtió el propósito de esta aplicación, ya que todo este trabajo, se podría hacer más eficiente y con un costo menor, pues los libros en algún momento se acaban.

En este caso, me puse en contacto con la subdirectora de dicha banda y directora de la banda juvenil para saber su veredicto sobre la idea y le pareció óptima, ya que facilitaría su trabajo en gran medida. Pues utilizando la aplicación estarían en contacto con sus músicos y también podría facilitar las diferentes *particellas* [1]que se ensayarían.

[1] *Particellas:* Coloquialmente conocido como partitura en el mundo de la música.

1.2. Objetivos

En esta sección, se van a incorporar los objetivos a cumplir en la propuesta de trabajo:

- Realizar un estudio para una aplicación web relacionada con la música orientada a las bandas para cumplir las necesidades que estas tienen y la creación y gestión de este sistema
- Diseñar un sistema informático, que está orientado a la utilización de arquitecturas y metodologías relacionadas con el desarrollo web.
- Desarrollar un prototipo de aplicación web, en el cual utilizamos por el lado del servidor tecnologías REST basadas en Spring boot y por parte del cliente una interfaz usando *HyperText Markup Language* (HTML) 5 y Angular 2.

1.3. Metodología

A continuación, se van a proporcionar los conceptos básicos de metodología que va a ser desarrollada a lo largo del proyecto:

- Realizar un estudio de los requisitos del sistema que se quiere desarrollar para cumplir en la aplicación
- Recopilación bibliográfica sobre los aspectos relacionados con la temática y tecnologías de la aplicación
- La metodología utilizada para el desarrollo ágil del proyecto está basada en incrementos. A esta, se le van a aplicar una serie de características de SCRUM ya que ha sido estudiada durante la carrera. De esta, se cogerá la organización de las tareas a realizar que proporciona mediante sprints, así como la evaluación del tiempo
- Realización del modelado *Unified Modeling Language* (UML) para el proyecto
- Desarrollo del diseño del sistema utilizando los patrones de diseño que considero necesarios y las arquitecturas relacionadas con Ingeniería del Software
- Elaborar la implementación del prototipo del sistema

1.4. Estructura del documento

En este apartado, se describirá brevemente cómo se han organizado los contenidos en la presente memoria donde se pueden ver los diferentes puntos a tener en cuenta a lo largo de la producción de una aplicación web.

El apartado 1, como ya hemos visto anteriormente, está dedicado a la introducción del proyecto. Esta información es utilizada para orientar al lector de los objetivos que se tendrán en cuenta a lo largo del desarrollo de dicho proyecto.

Se continúa con el apartado 2, en el cual se lleva a cabo el análisis inicial para presentar los diferentes puntos principales que deben tenerse en cuenta. Estos están basados en, por ejemplo, estimaciones iniciales de tareas y costes, requisitos del sistema, entre otras cosas.

Siguiendo con el apartado 3, aquí se tratan los aspectos del diseño de la aplicación web. Estos aportan los diferentes diagramas que se tendrán en cuenta en el funcionamiento del sistema que se va a desarrollar.

Prosiguiendo con las características del apartado 4, está formado por la información del desarrollo e implementación del código de la aplicación, tanto por parte del servidor como del cliente.

Para concluir la estructura del documento, se va a hablar del apartado 5, este está compuesto por las conclusiones obtenidas tras finalizar el proyecto propuesto en este TFG.

2. Análisis

En esta fase de análisis, se intenta que el lector obtenga mayor conocimiento del sistema que se va a desarrollar de forma gradual. Este punto se divide de la siguiente forma:

Análisis preliminar, utilizado para la explicación del sistema o el contexto del que se ha partido para realizar el desarrollo de la aplicación. Se hablará de las características, necesidades, problemáticas etc. Se procederá a hacer un estudio de algunas soluciones que existen en la web actualmente, sobre las que me apoyaré para así, obtener ideas para este proyecto.

Para las diferentes tecnologías o *frameworks*, también se desarrolla un análisis preliminar para introducir brevemente al lector en las características generales, historia, evolución… entre otros muchos puntos de información sobre ellas.

También se proporciona una propuesta de solución, con la que se quiere dar una visión global del sistema a desarrollar y de las metodologías y tecnologías a emplear.

Tras esto, se detallan los requisitos funcionales y no funcionales que deben ser cumplidos por la aplicación.

Después de los requisitos, se argumenta la metodología que se va a utilizar para organizar el desarrollo del sistema cumpliendo una serie de pautas; prosigue con la exposición de las historias de usuarios que me ayudan a facilitar la funcionalidad completa del sistema y, para terminar, se realiza una descomposición de historias de usuario en diferentes tareas y se muestra la planificación del tiempo que se va a emplear en el desarrollo de estas comentadas con anterioridad. Teniendo así en cuenta los puntos de esfuerzo, prioridad que tienen, y duración que se considera oportuna para cada una de ellas.

2.1. Análisis preliminar

En esta sección, se procede a la explicación del sistema o el contexto del que se ha partido para realizar el desarrollo de la aplicación. Se hablará de las características, necesidades, problemáticas, así como otros aspectos.

A continuación, se realiza un pequeño estudio de soluciones similares existentes con el objetivo de obtener los puntos fuertes, débiles y posibles mejoras que podrían llevarse a cabo para hacer una aplicación aún mejor.

Para concluir este punto, se procede a facilitar la introducción de los *frameworks* que han sido elegidos por mí para realizar el desarrollo. En dicho apartado, se informa sobre las características generales, los tipos de proyectos donde son utilizados comúnmente, historia, evolución y otros muchos puntos de información sobre ellas.

2.1.1. Análisis preliminar del sistema

El propósito que se tiene es crear un sistema que aporte soluciones a las necesidades del proyecto y objetivos ya comentados en el primer apartado de este documento. Por ello, el sistema debería de cumplir unos requisitos que son los siguientes:

Permitir a los usuarios/as de la aplicación obtener las diferentes informaciones de las partituras que quieren encontrar, de los compositores/as que buscan, poder escuchar la música que ellos/as quieran, ver información de distintas obras, marchas, pasacalles, pasodobles, asignarles comentarios, poder estar en contacto con otros usuarios/as etc.

Además, se procurará que un usuario/a que sea director/a pueda crear su propia banda de música dentro de la aplicación para así poder estar en contacto con los/las músicos que la componen o compartirle las *particellas* que van a tocar, por ejemplo.

2.1.2. Estudio de soluciones existentes similares

En esta sección, se va a proceder a presentar diferentes aplicaciones similares a la idea que se ha presentado de este proyecto.

En primer lugar, se encuentra:

- **Compartitura**[2]

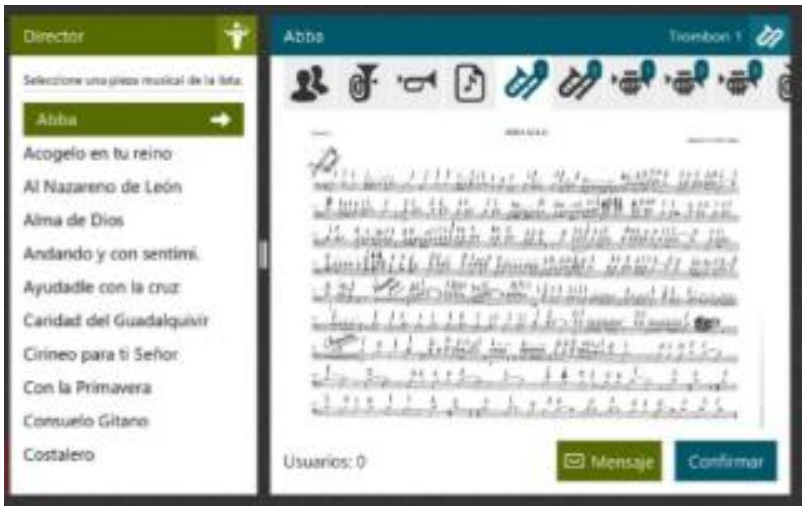
Esta aplicación está dedicada más específicamente a las bandas de música y a las agrupaciones musicales más que a los usuarios apasionados. Su objetivo es agilizar el reparto de *particellas* entre el/la directora/a y los/las músicos componentes de una banda mediante un punto wifi portátil. El director elige una partitura y, al seleccionarla, le aparece a cada músico dicha partitura la cual tienen que tocar.

[2] https://es.slideshare.net/Proconsi/app-de-msica-para-bandas-y-agrupaciones-musicales-compartitura

Desde el punto de vista de un/a director/a de una asociación musical, la aplicación está bastante bien para organizar las diferentes composiciones que se pueden tocar en una banda de música o agrupación musical. No obstante, la aplicación no permite realizar una organización de la propia banda (datos de músico, listado de componentes etc.).

Imagen 1. Compartitura

Imagen 2. Breve presentación de Compartitura



Otra aplicación similar es:

- **LaBanda**[3]

Este es un programa para ordenador que está enfocado a la gestión de bandas de música y agrupaciones musicales. Es una aplicación que facilita al/la director/a la organización de los actos, planificación de ensayos, realizar informes sobre los músicos, guardar la información de los músicos que componen su banda, los honorarios de cada músico, los instrumentos que tienen

[3] https://www.anapi.com/lb/labanda1.htm

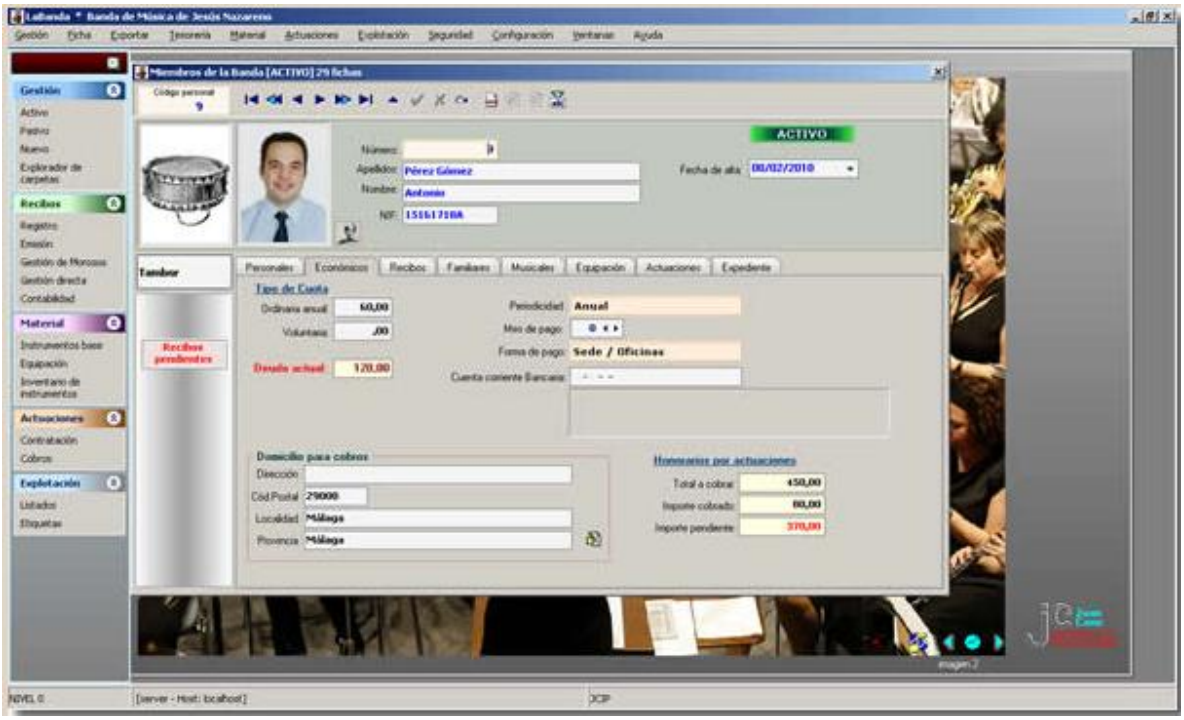
en activo, la gestión de la tesorería y muchos más puntos relacionados con la gestión de una banda.

En esta aplicación, como en la anteriormente citada, tampoco se tiene en cuenta a los usuarios apasionados de la música, ya que es sólo para directores/as, o miembros de la junta directiva para que puedan realizar cualquier tipo de gestión relacionada con dicha agrupación.

Imagen 3. Configuración de instrumentos de LaBanda

Imagen 4. Gestión de calendario y músicos de LaBanda

Imagen 5. Perfil de un músico en LaBanda



- **Cleartune**[4]

Esta es otra aplicación orientada a los/las directores/as de las bandas de música. Pues gracias a esta aplicación obtienen un afinador, para poder perfeccionar el sonido de los diferentes instrumentos que componen una banda de música.

Un punto desfavorable de esta aplicación es que no te aporta nada más, sólo un afinador, no te deja ni compartir diferentes composiciones y/o autores, crear tu propia banda y lista de músicos etc.

Esta aplicación es un producto comercial que está cerrado y debería de estudiarse como integrarlo con otras herramientas además de los costes derivados de su adquisición.

Para concluir este apartado, cabe decir que la elección de las aplicaciones vistas anteriormente, se debe a que tienen muchos puntos en común con la idea del proyecto que se quiere desarrollar en este TFG. Aunque todas están más relacionadas a las bandas de música que a los diferentes usuarios que tienen como hobbie la música y no tienen acceso a las diferentes composiciones y/o autores.

[4] https://apps.apple.com/es/app/cleartune-chromatic-tuner/id286799607

Imagen 6. Logotipo de Cleartune

Imagen 7. Vista de inicio de Cleartune

2.1.3. Introducción a los *frameworks* seleccionados

En este punto, se procederá a hablar sobre los *frameworks* que se utilizarán para el desarrollo de la aplicación web, tanto para el servidor como para el cliente.

Estos *frameworks* son muy completos y son complejos de aprender, pero han sido seleccionados porque facilitarán el proceso de desarrollo ya que han sido estudiados en la carrera. Además, permiten profundizar en los mismos complementando los conocimientos que se han adquirido sobre ellos.

2.1.3.1. Programación en el servidor Java: Spring Boot

- ¿Qué es Spring Boot?

Spring Boot [5]es una de las tecnologías que forma parte de la tecnología de Spring. Spring Boot nace con la intención de simplificar los pasos que se realizan para construir una aplicación en Spring Framework.

Con esta simplificación, Spring Boot reduce la complejidad del desarrollo de nuevos proyectos basados en Spring, proporcionando al sistema la configuración de la estructura básica del proyecto.

Esta incluye tanto las pautas para usar el marco, como todas las bibliotecas de terceros relevantes necesarias para la implementación de la aplicación. Con esta configuración que nos aporta, nos allana para comenzar lo más rápidamente posible.

- Características:
  - Incorpora directamente aplicaciones de servidores web/contenedores como, por ejemplo, pueden ser Apache Tomcat o Jetty, sin tener la necesidad de incluir en el proyecto archivos *Web Application Archive* (WAR)
  - En los archivos *Project Objects Models* (POM) se almacenan las distintas configuraciones de Maven facilitándonos una simplificación
  - Consigue una configuración automática de Spring
  - Incluye características como métricas o configuraciones externalizadas que se corresponden a características no funcionales

Imagen 8. Logotipo de Spring Boot

[5] https://spring.io/projects/spring-boot

2.1.3.2. Programación en el cliente web: Angular 2

- ¿Qué es Angular 2?

Angular 2 [6]es un *framework* de desarrollo que fue creado por Google para facilitar la creación de aplicaciones web. Este es de código abierto, se utiliza para crear y mantener aplicaciones web para conseguir un trabajo con los elementos necesarios de una manera más sencilla y óptima.

Su principal objetivo es que el número de aplicaciones basadas en web sea aumentado de manera considerable siguiendo la idea del estilo de arquitectura Modelo Vista Controlador (MVC), haciendo así que el desarrollo y las pruebas sean más fáciles para el/la desarrollador/a.

La biblioteca que contiene Angular está preparada para leer ficheros HTML donde están contenidos los atributos de las etiquetas adicionales que se necesitan que están personalizadas. Por ello, obedece a este contenido para poder unir las piezas necesarias (entrada o salida) de la página web a un modelo representado por JavaScript.

El predecesor de Angular es AngularJS y es incompatible con este.

Angular proporciona una recomendación, con la que comunica, que se use TypeScript. Este es un lenguaje de código abierto desarrollado por Microsoft. Dicho estilo añade un modo de tipado estático y objetos que está basado en clases.

- Características
    - Desarrollo móvil: Angular trabaja primero en el desarrollo móvil para obtener así los problemas de rendimientos y poder solucionarlos. De esta forma, consigue que el desarrollo de aplicaciones de escritorio o web sean mucho más sencillo
    - Modularidad: Angular trabaja con módulos, esto quiere decir, que cuando se desarrolla una nueva funcionalidad, se empaqueta en un módulo específico para conseguir así un núcleo más rápido y ligero
    - Compatibilidad: Angular es compatible con las últimas versiones de los navegadores más modernos

[6] https://angular.io/

Imagen 9. Logotipo de Angular

2.1.3.3. Utilización de base de datos: Docker

- ¿Qué es Docker?

Puede considerarse una virtualización "ligera" para máquinas anfitrión Linux, originalmente basado en la tecnología *Linux Containers* (LXC), ahora implementa su propia *runtime*. Un contenedor software empaqueta una aplicación en un entorno que incluye:

- Sistema de ficheros Linux
- Utilidades de sistema
- Bibliotecas de sistema
- Dependencias y configuración específicas de la aplicación

Los contenedores docker no instalan el sistema operativo completo, sino que comparten el núcleo y algunas bibliotecas con el sistema anfitrión.

En docker se pueden tener imágenes. Una imagen es un sistema operativo basado en alguna distribución Linux que contienen algunas aplicaciones estándar instaladas (Mysql, mongoDB, Apache, Tomcat…)

Un contenedor es una imagen que está configurada y en ejecución dentro de docker.

Docker puede gestionar distintos contenedores de manera simultánea, estos pueden ser parados, reanudados, pausados, creados y eliminados.

En este proyecto, Docker se va a utilizar para garantizar que el entorno en el que se ejecute la aplicación durante el desarrollo sea similar al que se utiliza en el entorno de producción

haciendo uso de la abstracción que aportan los contenedores proporcionados por este programa. De esta manera, se puede integrar el uso de la base de datos MySql para poder almacenar los datos con los que se va a trabajar.

Imagen 10. Logotipo de Docker

2.2. Propuesta de solución

En este apartado, se dará la visión del sistema que se quiere desarrollar, además de todas las tecnologías y metodologías que se van a utilizar en el transcurso de progreso de la aplicación.

2.2.1. Visión general del sistema a desarrollar

El objetivo del sistema que se quiere desarrollar es el de ayudar a directores/as, músicos y usuarios/as apasionados/as de la música a obtener información sobre los diferentes géneros de esta que hay para los diferentes tipos de agrupación musical. En esta aplicación, tendrán acceso a enlaces de vídeos de música para que puedan escuchar la pieza buscada y otras funcionalidades más como, por ejemplo, la de comentar la obra para dar su opinión.

Otra finalidad del sistema, es que si el usuario/a que está utilizando la aplicación es director/a, pueda crear esta su propia banda con las fichas de los diferentes músicos que la componen para poder también organizar su asistencia a ensayos y actos. Además de tener la opción de contactar con ellos mediante correo electrónico, para poder compartir las partituras que van a tocar y que las preparen como una de las diferentes opciones que tienen. Así, como también facilitarle la subida de sus propios conciertos o actos y así de esta manera obtener diferentes comentarios de los usuarios/as de la aplicación.

En este desarrollo se propone el uso de las tecnologías y *frameworks* comentados en el anterior apartado, ya que estos nos ayudan a cumplir con los objetivos del sistema que se quieren implementar. Con estas herramientas, se va a crear un sistema descompuesto en dos partes imprescindibles en una estructura, como son el servidor y el cliente web siendo estos comunicados mediante una API REST.

2.3. Requisitos del sistema

Según R. Pressman en [4] podemos entender por requisito: "*Éstas llevarán a la comprensión de cuál será el efecto que tendrá el software en el negocio, qué es lo que quiere el cliente y cómo interactuarán los usuarios finales con el software*".

En este punto y según lo aprendido durante el grado estos requisitos se pueden diferenciar entre funcionales y no funcionales que deben ser cumplidos por nuestro sistema. Ampliando de esta manera la información que ha sido comentada en el apartado anterior.

- Requisitos funcionales

Estos requisitos son los que plasman las funcionalidades generales y restricciones que deben estar en el desarrollo del sistema.

1. El usuario/a que utilice la aplicación puede hacer las funciones básicas de cualquier sistema como es el registro, *login*, modificar y ver su perfil etc.
2. El usuario/ puede buscar los títulos que quisiera por el buscador para obtener la información relacionada con ese título buscado, ya sea vídeos para escuchar la música buscada, historia de esa obra, compositor etc.
3. El usuario/a puede buscar un/a compositor/a para obtener sus datos y trabajos compuestos por él/ella
4. El usuario/a puede añadir comentarios en la obra que haya buscado para agregar sus sensaciones al tocarla, su dificultad, pequeños trucos para montarla mejor...
5. Si el usuario/a, en este caso es un/a director/a de una banda de música, éste/a puede crear su propia banda donde puede añadir fichas de los músicos que componen su banda donde introduce los datos (nombre, edad, año de entrada, instrumento, correo, ...) de sus músicos

6. Si es director/a puede adjuntar una planificación de ensayos con los días que habrá ensayo y también poner los/las músicos asistentes a ese ensayo mediante una lista de asistencia
7. Si es director/a puede compartir las *particellas* correspondientes a cada músico que tenga en su banda para prepararlas para el ensayo y/o concierto, mediante el correo de cada músico
8. Si es director/a puede subir los conciertos que han realizado a la aplicación para así poder obtener comentarios de otros directores/as, músicos y/o usuarios/as que den su opinión sobre el concierto, posibles mejoras entre otras opiniones

- Requisitos no funcionales

En esta sección, se van a tratar aspectos que vienen dados por el contexto o por un/a cliente, para obtener el correcto funcionamiento del sistema a desarrollar.

9. El sistema debe estar preparado para una rápida respuesta a las peticiones realizadas por el usuario/a
10. El contenido utilizado en la web debe poder adaptarse a los diferentes tamaños de ventana que utilicen los diferentes usuarios/as, para así no perder ninguna funcionalidad
11. La interfaz debe ser intuitiva y sencilla de manejar por el usuario/a para así facilitar su utilización, como también debe ser adaptada a nuestros usuarios/as con diversidades funcionales
12. El sistema debe garantizar cierta seguridad para proteger los datos de los usuarios/as

2.4. Historias de usuario

Según Cohn, en [2], las historias de usuarios aportan una serie de información sobre la funcionalidad que debe estar incorporada en el sistema que se está desarrollando y cuya implementación da valor a lo que el cliente quiere.

- Estructura:
    - Nombre descriptivo de la historia
    - La descripción de la funcionalidad que debe cumplir, siguiendo el formato de cómo(rol) querer(algo) para(objetivo). Por ejemplo, como usuario, quiero poder

buscar un autor para saber información sobre él, consiguiendo de esta manera el objetivo de lo que se quiere realizar

- Criterios de validación y verificación utilizados para dar las historias de usuario por terminadas y aceptadas por el cliente consiguiendo así el funcionamiento esperado. Una historia de usuario puede ser considerada hecha sólo si cumple con el criterio de aceptación específico satisfaciendo los diferentes requisitos funcionales y no funcionales correspondientes y el criterio de hecho definido para el sprint, este último afecta a todas las historias de usuario
- Para la realización de las tareas se decreta un orden siguiendo el método MoSCoW. Este es una técnica de priorización de requisitos donde se da más valor a aquellos que aportan mayor valor al sistema. Utiliza una escala con un significado intrínseco, de esta manera el responsable se encarga de asignar la prioridad. Estas son:
  (a) *Must* (M): El requisito que esté considerado como M, debe estar implementado en la versión final para considerar el desarrollo como éxito.
  (b) *Should* (S): El requisito que esté considerado como S, debería de estar en la versión final, pero si llegado a tal punto y si fuera necesario, se podría prescindir de este.
  (c) *Could* (C): El requisito que esté considerado como C, indica que es un requisito no necesario en la versión final, aunque sería deseable tenerlo en esta si hay posibilidades tanto de coste como de tiempo.
  (d) *Would not have* (W): El requisito que esté considerado como W, aporta que este está descartado, pero esto no indica que en un futuro en el que la aplicación esté en marcha se vuelva a tener en cuenta y se pueda implementar en el sistema.

Tras la introducción de historia de usuario, se va a presentar las diferentes historias de usuarios que se quieren realizar en el desarrollo de la aplicación.

Tabla 1. Desarrollo de las historias de usuario

Identificador (ID) de la historia	Rol	Característica / Funcionalidad	Razón / Resultado	Estimación	Prioridad
1 - Registro	Como un usuario,	quiero registrarme en la aplicación	para poder crear mi propia banda y tener los datos de los músicos.	5	M
2 - Login	Como un usuario,	quiero loguearme	para acceder a los datos de mi banda y músicos.	5	M
3 - Crear banda	Como un director	quiero crear una banda de música	para poder introducir los datos de mis músicos	3	M
4 - Buscar composición	Como un usuario	quiero buscar cualquier composición por título	para obtener información sobre ella	3	M
5 - Crear composición	Como un usuario	quiero crear una composición que no esté en la aplicación	para poder compartir la información con los demás usuarios	5	M
6 - Buscar autor	Como un usuario	quiero buscar cualquier autor por su nombre	para obtener la información de este	3	M
7 - Crear autor	Como un usuario	quiero introducir/crear un autor que no esté en la aplicación	para compartir la información con otros usuarios	5	M
8 - Añadir músico a banda	Como un director	quiero añadir músicos a mi banda de música	para tener los datos guardados	5	M

9 - Listado de músicos	Como un director	quiero tener un listado de mis músicos	para ver los datos de cada uno	3	M
10 - Listado de composiciones	Como un usuario	quiero ver el listado de todas las composiciones	para tener la información de cada una de ellas	3	M
11 - Modificar datos de perfil	Como un usuario	quiero modificar mis datos	para actualizar mi perfil	3	C
12 - Borrar datos de perfil	Como un usuario	quiero darme de baja	para borrar mi cuenta de la aplicación	5	C
13 - Borrar autor	Como un usuario	quiero borrar la información de un autor	para actualizar los autores disponibles	3	W
14 - Borrar composición	Como un usuario	quiero borrar la información de una composición	para actualizar el listado de composiciones	3	W
15 - Añadir un listado de ensayos	Como un director	quiero apuntar los músicos que van a un ensayo	para hacer recuento de ensayos	5	C
16 - Subir vídeos	Como un director	quiero subir vídeos de mis conciertos	para obtener las opiniones de otros usuarios	8	W
17 - Comentar las composiciones	Como un usuario	quiero comentar en las composiciones	para opinar sobre su dificultad, trucos que puede tener para tocar etc	8	C

18 - Planificación ensayos	Como un director	quiero hacer una planificación de ensayos	para organizar el trabajo a hacer en los ensayos	8	W
19 - Modificar un autor	Como un usuario	quiero modificar el perfil de un autor	para actualizar su información	3	C
20 - Modificar una composición	Como un usuario	quiero modificar los datos de una composición	para actualizar la información que contiene	3	C

Para la estimación de tiempos de cada historia de usuario se ha utilizado la técnica de *planning poker*, la cual consiste en evaluar la estimación mediante la serie de Fibonacci (1,2,3,5,8,13,20,40,100 etc.). Todos los trabajadores votan con una estimación y explican sus motivos de por qué debe tener ese valor y luego vuelven a votar hasta que se llega a un consenso.

En este caso, se ha procurado que todas las estimaciones estén entre 1 y 8 de dicha serie comentada.

2.5. Planificación inicial de tareas

En esta sección, se va a organizar la planificación de las diferentes tareas a realizar. Como se ha decidido seguir una metodología incremental se ha atendido a la estimación de puntos de historia que se ha realizado para hacer una selección de tareas para cada iteración.

Tras tener esta información en cuenta, se ha obtenido una estimación inicial del tiempo que se va a necesitar para hacer cada una de las iteraciones. Este tiempo debe estar entorno al tiempo calculado en realizar un TFG, el cuál es aproximadamente de unas 300 horas.

Se cuentan los puntos de estimación de las diferentes historias de usuarios y se obtiene un total de 89 puntos. Se pone una fecha de entrega de cuatro meses (unas 16 semanas aproximadamente).

Después de tener estos datos presentes, se establecen cuatro sprints que según Bahit en [1] se desarrollarán cada uno en cuatro semanas hasta finalizar todas las historias de usuario en el

plazo fijado. En cada sprint, se trabajarán 75 horas, para llegar a las 300 horas comentadas anteriormente.

Tras utilizar el método MoSCoW para organizar las historias de usuario mediante prioridad, las tareas que se realizarán antes serán las que tengan como prioridad M y las siguientes se realizarán con el mismo orden. Y también se tendrán en cuenta aquellas historias que dependan de otras, por ejemplo, no puedes crear una banda si no te has registrado o logueado previamente.

Para realizar la estimación de velocidad con la que se van a desarrollar las diferentes iteraciones, se va a utilizar una estimación optimista en la que los puntos de historia (véase Tabla 1) se utilizarán como la velocidad del proyecto. Esto se desarrolla así ya que no tenemos ningún dato histórico en el cual basarnos para realizar la estimación de la velocidad que se tiene que llevar para desarrollar las diferentes tareas. Por tanto, se tienen 89 puntos en 4 sprint, se realizarán 23 puntos de historia aproximadamente por sprint, quedando de la siguiente manera:

- Sprint 1: 24 puntos de historia

Tabla 2. Tareas a realizar en el primer sprint

Identificador (ID) de la historia	Tareas	Estimación en horas	Estimación en PH	Prioridad
1 - Registro	Creación del proyecto servidor y proyecto cliente	1	5	M
	Crear las diferentes clases y configuración de la aplicación	8		
	Crear y configurar los diferentes controladores	5		
	Crear la vista registro y configurar el envío de datos	5		
2 - Login	Realizar un estudio de mecanismos de seguridad en API REST	5	5	M
	Implementación de la función de logueo en servidor	5		
	Configurar el controlador de login y errores	5		
	Crear la vista de login y configurar el envío de datos	5		
3 - Crear banda	Crear y configurar la función en el servidor	4	3	M
	Crear vista y configuración de envío de datos	4		
	Comprobación de los datos recibidos	1		
4 - Buscar composición	Crear y configurar la función en el servidor	4	3	M
	Crear vista y configuración de envío de datos	4		
	Devolver los datos buscados en el cliente	1		

5 - Crear composición	Crear y configurar la función en el servidor Crear vista y configuración de envío de datos Comprobación de los datos recibidos	4 4 1	5	M
6 - Buscar autor	Crear y configurar la función en el servidor Crear vista y configuración de envío de datos Devolver los datos buscados en el cliente	4 4 1	3	M

Las historias de usuarios que no están desarrolladas en tareas (véanse Tablas 3,4,5), tienen la misma estructura todas, ya que es crear la vista correspondiente en el cliente y enviar los datos al servidor y crear la función y configuración correspondiente en este.

- Sprint 2: 22 puntos de historia

Tabla 3. Tareas a realizar en el segundo sprint

7 - Crear autor	Crear y configurar la función en el servidor Crear vista y configuración de envío de datos Comprobación de los datos recibidos	4 4 1	5	M
8 - Añadir músico a banda	Crear y configurar la función en el servidor para recibir los datos Crear y configurar la función para buscar la banda y director Crear vista y configuración de envío Mandar al cliente una respuesta	5 3 5 2	5	M
9 - Listado de músicos	Crear y configurar la función en el servidor para recibir los datos Crear vista y configuración de envío Mandar la respuesta al cliente	4 4 1	3	M
10 - Listado de composiciones	Crear y configurar la función en el servidor para recibir los datos Crear vista y configuración de envío Mandar la respuesta al cliente	4 4 1	3	M

19 - Modificar un autor	Como un usuario	quiero modificar el perfil de un autor	para actualizar su información	3	C
20 - Modificar una composición	Como un usuario	quiero modificar los datos de una composición	para actualizar la información que contiene	3	C

- Sprint 3: 21 puntos de historia

Tabla 4. Tareas a realizar en el tercer sprint

11 - Modificar datos de perfil	Como un usuario	quiero modificar mis datos	para actualizar mi perfil	3	C
12 - Borrar datos de perfil	Como un usuario	quiero darme de baja	para borrar mi cuenta de la aplicación	5	C
15 - Añadir un listado de ensayos	Como un director	quiero apuntar los músicos que van a un ensayo	para hacer recuento de ensayos	5	C
17 - Comentar las composiciones	Como un usuario	quiero comentar en las composiciones	para opinar sobre su dificultad, trucos que puede tener para tocar etc	8	C

- Sprint 4: 20 puntos de historia

Tabla 5. Tareas a realizar en el cuarto sprint

13 - Borrar autor	Como un usuario	quiero borrar la información de un autor	para actualizar los autores disponibles	3	W
14 - Borrar composición	Como un usuario	quiero borrar la información de una composición	para actualizar el listado de composiciones	3	W
16 - Subir vídeos	Como un director	quiero subir vídeos de mis conciertos	para obtener las opiniones de otros usuarios	8	W

18 - Planificación ensayos	Como un director	quiero hacer una planificación de ensayos	para organizar el trabajo a hacer en los ensayos	8	W

2.6. Planificación inicial de costes

Si se procede a explicar el coste inicial que va a ser necesario para realizar dicho proyecto, se pueden observar a continuación los detalles de los recursos y su estimación en coste, como también los recursos humanos, hardware y software.

Para realizar el cálculo del coste de los recursos humanos se va a tener en cuenta las horas que se van a dedicar al proyecto y el precio por hora que tiene cada trabajador.

Un programador informático tiene estimado un sueldo anual [7]de 18700€, por lo tanto, lleva a calcular un sueldo mensual de 1558€. Una jornada laboral está formada por 40 horas semanales, llegando así a 160 horas mensuales, por lo que se puede deducir que el precio por hora de un programador informático es de 10€/hora.

Un analista informático posee una estimación anual [8]de 27560€, por lo que, 2297€ es su sueldo mensual. La jornada laboral es la misma que se ha explicado en el programador, así se deduce que este tiene un precio por hora de 14€.

En la próxima tabla (véase Tabla 6), se explican las diferentes tareas que se llevan a cabo en el proyecto con sus respectivos costes. Se hace un desglose de los distintos trabajadores que hay en un proyecto.

[7] https://es.indeed.com/salaries/programador-junior-Salaries
[8] https://es.indeed.com/salaries/analista-programador-Salaries

Tabla 6. Estimación de costes en Recursos Humanos.

Tarea	Horas necesarias	Trabajadores	Coste
Búsqueda de información y estudios de aplicaciones similares	20	Analista	280 €
	10	Programador	100 €
Estudio y aprendizaje de las tecnologías necesarias	15	Analista	210 €
	15	Programador	150 €
Sprint 1 (24 puntos de historia)	45	Analista	630 €
	30	Programador	300 €
Sprint 2 (22 puntos de historia)	45	Analista	630 €
	30	Programador	300 €
Sprint 3 (21 puntos de historia)	45	Analista	630 €
	30	Programador	300 €
Sprint 4 (20 puntos de historia)	45	Analista	630 €
	30	Programador	300 €
Redacción de manual y memoria del proyecto	10	Analista	140 €
	10	Programador	100 €
Total	225	Analista	3.150 €
	155	Programador	1.550 €

La estimación de coste total en recursos humanos es de 4700€.

A este coste, se tienen que añadir los de hardware y software utilizados para el desarrollo del proyecto. Los costes de hardware son aquellos que están relacionados con la maquinaria con la que se trabaja (ordenador utilizado, discos duros externos, conexiones a internet, etc.) y

los de software son los relacionados con las diferentes herramientas que utilizamos para el trabajo (sistema operativo, programas de diseño, de implementación etc.).

Se pasa a calcular el coste de hardware (véase Tabla 7).

Tabla 7. Estimación de costes en Hardware

Tipo	Modelo	Coste
PORTATIL LENOVO	G50-80 I3 4005U 15.6" 4GB / 1TB	450 €
Fibra 300 MB : Campiña digital		100 € / 4 meses
Alta en conexión a internet		0 €
Total		550 €

Se procede a evaluar los costes de software (véase tabla 8).

Tabla 8. Estimación de costes en Software

Licencia	Coste
Windows 10 Pro	230 €
Visual Studio Code	0 €
Eclipse	0 €
Docker	0 €
Visual Paradigm (Estándar)	295 €
Postman	0 €
Google Chrome	0 €
NodeJS	0 €
Total	525 €

En software, la mayoría de las herramientas utilizadas son gratuitas ya que son programas fáciles de obtener mediante sus correspondientes enlaces web (esto es explicado a final del documento en el punto 7.1. Manual del Entorno de desarrollo).

Tabla 9. Estimación de costes eléctricos

Gasto	Coste
Luz	40€/4 meses

Tras la realización de los diferentes costes, recopilamos todos los importes y se obtiene una estimación inicial con un total de 5815€ (véase Tabla 10).

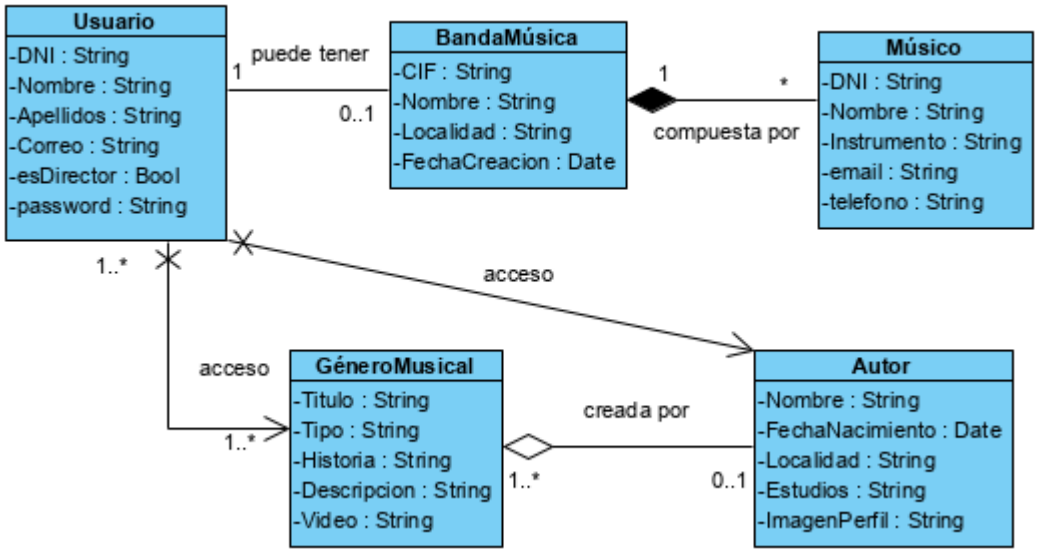
Tabla 10. Estimación de costes totales inicial

Costes	Total
Recursos Humanos	4.700 €
Hardware	550 €
Software	525 €
Eléctricos	40 €
Total	5.815 €

2.7. Modelo del dominio

En este punto, se desarrolla una visión de las diferentes clases que serán utilizadas para la implementación del servidor y la relación que tienen entre cada una de ellas.

Diagrama 1. Modelo del dominio.



Como se puede observar, han sido cinco clases las utilizadas para realizar la implementación del servidor, dónde se adjunta el nombre, los atributos y las relaciones de cada una de ellas. En el punto de diseño, se va a presentar un diagrama de clases más desarrollado, donde se podrá encontrar toda la información.

Para comenzar, se va a explicar la clase Usuario (véase Diagrama 1):

En esta clase, se encuentran los atributos típicos de un formulario de registro para obtener los datos del usuario y contiene estas relaciones:

- Usuario – BandaMusica: Si un usuario es director, tiene posibilidad de realizar la creación de su propia banda, para así poder desarrollar una lista de los músicos que la componen y tener una ficha de información de ellos
- Usuario – GeneroMusical: Esta relación, da acceso a cualquier usuario a los diferentes tipos de GeneroMusical que están disponibles en la aplicación
- Usuario – Autor: Esta relación, da acceso a cualquier usuario a los diferentes autores que están disponibles en la aplicación

Se continúa con la clase BandaMusica (véase Diagrama 1):

Para esta clase, los datos que se le van a pedir al usuario son los identificativos de una banda de música como son el CIF, nombre, localidad y fecha de creación. Contiene las siguientes relaciones:

- BandaMusica – Musico: Esta relación proporciona los músicos que componen la banda de música creada por el usuario
- BandaMusica – Usuario: Ha sido desarrollada en la clase anterior

Se desarrolla la clase Musico (véase Diagrama 1):

Esta clase contiene los datos necesarios para que el director pueda tener la ficha de cada músico de su banda de música, de la misma forma que puede ponerse en contacto con este mediante el teléfono o el correo electrónico. La relación que contiene esta clase, ha sido explicada anteriormente.

Se procede a explicar la clase GeneroMusical (véase Diagrama 1):

Aquí se pueden obtener los datos de las diferentes composiciones que están en nuestra aplicación. Las relaciones que tiene son:

- Usuario-GeneroMusical: Se ha comentado en la explicación de la clase Usuario
- GeneroMusical-Autor: Esta relación proporciona el autor que ha compuesto dicha composición

Para finalizar, se prosigue a comentar la clase Autor (véase Diagrama 1):

Esta clase va a proporcionar la información de los diferentes autores que se pueden encontrar en la aplicación. Las relaciones que contiene han sido desarrolladas en las clases comentadas antes.

2.8. Casos de Uso

Un diagrama de casos de usos [4] es utilizado para representar la forma en la que un cliente (Actor) trabaja con el sistema, además de expresar cómo interactúan los elementos que están en el diagrama (operaciones o casos de uso).

Está compuesto por elementos:

- Sistema

El sistema es representado mediante un rectángulo (en el Diagrama 2 expuesto más adelante se puede contemplar el de este proyecto que se va a desarrollar) que engloba los diferentes casos de uso que son trabajados en este.

- Actor

Los actores son los que existen fuera del sistema, que desarrollan un rol específico. Este rol hace que represente la labor que realiza frente al sistema.

Imagen 11. Representación de actor en diagrama de casos de uso

- Casos de Uso

Se representan mediante óvalos. Es una operación o tarea específica que se realiza tras ser ordenada por algún actor u otro caso de uso.

Imagen 12. Representación de caso de uso en diagrama de casos de uso

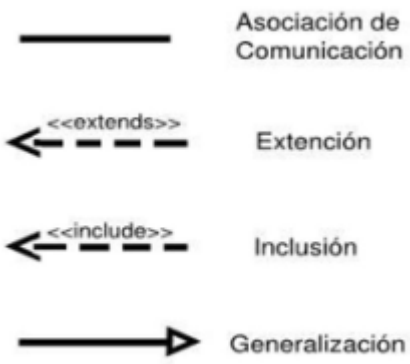
- Relaciones

Para representar las relaciones entre un actor y un caso de uso se utiliza una línea simple. Esta línea simple se le denomina asociación, ya que es la más básica.

Las relaciones entre casos de usos son representadas mediante flechas etiquetadas las cuales pueden ser:

- La relación "*include*" la cual indica que un caso de uso necesita de otro para poder realizar su tarea
- La relación "*extend*" que indica las opciones alternativas que tiene un caso de uso
- La relación de generalización es una de las más utilizadas, está orientada exclusivamente a casos de uso. Cumple con una doble función dependiendo de su estereotipo, puede ser de uso (<<*uses*>>) o de Herencia (<<*extends*>>)

Imagen 13. Representación de relaciones en diagrama de casos de uso



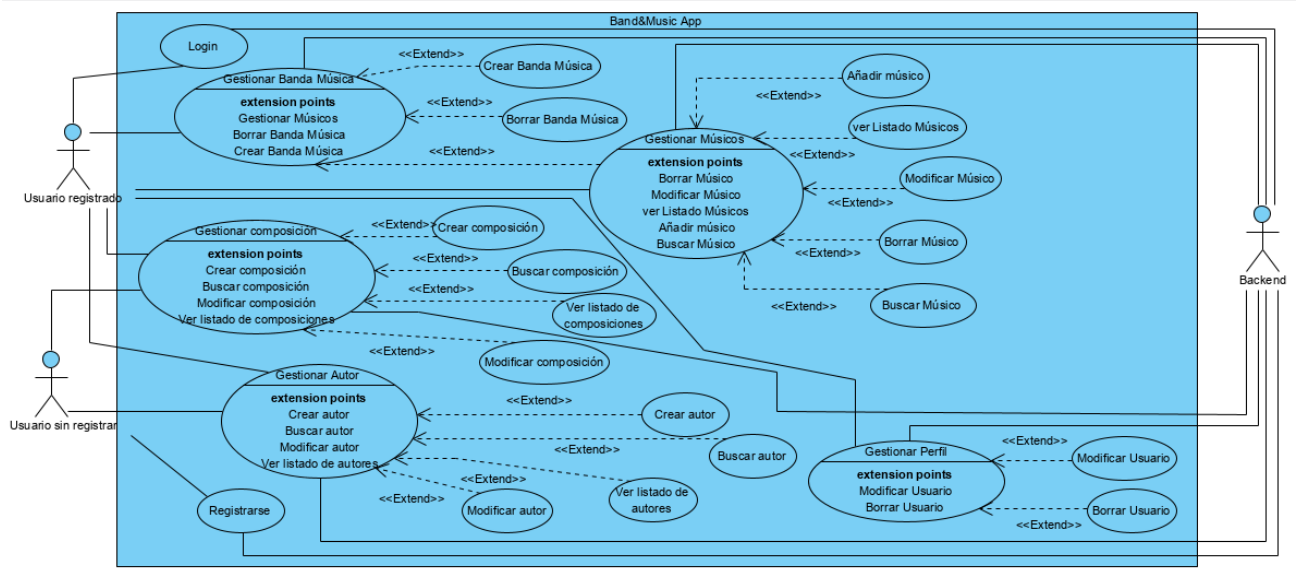
Tras la explicación del diagrama de casos de usos se procede a realizar los diferentes diagramas de casos de uso del proyecto:

En este proyecto se pueden encontrar tres actores:

- Backend: Encargado de recibir y realizar las operaciones de los usuarios
- Usuario sin registrar: Podrá registrarse, crear y consultar composiciones y autores

- Usuario registrado: Podrá loguearse, crear y consultar composiciones y autores, crear una banda de música y añadir músicos a su banda de música

Diagrama 2. Casos de uso



A continuación, se van a mostrar los diferentes casos de usos de forma detallada para así facilitar la posterior implementación de las diferentes funcionalidades que se pueden encontrar en el sistema (véase Diagrama 2).

Tabla 11. Descripción Caso de uso Registrarse

Caso de Uso	Registrarse
Actor	Usuario sin registrar
Descripción	Función para que se pueda registrar un usuario
Escenario principal	1. El usuario pulsa el botón de registro 2. El sistema recibe la petición y muestra el formulario de registro 3. El usuario introduce los datos y envia la confirmación 4. El sistema recibe y comprueba los datos 5. El sistema muestra la vista de login al usuario
Escenario alternativo	4: El sistema recibe y comprueba los datos 4.1: Si el sistema comprueba que hay datos erróneos 4.2: El sistema vuelve a mostrar el formulario de registro y muestra el error

Tabla 12. Descripción Caso de uso Login

Caso de Uso	Login
Actor	Usuario registrado
Descripción	El usuario quiere loguearse e introduce sus datos
Escenario principal	1. El usuario pulsa el botón de login 2. El sistema recibe la petición y muestra el formulario de login 3. El usuario introduce los datos y envia los datos 4. El sistema recibe y comprueba los datos 5. El sistema muestra la vista de inicio
Escenario alternativo	4: El sistema recibe y comprueba los datos 4.1: Si el sistema comprueba que hay datos erróneos 4.2: El sistema vuelve a mostrar el formulario de login y muestra el error

Tabla 13. Descripción Caso de uso Modificar Usuario

Caso de Uso	Modificar Usuario
Actor	Usuario registrado
Descripción	El usuario quiere modifcar sus datos
Escenario principal	1. El usuario accede a su perfil 2. El usuario pulsa el botón de modificar 3. El sistema recibe la petición 4. El sistema muestra el formulario con los datos del usuario 5. El usuario introduce los datos a modificar y los envía 6. El sistema recibe los datos, los comprueba y modifica el perfil
Escenario alternativo	6. El sistema comprueba datos erróneos 6.1. El sistema muestra los errores en el formulario y vuelve a paso 4

Tabla 14. Descripción Caso de uso Borrar Usuario

Caso de Uso	Borrar Usuario
Actor	Usuario registrado
Descripción	El usuario quiere eliminar su cuenta
Escenario principal	1. El usuario accede a su perfil 2. El usuario pulsa el botón de borrar 3. El sistema recibe la petición 4. El sistema muestra un mensaje de advertencia 5. El usuario confirma el borrado 6. El sistema borra al usuario y manda al usuario a la vista de inicio
Escenario alternativo	5. El usuario cancela el borrado 5.1. El sistema recibe la cancelación y manda al usuario a su perfil

Tabla 15. Descripción Caso de uso Crear Banda

Caso de Uso	Crear Banda
Actor	Usuario registrado
Descripción	El usuario quiere crear una banda de música
Escenario principal	1. El usuario pulsa el botón de creación de banda 2. El sistema recibe la petición y muestra el formulario de creación 3. El usuario introduce los datos y envia los datos 4. El sistema recibe y comprueba los datos 5. El sistema muestra la vista de la información de banda

Escenario alternativo	4A: El sistema recibe y comprueba los datos 4.1: Si el sistema comprueba que hay datos erróneos 4.2: El sistema vuelve a mostrar el formulario de creación y muestra el error 4B: El sistema recibe y comprueba los datos 4.1: El sistema comprueba que el usuario no es director 4.2: El sistema manda al usuario a la vista inicio e informa de que no es director al usuario

Tabla 16. Descripción Caso de uso Borrar Banda

Caso de Uso	Borrar Banda
Actor	Usuario registrado
Descripción	El usuario quiere eliminar la banda que tiene creada
Escenario principal	1. El usuario accede a la vista de banda 2. El usuario pulsa el botón de borrar banda 3. El sistema recibe la petición 4. El sistema muestra un mensaje de advertencia 5. El usuario confirma el borrado 6. El sistema borra la banda y manda al usuario a su perfil
Escenario alternativo	5. El usuario cancela el borrado 5.1. El sistema recibe la cancelación y manda al usuario a su perfil

Tabla 17. Descripción Caso de uso Añadir Músico

Caso de Uso	Añadir músico
Actor	Usuario registrado
Descripción	El usuario quiere añadir un nuevo músico a su banda
Escenario principal	1. El usuario pulsa el botón de añadir músico 2. El sistema recibe la petición y muestra el formulario de creación y añadir músico 3. El usuario introduce los datos y envia los datos 4. El sistema recibe y comprueba los datos 5. El sistema muestra la vista de la información de banda
Escenario alternativo	4A: El sistema recibe y comprueba los datos 4.1: Si el sistema comprueba que hay datos erróneos 4.2: El sistema vuelve a mostrar el formulario de creación y muestra el error 4B: El sistema recibe y comprueba los datos 4.1: El sistema comprueba que no es la banda del usuario en la que quiere añadir un nuevo músico 4.2: El sistema manda al usuario al formulario de creación y añadir músico

Tabla 18. Descripción Caso de uso Ver listado de músicos

Caso de Uso	Ver listado de músicos
Actor	Usuario registrado
Descripción	El usuario quiere ver el listado de músicos de su banda
Escenario principal	1. El usuario pulsa el botón de ver músicos 2. El sistema recibe la petición 3. El sistema muestra el listado de los músicos
Escenario alternativo	

Tabla 19. Descripción Caso de uso Modificar músico

Caso de Uso	Modificar Músico
Actor	Usuario registrado
Descripción	El usuario quiere modifcar los datos de un músico inscrito en su banda
Escenario principal	1. El usuario accede a la vista de músico 2. El usuario pulsa el botón de modificar músico 3. El sistema recibe la petición 4. El sistema muestra el formulario con los datos del músico 5. El usuario introduce los datos a modificar y los envía 6. El sistema recibe los datos, los comprueba y modifica al músico
Escenario alternativo	6. El sistema comprueba datos erróneos 6.1. El sistema muestra los errores en el formulario y vuelve a paso 4

Tabla 20. Descripción Caso de uso Borrar músico

Caso de Uso	Borrar Músico
Actor	Usuario registrado
Descripción	El usuario quiere eliminar el músico que hay inscrito en su banda
Escenario principal	1. El usuario accede a la vista de músico 2. El usuario pulsa el botón de borrar músico 3. El sistema recibe la petición 4. El sistema muestra un mensaje de advertencia 5. El usuario confirma el borrado 6. El sistema borra al músico y manda al usuario al listado de músicos
Escenario alternativo	5. El usuario cancela el borrado 5.1. El sistema recibe la cancelación y manda al usuario al listado de músicos

Tabla 21. Descripción Caso de uso Buscar músico

Caso de Uso	Buscar músico
Actor	Usuario registrado
Descripción	El usuario quiere encontrar a un músico
Escenario principal	1. El usuario pulsa el botón de buscar músico 2. El sistema muestra formulario para introducir el nombre 3. El usuario introduce el nombre a buscar 4. El sistema recibe el nombre y hace la búsqueda 5. El sistema devuelve un listado con los músicos encontrados con el nombre a buscar
Escenario alternativo	5. El sistema devuelve un listado vacío porque no hay músicos con ese nombre

Tabla 22. Descripción Caso de uso Crear composición

Caso de Uso	Crear composición
Actor	Usuario registrado y usuario sin registrar
Descripción	El usuario quiere crear una composición que no existe en la aplicación
Escenario principal	1. El usuario pulsa el botón de crear composición 2. El sistema recibe la petición y muestra el formulario de creación 3. El usuario introduce y envía los datos 4. El sistema recibe y comprueba los datos 5. El sistema manda al usuario a la vista de inicio
Escenario alternativo	4. El sistema recibe y comprueba los datos 4.1: El sistema comprueba que hay datos erróneos 4.2: El sistema vuelve a mostrar el formulario de creación y muestra el error

Tabla 23. Descripción Caso de uso Buscar composición

Caso de Uso	Buscar composición
Actor	Usuario registrado y usuario sin registrar
Descripción	El usuario quiere buscar una composición para obtener la información
Escenario principal	1. El usuario introduce los datos y pulsa el botón de búsqueda 2. El sistema recibe la petición y realiza la búsqueda 3. El sistema muestra la información de la composición

Escenario alternativo	2. El sistema recibe la petición y realiza la búsqueda 2.1: El sistema manda un mensaje al usuario de que no existe dicha composición 2.2: El sistema manda al usuario al formulario de creación de composición

Tabla 24. Descripción Caso de uso Modificar Composición

Caso de Uso	Modificar Composición
Actor	Usuario registrado y usuario sin registrar
Descripción	El usuario quiere modifcar la información de una composición
Escenario principal	1. El usuario accede a la vista de la composición 2. El usuario pulsa el botón de modificar 3. El sistema recibe la petición 4. El sistema muestra el formulario con los datos de la composición 5. El usuario introduce los datos a modificar y los envía 6. El sistema recibe los datos, los comprueba y modifica la composición
Escenario alternativo	6. El sistema comprueba datos erróneos 6.1. El sistema muestra los errores en el formulario y vuelve a paso 4

Tabla 25. Descripción Caso de uso Buscar autor

Caso de Uso	Buscar autor
Actor	Usuario registrado y usuario sin registrar
Descripción	El usuario quiere buscar un autor para obtener la información
Escenario principal	1. El usuario introduce los datos y pulsa el botón de búsqueda 2. El sistema recibe la petición y realiza la búsqueda 3. El sistema muestra la información del autor

Escenario alternativo	2. El sistema recibe la petición y realiza la búsqueda 2.1: El sistema manda un mensaje al usuario de que no existe dicho autor 2.2: El sistema manda al usuario al formulario de creación de autor

Tabla 26. Descripción Caso de uso Crear autor

Caso de Uso	Crear autor
Actor	Usuario registrado y usuario sin registrar
Descripción	El usuario quiere crear un autor que no existe en la aplicación
Escenario principal	1. El usuario pulsa el botón de crear autor 2. El sistema recibe la petición y muestra el formulario de creación 3. El usuario introduce y envía los datos 4. El sistema recibe y comprueba los datos 5. El sistema manda al usuario a la vista de inicio
Escenario alternativo	4. El sistema recibe y comprueba los datos 4.1: El sistema comprueba que hay datos erróneos 4.2: El sistema vuelve a mostrar el formulario de creación y muestra el error

Tabla 27. Descripción Caso de uso Modificar Autor

Caso de Uso	Modificar Autor
Actor	Usuario registrado y usuario sin registrar
Descripción	El usuario quiere modifcar la información de un autor
Escenario principal	1. El usuario accede a la vista del autor 2. El usuario pulsa el botón de modificar 3. El sistema recibe la petición 4. El sistema muestra el formulario con los datos del autor 5. El usuario introduce los datos a modificar y los envía 6. El sistema recibe los datos, los comprueba y modifica el autor
Escenario alternativo	6. El sistema comprueba datos erróneos 6.1. El sistema muestra los errores en el formulario y vuelve a paso 4

Tabla 28. Descripción Caso de uso Ver listado de composiciones

Caso de Uso	Ver listado de composiciones
Actor	Usuario registrado y usuario sin registrar
Descripción	El usuario quiere ver todas las composiciones de la aplicación
Escenario principal	1. El usuario accede a la vista de inicio de la aplicación 2. El sistema muestra los datos al usuario de todas las composiciones que hay
Escenario alternativo	

Tabla 29. Descripción Caso de uso Ver listado de autores

Caso de Uso	Ver listado de autores
Actor	Usuario registrado y usuario sin registrar
Descripción	El usuario quiere ver todos los autores de la aplicación
Escenario principal	1. El usuario pulsa el botón de listado de autores en la página de inicio 2. El sistema recibe la petición y carga todos los autores 3. El sistema muestra el listado de todos los autores
Escenario alternativo	5. El sistema devuelve un listado vacío porque no hay autores

3. Diseño

Para hablar del diseño de este proyecto, primero hay que detallar los diferentes aspectos relacionados con este y que lo conforman.

En primer lugar, se va a exponer el diagrama de entidad-relación que se ha obtenido mediante el modelo de dominio (véase Diagrama 1).

En segundo lugar, se va a explicar los diferentes diagramas de clases que tiene el proyecto, tanto el diagrama del servidor como el diagrama del cliente. Se comentan la información más relevante de cada diagrama, como las decisiones tomadas, estructura etc.

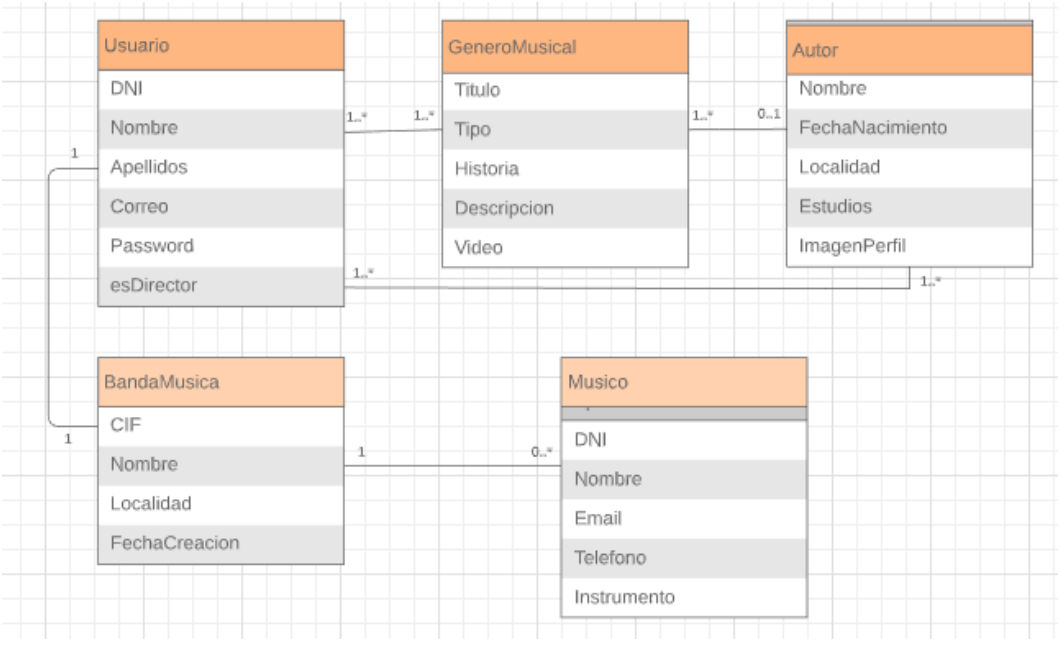
En tercer lugar, se van a encontrar los diferentes diagramas de secuencia de las historias de usuarios que han sido realizadas anteriormente, lo que se quiere conseguir es mostrar el funcionamiento interno del sistema.

Y, para terminar, se va a desarrollar un *storyboard* para mostrar las diferentes pantallas y las interacciones que se puede encontrar un cliente cuando haga uso de la aplicación.

3.1. Diagrama Entidad-Relación

El diagrama de Entidad-Relación, se ha desarrollado a partir del modelo de dominio (véase Diagrama 1).

Diagrama 3. Entidad – Relación



Con este diagrama, se obtiene que un usuario tiene acceso mediante sus relaciones a la información de las entidades BandaMusica, GeneroMusical y Autor (estas dos últimas son públicas para todos). Un usuario tiene acceso a BandaMusica si es director, en otro caso, no tiene acceso.

La entidad BandaMusica tiene acceso a la entidad músico porque son los que la componen y la entidad Musico guarda el id de la banda a la que pertenece.

La entidad GeneroMusical está relacionada con la entidad Autor porque guarda el id del autor que ha compuesto dicho GeneroMusical. En esta relación puede haber composiciones que no tengan autor, y en ese caso, se crearía un autor anónimo para asociarlo a dicha composición.

Las multiplicidades están desarrolladas en cada relación, los atributos de fechas son LocalDate, por tanto, solamente guarda la fecha de manera que guarda el año, el mes y el día, no tiene en cuenta las horas.

Todas las tablas, id's y relaciones son generadas automáticamente por el *Object-Relational Mapping* (ORM) que se ha usado en el desarrollo del proyecto a partir de las clases que se encuentran en el modelo.

El modelo propuesto está en la tercera forma normal ya que, está en segunda forma normal (con esto, se cumple que también está en primera forma normal) y los atributos que no son clave primaria no son dependientes transitivamente de esta

3.2. Diagrama de Clases

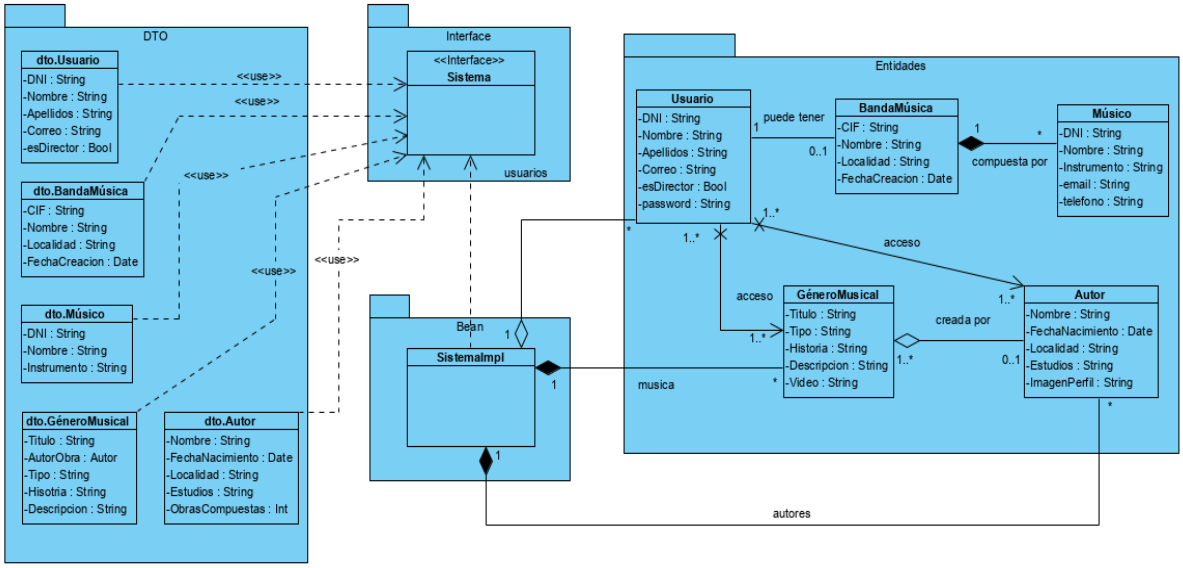
En este apartado, se van a exponer los diagramas de clases que dan información más detalladamente de la aplicación web. El apartado se divide en dos, pues la aplicación está desarrollada en dos partes diferentes (servidor y cliente).

En el lado del servidor, se realiza toda la lógica de negocio, modificaciones y consultas sobre los datos con los que se trabaja en la aplicación. Toda la funcionalidad implementada en el servidor está accesible mediante una API REST.

En el lado del cliente, se realiza la comunicación entre el usuario y el servidor para mostrar los datos y/o las modificaciones que el usuario quiera realizar. El cliente y el servidor están conectados mediante la API REST.

3.2.1. Diagrama de clases del sistema servidor

Diagrama 4. Diagrama de clases del servidor

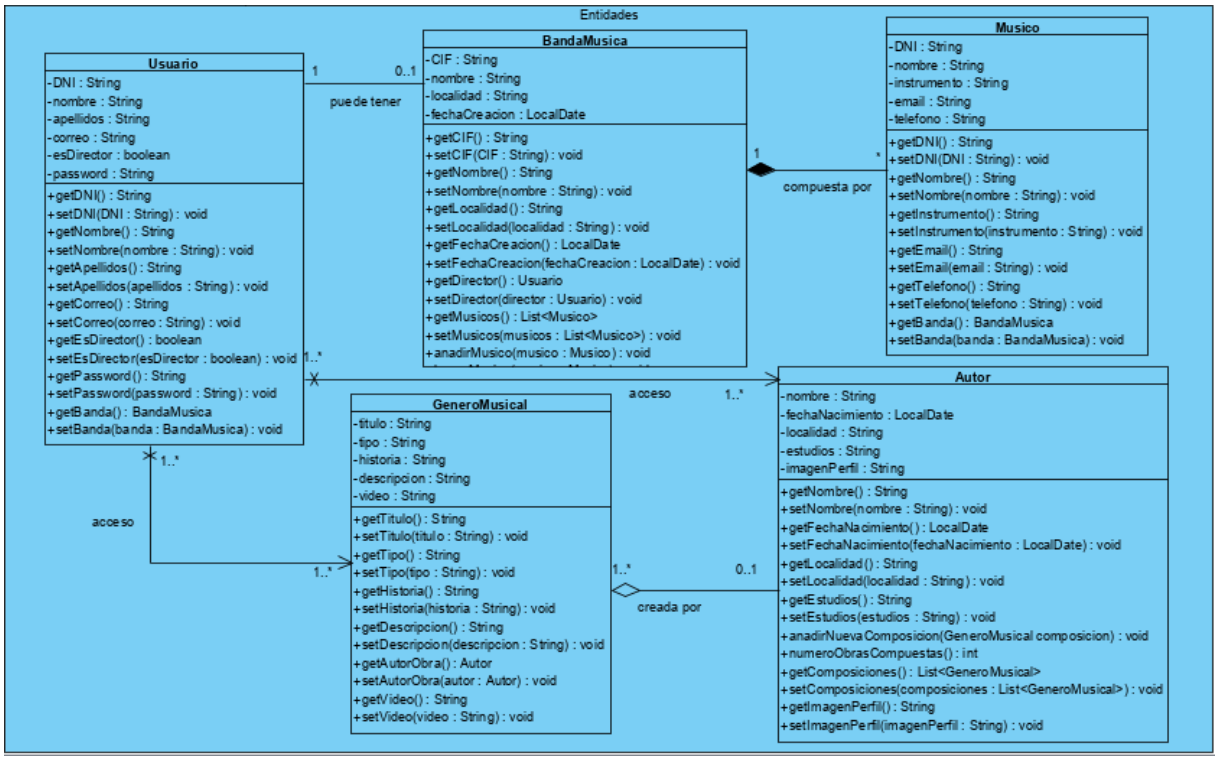


Este diagrama ayuda a facilitar la lectura del diagrama de clases y que se puedan ver las diferentes relaciones entre los paquetes y clases. Ahora para continuar, se va a dividir la sección en los diferentes paquetes que lo conforman, donde se podrán ver las diferentes clases con todos sus atributos y métodos.

3.2.1.1. Paquete Entidades

En el siguiente paquete, se pueden encontrar las diferentes entidades y relaciones con las que el servidor trabaja en la aplicación. Este paquete viene derivado del modelo de dominio (véase Diagrama 1) ya que está compuesto por las mismas clases, pero en este caso también vienen dadas las funcionalidades de cada una. En cada clase no aparece, pero esta tiene también su constructor parametrizado y su constructor por defecto.

Paquete 1. Diagrama de entidades



3.2.1.2. Paquete Beans

En el próximo paquete, se encuentra la clase SistemaImpl que es la que implementa todas las funcionalidades de la interfaz Sistema (véase Paquete 3). Los atributos que se pueden ver son inyecciones de cada una de las clases para que se pueda trabajar con los datos de la base de datos. Esos atributos *Data Access Objects* (DAOs) son interfaces que heredan de JPARepository[9] que es lo que se utiliza para mapear los datos.

Paquete 2. Diagrama de la clase Beans.

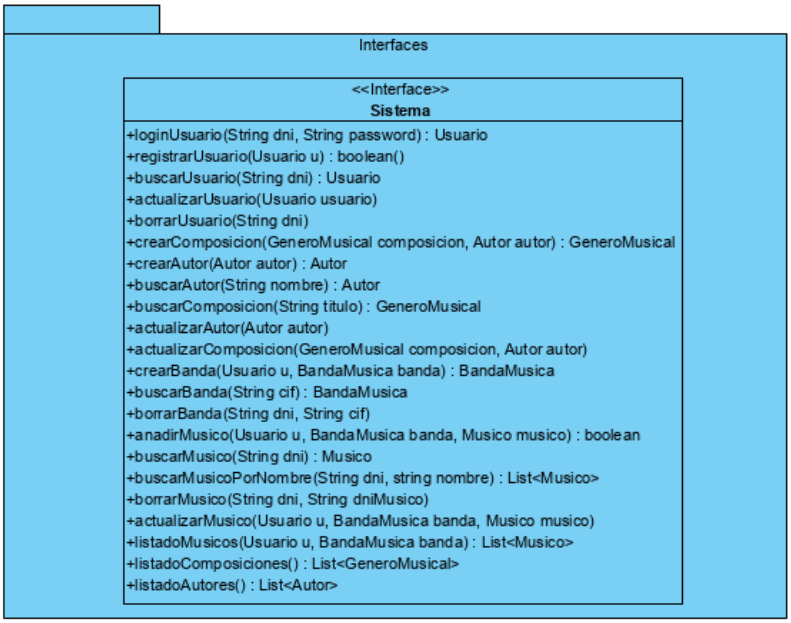
[9] JpaRepository: es una abstracción de springboot para simplificar la implementación de este tipo de clases.

3.2.1.3. Paquete Interfaz

En el paquete que se observa abajo, se puede encontrar la interfaz con la que se accede a las diferentes operaciones que tiene el servidor desde el cliente.

El cliente y la interfaz se envían datos mediante los *Data Transfer Objects* (DTOs) que actúan para trabajar con los datos necesarios y no tener que trabajar con todos los datos que tiene una entidad.

Paquete 3. Diagrama de la interfaz



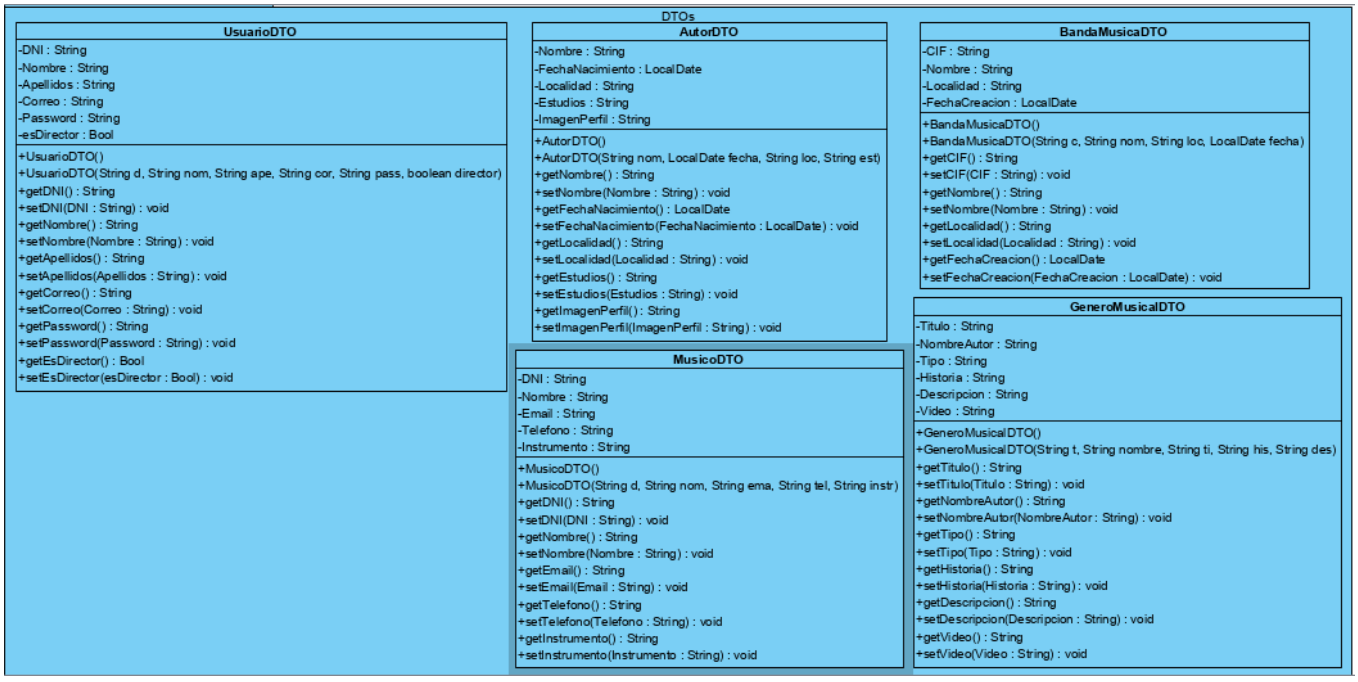
3.2.1.4. Paquete DTOs

Los DTOs tienen con finalidad crear un objeto plano con una serie de atributos que puedan ser enviados o recuperados del servidor en una sola invocación. De esta manera, un DTO puede contener información de diversas fuentes o tablas y concentrarlas en una clase simple.

Estos son utilizados para dar seguridad al sistema, ya que realiza la función de solo lectura para así evitar que se introduzcan en las distintas operaciones que trabajen con los datos. También son serializables ya que viajan mediante la red para hacer las diferentes relaciones entre cliente y servidor.

Como ya se ha comentado en el punto anterior, el cliente y la interfaz trabajan con los DTOs, y en el siguiente paquete se presentan los diferentes DTOs que existen en la aplicación para trabajar con los datos que el cliente quiere llevar al servidor. El servidor recibe estos datos y trabaja con ellos para realizar la función que el cliente quiere hacer. Después el servidor le manda los datos al cliente como DTO también; con esto se consigue no trabajar con todos los datos que puede tener una entidad.

Paquete 4. Diagrama de los DTOs



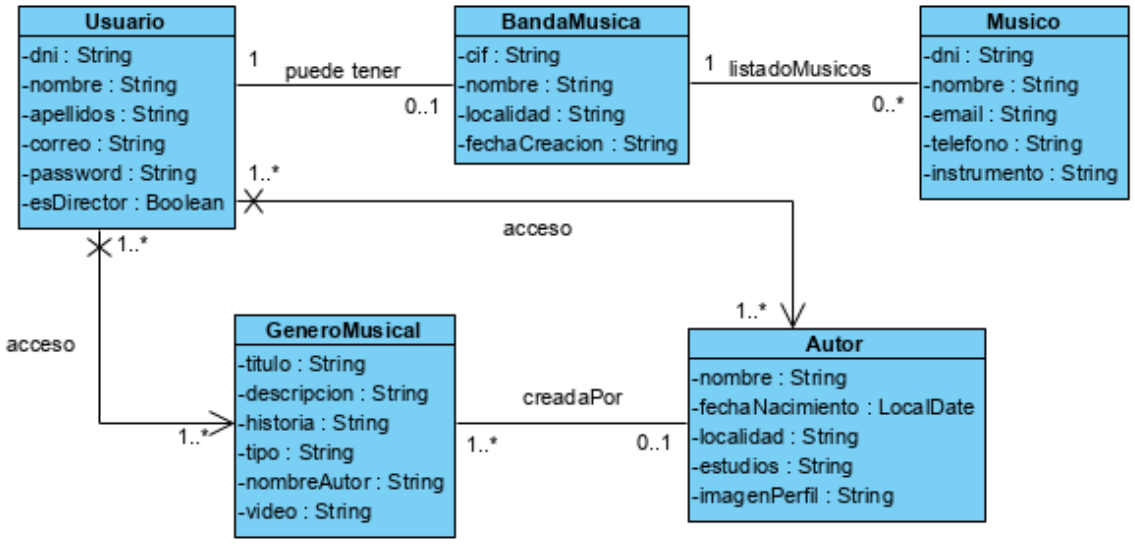
3.2.2. Diagrama de clases del sistema cliente

En el siguiente diagrama se pueden ver las diferentes clases que están implementadas en la parte del proyecto del cliente.

Como se observa, hay cinco clases de las cuáles, en la parte del cliente, sólo se necesitan los atributos de cada clase para trabajar con cada una de ellas. Esto es debido a que el servidor es el que se encarga de realizar todas las operaciones necesarias.

La estructura con la que trabaja el cliente viene determinada por la organización de los diferentes componentes que se utilizan en el framework angular. Esto se explicará con detalle en el apartado de implementación.

Diagrama 5. Diagrama de clases del cliente



La parte del cliente solamente se encarga de coger los datos de los atributos introducidos por el usuario, para enviárselos al servidor mediante las funciones de los servicios que se encargan de realizar las llamadas a la API REST, o de recuperar los datos de cada atributo desde el servidor mediante estas.

Por ello, solamente son necesarios los atributos que es con lo que va a trabajar el cliente en el proyecto.

Las relaciones que se pueden ver entre ellas son las siguientes:

- Usuario – BandaMusica: El usuario necesita identificarse para gestionar una banda de música y se necesita su Documento Nacional de Identidad (DNI) para asociarle dicha banda
- BandaMusica – Musicos: Esta relación es para poder obtener los datos de una banda de música en concreto
- Usuario – GeneroMusical: El usuario tiene acceso a todas las composiciones que haya en la aplicación
- Usuario – Autor: Misma funcionalidad que la anterior, pero en este caso con autores
- GeneroMusical – Autor: Esta relación que hay entre estas dos clases es para asociar a una composición el autor que la ha creado

3.3. Diagrama de Secuencia

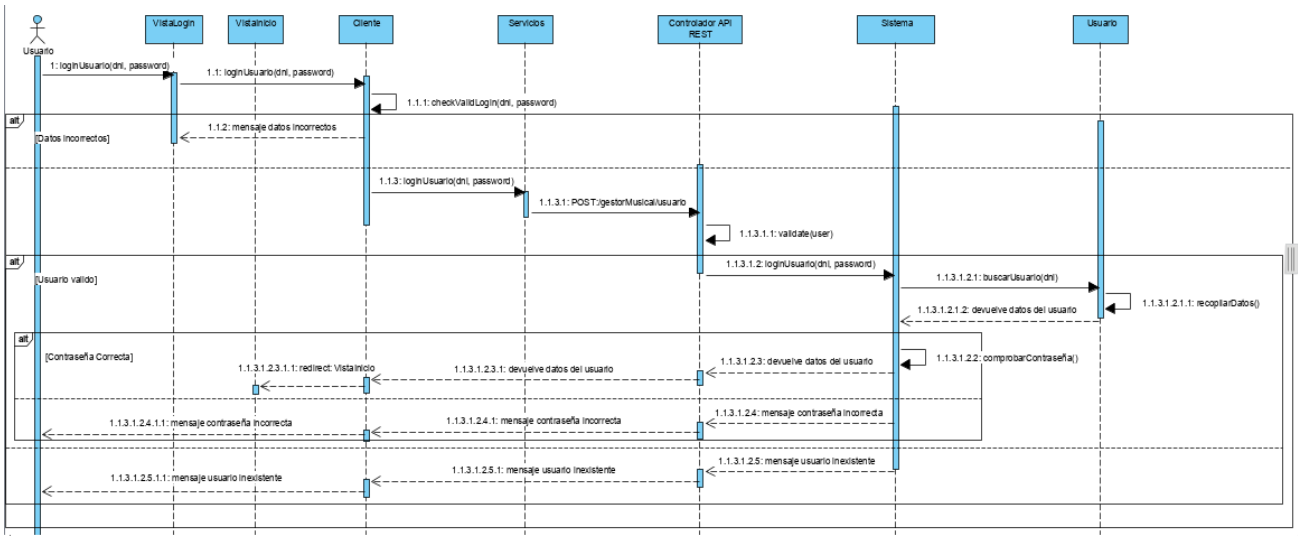
Un diagrama de secuencia [4] tiene como objetivo describir el comportamiento dinámico que tiene el sistema internamente, donde se muestran los diferentes mensajes que se intercambian entre los objetos que están activos en dicha secuencia.

A continuación, se van a observar los diagramas de secuencia más relevantes para que se pueda hacer un visionado extenso de cada uno de ellos y que puedan deducirse los comportamientos de otros diagramas más pequeños de manera más fácil a partir de los representados.

3.3.1. Login

En este diagrama de secuencia se puede ver el proceso que se realiza cuando un usuario quiere loguearse en la aplicación.

Diagrama 6. Secuencia Login



Para comenzar, dicho usuario entra en la vista de login e introduce los datos correspondientes, esta vista manda los datos al controlador del cliente y comprueba los datos, si los datos son incorrectos el cliente le manda al usuario mensaje de datos erróneos.

Si los datos son correctos el cliente manda el login al servicio y el servicio envía un post a la *Uniform Resource Identifier* (URI) que tiene el controlador API REST para el login de usuario. Este valida los datos y si está todo correcto manda los datos al sistema para que trabaje con ellos.

Cuando los datos están en el sistema, este se ocupa de buscar al usuario introducido, y comprobar si existe o no, aquí se pueden dar tres casos:

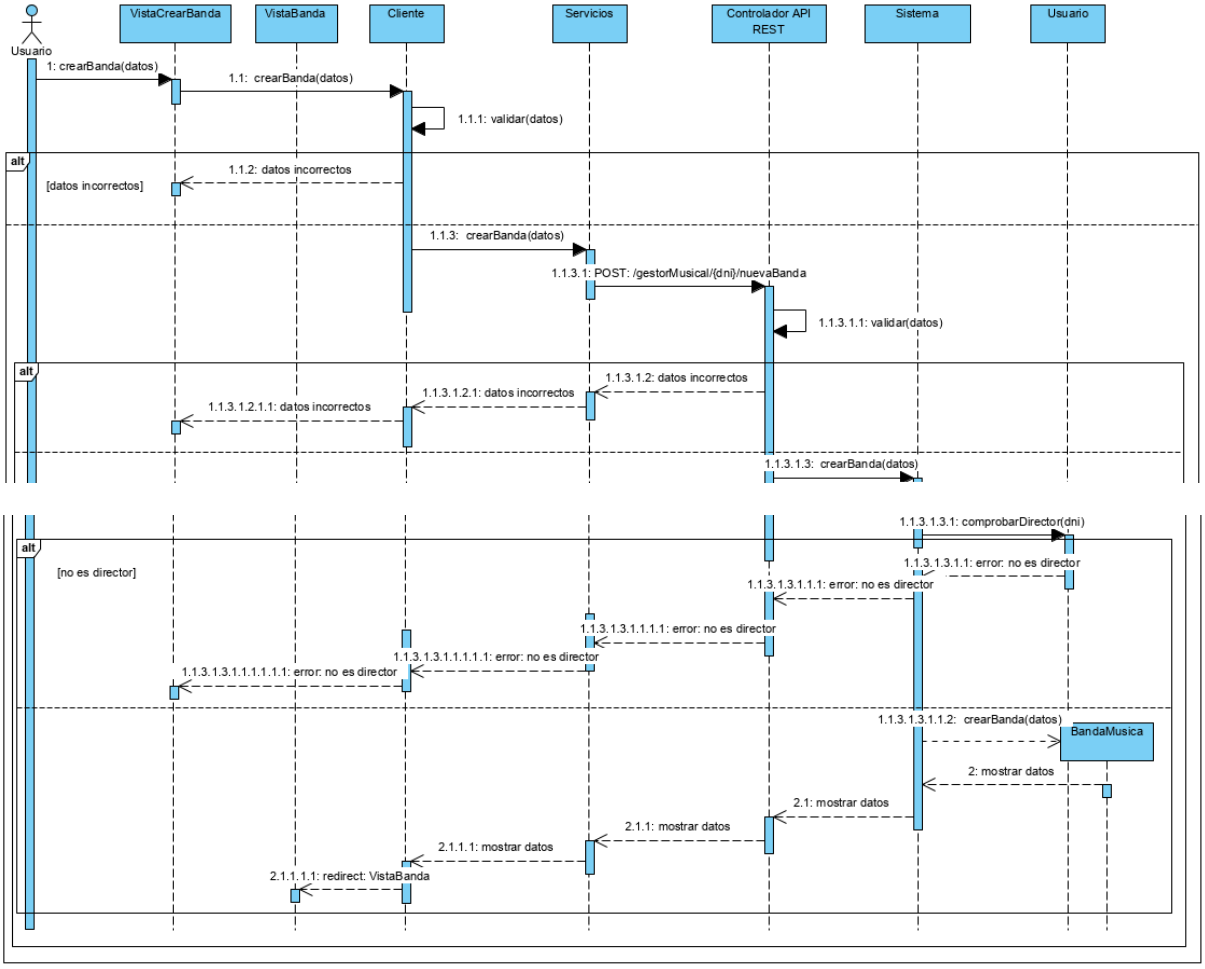
- Caso 1: Si el usuario existe, el sistema comprueba la contraseña introducida con la contraseña de ese usuario y, si todo está correcto, se realiza el proceso inverso desde el sistema hasta el usuario para devolver los datos del usuario
- Caso 2: El usuario existe, pero la contraseña es incorrecta. El sistema manda el mensaje de error de contraseña incorrecta al controlador, este al cliente y el cliente al usuario
- Caso 3: El usuario no existe, el sistema manda el mensaje de error de usuario inexistente por el camino inverso desde el servidor hasta llegar al usuario y mostrar el mensaje

En cada caso, se puede ver la respuesta que le llegará al usuario tras comprobar los diferentes datos introducidos tanto desde el servidor como del cliente.

3.3.2. Crear una banda

En el diagrama siguiente se muestra el proceso que conlleva crear una banda por parte del usuario.

Diagrama 7. Secuencia Crear una banda



Para empezar, el usuario introduce los datos en la vista de crear banda y estos datos son enviados al cliente para que sean validados. En caso de que estos sean incorrectos el cliente mandará un error al usuario.

Si no hay errores, el cliente manda los datos al servicio que es el que manda un post a la URI identificada para crear una banda por parte del controlador. Este se encarga de validar los datos de nuevo y si no hay errores, envía los datos de creación al sistema, este se encarga de comprobar si el usuario que quiere crear la banda es director, porque en caso de no ser director, no puede crear la banda.

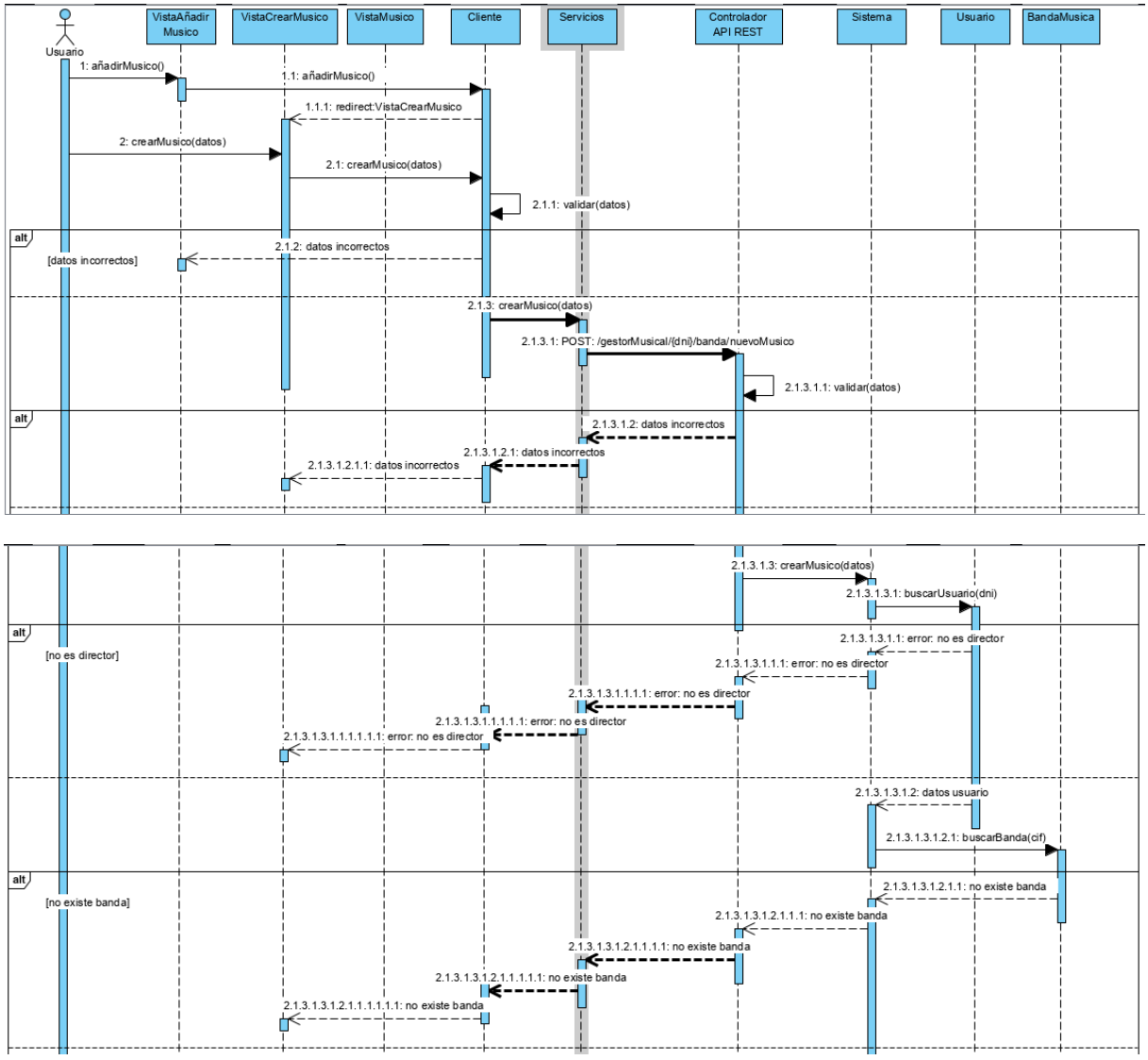
Cuando se comprueba y es director, entonces el sistema como bien se ve en el diagrama (véase Diagrama 6), crea una BandaMusica. Tras realizar la creación, el sistema muestra los datos de la banda hasta llegar al cliente y luego el cliente redirige al usuario a la vista de banda.

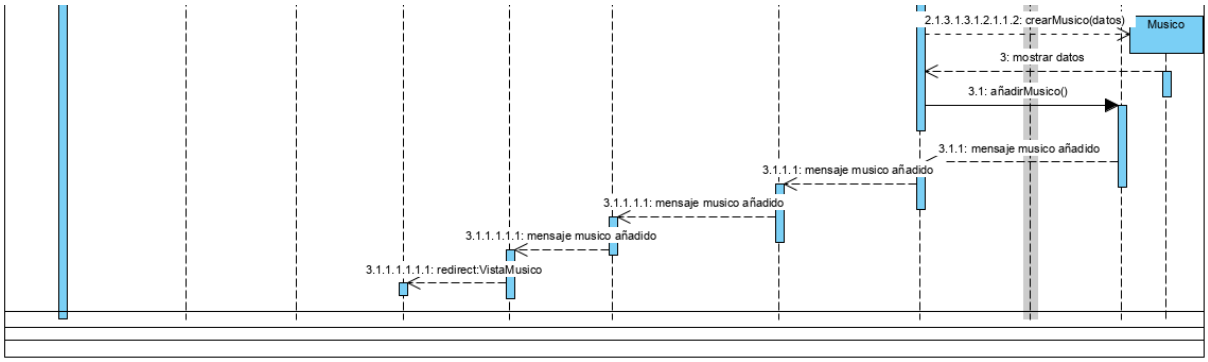
En caso de que haya algún problema de servicios solicitados, en los diferentes diagramas se ve que todos los errores se muestran al usuario sea por el tipo que sea.

3.3.3. Añadir músico a una banda

El próximo diagrama es uno de los más extensos debido a las comprobaciones que tiene que hacer internamente.

Diagrama 8. Secuencia Añadir músico





El usuario solicita la opción de añadir un músico a su banda, el cliente recibe la petición y lo redirige a la vista de crear músico para que introduzca los datos. El cliente los recibe y los comprueba, si hay algún error el cliente muestra el error al usuario.

En caso de no haber error en los datos introducidos, el cliente solicita el servicio de creación de músico y de añadirlo a la banda de música, el servicio solicitado manda un post al controlador rest y este se encarga de validar los datos por si hubiera errores. Si no hay errores manda los datos al sistema para hacer las comprobaciones necesarias, ya sea comprobar si el usuario es director y tiene banda, si la banda existe o no existe, y si todo está correcto el sistema crea el músico correspondiente.

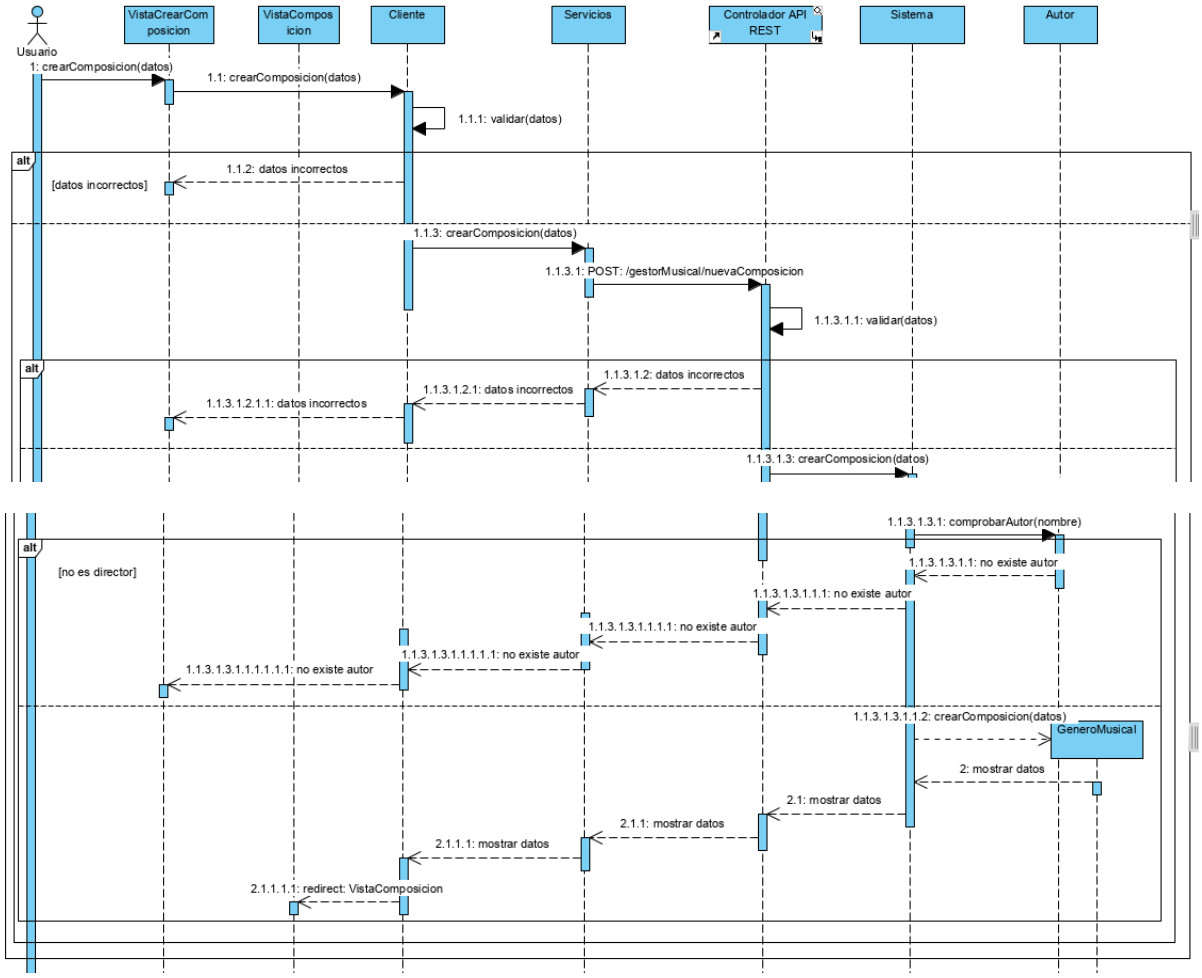
El músico devuelve los datos al sistema y este se encarga de añadir dicho músico a la banda correspondiente comprobada anteriormente.

En caso de que ocurra algún problema, el sistema hace que llegue este al usuario para que visualice el error que ha ocurrido, como se puede comprobar en el diagrama (véase Diagrama 7).

3.3.4. Añadir composición a un autor

En esta secuencia próxima, se puede ver el proceso que se sigue cuando se va a crear una nueva composición en nuestra aplicación.

Diagrama 9. Secuencia Crear composición



El usuario solicita la creación de dicha composición mediante la vista y el cliente recibe los datos, comprueba que no haya ningún dato erróneo y en caso de que lo haya, manda un error.

Tras la comprobación por parte del cliente, manda la funcionalidad al servicio que es el que se encarga de enviar los datos al controlador rest mediante su URI correspondiente; el controlador revisa los datos introducidos, si no hay error manda la función al sistema que es el que se encarga de manejar las diferentes funcionalidades y datos necesarios para trabajar en el servidor.

El sistema se encarga de comprobar si el autor al que se le va a asignar dicha composición existe o no, si el autor existe, continúa a crearla y cuando esta es creada envía los datos al usuario mediante la vista del género musical.

En caso de error en algún instante del proceso, se mandan los mensajes correspondientes al usuario como se puede ver en el diagrama (véase Diagrama 8).

3.4. Diseño de Interfaz

En este apartado, se van a encontrar dos subapartados, los cuales están dedicados al diseño de la aplicación. Uno trata sobre las diferentes metáforas utilizadas y el otro sobre el prototipo guiado de esta.

3.4.1. Metáforas

Una metáfora es un tipo de figura retórica en la cual se introduce el significado de un concepto a otro, realizando así una relación de analogía o semejanza entre ambos.

El uso de las metáforas se realiza cuando se quiere facilitar la familiaridad y accesibilidad del usuario a las funciones accesibles de la aplicación. En el diseño gráfico las metáforas suelen tener un papel bastante importante, ya que hace que el usuario se cree un modelo mental de la funcionalidad de la aplicación debido a que hace una comprensión más intuitiva de las funcionalidades del sistema.

En la aplicación se han aplicado varias metáforas:

- Visualizar datos de una composición, autor o músico

La siguiente metáfora que se puede observar en la imagen, es la que ha sido utilizada en los diferentes listados que tiene la aplicación, la cual va acompañada de cada uno de los datos visualizados (composición, autor o músico).

A través de ella, accedemos a los datos concretos de una composición, autor o músico seleccionado del listado que estamos visualizando.

Imagen 14. Icono de vista

- Entrar a tu perfil, login o registrarte

En la siguiente ilustración, se puede ver la metáfora utilizada para hacer referencia al acceso del perfil. Esta metáfora se encuentra en la cabecera para que siempre esté accesible por parte del usuario.

Cuando el usuario está logueado, esta metáfora da acceso al usuario a su perfil y cuando el usuario no está logueado, esta metáfora te lleva al login para acceder al sistema y después tras

el inicio de sesión muestra el perfil o, si no estás registrado, en el mismo formulario de login está el acceso al formulario de registro para inscribirte en la aplicación.

Imagen 15. Icono de perfil

- Desconectar la sesión activa

Esta metáfora que se puede percibir en la imagen que se muestra a continuación es la utilizada para la desconexión del usuario del sistema. Cuando el usuario la emplea, esta se ocupa de desconectar al usuario y de enviarlo a la página de inicio.

Esta metáfora aparece en la cabecera cuando el usuario inicia sesión en la aplicación y comienza a trabajar dentro de esta. En caso de que el usuario no esté logueado, la metáfora no aparecerá.

Imagen 16. Icono de desconexión

3.4.2. Prototipo

Un prototipo es un primer ejemplar de la aplicación para crear una simulación del producto final y con este, poder realizar cambios en el diseño si son necesarios y confirmar que están todas las funcionalidades necesarias en el diseño.

El prototipo se utiliza por los componentes del equipo de desarrollo y los usuarios para que se puedan hacer pruebas, testar y aprender el funcionamiento de la aplicación antes de comenzar con la implementación de esta.

Gracias a esto, se puede confirmar que el producto cumplirá con lo que busca el cliente como resultado final.

En este proyecto, para realizar el prototipo se ha utilizado Visual Studio Code, creando un proyecto Angular, donde se han desarrollado las diferentes vistas estáticas para poder hacer un recorrido por la aplicación.

Las vistas tienen diferentes números ilustrados con cierto significado:

- El número que se encuentra en el centro en lo más alto de la vista, es el que le corresponde a dicha vista. Por ejemplo, si una tiene el "1" significa que esa es la número 1.
- Los demás números que se pueden encontrar son para simular la interacción de los botones que se pueden ver. El número que esté arriba de ese botón que desea pulsar nos indica a qué vista hay que ir tras "pulsar" en este.

A continuación, se muestra el prototipo realizado antes de realizar la implementación de la aplicación. Este se puede agrupar en diferentes apartados:

- Vistas públicas

Estas vistas son las que están accesibles por todos los usuarios que utilicen la aplicación ya estén registrados, logueados o ninguna de las dos opciones.

Estas vistas se pueden dividir en tres tipos:

- Relacionadas con las composiciones

Las próximas vistas son aquellas que están relacionadas con los datos de las diferentes composiciones que están registradas en la aplicación, así como poder crear nuevas o modificar las que están.

Imagen 17. Vista inicio

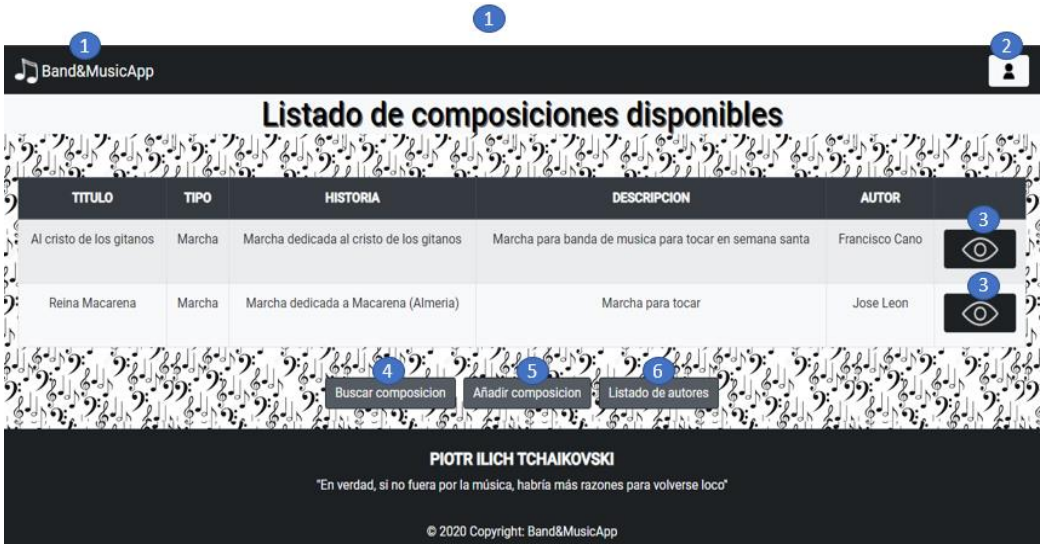


Imagen 18. Vista Composición

Imagen 19. Vista Búsqueda Composición

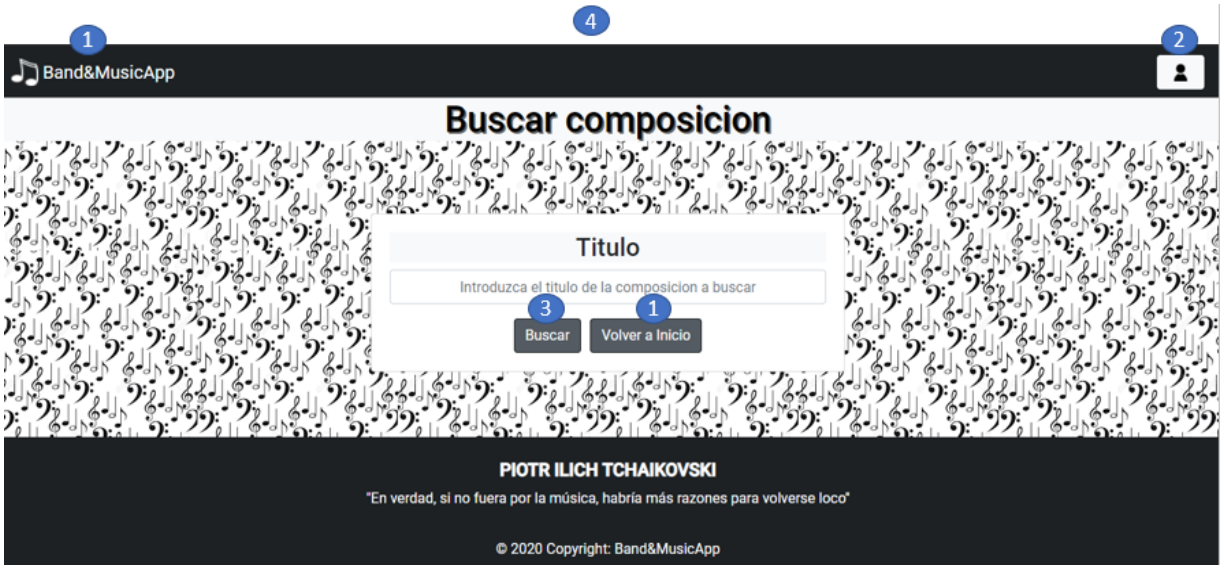
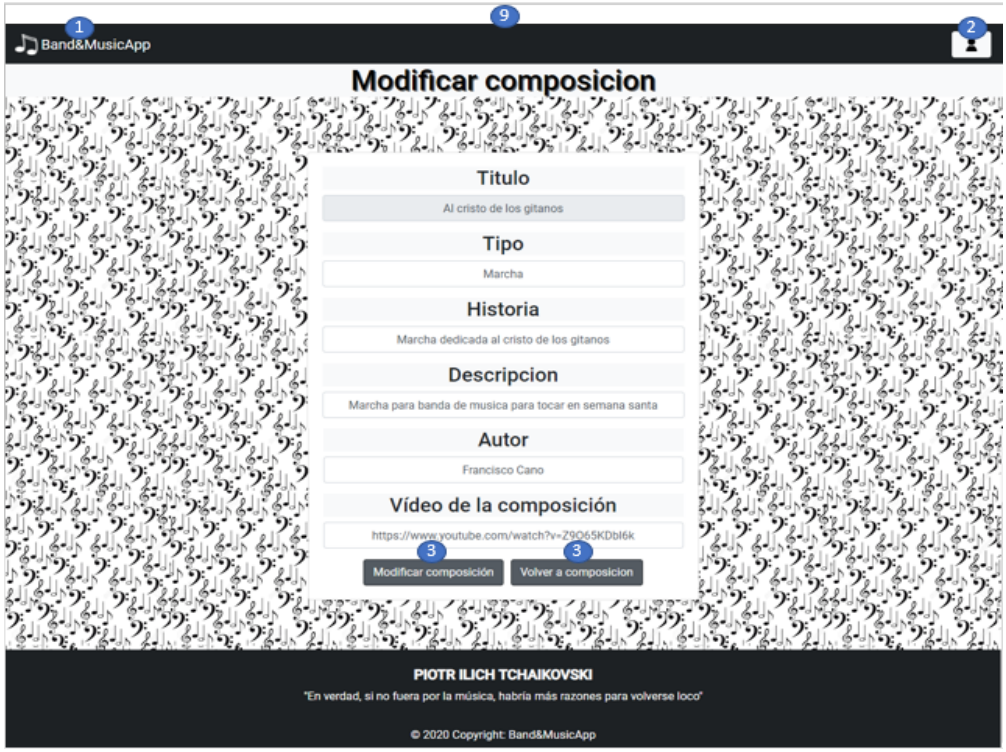


Imagen 20. Vista Nueva Composición

Imagen 21. Vista Modificación Composición



- Relacionadas con los autores

Las siguientes vistas son aquellas con las que se pueden trabajar con los diferentes datos de los autores que están guardados en la aplicación. También se pueden crear nuevos autores o modificar los que ya hay dentro.

Imagen 22. Vista Listado Autores

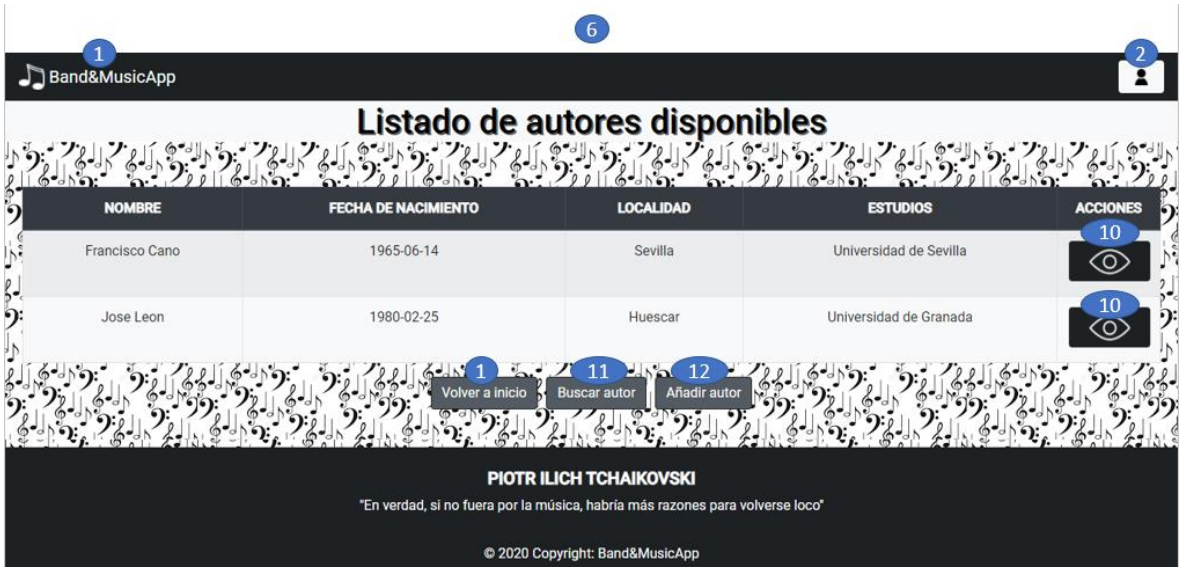


Imagen 23. Vista Autor

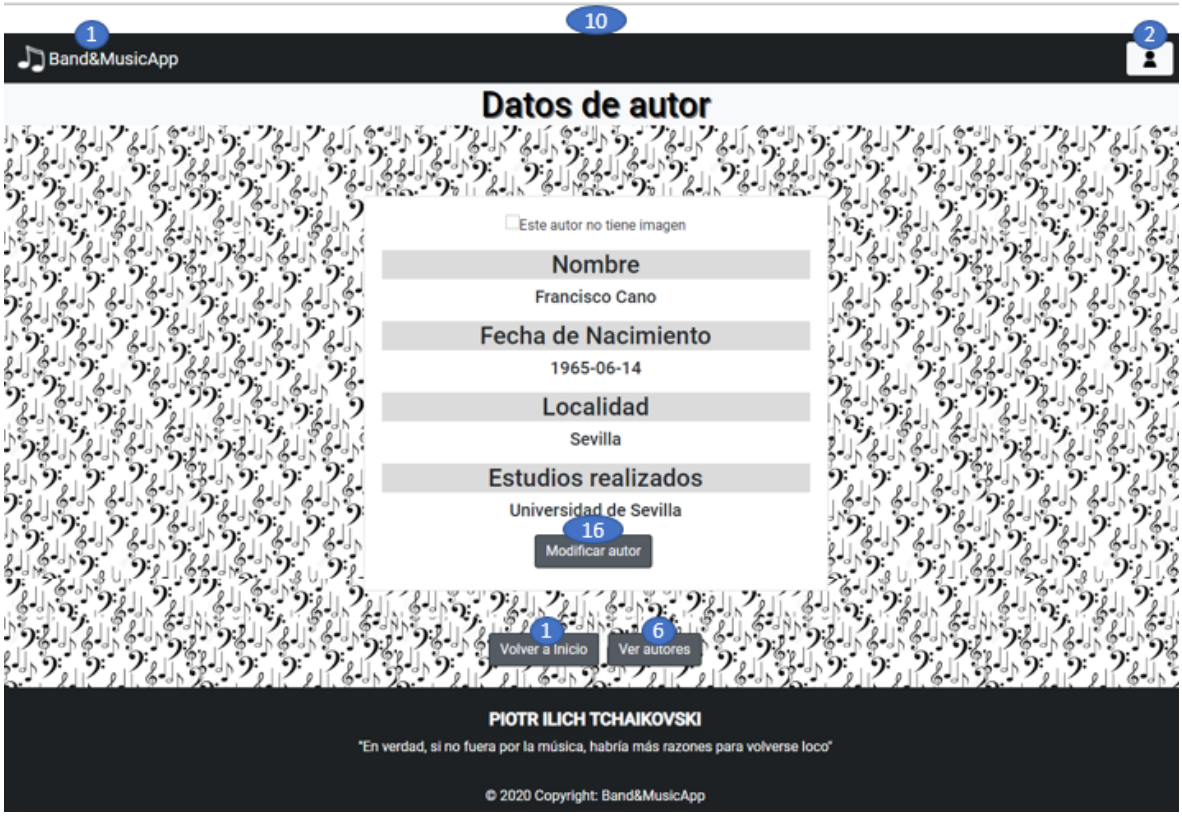


Imagen 24. Vista Búsqueda Autor

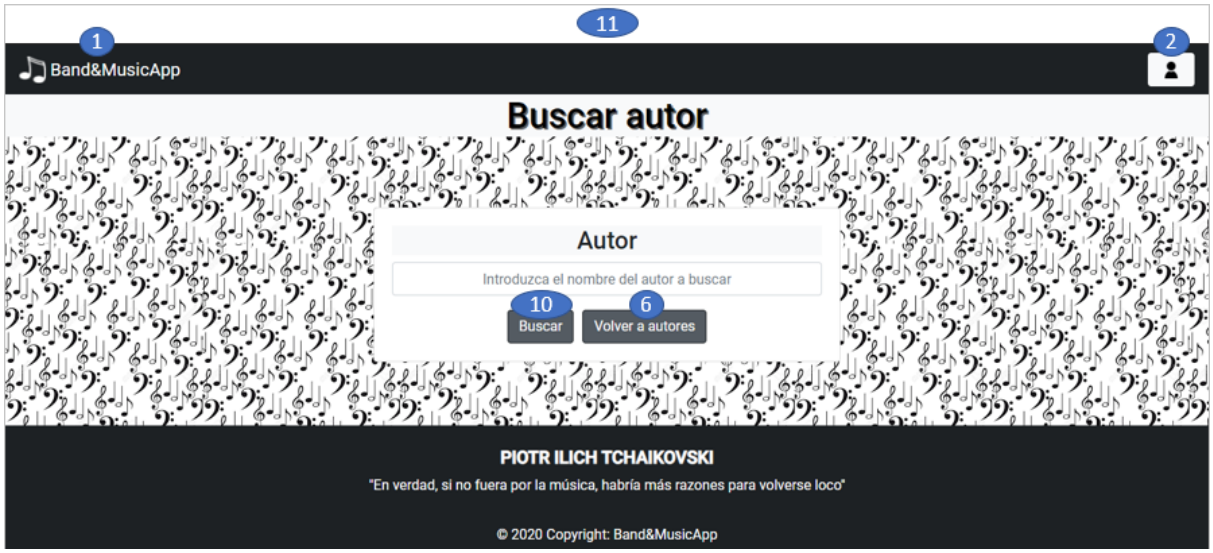


Imagen 25. Vista Crear Autor

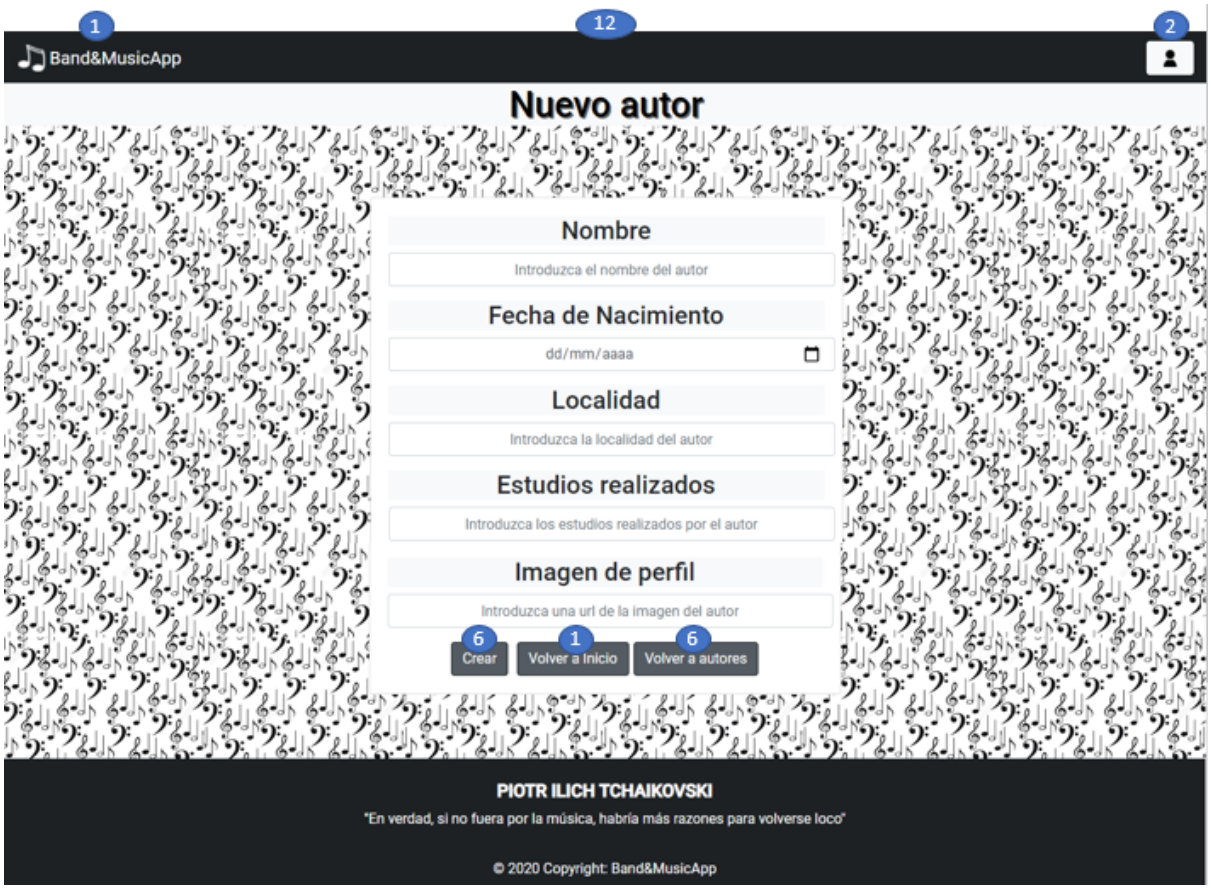
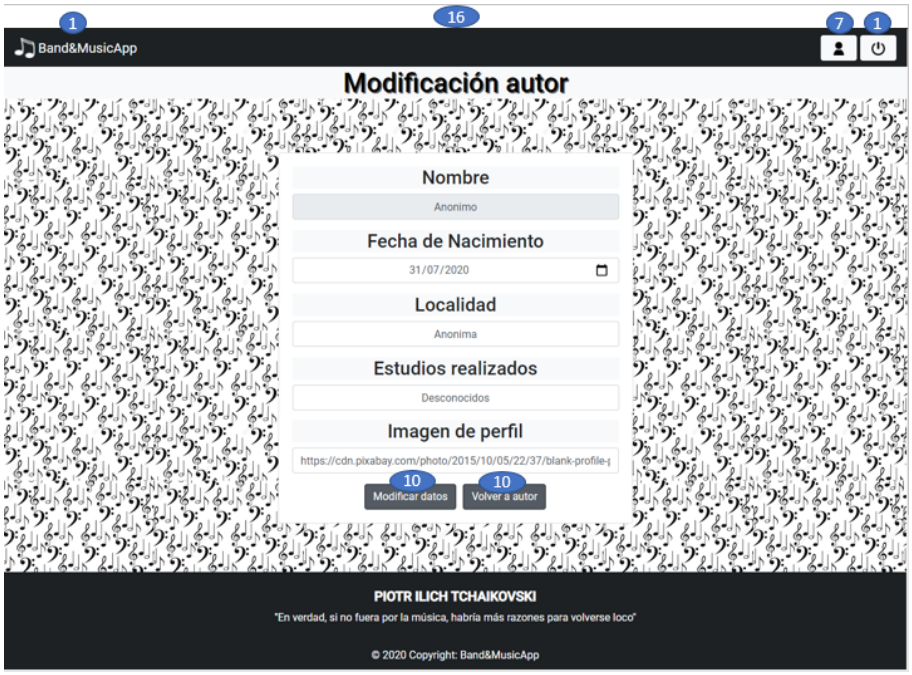


Imagen 26. Vista Modificación Autor



- Relacionadas con los usuarios

En las siguientes ilustraciones se muestran las dos vistas públicas para los usuarios que se quieren loguear o registrar en la aplicación.

Imagen 27. Vista Login

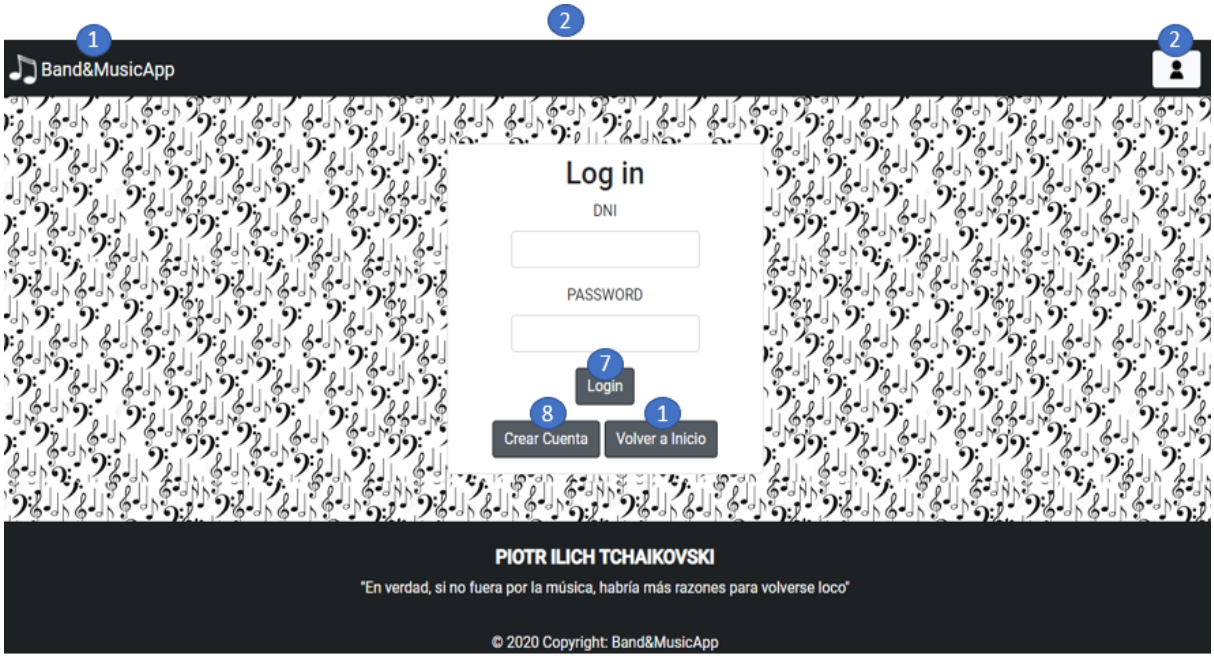


Imagen 28. Vista Crear Usuario

- Vistas privadas

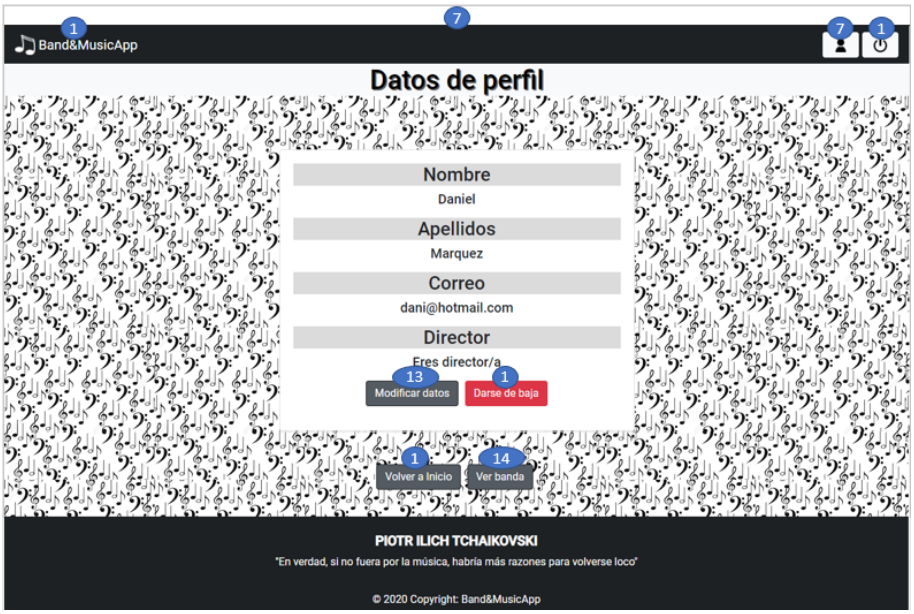
Estas vistas son aquellas accesibles por los usuarios que han sido registrados en la aplicación y han iniciado sesión en esta. Tras realizar esto, aparecen las diferentes vistas siguientes.

En este apartado, también se pueden dividir las diferentes vistas en subapartados dedicados cada uno de ellos a diferentes cosas de la aplicación.

- Relacionadas con el usuario

En las siguientes imágenes se podrán contemplar las diferentes vistas que trabajan específicamente con los datos de los usuarios y lo que puede realizar cada uno de ellos.

Imagen 29. Vista Perfil Usuario



En esta vista, si el usuario no tuviera banda, el botón de "Ver banda" no aparecería, y se vería un botón "Crear banda" (15).

En caso de que el usuario no sea director, no se verán ninguna de las dos opciones ni "Ver banda" ni "Crear banda". Ya que si no es director la función de gestionar una banda de música no es necesaria.

Imagen 30. Vista Modificación Perfil Usuario

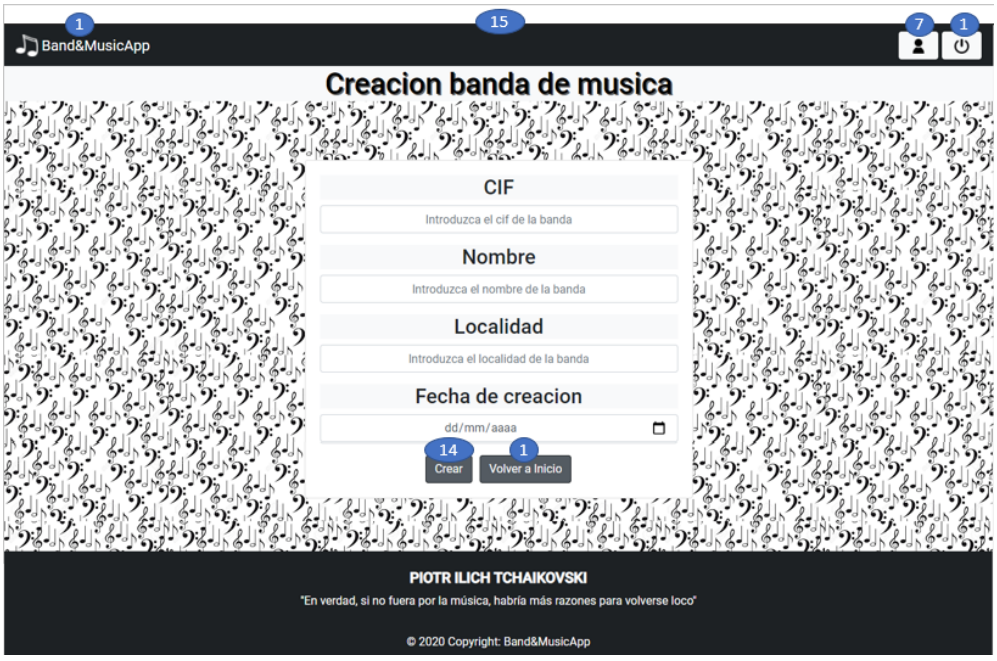
- Relacionadas con la banda de música

Las próximas vistas están relacionadas con los diferentes datos que están guardados en la aplicación sobre la banda de música de un usuario.

Estas vistas también están relacionadas con el usuario ya que es el que se encarga de gestionar los diferentes datos que conlleva una banda de música (creación, añadir músicos, etc.).

Imagen 31. Vista Banda de Música

Imagen 32. Vista Crear Banda de Música



- Relacionadas con los músicos de una banda

Las siguientes vistas mostrarán las diferentes opciones que hay en la aplicación para trabajar con los datos de los diferentes músicos que hay en una banda de música que ha creado el usuario. Por tanto, estas ilustraciones, también están relacionadas con el usuario y la banda de música.

Imagen 33. Vista Listado Músicos.

Imagen 34. Vista Músico

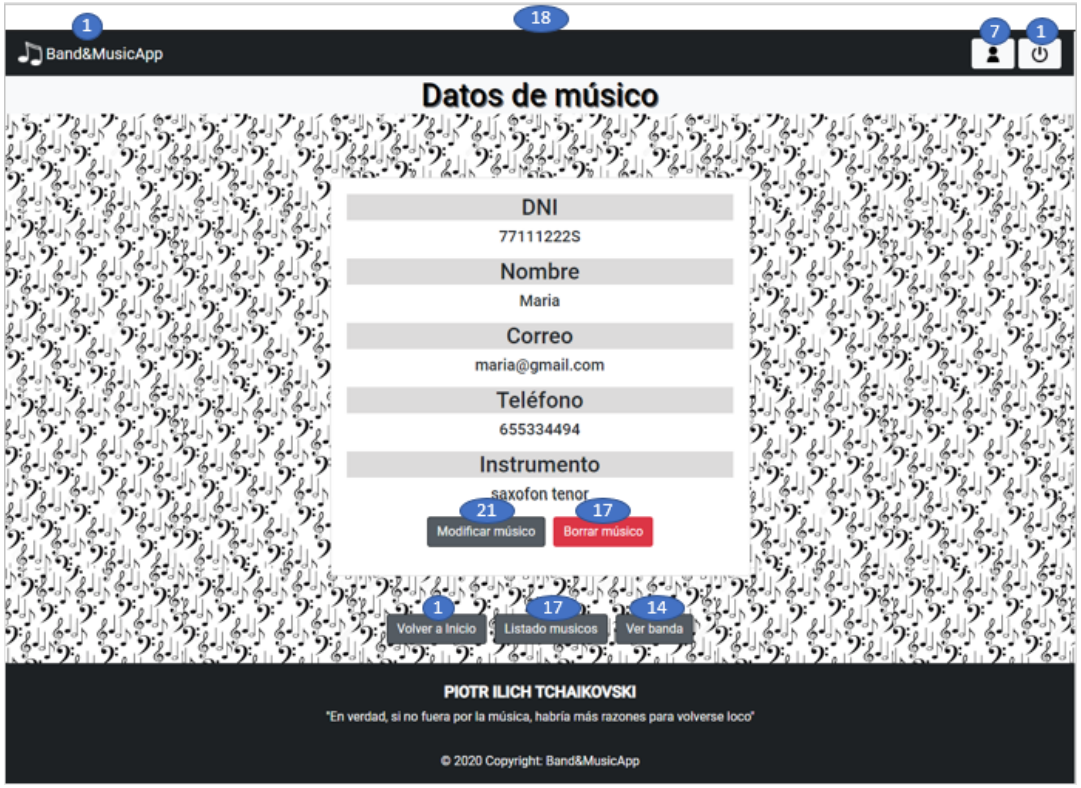


Imagen 35. Vista Búsqueda Músico

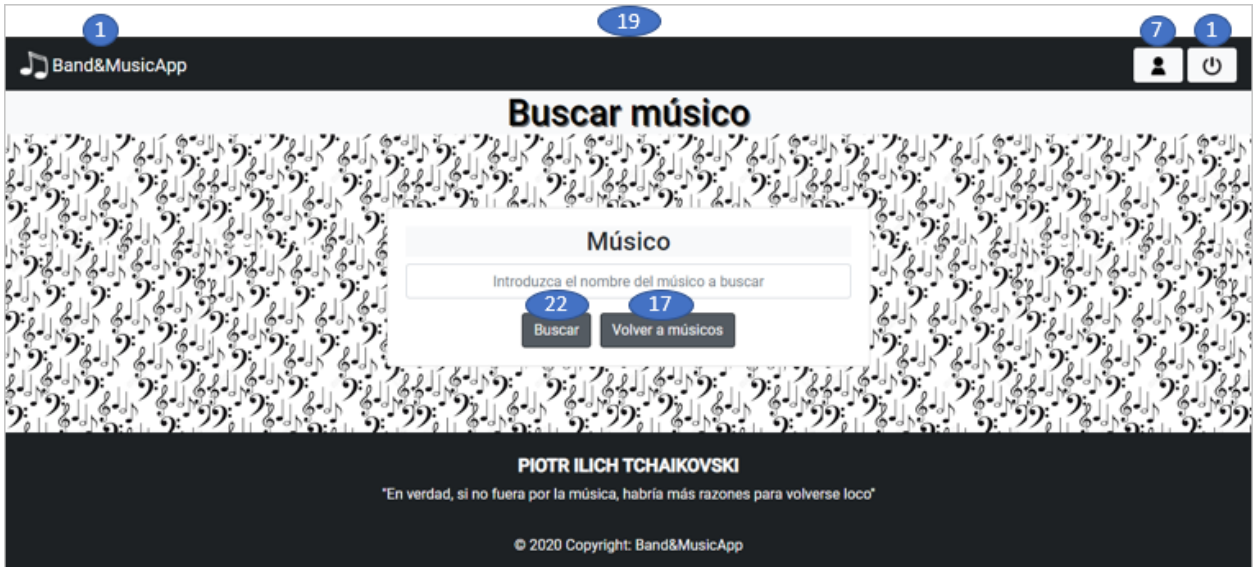


Imagen 36. Vista Añadir Músico

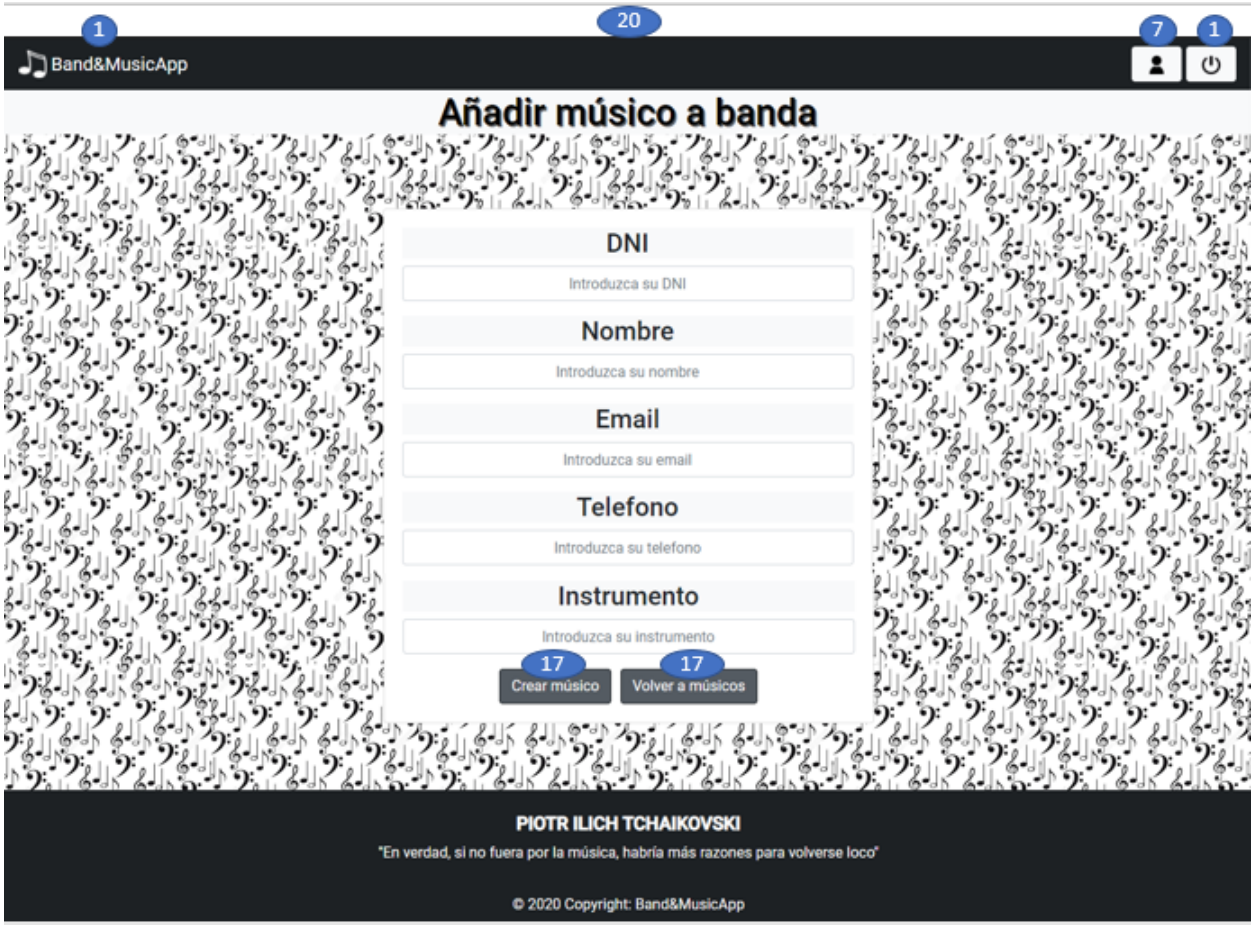


Imagen 37. Vista Modificar Músico

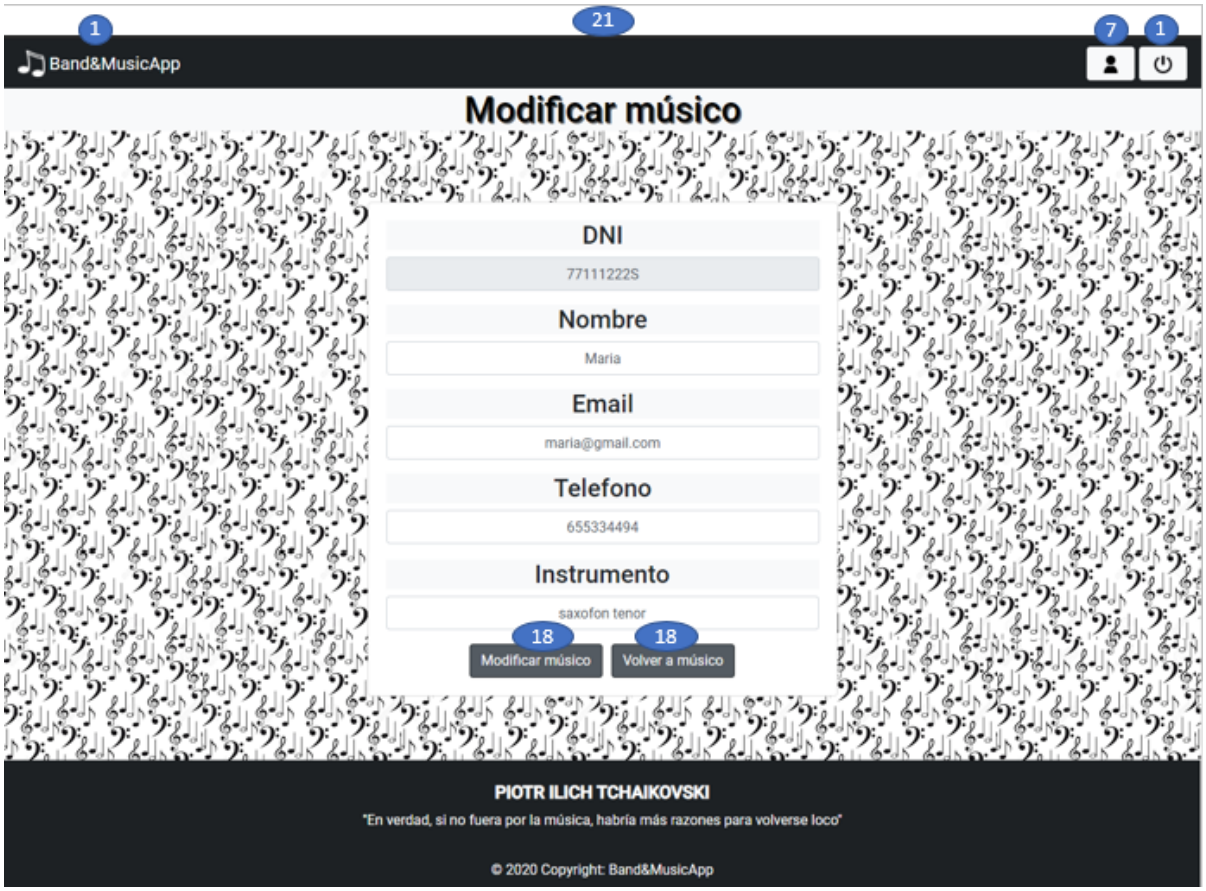
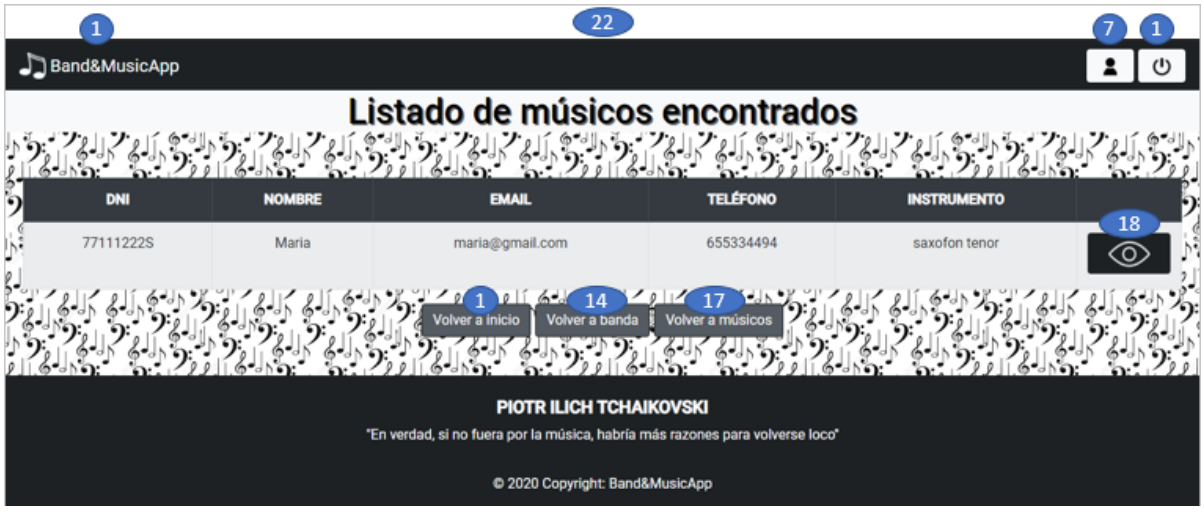


Imagen 38. Vista Listado de Músicos buscados



En esta imagen, aparecerán todos los músicos que tengan el nombre que se haya puesto a buscar y el/la directora/a elegirá que músico está buscando. Como estamos en una versión estática, se ha buscado el nombre de María y han salidos los datos de esta músico.

4. Implementación

En esta sección se tratarán aspectos relevantes relacionados con la implementación de cada uno de los elementos que componen la aplicación.

4.1. Arquitectura

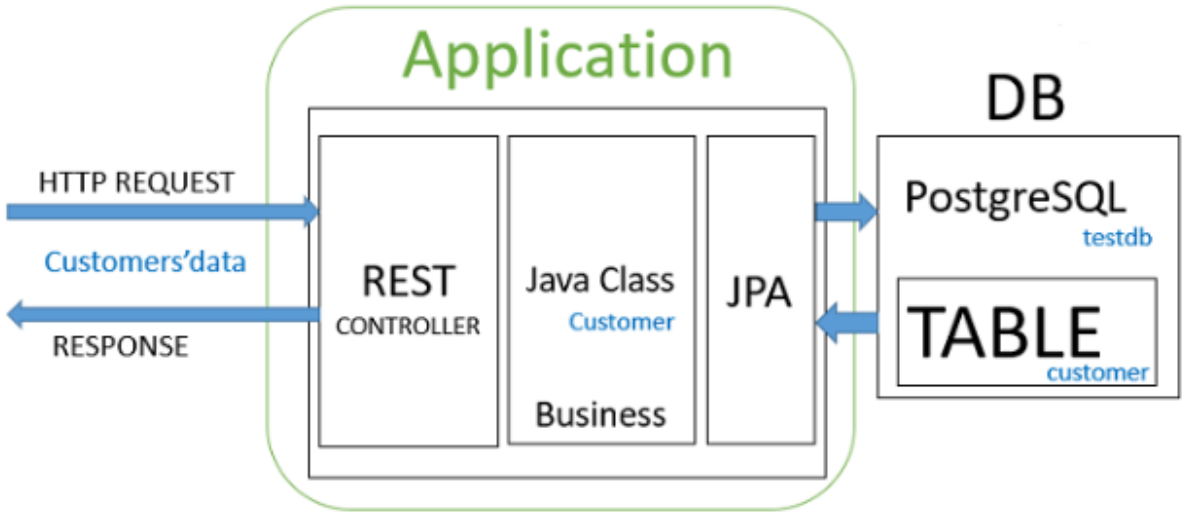
En este apartado, se mostrarán los diferentes diagramas arquitectónicos del sistema que ha sido implementado, para que se observen los diferentes elementos constituyentes y la relación que hay entre cada uno de ellos.

Como se dijo anteriormente, la aplicación está compuesta de dos aplicaciones (servidor y cliente), por tanto, se va a explicar la arquitectura de cada una de ellos.

4.1.1. Servidor

En este punto, se va a mencionar el funcionamiento de un servidor con servicios RESTFUL, ya que el funcionamiento del servidor ha sido mencionado en el apartado 3.2.1. donde se han explicado las diferentes relaciones que tienen las clases y los paquetes que están en el servidor implementados.

Imagen 39. Arquitectura de Spring Boot.



En esta imagen se puede ver la arquitectura que sigue una aplicación que está desarrollada con un API REST. El cliente si quiere utilizar alguna de las diferentes funcionalidades del servidor, lo hace mediante llamadas *Hypertext Transfer Protocol* (HTTP). Estas llamadas llegan al controlador rest que está implementado para acceder a la funcionalidad del servidor, internamente el controlador entra dentro del sistema para trabajar con los datos de las clases

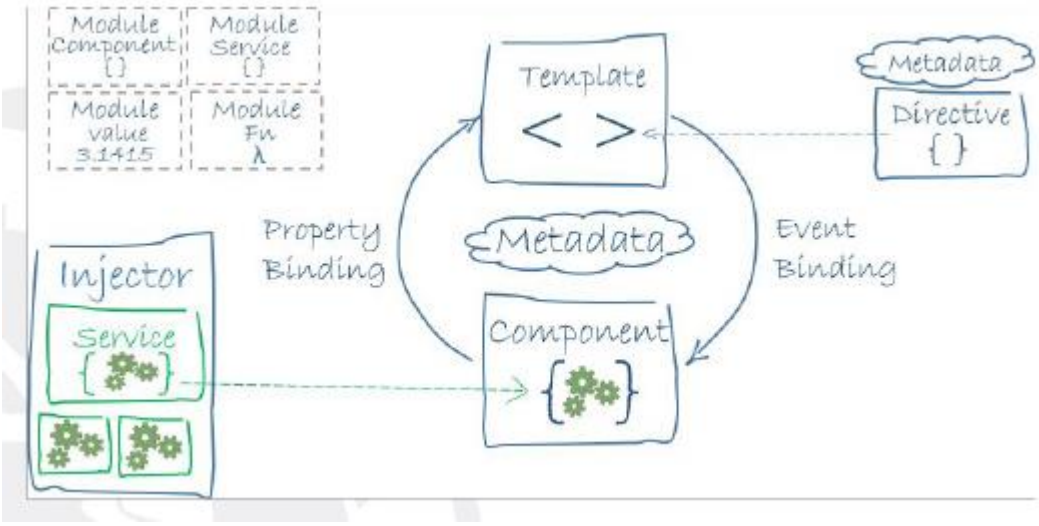
que hay implementadas. La capa de persistencia es la que se encarga de interactuar enviando y recibiendo datos de la base de datos. A su vez esta capa también trabaja con la capa donde están las clases para poder trabajar con sus datos.

El servidor cuando ya ha realizado la funcionalidad que ha solicitado el cliente, le responde con una respuesta HTTP y cuando sea necesario el envío de datos lo hace en formato JSON.

4.1.2. Cliente

El cliente en esta aplicación está orientado en Angular 2, por tanto, se va a estudiar el funcionamiento de la arquitectura de Angular 2.

Imagen 40. Arquitectura de Angular 2



Angular se basa en componentes y estos pueden estar compuestos por varios archivos: vistas, estilos, controladores, servicios etc. En los servicios es donde se realizan las diferentes llamadas para gestionar los datos. Los controladores son aquellos que se encargan de controlar las vistas y de atender los datos de estas.

Las plantillas son las que permiten definir la vista de un componente. Por tanto, como se puede observar en la imagen (véase Imagen 41), hay un ciclo continuo entre los componentes y las plantillas porque es donde se está trabajando con los datos.

Para sincronizar el modelo y la vista, se utilizan ciertas directivas, esto lo que hace es sincronizar la información tanto si hay cambios en la vista como si hay cambios en el modelo.

El paquete Injector es el que se utiliza para inyectar los servicios mediante inyección de dependencias.

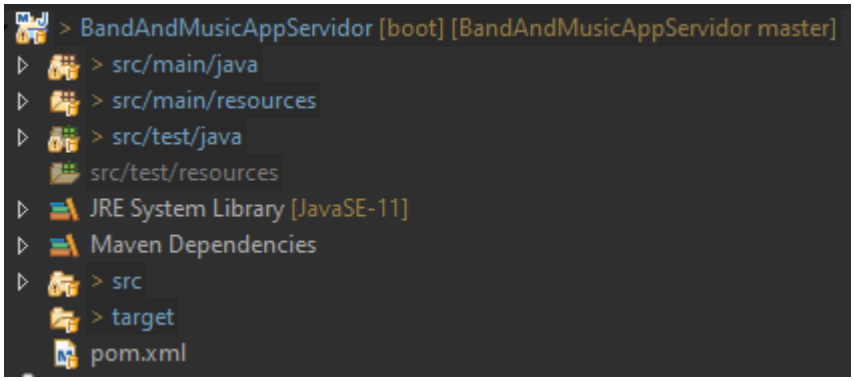
4.2. Estructuras de los proyectos

En dicha sección se va a realizar la explicación de cómo están organizados los dos proyectos que han sido implementados.

4.2.1. Estructura del proyecto servidor

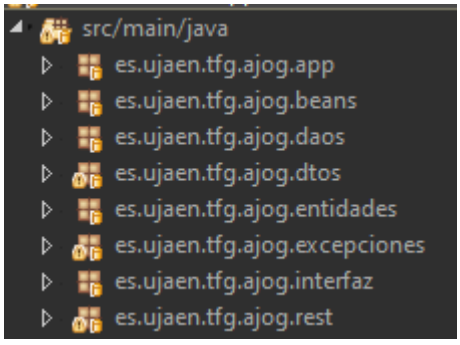
El proyecto Maven de Spring Boot está estructurado de la siguiente manera

Imagen 41. Estructura Servidor



Como se puede observar en la imagen, el proyecto consta varias carpetas donde están almacenadas las distintas clases y paquetes.

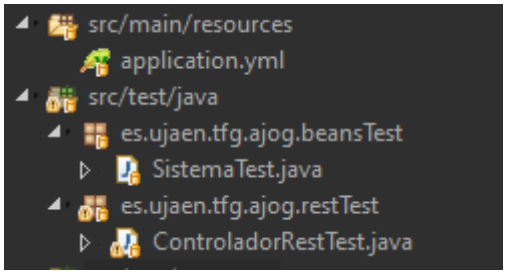
Imagen 42. Paquete Java



En la primera carpeta Java, se pueden ver los diferentes paquetes que contiene. Cada paquete contiene las clases correspondientes que han sido implementadas.

- En el paquete app nos encontramos la clase principal que lanza nuestro backend
- El paquete beans contiene la clase donde están implementadas todas nuestras funcionalidades
- El paquete daos está compuesto por todas las interfaces utilizadas para trabajar con los datos en la Base de Datos (BBDD)
- El paquete DTOs incluye las clases que actúan como DTOs para hacer la transferencia de los datos
- El paquete entidades envuelve todas las entidades utilizadas en el proyecto como las relaciones que tienen entre ellas
- El paquete excepciones está formado por las diferentes excepciones que han sido implementadas en el servidor cuando surgen diferentes errores
- El paquete interfaz está construido por la interfaz de la aplicación con la cual el controlador rest accede a las diferentes funcionalidades implementadas por la clase beans
- El paquete rest está hecho por la clase que actúa como controlador rest donde se han implementado las diferentes funciones para adaptarlas a API REST y puedan ser accesibles por diferentes clientes

Imagen 43. Paquetes Resource y Test



Continuando con la estructura del servidor, se pueden encontrar dos carpetas más que son la de resource, la cual contiene el archivo de configuración que facilita la conexión con la BBDD que se esté utilizando.

Por otro lado, se encuentra la carpeta de test, esta carpeta contiene las diferentes pruebas que se realizan al sistema implementado para ver si su funcionamiento es correcto. Se pueden observar en la imagen dos paquetes:

- beansTest: Este paquete contiene la clase que realiza las diferentes pruebas al sistema beans implementado

- restTest: Este paquete está formado por la clase que hace los diferentes tests a los métodos implementados en el controlador rest para comprobar si todo está correcto

De las carpetas que nos faltan por definir (véase Imagen 42), la carpeta JRE System Library y Maven Dependencies son carpetas donde se encuentran las diferentes configuraciones y dependencias de Java y Maven.

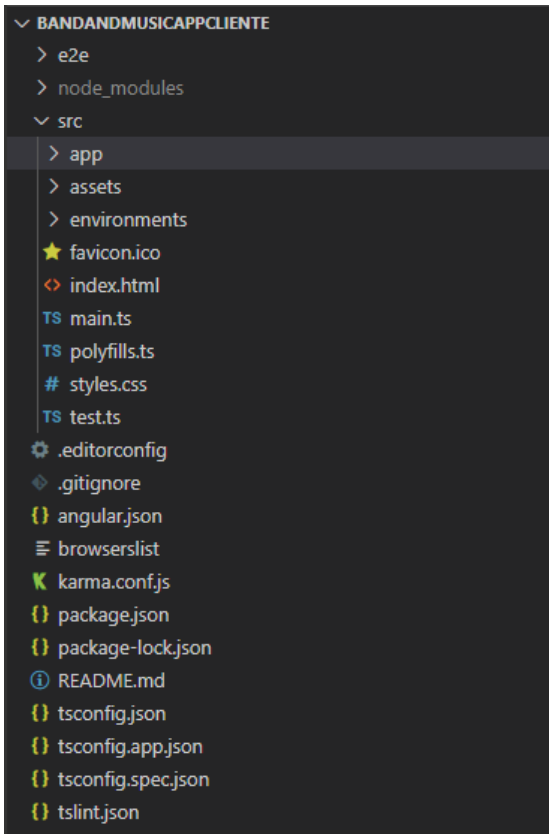
La carpeta source (SRC) está formada por el conjunto de las tres carpetas comentadas anteriormente como, por ejemplo: java, resources y test.

Y, para terminar, el proyecto contiene el archivo pom.xml que es donde se introducen las diferentes dependencias que se necesitan en el proyecto para que todo se ejecute correctamente.

4.2.2. Estructura del proyecto cliente

Cuando se crea un proyecto Angular, este proporciona una estructura de inicio dónde se puede ver la organización de las diferentes carpetas y archivos en el proyecto.

Imagen 44. Estructura cliente



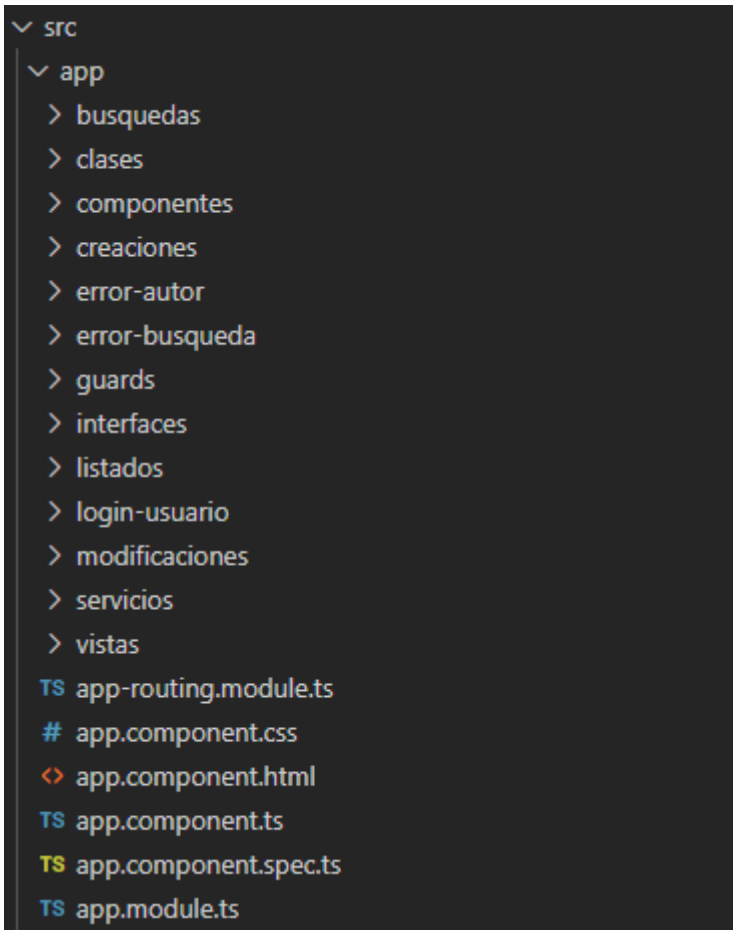
Como podemos observar en la ilustración, estas son las diferentes carpetas y archivos que nos genera Angular por defecto cuando creamos un proyecto nuevo.

Se va a comenzar a explicar las diferentes carpetas que componen el proyecto:

- e2e: esta carpeta está denominada como *"end to end"* la cual está formada por unos ficheros cuya funcionalidad es realizar tests de manera automática, imitando una interacción de un usuario y la aplicación
- node_modules: en esta carpeta se encuentran las diferentes dependencias instaladas que se han utilizado en el proyecto
- src: esta carpeta es el directorio donde se trabajan los diferentes módulos del proyecto. Además, es la carpeta más importante ya que contiene el código que se ha implementado. Dentro de esta carpeta nos podemos encontrar diferentes carpetas y archivos:
  - app: en esta carpeta se encuentra toda la implementación de los componentes que construyen la aplicación, junto a sus archivos de estilos y plantilla HTML
  - assets: esta sección está formada por todos los assets y archivos adicionales que son necesarios para hacer que el proyecto funcione
  - enviroments: aquí se pueden encontrar las distintas configuraciones y variables de entorno para poner el proyecto en desarrollo y en producción
  - index.html: archivo HTML el cual es la página principal del proyecto
  - main.ts: es el archivo TypeScript inicial que te proporciona el proyecto para que puedas configurar las configuraciones que engloben a todas las demás
  - styles.css: en este archivo se configuran los diferentes estilos *Cascading Style Sheets* (CSS) que van a ser comunes para los diferentes componentes que se utilizan en el proyecto
- .editorconfig: el editor de código es configuración en este archivo
- .gitignore: en esta carpeta se almacenan las diferentes carpetas o archivos que deben ser ignorados por el git
- angular.json: contiene la configuración de angular
- package.json: este archivo es la configuración de la aplicación. Contiene los diferentes datos de la aplicación (nombre y dependencias)
- Tsconfig.json: aquí se puede encontrar la configuración de TypeScript

Tras la explicación de la estructura general del proyecto se va a mostrar las carpetas y archivos contenidos en la carpeta app.

Imagen 45. Estructura carpeta app del cliente

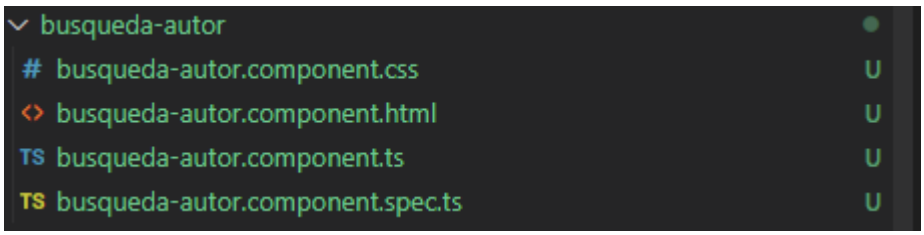


Como nos ilustra esta imagen, podemos encontrar las diferentes carpetas y componentes que están contenidos en la carpeta app. Se va a explicar cada una de las carpetas y componentes.

- Carpeta búsquedas: En esta carpeta, se encuentran los diferentes componentes que están orientados a la búsqueda (de una composición, un autor o un músico)
- Carpeta clases: Está formada por las diferentes clases que están implementadas en el proyecto del cliente
- Carpeta componentes: Aquí se pueden encontrar los dos componentes comunes de los archivos HTML, la cabecera y el pie de cada vista
- Carpeta creaciones: Esta carpeta está compuesta por los diferentes componentes relaciones con la creación de cualquier objeto del proyecto (composición, autor, banda de música, usuario, músico)

- Componente error-autor: Este nos muestra un error cuando al crear una nueva composición el autor no existe aún en la aplicación
- Componente error-búsqueda: Este componente es para mostrar una vista de error cuando al realizar una búsqueda en la aplicación no se ha obtenido resultado
- Componente guards: Este se utiliza para guardar los datos de la sesión del usuario en el navegador hasta que el usuario se desconecte
- Carpeta interfaces: Aquí dentro está contenido la interfaz que actúa para recibir los datos del login de usuario
- Carpeta listados: En esta carpeta, se pueden encontrar los componentes que llevan a cabo mostrar los listados de composiciones, autores, músicos y el resultado de la búsqueda de músicos que hay dentro de la aplicación
- Componente login-usuario: Este componente es el que se encarga de mostrar al usuario el formulario de login para loguearse y el que comprueba los datos y recibe los datos del usuario desde el servidor
- Carpeta modificaciones: Dentro de esta, se pueden ver los componentes que están dedicados a las modificaciones de datos de un usuario, músicos, composiciones y autores
- Carpeta servicios: En esta sección se encuentran los diferentes servicios implementados en el proyecto para hacer la conexión mediante el API REST con el servidor
- Carpeta vistas: Aquí están los diferentes componentes que se utilizan para mostrar los datos de perfil de cada clase
- App-routing.module.ts: Este es el archivo TypeScript en el cual se configuran las diferentes rutas para acceder a los componentes implementados
- App.module: Es el módulo que actúa de raíz, se encarga de iniciar el proyecto con los diferentes componentes, funcionalidades y servicios que son necesarios para inicializar el proyecto. Aquí es donde se introducen todos los componentes que se crean
- App.component.html: Es el archivo encargado de inicializar la primera vista que hemos implementado en el proyecto. En este proyecto, la vista principal es el listado de composiciones que hay en la aplicación

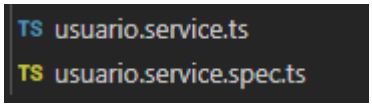
Imagen 46. Estructura de un componente



En esta imagen, se puede ver la estructura que tiene un componente el cual está compuesto por:

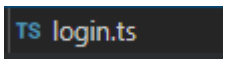
- Archivo css: en este archivo se realizan la introducción de los estilos específicos que se quieren utilizar para este componente
- Archivo html: aquí se implementa toda la vista HTML que va a ser vista por el usuario que utilice la aplicación
- Archivo ts: es el archivo TypeScript donde se realizan las diferentes funcionalidades necesarias para este componente. En este ejemplo de búsqueda de autor, este archivo tiene las funcionales de enviar los datos del autor a buscar y recibirá los datos del autor en caso de que exista
- Archivo spec.ts: archivo para ejecutar los tests del código del componente

Imagen 47. Estructura de un servicio



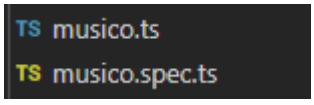
En esta imagen, se pueden contemplar los archivos que se crean cuando se hace un servicio nuevo. Toda la implementación que se hace en el servidor se ejecuta en el archivo TypeScript (ts). El archivo spec.ts es para realizar las pruebas internamente.

Imagen 48. Estructura de una interfaz



En esta ilustración, se muestra el archivo que se crea al componer una interfaz.

Imagen 49. Estructura de una clase



En esta fotografía, se pueden observar los dos archivos que se hacen al crear una nueva clase, como en el caso de los servicios, se trabaja solamente en el archivo ts.

4.3.Detalles sobre implementación

En este punto, se va a desarrollar los diferentes aspectos metodológicos y técnicos que se han utilizado en la implementación, recursos usados, diferentes bibliotecas y servicios externos utilizados. Se van a tratar las versiones, dependencias, configuraciones utilizadas etc. Alguna de esta información se acompañará con imágenes concretas del código implementado para ayudar en la comprensión.

4.3.1. Implementación en el servidor

Para implementar la parte del servidor se ha utilizado Java Enterprise Edition 11 Web. Para realizar el proyecto servidor se ha utilizado el entorno de desarrollo Eclipse y se ha creado un proyecto MAVEN que permite construir la aplicación y controlar las dependencias y versiones utilizadas en el desarrollo. La versión de Spring Boot en la que se ha trabajado ha sido la 2.2.6.RELEASE.

Tras la introducción del entorno en el que se ha trabajado se va a continuar con el desarrollo de los diferentes aspectos implementados en el proyecto.

4.3.1.1. Persistencia

Para ejecutar la persistencia de los datos con los que trabaja la aplicación se va a utilizar *Java Persistence API* (JPA) sobre el ORM, para así mediante el código implementado conseguir que las tablas de la base de datos relacional se autogeneren.

Con la utilización de este framework se puede utilizar el código en diferentes bases de datos haciendo sólo cambios en la configuración del archivo application.yml donde se introducen los datos de la base de datos que queremos utilizar.

La manera en la que se puede realizar el mapeo de las entidades es mediante el uso las diferentes anotaciones que nos aporta JPA sobre los atributos o métodos de las clases implementadas.

Imagen 50. Entidad BandaMusica con JPA.

```java
@Entity
public class BandaMusica {
    @Id
    String CIF;
    String nombre;
    String localidad;
    LocalDate fechaCreacion;
    @OneToOne(mappedBy="banda", cascade = CascadeType.MERGE, fetch=FetchType.EAGER)
    Usuario director;

    @OneToMany(mappedBy="banda", cascade = CascadeType.MERGE, fetch=FetchType.EAGER)
    List<Musico> musicos;

    public BandaMusica() {

    }

    public BandaMusica(String c, String nom, String loc, LocalDate fecha) {
        this.CIF = c;
        this.nombre = nom;
        this.localidad = loc;
        this.fechaCreacion = fecha;
        this.director = null;
        this.musicos = new ArrayList<>();
    }
```

En esta imagen se puede ver qué etiquetas se utilizan para mapear los datos, se va a explicar cada una de ellas:

- @Entity esta anotación especifica que esta clase debe ser persistida en BBDD
- @Id comunica a la BBDD cual es la clave primaria de la clase
- @OneToOne es utilizada para que la BBDD relacione dos entidades (en este caso, Usuario y BandaMusica), ya que, en este ejemplo, un usuario (si es director) puede tener una banda y una banda tiene un director. Esta relación es mapeada en la BBDD con esta etiqueta
- @OneToMany es una anotación similar a la anterior, pero en este caso, la relación es de uno a muchos, una banda puede tener muchos músicos y un músico sólo puede estar en una banda. Por lo que, esta hace que la BBDD guarde la relación que tiene una banda de música con todos sus músicos
- El atributo "*mappedBy*" que está dentro de las dos últimas anotaciones comentadas, es el que hace referencia al atributo por el cual están relacionadas las dos clases
- "*cascade*" es el atributo que indica de qué manera se hace la relación. En este caso, la opción que se utiliza es MERGE, el cual une las dos tablas guardando la clave primaria de una entidad en la tabla de la otra entidad con la que está relacionada

- "*fetch*" este nos proporciona el arreglo de los problemas que nos pueden dar las relaciones de tipo *lazy*. Por ello, la utilización de la opción EAGER es la que define las relaciones como *no-lazy*

Para la configuración de la BBDD hay que trabajar en el archivo application.yml en el cual metemos los datos necesarios para la *Uniform Resource Locator* (URL).

Imagen 51. Configuración BBDD

```
spring.datasource.url: jdbc:mysql://localhost:3306/TFG
spring.datasource.username: root
spring.datasource.password: password
spring.jpa.properties.javax.persistence.schema-generation.database.action: none
```

Como se puede ver en la imagen, la primera línea es para indicar la URL en la que está la BBDD; la segunda y tercera es para el usuario y la contraseña para trabajar con ella; y la última línea es para la acción que tiene que realizar el proyecto cada vez que se ejecute. En este caso, la opción utilizada es none, para que trabaje según la información que le llegue desde el cliente. Si se quiere borrar la base de datos siempre que se arranque, se puede utilizar la opción drop-and-create para borrar y crear la base de datos cada vez que arranque el servidor.

4.3.1.2. Beans

La clase Bean es utilizada para el almacenamiento de los datos para trabajar en nuestra aplicación. Esta clase está gestionada por el contenedor de Beans de Spring Boot.

Imagen 52. Clase Beans

```
@Repository
public class SistemaImpl implements Sistema{

    @Autowired
    UsuarioDAO usuarioDAO;

    @Autowired
    GeneroMusicalDAO generoMusicalDAO;

    @Autowired
    AutorDAO autorDAO;

    @Autowired
    BandaMusicaDAO bandaMusicaDAO;

    @Autowired
    MusicoDAO musicoDAO;

    public SistemaImpl() {
    }
```

Como se ve en la imagen, esta clase se realizan las inyecciones de dependencias de cada clase que va a utilizar este Beans para trabajar con los datos mediante la anotación @Autowired. Las clases que se inyectan son aquellas que tienen la etiqueta @Repository

Imagen 53. Interfaz DAO Músico

```
@Repository
public interface MusicoDAO extends JpaRepository<Musico, String>{
    List<Musico> findByNombreContainsIgnoreCase(String nombre);
}
```

Esta interfaz es un ejemplo de las diferentes interfaces que se utilizan en la aplicación, las cuáles son inyectadas en el Beans como se ha mencionado y mostrado antes.

4.3.1.3. Controladores

Los controladores son los que se utilizan para realizar la conexión entre el cliente y el servidor mediante la comunicación API REST.

Imagen 54. Controlador Rest

```
@CrossOrigin
@RestController
@RequestMapping("/gestorMusical")
public class ControladorRest {
    @Autowired
    Sistema sistema;

    @Autowired
    ModelMapper modelMapper;

    @PostMapping("/nuevoUsuario")
    @ResponseStatus(HttpStatus.CREATED)
    public void registrarUsuario(@Valid @RequestBody UsuarioDTO usuario) {
        Usuario u = modelMapper.map(usuario, Usuario.class);
        sistema.registrarUsuario(u);
    }

    @PostMapping("/usuario")
    @ResponseStatus(HttpStatus.ACCEPTED)
    public UsuarioDTO loginUsuario(@RequestBody UsuarioDTO usuario) {
        Usuario u = modelMapper.map(usuario, Usuario.class);
        return modelMapper.map(sistema.loginUsuario(u.getDNI(), u.getPassword()), UsuarioDTO.class);
    }
```

En esta imagen se puede ver la configuración de la clase Controlador y dos funciones que se utilizan en este controlador.

Se van a explicar las etiquetas:

- @CrossOrigin esta anotación se utiliza para que el controlador sea accesible desde cualquier puerto que utilice el cliente

- @RestController se utiliza para poder ser manejado por el Dispatcher Servlet
- @RequestMapping esta denomina la ruta de entrada al controlador
- @Autowired para la inyección de dependencias
- @PostMapping (enviar los datos) para realizar un POST al controlador y añadir un nuevo usuario o enviar los datos de login según la ruta que haya elegido el cliente. En este ejemplo, se ha utilizado @PostMapping, pero también están @GetMapping (obtener los datos), @PutMapping (modificar los datos), @DeleteMapping (borrar los datos) entre otros
- @ResponseStatus esta anotación es utilizada para mostrar el resultado que debe de salir del método utilizado si todo está correcto
- @Valid se explica en el siguiente punto
- @RequestBody este se hace uso para indicar que tipo de objeto es donde se almacenará la información recibida por el json introducido por el cliente

4.3.1.4. Validación

Para realizar una validación de datos de objetos, Java nos proporciona una dependencia que se encarga de esto. Hay diversas anotaciones que se utilizan en los DTOs para que se tengan en cuenta en la validación.

Imagen 55. Clase ejemplo @NotBlank

```
public class AutorDTO {
    @NotBlank(message = "El nombre es obligatorio")
    String nombre;
    LocalDate fechaNacimiento;
    String localidad;
    String estudios;
    String imagenPerfil;

    public AutorDTO() {

    }
```

En la imagen se puede ver, la anotación @NotBlank, esta es empleada para indicar que el nombre de un autor en este caso, no puede ser nulo y tiene que ser rellenado obligatoriamente.

Imagen 56. Función ejemplo @Valid

```
@PostMapping("/nuevoAutor")
@ResponseStatus(HttpStatus.CREATED)
public void crearAutor(@Valid @RequestBody AutorDTO autor) {
    Autor a = modelMapper.map(autor, Autor.class);
    sistema.crearAutor(a);
}
```

La anotación @Valid que se puede ver en la imagen anterior, es la encargada de ejecutar la comprobación de los datos recibidos para indicar si son correctos o no son correctos.

Esto lo realiza teniendo en cuenta las etiquetas que se encuentran en las clases DTOs utilizadas. En este ejemplo, comprobaría si el nombre del autor está vacío o no, ya que es obligatorio porque está especificado en la clase DTO del autor (véase Imagen 56).

4.3.1.5. Autenticación

En este apartado, para realizar la autenticación se ha utilizado el DNI y la contraseña del usuario para realizar las diferentes operaciones que puede utilizar un usuario registrado.

Imagen 57. Autentificación de usuario

```
@PostMapping("/usuario")
@ResponseStatus(HttpStatus.ACCEPTED)
public UsuarioDTO loginUsuario(@RequestBody UsuarioDTO usuario) {
    Usuario u = modelMapper.map(usuario, Usuario.class);
    return modelMapper.map(sistema.loginUsuario(u.getDNI(), u.getPassword()), UsuarioDTO.class);
}
```

Como se puede observar en dicha imagen, el controlador recibe el DNI y la *password* de un usuario que se quiere loguear desde el cliente, este comprueba esos datos y como respuesta, manda los datos del usuario para que pueda trabajar el cliente con ellos y poder acceder a las diferentes funcionalidades.

4.3.1.6. Tests de funcionalidades

En dicha sección, se va a mostrar un ejemplo de las pruebas unitarias realizadas a las funciones que están implementadas en el sistema bean y en el controlador rest de la aplicación.

Estas pruebas son utilizadas para comprobar que una función concreta del código implementado en la aplicación funciona correctamente.

Las diferentes pruebas son realizadas mediante JUnit que se encarga de ejecutar todos los tests que tengamos implementados y mostrar los resultados obtenidos de dichos tests.

Imagen 58. Prueba unitaria crearBanda en el sistema

```
@Test
@Order(7)
public void crearBanda() {
    LocalDate fecha = LocalDate.of(1870, 8, 10);
    BandaMusica banda = new BandaMusica("11222334445CF", "Banda de musica de Torredonjimeno", "Torredonjimeno", fecha);
    Usuario usuario = new Usuario("77333444-P", "Evaristo", "García", "evaristodirector@hotmail.com", "12345", true);
    Assertions.assertEquals("11222334445CF", sistema.crearBanda(usuario, banda).getCIF());
    Assertions.assertEquals(usuario.getNombre(), usuario.getBanda().getDirector().getNombre());
}
```

En esta imagen se puede ver el desarrollo de una prueba unitaria de dicha función, para realizar la creación de una banda, hay que crear un usuario que sea director, para poder mandar la información del director y de la banda al sistema para que este cree la banda.

La anotación @Test es para indicar que esta función es una prueba, @Order es para realizar las diferentes pruebas en un orden concreto.

Las líneas de Assertions son las utilizadas para comprobar que los resultados que nos devuelven las funciones a las que se ha llamado son los resultados que deberían ser los esperados.

Imagen 59. Prueba unitaria modificarMusico en el controlador Rest

```
@Test
@Order(17)
void modificarMusico() {
    String dni = "25333444L";
    String dniMusico = "77111222S";
    Musico m = sistema.buscarMusico(dniMusico);
    MusicoDTO musDTO = new MusicoDTO(m.getDni(), m.getNombre(), m.getEmail(), m.getTelefono(), m.getInstrumento());
    musDTO.setEmail("nuevocorreo@gmail.com");
    restTemplate.put("/usuario/"+dni+"/banda/musico/" + dniMusico, musDTO);

    m = sistema.buscarMusico(m.getDni());
    Assertions.assertEquals("nuevocorreo@gmail.com", m.getEmail());
}
```

Aquí se muestra una prueba unitaria de una función del controlador Rest, el objetivo es modificar los datos de un músico ya creado anteriormente en la aplicación, se introducen los datos necesarios por la URL que llama a la función de modificar un músico y tras realizar esta llamada comprobamos con el Assertions si el resultado de la modificación es el esperado.

4.3.2. Implementación en el cliente

Para la implementación que se ha realizado en el cliente se ha utilizado Angular 2. Para implementar el proyecto cliente se ha utilizado el entorno de desarrollo Visual Studio Code y se ha creado un proyecto angular.

Tras la introducción del entorno en el que se ha trabajado se va a continuar con el desarrollo de los diferentes aspectos implementados en el proyecto.

4.3.2.1. Módulos

Las aplicaciones que son desarrolladas en Angular están compuestas por un bloque NgModule que es el que se define como una colección de código dedicado a una funcionalidad.

Una aplicación Angular tiene un módulo llamado AppModule el cual es el módulo raíz de la aplicación y proporciona el mecanismo que inicia la aplicación.

Una aplicación está compuesta de muchos módulos funcionales. Estos módulos pueden importar diferentes funcionalidades de externos y también puede exportar su propia funcionalidad para que sea accesible por otros.

En la siguiente imagen se puede observar los diferentes módulos que componen el proyecto del cliente y que aportan diferentes funcionalidades.

Imagen 60. AppModule del cliente

```
@NgModule({
  declarations: [
    AppComponent,
    ComposicionesComponent,
    NuevaComposicionComponent,
    VistaComposicionComponent,
    VistaAutorComponent,
    NuevoAutorComponent,
    AutoresComponent,
    HeaderComponent,
    FooterComponent,
    NuevoUsuarioComponent,
    LoginUsuarioComponent,
    VistaUsuarioComponent,
    NuevaBandaComponent,
    VistaBandaComponent,
    NuevoMusicoComponent,
    MusicosComponent,
    VistaMusicoComponent,
    ModificarUsuarioComponent,
    ModificarAutorComponent,
    ModificarComposicionComponent,
    ModificarMusicoComponent,
    BusquedaComposicionComponent,
    ErrorBusquedaComponent,
    BusquedaAutorComponent,
    BusquedaMusicoComponent,
    MusicosEncontradosComponent,
    ErrorAutorComponent
  ],
```

```
  imports: [
    BrowserModule,
    AppRoutingModule,
    FormsModule,
    HttpClientModule,
    BrowserAnimationsModule,
    MatInputModule,
    MatDatepickerModule,
    MatNativeDateModule,
    ReactiveFormsModule
  ],
  providers: [GeneroMusicalServiceService, CookieService, AuthGuard],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

4.3.2.2. Rutas

Las rutas son aquellas que especifican la navegación concreta de cada componente para acceder este. Para introducir las diferentes URLs utilizadas por cada uno de estos elementos, angular tiene un archivo llamado app-routing.module.ts que es donde se almacena toda esta información.

En la siguiente ilustración se pueden distinguir las diferentes rutas que tienen cada uno de los componentes que son utilizados en el proyecto.

Imagen 61. App-routing del cliente

```
const routes: Routes = [
  {path: '', component: ComposicionesComponent},
  {path: 'nuevaComposicion', component: NuevaComposicionComponent},
  {path: 'composicion/:titulo', component: VistaComposicionComponent},
  {path: 'nuevoAutor', component: NuevoAutorComponent},
  {path: 'autor/:nombre', component: VistaAutorComponent},
  {path: 'autores', component: AutoresComponent},
  {path: 'usuario', component: LoginUsuarioComponent},
  {path: 'nuevoUsuario', component: NuevoUsuarioComponent},
  {path: 'perfilUsuario/:dni', component: VistaUsuarioComponent},
  {path: 'usuario/nuevaBanda', component: NuevaBandaComponent},
  {path: 'usuario/banda', component: VistaBandaComponent},
  {path: 'usuario/banda/nuevoMusico', component:NuevoMusicoComponent},
  {path: 'usuario/banda/listadoMusicos', component:MusicosComponent},
  {path: 'usuario/banda/musico/:dni', component:VistaMusicoComponent},
  {path: 'usuario/edicion', component:ModificarUsuarioComponent},
  {path: 'autor/edicion/:nombre', component:ModificarAutorComponent},
  {path: 'composicion/edicion/:titulo', component:ModificarComposicionComponent},
  {path: 'usuario/banda/musico/edicion/:dni', component:ModificarMusicoComponent},
  {path: 'busquedaComposicion', component: BusquedaComposicionComponent},
  {path: 'busquedaAutor', component:BusquedaAutorComponent},
  {path: 'busquedaMusico', component:BusquedaMusicoComponent},
  {path: 'usuario/banda/musicosEncontrados/:nombre', component:MusicosEncontradosComponent},
  {path: 'error', component:ErrorBusquedaComponent},
  {path: 'errorAutor', component:ErrorAutorComponent}
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```

4.3.2.3. Clases

Las clases son utilizadas para guardar la información de cada formulario que se emplea en el cliente y poder crear los objetos con la información que tiene la clase.

En la próxima imagen se observan los atributos que tiene un usuario y los que son necesarios desde un formulario para crear un objeto de esta clase, por ejemplo.

Imagen 62. Clase Usuario del cliente

```
export class Usuario {
    dni: String;
    nombre: String;
    apellidos: String;
    correo: String;
    password: String;
    esDirector: Boolean;

    getNombre(): String{
        return this.nombre;
    }

    getApellidos(): String {
        return this.apellidos;
    }

    getCorreo(): String {
        return this.correo;
    }

    getPassword(): String {
        return this.password;
    }

    getEsDirector(): Boolean{
        return this.esDirector;
    }
}
```

4.3.2.4. Componentes

Los componentes están compuestos de una clase codificada en TypeScript donde estarán almacenados los distintos datos con los que se trabaje y por una vista HTML que es lo que se muestra al usuario por pantalla.

Una aplicación Angular tiene como mínimo un componente el cual actúa como raíz. Este es el líder de la jerarquía de todos los componentes que están implementados en la aplicación.

Imagen 63. Componente TypeScript nuevoUsuario del cliente

```
export class NuevoUsuarioComponent implements OnInit {

  usuario: Usuario;
  usuarioForm: FormGroup;
  submitted = false;

  constructor(private route: ActivatedRoute,
    private router: Router,
    private usuarioService: UsuarioService,
    private formBuilder: FormBuilder) {
    this.usuario = new Usuario();
  }

  ngOnInit() {
    this.usuarioForm = this.formBuilder.group({
      dni: ['', [Validators.required, Validators.pattern('^[0-9]{8,8}[A-Za-z]$')]],
      nombre: ['', [Validators.required, Validators.minLength(4)]],
      apellidos: ['', [Validators.required, Validators.minLength(4)]],
      correo: ['', [Validators.required, Validators.pattern('^[a-z]+[a-z0-9._]+@[a-z]+\.[a-z.]{2,5}$')]],
      password: ['', [Validators.required, Validators.minLength(8)]],
      esDirector: ['']
    });
  }

  get f() { return this.usuarioForm.controls; }
```

```
  enviarDatos() {
    this.submitted = true;
    if (this.usuarioForm.invalid) {
      return;
    }
    else {
        this.usuario.dni = this.f.dni.value;
        this.usuario.nombre = this.f.nombre.value;
        this.usuario.apellidos = this.f.apellidos.value;
        this.usuario.correo = this.f.correo.value;
        this.usuario.password = this.f.password.value;
        this.usuario.esDirector = this.f.esDirector.value;
        this.usuarioService.save(this.usuario).subscribe(result => this.login());
    }
  }

  login() {
    this.router.navigate(['/usuario']);
  }

  volverInicio() {
    this.router.navigate(['/']);
  }
}
```

En esta imagen, se puede contemplar el código desarrollado en el componente nuevoUsuario. Este código tiene implementado un formulario por donde el usuario que esté utilizando la aplicación introducirá los datos, y en el TypeScript el proyecto los recibirá y lo creará si no hay ningún problema. La función de enviarDatos() es mediante la cual el cliente

los envía al servidor a través de un servicio para guardar la información nueva de este en la BBDD.

Las otras funciones login() y volverInicio() son aquellas con las que se navega hacia diferentes componentes mediante sus rutas específicas.

Imagen 64. Componente HTML nuevoUsuario del cliente

```
<body>
    <!-- Cabecera -->
    <app-header></app-header>
    <h1 class="text-center bg-light">Creacion de cuenta</h1>
    <br>
    <div style="justify-content: center;display: grid;">
        <div class="card my-5" style="margin: 30px; width: 530px;">
            <div class="card-body">
                <form [formGroup]="usuarioForm" (ngSubmit)="enviarDatos()">
                    <div class="form-group">
                        <h3 for="dni" class="bg-light text-center">DNI</h3>
                        <input type="text" class="form-control text-center" id="dni" name="dni" formControlName="dni"
                            placeholder="Introduzca su DNI" [ngClass]="{ 'is-invalid': submitted && f.dni.errors }">
                        <div *ngIf="submitted && f.dni.errors" class="invalid-feedback">
                            <div *ngIf="f.dni.errors.required">El DNI es obligatorio</div>
                            <div *ngIf="f.dni.errors.pattern">El formato tiene que ser 11111111A</div>
                        </div>
                    </div>
                    <div class="form-group">
                        <h3 for="nombre" class="bg-light text-center">Nombre</h3>
                        <input type="text" class="form-control text-center" id="nombre" name="nombre"
                            formControlName="nombre" placeholder="Introduzca su nombre"
                            [ngClass]="{ 'is-invalid': submitted && f.nombre.errors }">
                        <div *ngIf="submitted && f.nombre.errors" class="invalid-feedback">
                            <div *ngIf="f.nombre.errors.required">El nombre es obligatorio</div>
                            <div *ngIf="f.nombre.errors.minlength">Longitud mínima 4 caracteres</div>
                        </div>
                    </div>
                    <div class="form-group">
                        <h3 for="apellidos" class="bg-light text-center">Apellidos</h3>
                        <input type="text" class="form-control text-center" id="apellidos" name="apellidos"
                            formControlName="apellidos" placeholder="Introduzca sus apellidos"
                            [ngClass]="{ 'is-invalid': submitted && f.apellidos.errors }">
                        <div *ngIf="submitted && f.apellidos.errors" class="invalid-feedback">
                            <div *ngIf="f.apellidos.errors.required">Los apellidos son obligatorio</div>
                            <div *ngIf="f.apellidos.errors.minlength">Longitud mínima 4 caracteres</div>
                        </div>
                    </div>
                    <div class="form-group">
                        <h3 for="correo" class="bg-light text-center">Correo</h3>
                        <input type="text" class="form-control text-center" id="correo" name="correo"
                            formControlName="correo" placeholder="Introduzca su correo"
                            [ngClass]="{ 'is-invalid': submitted && f.correo.errors }">
                        <div *ngIf="submitted && f.correo.errors" class="invalid-feedback">
                            <div *ngIf="f.correo.errors.required">El correo es obligatorio</div>
                            <div *ngIf="f.correo.errors.pattern">El formato es ejemplo@ejemplo.com</div>
                        </div>
                    </div>
                    <div class="form-group">
                        <h3 for="password" class="bg-light text-center">Password</h3>
                        <input type="password" class="form-control text-center" id="password" name="password"
                            formControlName="password" placeholder="Introduzca su password"
                            [ngClass]="{ 'is-invalid': submitted && f.password.errors }">
                        <div *ngIf="submitted && f.password.errors" class="invalid-feedback">
                            <div *ngIf="f.password.errors.required">La password es obligatoria</div>
                            <div *ngIf="f.password.errors.minlength">Longitud mínima 8 caracteres</div>
                        </div>
                    </div>
                    <div class="form-group">
                        <h3 for="esDirector" class="bg-light text-center">¿Eres director/a?</h3>
                        <input type="checkbox" class="form-control text-center" id="esDirector" name="esDirector"
                            formControlName="esDirector" style="height: 25px;">
                    </div>
                    <div class="text-center">
                        <button type="submit" style="margin-right: 5px;"
                            class="btn btn-secondary">Crear Cuenta</button>
                        <button class="btn btn-secondary my-2 my-sm-0" style="margin-left: 5px;"
                            (click)="volverInicio()">Volver a Inicio</button>
                    </div>

                </form>

            </div>
        </div>
    </div>

    <br>
    <!-- Footer -->
    <app-footer></app-footer>
</body>
```

En dicha imagen, se puede ver el código HTML utilizado para crear la vista correspondiente para que el usuario pueda visualizar el formulario de creación de usuario. A través de esta vista el TypeScript del componente recibe los datos que ha introducido el usuario en el formulario que ha sido mostrado.

4.3.2.5. Servicios

Los servicios en angular son clases las cuales tienen como tarea hacer la comunicación entre el servidor y el cliente. Estos obtienen los datos que son introducidos por el usuario y hace la conexión con el servidor para que este reciba los datos y pueda trabajar con ellos. También es el que recibe los datos que son traídos desde el servidor para que pueda dárselos al cliente.

Imagen 65. Servicio GeneroMusical del cliente

```
export class GeneroMusicalServiceService {
  private composicionesUrl: string;
  private nuevaComposicionUrl: string;
  private vistaComposicionUrl: string;
  private autoresUrl: string;

  constructor(private http: HttpClient) {
    this.composicionesUrl = 'http://localhost:8080/gestorMusical/composiciones';
    this.nuevaComposicionUrl = 'http://localhost:8080/gestorMusical/nuevaComposicion';
    this.vistaComposicionUrl = 'http://localhost:8080/gestorMusical/composicion/';
    this.autoresUrl = 'http://localhost:8080/gestorMusical/autores';
  }

  public findAll(): Observable<GeneroMusical[]> {
    return this.http.get<GeneroMusical[]>(this.composicionesUrl);
  }

  public save(generoMusical: GeneroMusical){
    return this.http.post<GeneroMusical>(this.nuevaComposicionUrl,generoMusical);
  }

  public findById(titulo: String): Observable<GeneroMusical>{
    return this.http.get<GeneroMusical>(this.vistaComposicionUrl + titulo);
  }

  public modificar(titulo: String, generoMusical: GeneroMusical){
    return this.http.put<GeneroMusical>(this.vistaComposicionUrl + titulo, generoMusical);
  }

  public findByAutores(): Observable<Autor[]>{
    return this.http.get<Autor[]>(this.autoresUrl);
  }
}
```

En la ilustración presentada, se pueden ver las diferentes funciones que lleva este servicio a cabo entre el cliente y el servidor haciendo las llamadas correspondientes a las API REST implementadas en el servidor.

En este caso, se han creado cuatro variables para hacer referencia a las API REST correspondientes del servidor para hacer las consultas get, post, put etc. Cada una de las funciones hace la llamada y recibe o envía según sea lo correspondiente de la función.

5. **Conclusiones**

Para concluir este trabajo, en primer lugar, una serie de argumentos y razonamientos, sobre los diferentes puntos cumplidos de los objetivos iniciales para el desarrollo de la aplicación.

También se va a hablar de las sensaciones obtenidas durante el progreso del proyecto como, por ejemplo, de las tecnologías aplicadas, ventajas e inconvenientes que han surgido etc.

Para finalizar dicho apartado, se va a presentar una opinión personal sobre los conocimientos adquiridos en este trabajo como finalización de los obtenidos durante el grado.

5.1. Cumplimiento de los objetivos iniciales

En este proyecto, se comenzaba con la idea de buscar una solución a los usuarios apasionados de la música creando una aplicación donde pudieran obtener la información de los distintos géneros musicales que son interpretados por las diferentes bandas de músicas, así como poder escucharlas y comentarlas.

Por parte de los directores, el objetivo era poder utilizar la aplicación para que estos pudieran administrar su banda de música, así como los músicos que la componen teniendo las fichas de información correspondientes de cada uno de ellos.

Toda esta información está desarrollada en el apartado de análisis (véase apartado 2), donde se estudiaron los diferentes requisitos que debería de cumplir la aplicación para llegar a su objetivo principal.

Dentro de este apartado también se introdujeron los diferentes *frameworks* que se utilizarían en el desarrollo del sistema. Además, se pueden obtener los estudios iniciales tanto de coste como de tareas para tener en cuenta en el desarrollo. También se pueden observar el modelo del dominio el cuál sirve para hacer una idea de las clases que nos vamos a encontrar en la aplicación y el diagrama de caso de uso general donde se ven las diferentes tareas a las que van a tener acceso los usuarios.

En la sección de diseño se exponen los diferentes diagramas estudiados durante la rama de ingeniería del software, así como las metáforas y prototipo de la aplicación.

En el cuarto punto, se observan los objetivos de la implementación del proyecto web, en el que se comentan las tecnologías REST basadas en Spring Boot por parte del servidor y la

interfaz implementada en Angular por parte del cliente. En este, se comenta detalladamente el código implementado en cada una de las partes.

Se va a finalizar comentando los objetivos cumplidos y no cumplidos. En general, los puntos a desarrollar han sido realizados exceptos por algunos requisitos que no han sido posibles implementar en la aplicación. Estos requisitos serán comentados en el apartado de mejoras, pero gracias a la priorización de tareas y diseño del sistema propuesto, podrían añadirse fácilmente en una futura iteración sin afectar a la puesta en producción inicial del sistema realizado.

5.2. Algunas reflexiones sobre el proyecto

Se inicializa esta sección, comentando los frameworks utilizados en la aplicación. Los frameworks utilizados han sido totalmente acertados, ya que estos son muy sencillos de entender y de utilizar para el desarrollo de un proyecto web.

Por parte del servidor, Spring Boot es una tecnología bastante sencilla, la cual aporta muchísimas facilidades al programador para que se centre más bien en la implementación del código. Pues Spring Boot se encarga de la mayoría de la configuración interna como, por ejemplo, conexiones a base de datos diferentes, el mapeado de las clases a las bases de datos, recibir los datos como formato json para hacer la gestión de las consultas *Structured Query Language* (SQL) y transacciones, entre otras muchas facilidades más. Por mi parte, tenía un poco de experiencia en el desarrollo de un backend utilizando Spring Boot y ha sido bastante sencillo realizar el desarrollo del servidor de la aplicación ya que cómo se ha comentado antes, este framework es simple y sencillo.

Por parte del cliente, Angular también es una tecnología fácil de aprender y de utilizar, desde mi punto de vista, nunca había desarrollado un proyecto con este framework y me ha resultado bastante asequible y sencillo. En el desarrollo del cliente, se puede decir que los primeros pasos con Angular han sido lentos puesto que era una tecnología nueva y se necesitaba un tiempo de aprendizaje extra. Pero una vez adquiridos dichos conocimientos básicos de esta tecnología, el rendimiento sobre ella y el desarrollo es muy óptimo y eficiente. La forma de conectar el cliente con el servidor mediante API REST, la manera de recibir y enviar datos, la representación e implementación de las diferentes vistas HTML y las diferentes funcionalidades de un cliente son bastantes sencillas de implementar en Angular.

Tras realizar este TFG, mi opinión es positiva, ya que es un proyecto donde se desarrollan muchos conceptos aprendidos durante las diferentes asignaturas cursadas en el grado y se puede ver las relaciones que hay entre todas ellas y lo importante que son todos estos conocimientos en el desarrollo de una aplicación, desde el análisis hasta la implementación.

Para concluir, considero de manera muy favorable la realización del proyecto con los *frameworks* elegidos ya que estos son tecnologías recientes y nuevas que son utilizadas en el mundo empresarial. De esta manera, siento más seguridad cuando tenga que desarrollar una aplicación en este ámbito.

5.3. Mejoras y trabajos futuros

Este punto es desarrollado porque el trabajo realizado en este TFG, como es de comprender puede tener ciertas carencias. Hay diversos puntos donde se puede mejorar o incluir aportaciones que mejoren el estado actual de la aplicación para tener un mejor proyecto futuro. Por ejemplo, se podrían incluir los requisitos que no han sido introducidos. que son los siguientes (estos requisitos pueden encontrarse en el apartado 2.3):

- Requisito 4
- Requisito 6
- Requisito 7
- Requisito 8

También, se presenta una lista de mejoras para realizar en un futuro:

- Seguridad: tanto en el servidor como en el cliente, no se ha tenido tan en cuenta cuando se han implementado funcionalidades relacionadas con los datos de los usuarios
- Servidor: cuando se ha implementado cualquier funcionalidad que trabajaba con los datos del usuario, se trabaja con el DNI de este, en lugar de utilizar un token que lo haga más seguro
- Cliente: cuando almacena los datos de la sesión, guarda el DNI del usuario en el LocalStorage del navegador, cuando también se podría guardar un token que represente ese DNI
- Código: este se ha tratado de organizar de la forma más adecuada posible, usando los diferentes identificadores en cada proyecto, como funciones específicas de Spring Boot (beans, interfaz, entidades…) o de Angular (servicios, clases, componentes entre otros)

- Funcionalidad: se pueden añadir más funcionalidades que no se hayan podido cumplir a lo largo del progreso de la aplicación como, por ejemplo, que un director pueda crear un listado de ensayos para organizar qué músicos asisten a los ensayos propuestos
- Aspecto visual: seguramente habrá mejoras que se puedan introducir para hacer la aplicación mucho más simple e intuitiva para cualquier usuario

Este tipo de mejoras deberían estudiarse con un cliente real que planteara sus necesidades específicas al respecto.

6. **Bibliografía**

Libros o monografías:

[1] BAHIT. E. (2012). *Scrum & eXtreme Programming para programadores.* Editorial: safeCreative.

[2] COHN. M. (2004). *User Stories Applied. For Agile Software Development.* Editorial: Addison-Wesley.

[3] FREEDAM, A. (2018). *Pro Angular 6.* Editorial: Apress.

[4] PRESSMAN, R.S. (2010). *Ingeniería del Software. Un enfoque práctico.* P.101. Editorial: McGraw-Hill

[5] SHARMA. S (2019). *Mastering Microservices with Java.* Packt Publishing Ltd

[6] WALLS. C. (2016). *Spring Boot In Action.* Editorial: Manning

Páginas web:

Spring Boot. [Online]. Disponible: https://spring.io/projects/spring-boot Última visita: 13/04/2020 12:30

Cecilio Álvarez Caules. (marzo, 2016). *¿Qué es Spring Boot?.* Blog sobre Java EE. [Online]. Disponible: https://www.arquitecturajava.com/que-es-spring-boot/ Última visita: 13/04/2020 12:30

Spring boot: tutorial para crear aplicaciones en Java. (septiembre, 2019). Digital Guide IONOS. [Online]. Disponible: https://www.ionos.es/digitalguide/paginas-web/desarrollo-web/spring-boot-tutorial/ Última visita: 13/04/2020 12:30

Angular (framework). (agosto, 2020). Wikipedia. [Online]. Disponible: https://es.wikipedia.org/wiki/Angular_(framework) Última visita: 13/04/2020 13:30

Introduction to the Angular Docs. Angular. [Online]. Disponible: https://angular.io/docs Última visita: 13/04/2020 13:30

Víctor. *¿Qué es Angular y para qué sirve?.* Víctor Robles. [Online]. Disponible: https://victorroblesweb.es/2017/08/05/que-es-angular-y-para-que-sirve/ Última visita: 13/04/2020 13:30

Alberto Basalo (junio, 2016). *Introducción a Angular*. Desarrolloweb.com. [Online]. Disponible: https://desarrolloweb.com/articulos/introduccion-angular2.html Última visita: 13/04/2020 13:30

J. Carlos Valerio Barreto. (noviembre, 2016). *Angular 2: Historia, características y métodos de instalación.* Medium. [Online]. Disponible: https://medium.com/@jc.valerio.b/angular-2-historia-caracter%C3%ADsticas-y-m%C3%A9todos-de-instalaci%C3%B3n-11492ea67e2b Última visita: 14/04/2020 12:30

Gastón Iriarte. (octubre, 2018). *¿Para qué nos sirve Docker?*. DataArt. [Online]. Disponible: https://www.dataart.com.ar/news/para-que-nos-sirve-docker/ Última visita: 18/08/2020 10:00

¿Qué es Docker?. Red Hat. [Online]. Disponible: https://www.redhat.com/es/topics/containers/what-is-docker Última visita: 18/08/2020 10:00

Proconsi. (agosto, 2014). *App de Música para bandas y agrupaciones musicales.* Slideshare. [Online]. Disponible: https://es.slideshare.net/Proconsi/app-de-msica-para-bandas-y-agrupaciones-musicales-compartitura Última visita: 30/07/2020 13:45

5 aplicaciones para músicos que debes llevar en tu smartphone. Musicalizza. [Online]. Disponible: https://musicalizza.com/aplicacionesparamusicos/ Última visita: 30/07/2020 16:30

Gestión de Bandas de música, Orquestas y Bandas de Cornetas y Tambores. Anapi. [Online]. Disponible. https://www.anapi.com/lb/labanda1.htm Última visita: 30/07/2020 16:30

Historia de usuario. (marzo,2014). ScrumManager. [Online]. Disponible: https://www.scrummanager.net/bok/index.php/Historia_de_usuario Última visita: 14/04/2020 13:00

Método MoSCoW. (abril, 2013). Jummp. [Online]. Disponible: https://jummp.wordpress.com/2013/04/27/metodo-moscow/ Última visita: 14/04/2020 13:15

Julio Roche. *Técnicas de estimación en SCRUM.* Deloitte. [Online]. Disponible: https://www2.deloitte.com/es/es/pages/technology/articles/tecnicas-de-estimacion-en-scrum.html Última visita: 29/07/2020 12:25

Diagramas de casos de uso. UNAD. [Online]. Disponible: http://stadium.unad.edu.co/ovas/10596_9839/diagramas_de_casos_de_uso.html Última visita: 30/07/2020 11:45

Karla Cevallos. (junio, 2015). *UML: Casos de Uso.* Ingeniería del Software. [Online]. Disponible: https://ingsotfwarekarlacevallos.wordpress.com/2015/06/04/uml-casos-de-uso/ Última visita: 30/07/2020 11:45

Oscar Blancarte. (noviembre, 2018). *Data Transfer Object (DTO) – Patrón de diseño.* Blog de Oscar Blancarte. [Online]. Disponible: https://www.oscarblancarteblog.com/2018/11/30/data-transfer-object-dto-patron-diseno/ Última visita: 01/08/2020 17:30

Manuel Cillero. *Diagrama de Secuencia.* Manuel-cillero.es. [Online]. Disponible: https://manuel.cillero.es/doc/metrica-3/tecnicas/diagrama-de-interaccion/diagrama-de-secuencia/ Última visita: 02/08/2020 11:30

OKDIARIO. (mayo, 2015). *¿Qué son las metáforas y ejemplos?.* OKDIARIO. [Online]. Disponible: https://okdiario.com/curiosidades/que-son-metaforas-ejemplos-2274559 Última visita: 18/08/2020 19:15

Eliza Arias. (marzo, 2020). *Significado de metáfora.* Significados. [Online]. Disponible: https://www.significados.com/metafora/ Última visita: 18/08/2020 19:15

¿Qué es un prototipo?. MPIU+A. [Online]. Disponible: https://mpiua.invid.udl.cat/fases-mpiua/prototipado/que-es-un-prototipo/ Última visita: 18/08/2020 19:30

spring-boot-vue-example-spring-data-jpa-rest-api-postgresql-architecture-server. (septiembre, 2018). grokOnoz. [Online]. Disponible: https://grokonez.com/spring-framework/spring-data/spring-boot-vue-example-spring-data-jpa-rest-postgresql-crud-example/attachment/spring-boot-vue-example-spring-data-jpa-rest-api-postgresql-architecture-server Última visita: 17/08/2020 11:30

Jnj. (marzo, 2013). *¿Qué es un bean de Java?.* JnjSite.com. [Online]. Disponible: https://jnjsite.com/que-es-un-bean-de-java/ Última visita: 17/08/2020 11:45

Alejandro Ugarte. (febrero, 2020). *Building a Web Application with Spring Boot and Angular.* Baeldung. [Online]. Disponible: https://www.baeldung.com/spring-boot-angular-web Última visita: 03/08/2020 9:30

Angular 7 – Components. Tutorialspoint.com. [Online]. Disponible: https://www.tutorialspoint.com/angular7/angular7_components.htm Última visita: 03/08/2020 9:30

Angular – Cómo crear rutas y componentes de forma sencilla. (mayo, 2018). Codings Potions. [Online]. Disponible: https://codingpotions.com/angular-componentes-routing Última visita: 03/08/2020 9:30

Francisco Bello Díaz. (septiembre, 2019). *Como integrar el framework Bootstrap en un proyecto Angular*. Medium. [Online]. Disponible: https://medium.com/@fbellod/como-integrar-el-framework-bootstrap-en-un-proyecto-angular-a5d53fa79e03 Última visita: 04/08/2020 10:30

Dhiraj. (enero, 2019). *Spring Boot + Angular 5 + Spring Data + Rest Example (CRUD).* Devglan. [Online]. Disponible: https://www.devglan.com/spring-boot/spring-boot-angular-example Última visita: 07/08/2020 18:30

Víctor. *Como incrustar un vídeo de Youtube en Angular.* Víctor Robles. [Online]. Disponible: https://victorroblesweb.es/2019/04/15/como-incrustar-un-video-de-youtube-en-angular/ Última visita: 12/08/2020 9:30

Salarios para empleos de Programador/a Junior en España. Indeed. [Online]. Disponible: https://es.indeed.com/salaries/programador-junior-Salaries Última visita: 03/09/2020 11:30

Salarios para empleos de Analista programador/a Junior en España. Indeed. [Online]. Disponible: https://es.indeed.com/salaries/analista-programador-Salaries Última visita: 03/09/2020 11:30

7. Apéndices

7.1. Apéndice I. Manual del Entorno de desarrollo

En este apéndice, se va a explicar los diferentes pasos para instalar las herramientas utilizadas para desarrollar la aplicación web y poder compilarla y ejecutarla en los distintos ordenadores.

Para los diagramas presentados en los puntos de análisis y diseño se ha utilizado el programa de Visual Paradigm 16.0, este programa está preparado para la creación de los diversos tipos de diagramas que se realizan durante el desarrollo de un proyecto web.

Para realizar la descarga de este programa, hay que acceder a la página oficial de Visual Paradigm al apartado de descargas que se puede entrar desde este enlace: https://www.visual-paradigm.com/download/

En este link, se puede obtener la versión FREE TRIAL del programa. Para obtener la versión oficial se necesita una licencia, que en esta ocasión ha sido proporcionada por la Universidad de Jaén.

Tras descargar el programa, ejecutamos el archivo .exe de instalación y se realizan los pasos correspondientes para hacerse con los servicios de este. De esta manera obtendremos el programa Visual Paradigm para poder trabajar con los distintos diagramas.

Imagen 66. Logotipo de Visual Paradigm

Se prosigue con los programas utilizados para el desarrollo del servidor de la aplicación.

En este caso, han sido utilizados estos programas:

- Eclipse

Este programa, se ha utilizado para implementar el código correspondiente a la parte del servidor de la aplicación. Para realizar su descarga, se puede desde este enlace: https://www.eclipse.org/downloads/ .

Tras la descarga, ejecutamos el archivo de instalación y comenzará el proceso con los pasos de configuración pertinentes. Se realizan dichos pasos y cuando estos se acaben, se tendrá el programa instalado y listo para comenzar a programar.

Este programa es gratuito y no necesita ninguna licencia para que se pueda trabajar con todas las funcionalidades que ofrece este.

Para este proyecto son necesarios instalar los siguientes *plugins* en el programa:

- Maven (Java EE) integration for Eclipse WTP
- Wild Web Developer
- Spring Tools

Para la instalación de estos hay que seleccionar la opción *help* de eclipse y después entrar en *marketplace* y buscar en la tienda los *plugins* correspondientes.

La versión de Java con la que se ha trabajado ha sido OpenJDK 11 la cual se puede obtener aquí: https://adoptopenjdk.net/ . Se descarga cierta versión y se guarda en carpetas localizables para el proyecto.

Imagen 67. Logotipo de Eclipse

- Docker

La función de docker en el proyecto ha sido la de proporcionar un contenedor en el cual se ha creado la base de datos con la que se ha trabajado. Esta función ha facilitado la configuración para trabajar con dicha base de datos.

Para obtener esta aplicación, se accede a este link: https://www.docker.com/products/docker-desktop .

Hay dos opciones de descarga *"stable"* o *"edge"* en este caso, se necesita el programa con la versión *"stable"*, por defecto, aparece esa opción para descargar.

Tras realizar la descarga, se ejecuta el archivo .exe para proceder a su instalación y se realizan los diversos pasos de configuración necesarios. Cuando el programa esté instalado en el ordenador, para que este se ejecute correctamente hay que activar la virtualización en la máquina y en el Sistema Operativo (SO) que estemos utilizando.

Para realizar esta activación se puede seguir el procedimiento de esta página, donde están explicados los pasos a seguir: https://docs.microsoft.com/es-es/virtualization/hyper-v-on-windows/quick-start/enable-hyper-v .

Se ha de mencionar que esta activación es para ejecutarla en Windows.

Para ver el logotipo de Angular diríjase a la página 19 y véase Imagen 10.

- Postman

Este programa se ha usado para la realización de pruebas de la API REST implementada en el servidor. A través de este, se pueden acceder a las diferentes rutas creadas para comprobar si su funcionamiento es correcto. Este programa tiene las diferentes opciones que se pueden ejecutar en una API REST.

Para obtener dicho programa se visita esta página: https://www.postman.com/downloads/ . Tras la descarga, se procede a ejecutar el archivo que nos trae el programa y se realiza el seguimiento de los pasos. Cuando el programa esté en nuestro sistema, ya se puede hacer uso de este y poder trabajar sin ninguna restricción.

Imagen 68. Logotipo de Postman

Se procede a continuación a explicar la instalación de los programas utilizados para el desarrollo en la parte del cliente.

Estos han sido los programas instalados para la implementación:

- NodeJS

Este programa es necesario debido a que se utiliza como conductor de otras aplicaciones que se usan en el proyecto. Con este se pueden instalar otras herramientas o gestionar las dependencias de la aplicación.

Para obtener el programa se puede hacer mediante este enlace: https://nodejs.org/es/ . Tras realizar la descarga, se ejecuta el archivo de instalación y se sigue el procedimiento necesario para este proceso.

En estos pasos comentados, Windows lanzará una ventana de Windows Firewall en la cual pide permiso para acceder a las diferentes redes. En este punto, hay que permitir que tenga acceso y seguir con la guía correspondiente.

Imagen 69. Logotipo de Node

- Angular

Tras la instalación del programa Node explicado anteriormente, ahora se procede a instalar el programa Angular. Este se instala gracias a las diferentes opciones que presenta Node.

Para instalar Angular, se tiene que abrir el terminal de nuestro ordenador e introducir el siguiente comando: npm i @angular/cli@8.3.25

La versión 8.3.25 es con la que se ha trabajado en este proyecto, por ello, es la que se explica en el manual de instalación.

Tras ejecutar el comando comentado, se tendrá instalado Angular en el ordenador y listo para ser trabajado.

Para crear un proyecto Angular en el cual, se pueda empezar a trabajar se ejecuta el siguiente comando: ng new NOMBRE_APP.

Para instalar las dependencias necesarias se introduce: npm install

Tras realizar estos pasos, ya se tiene un proyecto Angular preparado para ser implementado por parte del usuario.

Para ver el logotipo de Angular diríjase a la página 18 y véase Imagen 9.

- Visual Studio Code

Para realizar la implementación del código por parte del cliente ha sido utilizado este programa. Este lo podemos obtener visitando este link: https://code.visualstudio.com/

Cuando se descargue, se ejecuta el archivo .exe de instalación y se prosiguen los pasos que se proporcionan para conseguir obtener el programa en nuestro ordenador.

El programa es gratuito y no necesita licencias para obtener todas las funcionalidades que ofrece al usuario, por tanto, cuando la instalación esté terminada en el ordenador, ya se podrá comenzar a trabajar sin restricciones en él.

Imagen 70. Logotipo de Visual Studio Code

Y, para terminar, se va a proceder a explicar la instalación del programa utilizado para ver si el funcionamiento de la aplicación web es correcto y las diferentes vistas que contiene la interfaz.

- Google Chrome

Como ya se ha comentado antes, este navegador ha sido el utilizado para ver la interfaz de la aplicación web desarrollada y comprobar que el funcionamiento de esta es correcto y no presenta fallos.

Para descargar esta aplicación se puede realizar desde este link: https://www.google.com/chrome/

Tras la descarga de este, ejecuta el archivo descargado y continúa con los pasos correspondientes para el proceso de instalación y cuando ya esté terminado, estará disponible para el uso del usuario sin restricción alguna.

Imagen 71. Logotipo de Google Chrome

7.2. Apéndice II. Descripción de contenidos suministrados

Este TFG se entrega mediante un archivo .zip el cual contiene la siguiente estructura:

- Memoria: Este documento PDF es donde se encuentra toda la información realizada en el presente trabajo.
- Aplicación TFG.zip: Este archivo está compuesto por:
    - BandAndMusicAppServidor.zip: Aquí se encuentra el código fuente del proyecto del servidor. Este debe ser abierto en Eclipse, para poder trabajar con él
    - BandAndMusicAppCliente.zip: En este archivo se puede observar la implementación por parte del cliente. Para funcionar, se tiene que abrir con Visual Studio Code
    - Band & Music App.vpp: Este archivo contiene todos los diagramas realizados en este proyecto mediante el programa Visual Paradigm