Escuela Superior
de Ingeniería y Tecnología
Universidad de La Laguna

Trabajo de Fin de Grado

# Aplicación Full Stack para músicos basada en Crowdfunding

*Full Stack App for Musicians Based on Crowdfunding*

Jonay Estévez Díaz

La Laguna, 13 de julio de 2023

D. **Alejandro Pérez Nava**, con N.I.F. 43.821.179-S profesor asociado de Universidad adscrito al Departamento de Ingeniería Informática de la Universidad de La Laguna, como tutor

D. **Francisco Javier Rodríguez González**, con N.I.F. 43.618.712-V profesor asociado de Universidad adscrito al Departamento de Ingeniería Informática de la Universidad de La Laguna, como cotutor

**C E R T I F I C A (N)**

Que la presente memoria titulada:

”*Aplicación Full Stack para músicos basada en Crowdfunding*”

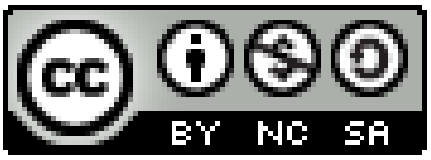
ha sido realizada bajo su dirección por D. **Jonay Estévez Díaz**, con N.I.F. 78.645.288-P.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 13 de julio de 2023

# Agradecimientos

A mis familiares, amigos y pareja.

# Licencia



**Resumen**

*La presente memoria aborda el desarrollo de la aplicación full stack CrowdSound, cuyo propósito es facilitar el financiamiento de discos, conciertos y merchandising para músicos nuevos en la industria a través de una plataforma de crowdfunding.*

*El enfoque principal del trabajo se centra en la implementación de las tecnologías MERN (MongoDB, Express, React, Node.js) para construir una aplicación web robusta y escalable. Se describe la arquitectura y la estructura de la aplicación, resaltando la importancia de cada una de las tecnologías utilizadas y como se integran entre sí.*

*Además, se analizan las funcionalidades clave de CrowdSound, como la creación de usuarios, gestión de la información presentada por el artista y a las contribuciones de los mismos. Se profundiza en los aspectos técnicos y de diseño para lograr una experiencia de usuario intuitiva y atractiva.*

*Finalmente, se lleva a cabo una evaluación exhaustiva de CrowdSound, destacando los resultados obtenidos, las mejoras realizadas y las posibles vías de desarrollo futuro.*

*En resumen, esta memoria del TFG presenta el desarrollo de la aplicación CrowdSound, basada en las tecnologías MERN, que proporciona una plataforma de crowdfunding para músicos emergentes. Se cubren aspectos técnicos y funcionales.*

**Palabras clave:** Full Stack, CrowdSound, Tecnologías MERN, Financiamiento, Crowdfunding, Músicos nuevos, Experiencia de usuario, Proyectos.

**Abstract**

*This report addresses the development of the full-stack application CrowdSound, which aims to facilitate the funding of albums, concerts, and merchandise for emerging musicians in the industry through a crowdfunding platform. The main focus of the work is centered around the implementation of the MERN technologies (MongoDB, Express, React, Node.js) to build a robust and scalable web application. The architecture and structure of the application are described in detail, highlighting the importance of each technology used and how they integrate with each other.*

*Furthermore, key functionalities of CrowdSound are analyzed, such as user creation, management of artist-presented information, and user contributions. Technical and design aspects are explored to achieve an intuitive and appealing user experience.*

*Finally, a comprehensive evaluation of CrowdSound is conducted, highlighting the obtained results, improvements made, and potential avenues for future development.*

*In summary, this Bachelor's thesis presents the development of the CrowdSound application based on the MERN technologies, providing a crowdfunding platform for emerging musicians. Both technical and functional aspects are covered.*

**Keywords:** Full Stack, CrowdSound, MERN Technologies, Financing, Crowdfunding, New musicians, User experience, Projects.

# Índice general

# Índice de Figuras

# Índice de Tablas

# Capítulo 1

# Introducción

## 1.1. Problemática

Los músicos emergentes en la industria musical a menudo se enfrentan a desafíos relacionados con la visibilidad y la financiación de sus proyectos. Esta problemática surge debido a la naturaleza altamente competitiva y saturada de la industria, lo que dificulta que muchos de estos músicos noveles logren que sus proyectos destaquen.

La industria musical presenta numerosos desafíos y problemáticas para los nuevos músicos que desean ingresar en ella. Algunas de estas dificultades se encuentran en la feroz competencia que caracteriza a esta industria, la cual se encuentra altamente saturada. Existen miles de artistas emergentes que buscan destacarse, lo que dificulta el acceso a contratos discográficos, oportunidades de presentaciones en vivo y la atención de los medios.

Por otro lado, la mayoría de estos músicos noveles carecen de una red establecida, lo cual también dificulta el establecimiento de contactos y la conexión con profesionales de la industria, como productores, sellos discográficos y agentes de reserva. Esta falta de conexiones limita el crecimiento y la promoción del propio músico.

Producir música de calidad y promocionarla requiere una inversión económica significativa, y es en este aspecto donde radica una de las mayores dificultades a las que se enfrenta un artista emergente. Esta limitación financiera disminuye considerablemente su capacidad para competir en la industria y llegar a un público más amplio. Sin embargo, muchos músicos emergentes han encontrado la manera de superar estos obstáculos y abrirse paso en la industria.

Con el objetivo de abordar esta situación, se plantea el desarrollo de una aplicación web basada en crowdfunding que permita a los artistas registrados visibilizar y financiar sus proyectos de manera efectiva y sencilla. La aplicación proporcionará a los artistas una plataforma para gestionar sus proyectos, así como centralizar su información personal y profesional asociada a los proyectos activos y los datos que ellos mismos proporcionen.

Además de acceder a los proyectos musicales, los usuarios también podrán participar y apoyar a los artistas que elijan dentro de la aplicación. La plataforma ofrecerá facilidades para realizar pagos y devolver las aportaciones realizadas por los usuarios.

Al desarrollar esta aplicación de crowdfunding para músicos noveles, se pretende solventar los desafíos de visibilidad y financiamiento a los que se enfrentan, permitiéndoles promover sus proyectos de manera más efectiva y obtener los recursos necesarios para llevar a cabo sus metas musicales.

## 1.2. Antecedentes y estado actual

El crowdfunding, también conocido como financiación colectiva, financiación en masa o micromecenazgo, se refiere a la práctica en la que las personas se unen en red para obtener fondos u otros recursos destinados a un proyecto o propósito específico. Este fenómeno se utiliza en una amplia gama de actividades, tanto industriales como creativas, campañas políticas, creación de empresas, así como en proyectos sociales y culturales.

Estas nuevas formas de financiamiento son un claro ejemplo de que la crisis experimentada en el ámbito social, cultural y solidario no se limita únicamente a lo económico, sino que también afecta a los modelos que ya no resultan efectivos, así como a la adaptación a nuevos modelos que aún nos resultan desafiantes. [1]

Existen algunas plataformas de crowdfunding famosas como Kickstarter y Patreon, que no están centradas en proyectos musicales, ya que puedes realizar propuestas de cualquier índole. Por otro lado, existen plataformas especializadas en este ámbito como PledgeMusic, donde se permite a los músicos y bandas financiar y promover sus proyectos musicales, así como interactuar directamente con sus seguidores. Los fans pueden respaldar proyectos y recibir recompensas exclusivas, como descargas anticipadas, mercancía autografiada y experiencias únicas.

El panorama actual de la industria musical ha facilitado la distribución digital y la producción, ya que el auge del streaming ha cambiado el escenario. Plataformas como Spotify, Apple Music, Amazon Music y YouTube Music han experimentado un crecimiento significativo en los últimos años, reemplazando en gran medida las descargas y la compra física de música. El streaming ha permitido un acceso mucho más amplio a la música, lo que ha contribuido a que este camino no esté tan obstaculizado.

Los videos musicales también han adquirido una importancia aún mayor gracias a YouTube y las redes sociales en general. La creación de contenido visual que acompañe a la música lanzada ha permitido llegar a un público más amplio y, a su vez, que ese contenido sea compartido por los usuarios.

Además, las redes sociales se han convertido en herramientas fundamentales para la promoción de la música y la creación de una base de fans. Su impacto permite establecer conexiones directas y personales entre artistas y fans, y también contribuyen a la globalización de la música al llegar a audiencias de diferentes partes del mundo sin necesidad de grandes inversiones en promoción. Las colaboraciones entre músicos también han contribuido a este proceso.

A pesar de todos estos aspectos, es importante tener en cuenta que la industria musical es un entorno en constante cambio y evolución. La tecnología y los cambios en los hábitos de consumo de música siguen avanzando rápidamente, lo que requiere una adaptación constante por parte de los artistas y profesionales de la industria.

## 1.3. Fases del desarrollo

El desarrollo de la aplicación se ha dividido en fases, las cuales se han realizado en el mismo orden en el que aparecen a continuación:

- **Diseño de modelos y estructura**
  - Diseño de los modelos presentes en la base de datos
  - Diseño de los middlewares, API y estructura del Back End
  - Mockups de las diferentes partes del Front End
- **Creación de la base de datos**
  - Creación de organización en MongoDB Atlas
  - Creación de proyecto dentro de la organización
  - Despliegue de la base de datos
- **Desarrollo del Servidor (Back End)**
  - Sistema de Autenticación
  - Sistema de Usuarios
  - Sistema de Newsletter
  - Sistema de Artistas
  - Sistema de Discos
  - Sistema de Conciertos
  - Sistema de Merchandising
- **Desarrollo del Cliente (Front end)**
  - Sistema de Rutas
  - Sistema de Autenticación
  - Sistema de Sesión
  - Sistema y páginas de Usuarios
  - Sistema y página de Newsletter
  - Sistema y páginas de Artistas
  - Sistema y páginas de Discos
  - Sistema y páginas de Conciertos
  - Sistema y páginas de Merchandising
  - Página de Datos
- **Despliege**

## 1.4. Estructura del documento

- **Capítulo 2** - Solución adoptada

  Se proporciona una descripción funcional de la solución al problema, así como un esquema de la misma.

- **Capítulo 3** - Tecnologías

  Se explican las diferentes tecnologías que se han utilizado para llevar a cabo el desarrollo de la aplicación.

- **Capítulo 4** - Desarrollo

  Se proporciona una explicación minuciosa de las etapas divididas en tres segmentos del desarrollo: Base de datos, Back End y Front End, junto con los pasos realizados en cada uno de ellos.

- **Capítulo 5** - Despliegue

  Se detalla la configuración del despliegue de los diferentes módulos en este proyecto.

- **Capítulo 6** - Conclusiones y líneas futuras

  Se presentan los resultados obtenidos tras completar este proyecto, así como posibles mejoras que podrían implementarse en el futuro para enriquecer la experiencia del usuario.

- **Capítulo 7** - Summary and Conclusions

  Se presenta el mismo contenido del apartado anterior pero adaptado al inglés.

- **Capítulo 8** - Presupuesto

  Se obtienen los costes del proyecto tanto de desarrollo como de despliegue y alojamiento.

# Capítulo 2

# Solución Adoptada

## 2.1. Descripción del producto

Se ha optado por desarrollar una aplicación web que se divide en dos áreas distintas. La primera es la zona web, es accesible para cualquier persona, sin importar si están registradas en la aplicación o no. En esta sección, se encuentra una página principal que explica el propósito del proyecto, así como enlaces directos a los principales proyectos de crowdfunding, como discos, conciertos y merchandising. Además, se pueden explorar los distintos artistas y acceder a información detallada sobre ellos, incluyendo sus discos, conciertos y merchandising creados.

Por otro lado, también se puede acceder a todos los proyectos de manera separada, clasificados según sean discos, conciertos o merchandising. Al seleccionar un proyecto específico, se muestra información completa sobre el mismo, incluyendo su estado de financiamiento y el artista responsable de su creación.

Además, se proporciona una página que recopila todos los datos de la aplicación, como el número de artistas y proyectos disponibles, junto con gráficos que muestran la cantidad de proyectos fallidos, en curso y exitosos.

La segunda zona de la aplicación, denominada zona de autenticación, requiere iniciar sesión para acceder. Sin embargo, se ha habilitado una página especial para el registro e inicio de sesión, que está disponible para todos los usuarios. Al registrarse, se pueden crear dos tipos diferentes de cuentas: usuarios comunes y artistas. Los usuarios comunes tienen la capacidad de administrar su participación en los proyectos, es decir, comprar o devolver productos. Por otro lado, los artistas tienen la posibilidad de crear, actualizar y eliminar cualquiera de sus proyectos, así como gestionar su página de artista, incluyendo la opción de eliminarla si así lo desean. Ambos tipos de usuarios tienen la capacidad de gestionar su cuenta personal, actualizando o eliminándola según sea necesario. Además, pueden modificar su contraseña y decidir si desean suscribirse o no a las noticias y actualizaciones de la aplicación.

## 2.2. Esquema de funcionalidad

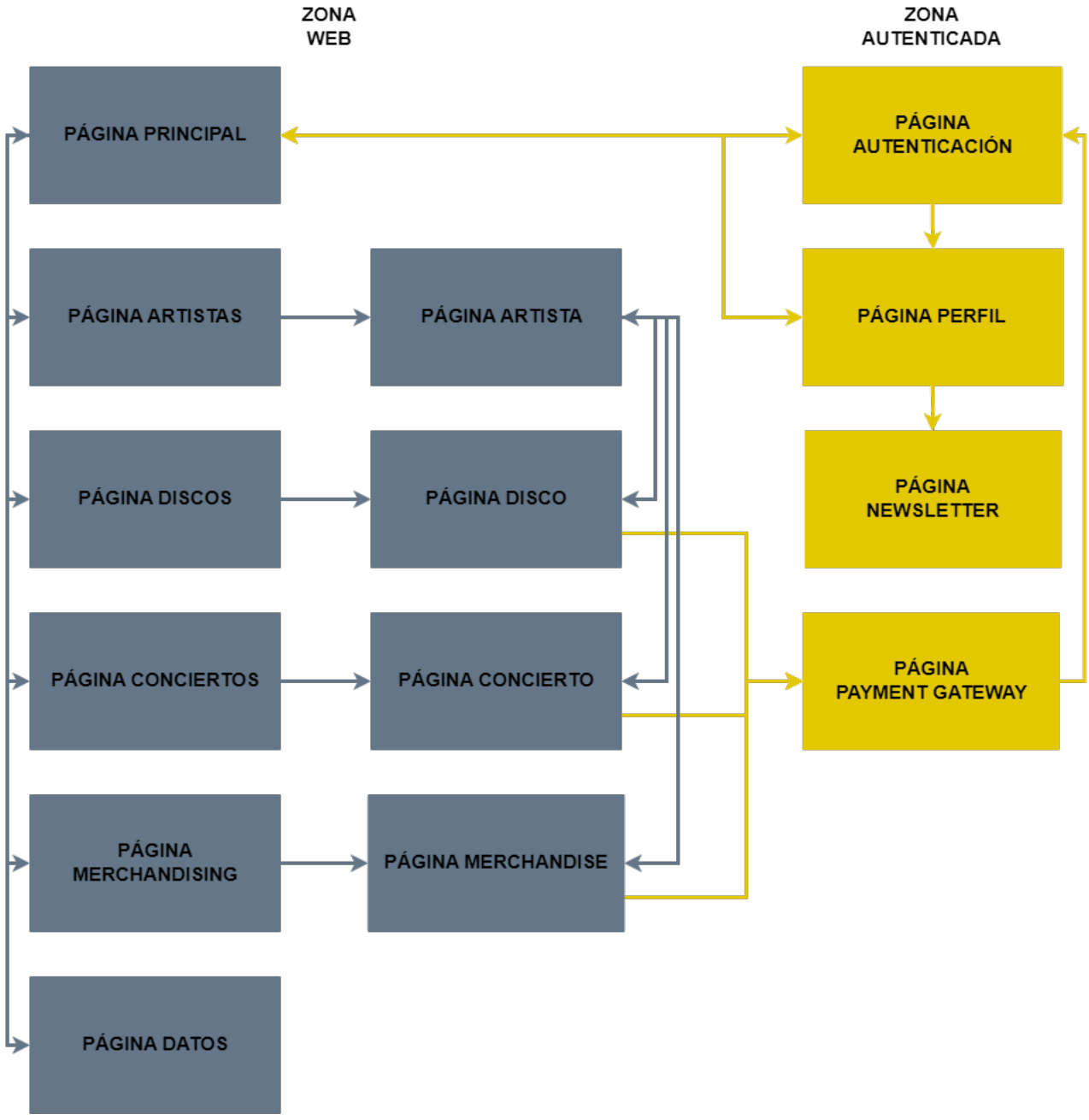



Figura 2.1: Funcionalidad de la aplicación

# Capítulo 3

# Tecnologías

Para la implementación de la aplicación web, se tomó la decisión de utilizar la pila tecnológica MERN, la cual es reconocida como uno de los conjuntos de herramientas más ampliamente utilizados para este propósito. Esta pila se compone de MongoDB, Express, Node.js y React. Con respecto a este proyecto en particular, se eligió utilizar MongoDB Atlas como solución para almacenar la base de datos en la nube, lo cual ofrece una gran facilidad de uso. Para el desarrollo del servidor o Back End, se empleó Express en conjunto con Node.js, mientras que para el cliente o Front End se utilizó el framework React. Por último, tanto el servidor como el cliente fueron programados utilizando el lenguaje JavaScript.

## 3.1. MongoDB Atlas

MongoDB Atlas es un servicio de base de datos en la nube ofrecido por MongoDB, la popular base de datos NoSQL. Atlas permite a los desarrolladores alojar su base de datos MongoDB en la nube, eliminando la necesidad de administrar la infraestructura de servidores y simplificando la configuración y el mantenimiento. Al utilizar MongoDB Atlas, los datos se almacenan de manera segura en la nube, lo que garantiza la disponibilidad y la redundancia.

## 3.2. Express

Express es un framework web rápido, minimalista y flexible para Node. Express proporciona una capa de abstracción sobre el servidor HTTP de Node. Además, Express ofrece una variedad de funciones y middleware que facilitan el procesamiento de datos, el manejo de sesiones, la autenticación y la autorización, entre otros aspectos comunes en el desarrollo web. Una de las principales ventajas de Express es su enfoque minimalista, lo que significa que proporciona solo las funcionalidades esenciales necesarias para construir aplicaciones web.

Express también es altamente extensible y se integra fácilmente con otras bibliotecas y módulos de Node. En resumen, Express es un framework web minimalista y flexible para Node. Proporciona una capa de abstracción sobre el servidor HTTP de Node.

## 3.3. React

React es una biblioteca de JavaScript de código abierto utilizada para construir interfaces de usuario interactivas y reactivas. Se centra en la construcción de componentes reutilizables que representan diferentes partes de la interfaz de usuario.

Cuando el estado de un componente cambia, React actualiza de manera eficiente solo las partes relevantes de la interfaz de usuario, evitando renderizaciones innecesarias y mejorando el rendimiento general de la aplicación.

React también utiliza una sintaxis llamada JSX, que combina HTML y JavaScript, lo que facilita la creación de componentes visuales y la manipulación dinámica de datos. Además, React cuenta con una gran comunidad de desarrolladores y una amplia gama de herramientas y bibliotecas complementarias disponibles, lo que facilita el desarrollo de aplicaciones robustas y escalables. En resumen, React es una biblioteca de JavaScript utilizada para construir interfaces de usuario interactivas y reactivas.

## 3.4. NodeJs

Node.js es un entorno de ejecución de código abierto basado en el motor JavaScript V8 de Google Chrome. Permite a los desarrolladores ejecutar código JavaScript en el lado del servidor y construir aplicaciones escalables y de alto rendimiento.

Una de las características principales de Node.js es su modelo de E/S sin bloqueo y basado en eventos. Node.js también cuenta con un ecosistema de paquetes y módulos extenso y activo a través de npm. Con npm, los desarrolladores pueden acceder a una amplia variedad de módulos preexistentes para agregar funcionalidades adicionales a sus aplicaciones de manera rápida y sencilla.

Además, Node.js es especialmente adecuado para el desarrollo de aplicaciones web, ya que puede actuar como un servidor web en sí mismo o integrarse con otros servidores web como Apache o Nginx. En resumen, Node.js es un entorno de ejecución de código abierto que permite a los desarrolladores utilizar JavaScript en el lado del servidor.

## 3.5. VsCode

VSCode, abreviatura de Visual Studio Code, es un editor de código fuente desarrollado por Microsoft. VSCode es conocido por su amplia compatibilidad con múltiples lenguajes de programación y su capacidad para adaptarse a una amplia gama de proyectos de desarrollo. Una de las ventajas clave de VSCode es su extensibilidad. Estas extensiones abarcan desde temas visuales hasta herramientas de control de versiones y paquetes de lenguaje específicos, ampliando las capacidades de VSCode y adaptándose a las preferencias individuales de los desarrolladores.

Además, VSCode tiene integración con control de versiones como Git y ofrece una interfaz de línea de comandos y una terminal integrada para facilitar el flujo de trabajo de desarrollo.

## 3.6. Postman

Postman es una herramienta de desarrollo de API y plataforma de colaboración que permite a los desarrolladores diseñar, probar y documentar API de manera eficiente. Es ampliamente utilizado en el desarrollo de software para facilitar la interacción con servicios web y realizar pruebas de API de forma efectiva.

En resumen, Postman brinda la capacidad de enviar solicitudes HTTP a una API y recibir respuestas, lo que ayuda a los desarrolladores a probar y depurar el funcionamiento de la API. Su interfaz intuitiva y amigable permite a los usuarios especificar parámetros, definir encabezados, enviar solicitudes y analizar las respuestas recibidas.

Una de las ventajas clave de Postman es su capacidad para trabajar con diversos tipos de solicitudes, como GET, POST, PUT, DELETE, y admitir diferentes formatos de datos, como JSON, XML y formularios. Esto permite a los desarrolladores interactuar con API de manera flexible y validar su funcionalidad.

Además de la funcionalidad básica de envío de solicitudes, Postman ofrece características avanzadas que mejoran el flujo de trabajo de desarrollo de API. Los usuarios pueden organizar las solicitudes en colecciones, crear entornos de prueba para gestionar configuraciones diferentes, realizar pruebas automatizadas y generar documentación completa para la API.

## 3.7. GitHub

GitHub es una plataforma de desarrollo colaborativa basada en la nube que permite a los desarrolladores trabajar en proyectos de software de manera colaborativa y controlar las versiones del código fuente. En GitHub, los desarrolladores pueden alojar y administrar repositorios de código, que contienen todos los archivos y el historial de cambios de un proyecto. Una de las características clave de GitHub es su sistema de control de versiones distribuido basado en Git. Además de la gestión del código fuente, GitHub ofrece una amplia gama de características adicionales.

Los desarrolladores pueden utilizar GitHub como una plataforma para colaborar en problemas, realizar seguimiento de tareas, documentar proyectos y colaborar en proyectos de código abierto. GitHub es reconocido por su gran comunidad de desarrolladores y por ser un lugar central para compartir y descubrir proyectos de código abierto. En resumen, GitHub es una plataforma de desarrollo colaborativa basada en la nube que utiliza el sistema de control de versiones Git. GitHub es ampliamente utilizado para alojar, compartir y colaborar en proyectos de código abierto, y también proporciona herramientas adicionales para la gestión de tareas, la documentación y la integración continua.

# Capítulo 4

# Desarrollo

## 4.1. Base de Datos

La base de datos se preparó mediante la creación de un clúster en MongoDB Atlas. Para ello, se inició un nuevo proyecto que permitió desplegar la base de datos. Se optó por la opción gratuita llamada ”Shared”, que utiliza Amazon Web Services en la región de París (eu-west-3). El clúster se configuró como M0 Sandbox, lo que proporciona RAM compartida y 512 MB de almacenamiento. Además, se seleccionó MongoDB 6.0 como versión adicional de configuración.

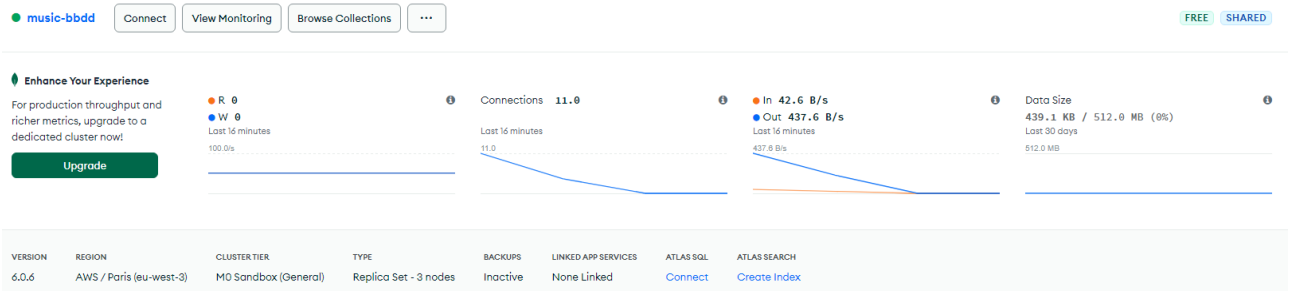



Figura 4.1: Cluster MongoDB Atlas

Una vez creado el clúster, se procedió a establecer un usuario con privilegios de administrador para la base de datos. También se configuraron las redes autorizadas para acceder a la base de datos.

## 4.2. Back End

### 4.2.1. Creación del Proyecto

Al ejecutar el comando “yarn init” y completar los campos requeridos, se inicializa el proyecto y se configuran las distintas funcionalidades del servidor, así como las API necesarias. A continuación, se crea la estructura de carpetas que organiza el backend en controladores, middlewares, modelos, enrutadores, cargas y utilidades.

El siguiente paso consistió en establecer la conexión entre el servidor y la base de datos. Para lograrlo, se instaló la dependencia “mongoose” y se siguieron los pasos

proporcionados por MongoDB Atlas para establecer la comunicación entre estos dos componentes.

```
const moongose = require('mongoose')
const app = require('./app')

const {
  DB_HOST,
  DB_PASSWORD,
  DB_USER,
  API_VERSION,
  IP_SERVER,
} = require('./constants')

const PORT = process.env.PORT || 3977

const startServer = async () => {
  try {
    await moongose.connect(`mongodb+srv://${DB_USER}:${DB_PASSWORD}@${DB_HOST}/`
    app.listen(PORT, () => {
      console.log('-------------------------------------')
      console.log('---------- Music API ---------------')
      console.log('-------------------------------------')
      console.log(
        `Server running on http://${IP_SERVER}:${PORT}/api/${API_VERSION}`
      )
    });
  } catch (error) {
    console.error('Database connection error:', error);
  }
}

startServer();

module.exports = app;
```

Figura 4.2: Conexión Back End - Base de Datos

Después de eso, se procedió a configurar el servidor utilizando Express. Con la ayuda de este framework, se realizaron las configuraciones necesarias, como el “body parser”, que permite analizar y procesar los datos de las solicitudes HTTP, específicamente en formatos JSON y datos de formularios. Para garantizar la seguridad de las solicitudes, se activó el CORS. Por último, se especificaron las rutas disponibles en el servidor, así como la configuración de la carpeta de “uploads”.

Durante el proceso de desarrollo, se ha seguido una metodología consistente que se repite para cada sistema, y se define mediante los siguientes pasos:

- Creación de los test correspondientes con ”supertest” y ”mocha”.
- Creación de los esquemas necesarios en la carpeta "models".
- Desarrollo de las funcionalidades del sistema en la carpeta "controllers".
- Creación de middlewares, en caso de ser necesarios, en la carpeta "middlewares".

- Establecimiento de los endpoints del sistema, donde se especifica el tipo de petición, los middlewares involucrados y el controlador correspondiente. Cada sistema dispone de un fichero en la carpeta "router".
- Importación y uso de los endpoints en la aplicación definida con Express.

```
api.post(
  '/user',
  [mwAuth.asureAuthenticated, mw_upload],
  userController.createUser
)
```

Figura 4.3: Configuración Endpoint

### 4.2.2. Sistema de Autenticación

Se diseñaron los modelos de usuarios y artistas antes de implementar los endpoints. Los usuarios tienen atributos como nombre de usuario, nombre y apellido, correo electrónico, contraseña, fecha de nacimiento o inicio, y un rol que indica si son usuarios comunes o artistas. Además, incluyen atributos como avatar, géneros musicales, y registros de discos, conciertos y merchandising. Los artistas son la parte visible de los usuarios artistas y tienen un atributo llamado “ownerId” que establece una relación directa con el usuario al que pertenecen. También incluyen atributos como nombre, fecha de inicio, avatar, discos, conciertos, merchandising y los géneros de sus proyectos.

```
const express = require('express')
const authenticationController = require('../controllers/authentication.controller')

const api = express.Router()

api.post('/auth/sign-up', authenticationController.signUp)

api.post('/auth/sign-in', authenticationController.signIn)

api.post(
  '/auth/refresh-access-token',
  authenticationController.refreshAccessToken
)

api.patch('/auth/change-password', authenticationController.changePassword)

module.exports = api
```

Figura 4.4: Endpoints del sistema de autenticación

El proceso de registro se realiza en la ruta ”/auth/sign-up” y utiliza el controlador de autenticación para invocar la función ”signUp”. En esta función, se recopilan los campos necesarios del cuerpo de la solicitud para crear un nuevo usuario. Si falta algún campo

requerido, se responde con un código de estado 400 y se muestra un mensaje indicando el elemento faltante. Si todos los campos necesarios están presentes, se crea un nuevo usuario. La contraseña se encripta de forma segura utilizando la función bcrypt para generar un hash. En última instancia, si el usuario tiene el rol ”common”, se guarda en la base de datos como un usuario común. De lo contrario, se crea un artista utilizando los datos proporcionados por el usuario y se asegura de que esté asociado a este.

Ubicado en la ruta ”/auth/sign-in”, se encuentra el punto de inicio de sesión que hace uso del controlador de autenticación para invocar la funcionalidad ”signIn”. Esta función recibe como parámetros tanto el correo electrónico como la contraseña extraída del cuerpo de la solicitud. Posteriormente, se realiza una búsqueda en la base de datos utilizando el correo electrónico proporcionado y se compara el hash de la contraseña almacenada con la contraseña ingresada. Si se encuentran coincidencias, se devuelve un token de acceso y un token de actualización. En caso contrario, se envía una respuesta de error con el código 500.

```
function createAccessToken(user) {
  const expToken = new Date()
  expToken.setHours(expToken.getHours() + 12)

  const payload = {
    token_type: 'access',
    user_id: user._id,
    role: user.role,
    iat: Date.now(),
    exp: expToken.getTime(),
  }

  return jwt.sign(payload, JWT_SECRET_KEY)
}

function createRefreshToken(user) {
  const expToken = new Date()
  expToken.setDate(expToken.getDate() + 7)

  const payload = {
    token_type: 'refresh',
    user_id: user._id,
    role: user.role,
    iat: Date.now(),
    exp: expToken.getTime(),
  }

  return jwt.sign(payload, JWT_SECRET_KEY)
}

function decodeToken(token) {
  return jwt.decode(token, JWT_SECRET_KEY, true)
}
```

Figura 4.5: Creación y decodificación de los tokens

Además, se ha implementado en la ruta ”/auth/refresh-access-token” un procedimiento que posibilita la obtención de un nuevo token de acceso en situaciones en las que este

no esté presente o haya expirado. En este proceso, se utiliza el token de refresco para buscar al usuario correspondiente en la base de datos. Si se encuentra el registro del usuario, se genera un nuevo token de acceso y se retorna al cliente.

Finalmente, se encuentra ”/auth/change-password” que permite llevar a cabo la modificación de la contraseña. En este proceso, se requiere que se ingresen tanto la contraseña antigua como la nueva, verificando de esta manera que la antigua coincida con el hash almacenado en la base de datos. En caso de coincidencia, se procede a aplicar una función de ”hashing” a la nueva contraseña y se actualiza el valor correspondiente en la base de datos.

### 4.2.3. Sistema de Usuarios

Con el propósito de llevar a cabo la implementación de los endpoints de este sistema, se ha desarrollado un componente de software intermediario (middleware) cuya finalidad es asegurar que el usuario que acceda a este sistema esté autenticado en la aplicación. Dicho middleware se encarga de obtener los tokens de acceso y verificar que no hayan expirado. En caso de que el token se encuentre vencido o no se haya otorgado uno, se denegará el acceso al endpoint correspondiente.

```
const express = require('express')
const multiparty = require('connect-multiparty')
const userController = require('../controllers/user.controller')
const mwAuth = require('../middlewares/authentication.middleware')

const mw_upload = multiparty({ uploadDir: './uploads/avatar' })
const api = express.Router()

api.get('/user/me', [mwAuth.asureAuthenticated], userController.getMe)

api.get('/users', [mwAuth.asureAuthenticated], userController.getUsers)

api.post(
  '/user',
  [mwAuth.asureAuthenticated, mw_upload],
  userController.createUser
)

api.patch(
  '/user/me',
  [mwAuth.asureAuthenticated, mw_upload],
  userController.updateUser
)

api.delete('/user/me', [mwAuth.asureAuthenticated], userController.deleteUser)

api.patch(
  '/user/musical-genre',
  [mwAuth.asureAuthenticated],
  userController.updateUserMusicalGenre
)

module.exports = api
```

Figura 4.6: Endpoints del sistema de usuarios

El código utiliza la librería multiparty para procesar las solicitudes que contienen archivos adjuntos (como imágenes) enviados desde el cliente a través de un formulario HTML o una solicitud HTTP.

Los endpoints ”GET” del sistema tienen la función de recuperar la información almacenada en la base de datos. Específicamente, la ruta ”/user/me” proporciona los datos del usuario autenticado, mientras que ”/users” brinda información sobre todos los usuarios. Además, este sistema también ofrece la posibilidad de crear un nuevo usuario, utilizando un proceso similar al del sistema de autenticación.

Adicionalmente, se han implementado dos actualizaciones en el sistema. La primera permite que el usuario autenticado pueda actualizar su información personal. La segunda actualización se utiliza para modificar el atributo de géneros musicales ”musicalGenre”. Este atributo se actualiza automáticamente basándose en los discos y conciertos que el usuario tenga en su información. En otras palabras, este atributo se actualiza automáticamente al agregar o eliminar discos y conciertos en la información del usuario.

Por último, se ha incorporado la funcionalidad que permite al usuario autenticado eliminar su información de forma segura y definitiva del sistema.

### 4.2.4. Sistema de Artistas

Para la implementación de este sistema, se creó un nuevo middleware, en conjunto con el middleware que se asegura de la autenticación el usuario, verifica si el usuario tiene el rol de artista, ya que muchas de las funcionalidades están reservadas para esos usuarios. ”AsureIsArtist” es un middleware que recoge del accestoken el rol del usuario y si este coincide con el rol esperado, se deja pasar.

Se encuentran disponibles tres métodos ”GET” dentro del sistema. El primero, cuya dirección es ”/artists”, proporciona la totalidad de la información correspondiente a todos los artistas almacenados en la base de datos. El segundo método, localizado en ”/artist/:id”, proporciona la información del usuario específico al que se le ha asignado el ID ingresado en la dirección. Por último, el tercer método, ubicado en ”/owned/artist”, permite al usuario autenticado cuyo rol es artista obtener la información del artista asociado, en caso de que exista una asociación entre ambos.

Por otro lado, en caso de que el usuario no disponga de un artista asociado, el sistema ofrece una funcionalidad ”POST” en la ruta ”/artist”, que le permite crear un artista personalizado. Mediante esta funcionalidad, el usuario puede proporcionar la información deseada, como el avatar, nombre y fecha de inicio, para configurar su propio artista. Una vez creado, el sistema asignará automáticamente al artista los proyectos de discos, conciertos, merchandising y géneros musicales que estén asociados al usuario.

Adicionalmente, se brinda la opción de actualizar el artista asociado, lo que implica la capacidad de modificar toda la información del artista, incluyendo la imagen de avatar.

La eliminación también es una característica integral de este sistema, ya que permite al usuario eliminar sin problemas el artista asociado en caso de que sea necesario, sin tener repercusiones en su propia información.

```
const express = require('express')
const multiparty = require('connect-multiparty')
const artistController = require('../controllers/artist.controller')
const mwAuth = require('../middlewares/authentication.middleware')
const mwIsArtist = require('../middlewares/IsArtist.middleware')
const mwUpload = multiparty({ uploadDir: './uploads/avatar' })

const api = express.Router()

api.get('/artists', artistController.getArtists)

api.get('/artist/:id', artistController.getArtist)

api.get(
  '/owned/artist',
  [mwAuth.asureAuthenticated],
  artistController.getOwnerArtist
)

api.post(
  '/artist',
  [mwAuth.asureAuthenticated, mwUpload, mwIsArtist.asureIsArtist],
  artistController.createArtist
)

api.patch(
  '/artist/me',
  [mwAuth.asureAuthenticated, mwUpload, mwIsArtist.asureIsArtist],
  artistController.updateArtist
)

api.delete(
  '/artist/me',
  [mwAuth.asureAuthenticated, mwIsArtist.asureIsArtist],
  artistController.deleteArtist
)

module.exports = api
```

Figura 4.7: Endpoints del sistema de artistas

### 4.2.5. Sistema de Discos

Antes de proceder al desarrollo del sistema de discos, se llevaron a cabo una serie de etapas preliminares. En primer lugar, se implementó un modelo que incluye diversos atributos, como el ID del propietario, el nombre del disco, la fecha de lanzamiento, el límite de dinero, el precio, el dinero acumulado, los géneros musicales, las canciones que componen el disco y su portada. Además, este modelo cuenta con el complemento ”moongoose-paginate”, el cual permite obtener información de la base de datos paginada, es decir, dividida en secciones.

Por otro lado, era necesario incorporar un nuevo middleware denominado ”AsuraIs-Common”. Al igual que el middleware mencionado anteriormente, este nuevo componente tiene la capacidad de distinguir si un usuario autenticado posee el rol de ”common”, ya que en este sistema, se presentan funcionalidades que están disponibles tanto para los artistas como para los usuarios comunes.

El sistema cuenta con cuatro funcionalidades distintas para la lectura de datos. La primera funcionalidad se encuentra en la ruta ”/discs” y devuelve una lista de todos los discos almacenados en la base de datos. La segunda funcionalidad, ubicada en la ruta ”/disc/:id”, proporciona la información específica del disco solicitado, identificado por su id. Las dos últimas funcionalidades tienen un funcionamiento similar, ya que permiten la paginación de la información extraída de la base de datos, mostrando un máximo de tres discos por página. La primera funcionalidad de paginación permite obtener los discos según el artista, y se encuentra en la ruta ”/discs/artist/:id”, donde id representa el identificador del artista. La segunda funcionalidad de paginación permite obtener los discos según el usuario autenticado, y se encuentra en la ruta ”/discs/user”.

```
api.get('/discs', discController.getDiscs)

api.get('/disc/:id', discController.getDisc)

api.get('/discs/artist/:id', discController.getDiscsByArtist)

api.get(
  '/discs/user',
  [mwAuth.asureAuthenticated],
  discController.getDiscsByUser
)
```

Figura 4.8: Lectura del sistema de discos

Los endpoints accesibles para los usuarios artistas proporcionan la capacidad de realizar diversas acciones relacionadas con los discos. Para la creación de discos, se utiliza la ruta ”/disc”, mientras que para la actualización y eliminación se emplea la ruta ”/disc/:id”. Además, se han incorporado dos funcionalidades adicionales en la ruta ”/disc/song/:id”. Estas funcionalidades permiten agregar o eliminar canciones de un disco específico.

La razón detrás de la implementación de estas funcionalidades adicionales se basa en una limitación inherente a la funcionalidad de actualización. Debido a la naturaleza del atributo involucrado, no es posible añadir o eliminar canciones de manera específica utilizando la función de actualización. En su lugar, dicha función solo permite modificar el valor del atributo en su conjunto.

Dado que el atributo en cuestión es un array de cadenas de texto, se ha optado por implementar las funcionalidades que modifican directamente dicho arreglo como la solución más sencilla y eficiente.

```
api.post(
  '/disc',
  [mwAuth.asureAuthenticated, mwUpload, mwIsArtist.asureIsArtist],
  discController.createDisc
)

api.patch(
  '/disc/:id',
  [mwAuth.asureAuthenticated, mwUpload, mwIsArtist.asureIsArtist],
  discController.updateDisc
)

api.delete(
  '/disc/:id',
  [mwAuth.asureAuthenticated, mwIsArtist.asureIsArtist],
  discController.deleteDisc
)

api.post(
  '/disc/song/:id',
  [mwAuth.asureAuthenticated, mwIsArtist.asureIsArtist],
  discController.addSong
)

api.delete(
  '/disc/song/:id',
  [mwAuth.asureAuthenticated, mwIsArtist.asureIsArtist],
  discController.deleteSong
)
```

Figura 4.9: Endpoints para artistas en el sistema de discos

Para los usuarios comunes, el sistema habilita exclusivamente la adquisición y el reembolso de productos, accesibles a través de las rutas ”/buy/disc/:id” y ”/return/disc/:id”. Estas funciones se encargan de gestionar el atributo del disco que concierne tanto al saldo acumulado como al precio, garantizando que el saldo acumulado refleje la suma de su valor anterior y el costo de compra respectivo.

```
api.patch(
  '/buy/disc/:id',
  [mwAuth.asureAuthenticated, mwIsCommon.asureIsCommon],
  discController.buyDisc
)

api.patch(
  '/return/disc/:id',
  [mwAuth.asureAuthenticated, mwIsCommon.asureIsCommon],
  discController.returnDisc
)
```

Figura 4.10: Endpoints para usuarios en el sistema de discos

### 4.2.6. Sistema de Conciertos

Teniendo en cuenta que ya se han creado los middlewares necesarios, el proceso previo a la implementación del sistema se centró en la introducción del modelo de conciertos. Este modelo también utiliza la técnica de paginación y está compuesto por varios elementos esenciales, como el identificador del propietario, el nombre del concierto, la fecha en la que se celebrará, la ubicación del evento, el límite de dinero establecido, el precio de las entradas y la cantidad de dinero acumulada hasta el momento. Además, se incluyen detalles sobre los géneros musicales que se presentarán, los participantes del concierto y el póster correspondiente a cada evento.

```
api.get('/concerts/artist/:id', concertController.getConcertsByArtist)

api.get(
  '/concerts/user',
  [mwAuth.asureAuthenticated],
  concertController.getConcertsByUser
)

api.patch(
  '/concert/:id',
  [mwAuth.asureAuthenticated, mwUpload, mwIsArtist.asureIsArtist],
  concertController.updateConcert
)

api.delete(
  '/concert/:id',
  [mwAuth.asureAuthenticated, mwIsArtist.asureIsArtist],
  concertController.deleteConcert
)

api.patch(
  '/buy/concert-ticket/:id',
  [mwAuth.asureAuthenticated, mwIsCommon.asureIsCommon],
  concertController.buyTicket
)

api.patch(
  '/return/concert-ticket/:id',
  [mwAuth.asureAuthenticated, mwIsCommon.asureIsCommon],
  concertController.returnTicket
)
```

Figura 4.11: Ejemplo Endpoints del sistema de conciertos

El diseño del sistema para la gestión de conciertos sigue la misma estructura que el sistema utilizado para los discos, y mantiene las cuatro formas principales de obtener información: todos los conciertos (”/concerts”), conciertos específicos (”/concert/:id”), conciertos de un artista en particular (”/concerts/artist/:id”) y conciertos pertenecientes al usuario autenticado (”/concerts/user”).

Además de estas funcionalidades básicas, se han añadido características adicionales relacionadas con los artistas, lo cual permite la creación de nuevos conciertos en la ruta ”/concert”, la edición y eliminación de conciertos existentes en la ruta ”/concert/:id”, así como la capacidad de agregar y eliminar participantes en la ruta ”/concert/participant/:id”.

Por otro lado, se han desarrollado funcionalidades específicas para los usuarios comunes, brindándoles la oportunidad de comprar entradas para conciertos en la ruta ”/buy/concert-ticket/:id” y solicitar reembolsos por las entradas adquiridas en la ruta ”/return/concert-ticket/:id”. Estas características adicionales permiten a los usuarios disfrutar de una experiencia más completa al interactuar con el sistema, ofreciendo opciones de compra y manejo de entradas para los conciertos de su interés.

### 4.2.7. Sistema de Merchandising

La implementación del modelo de Mercancía implica la inclusión de una serie de atributos que definen de manera precisa cada artículo. Estos atributos incluyen el ID del propietario, el nombre, la fecha de lanzamiento, el límite de dinero, el precio, el total de dinero acumulado hasta el momento, las tallas disponibles (que varían desde XS hasta XL), la descripción detallada del producto y su correspondiente imagen visual.

Posteriormente, se llevó a cabo la creación del sistema de merchandising, siguiendo la misma estructura que se utiliza para los sistemas de discos y conciertos. La única diferencia radica en que, en este caso, el modelo y las rutas se han adaptado específicamente para el contexto de la mercancía, utilizando nombres pertinentes, como ”merchandise”.

```
api.get('/merchandises/artist/:id', merchController.getMerchandisesByArtist)

api.get(
  '/merchandises/user',
  [mwAuth.asureAuthenticated],
  merchController.getMerchandiseByUser
)

api.patch(
  '/merchandise/:id',
  [mwAuth.asureAuthenticated, mwUpload, mwIsArtist.asureIsArtist],
  merchController.updateMerchandise
)

api.patch(
  '/buy/merchandise/:id',
  [mwAuth.asureAuthenticated, mwIsCommon.asureIsCommon],
  merchController.buyMerchandise
)
```

Figura 4.12: Ejemplo Endpoints del sistema de merchandising

### 4.2.8. Sistema de Newsletters

El modelo de boletines informativos indispensable para el sistema se caracteriza por tener un único atributo, el cual es la dirección de correo electrónico. Consta únicamente de tres endpoints, los cuales requieren autenticación. Estos puntos de acceso permiten obtener la lista completa de correos electrónicos de los usuarios suscritos, además de posibilitar que un usuario pueda suscribirse o cancelar su suscripción a las newsletters.

```
api.get(
  '/get-emails',
  [mwAuth.asureAuthenticated],
  newsletterController.getEmails
)

api.post(
  '/new-subscription',
  [mwAuth.asureAuthenticated],
  newsletterController.newSubscription
)

api.delete(
  '/cancel-subscription',
  [mwAuth.asureAuthenticated],
  newsletterController.cancelSubscription
)
```

Figura 4.13: Endpoints del sistema de newsletter

## 4.3. Front End

### 4.3.1. Creación del Proyecto

En la elaboración del proyecto para la aplicación del cliente, se procedió a ejecutar el comando ”npx create-react-app”, especificando el nombre del proyecto. Posteriormente, se eliminaron los archivos generados que no eran necesarios para el desarrollo. A continuación, se instalaron las dependencias de manera secuencial, siendo estas: ”sass” para la gestión de estilos, ”semantic-ui-react” para la implementación de componentes visuales, ”lodash” para facilitar la manipulación de datos, y finalmente, se agregaron las dependencias ”formik” y ”yup” para el manejo de formularios en la aplicación.

También se modificó la información de la pestaña del navegador, ya que se realizó el cambio de icono, quitando el que viene por defecto, y se cambió el título al nombre de la aplicación.

A continuación, se procedió a crear la carpeta ”assets”, la cual está organizada en subcarpetas (jpg, png, svg) con el propósito de alojar todas las imágenes e iconos necesarios para la parte visual de la aplicación. Además, se realizó la configuración del archivo que facilita la exportación de todos los contenidos de dicha carpeta.

Además, se llevó a cabo el diseño e implementación del sistema de enrutamiento de la aplicación. En este sentido, se tomó la decisión de dividir la aplicación en dos áreas

claramente distintas: la zona de autenticación, restringida exclusivamente a usuarios registrados y autenticados, y la zona web, accesible para todos los usuarios. Estas áreas fueron denominadas ”AuthRouter” y ”WebRouter”, respectivamente.

```jsx
import React from 'react'
import { BrowserRouter } from 'react-router-dom'
import { AuthRouter, WebRouter } from './router'
import { AuthProvider } from './context'

export default function App() {
  return (
    <AuthProvider>
      <BrowserRouter>
        <WebRouter />
        <AuthRouter />
      </BrowserRouter>
    </AuthProvider>
  )
}
```

Figura 4.14: Lógica de enrutamiento de la aplicación

### 4.3.2. Zona Autenticación

La sección de autenticación cuenta con tres rutas: ”/auth”, ”/auth/newsletters” y ”/auth/payment”. En la primera ruta se encuentran los formularios de inicio de sesión y registro para aquellos usuarios que no hayan iniciado sesión. En caso de que exista un usuario autenticado, se mostrará la página correspondiente al perfil del usuario. Por otro lado, en la ruta ”/auth/newsletters” se encuentra la página destinada a la gestión de suscripciones del usuario. Y en la ruta ”/auth/payment” se pueden llevar a cabo las transacciones de compra dentro de la aplicación, las cuales están simuladas.

Todas las páginas en esta sección de la aplicación, excluyendo la pasarela de autenticación, siguen un diseño de presentación coherente. Esto significa que todas ellas comparten un mismo ”layout”, para el cual, se ha creado un componente que se encarga de la lógica de cierre de sesión, ya que desde esta sección, es necesario que los usuarios puedan salir de la aplicación. En cuanto a la estructura del ”layout”, consta de tres columnas, siendo la columna central la más amplia, donde se encuentra toda la información relevante de la página. Las otras dos columnas, situadas a cada lado de la columna principal, presentan los colores principales de la aplicación, así como el logotipo. Si se hace clic en el logotipo, se redirige al usuario a la página principal.

```
return (
  <Routes>
    {!user ? (
      <Route path="/auth/*" element={<Auth />} />
    ) : (
      <>
        {['/auth/my-profile', '/auth/*'].map((path) => (
          <Route
            key={path}
            path={path}
            element={loadLayout(LogInLayout, loadUserPage)}
          />
        ))}
        <Route
          path="/auth/newsletters"
          element={loadLayout(LogInLayout, Newsletters)}
        />
        <Route
          path="/auth/payment/:id"
          element={loadLayout(LogInLayout, PaymentGateway)}
        />
      </>
    )}
  </Routes>
)
```

Figura 4.15: Rutas de la zona de autenticación

### Inicio Sesión y Registro

La aplicación ofrece la funcionalidad de autenticación, tal como se ha mencionado previamente. Con este fin, se han desarrollado dos formularios que han sido validados y controlados mediante el uso de formik y yup. Estos formularios permiten tanto iniciar sesión como registrarse en la aplicación. En el formulario de registro, se solicita toda la información necesaria para crear un nuevo usuario, incluyendo su nombre, rol, dirección de correo electrónico, contraseña, entre otros datos relevantes. Por otro lado, el formulario de inicio de sesión consta únicamente de dos campos: email y contraseña.

Ambos formularios almacenan los datos ingresados por el usuario y los envían al backend, específicamente a la API del sistema de autenticación. Además de los formularios, la interfaz de la aplicación también incluye el logotipo y un botón que permite regresar a la página de inicio de la aplicación.


Figura 4.16: Zona de registro e inicio de sesión

### Página de Usuario

En la figura 3.15 se observa que, para la ruta de la página de usuario, se invoca al ”layout” y a la función ”loadUserPage”. Esta última se encarga de determinar el contenido que se mostrará al usuario autenticado, ya que la página de usuario difiere para un artista en comparación con un usuario común. Con este propósito, se han diseñado tres componentes para esta página. El componente compartido por ambos tipos de usuarios es ”UserContent”, el cual se encarga de formatear y mostrar la información del usuario, así como los proyectos de crowdfunding en los que ha participado en el caso de usuarios comunes, o los proyectos creados por un artista.

Adicionalmente, en nuestro proyecto contamos con dos componentes distintos pero similares: los menús ”CommonMenu” y ”ArtistMenu”. Estos menús permiten a cada usuario realizar todas las operaciones correspondientes a sus respectivos roles. Para implementar estos menús, se creó un componente llamado ”BasicModal” que tiene la funcionalidad de mostrar una ventana emergente al hacer clic en cualquiera de los elementos del menú. A este modal se le pasa el formulario específico correspondiente a cada operación.

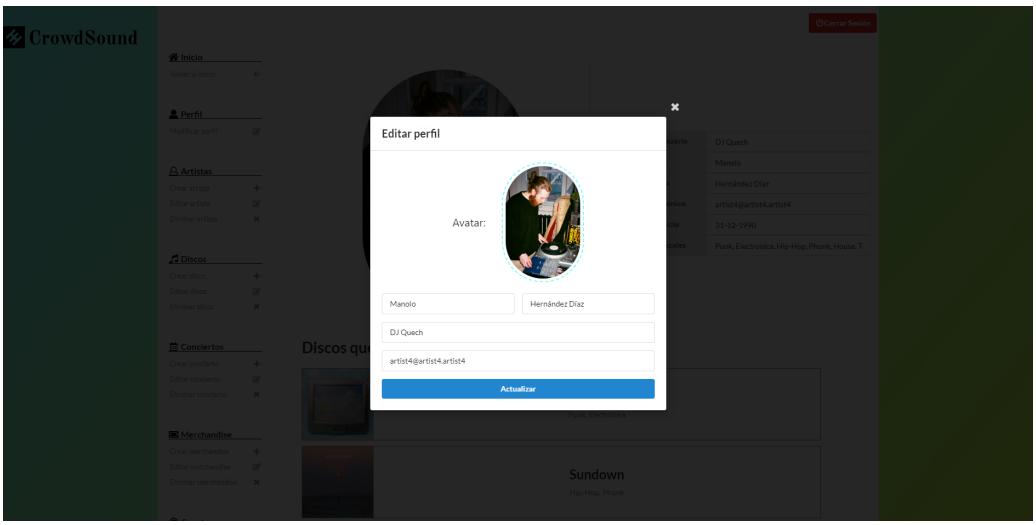



Figura 4.17: Ejemplo de modal del menú

Los usuarios con el rol de artistas tienen la capacidad de realizar diversas acciones. Pueden modificar su perfil, cambiar su contraseña, acceder a la suscripción de la newsletter o eliminar su cuenta. Además, tienen la posibilidad de crear, editar o eliminar su página de artista. Por supuesto, también pueden crear, eliminar y actualizar sus proyectos, ya sean discos, conciertos o merchandise.

Por otro lado, los usuarios comunes tienen la capacidad de gestionar su cuenta de manera similar a los artistas, con algunas diferencias. Aunque no pueden crear proyectos, solo participar en ellos, su menú incluye opciones para realizar devoluciones de participaciones en los proyectos de crowdfunding.

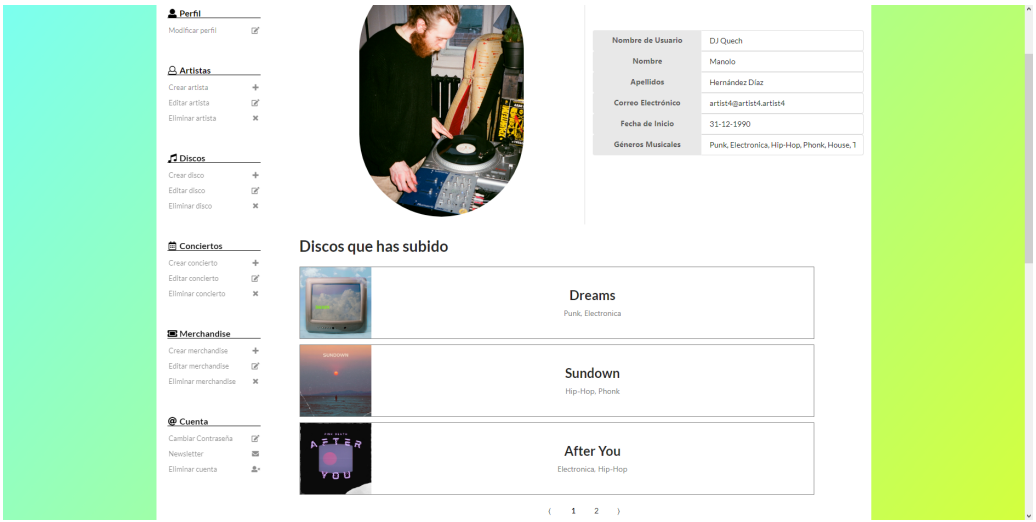



Figura 4.18: Página de usuario

### Plataforma de pago

La página de pagos está diseñada exclusivamente para usuarios que no son artistas. Esta se encarga de mostrar toda la información relacionada con la transacción en la que el usuario desea participar. Además, cuenta con un formulario que solicita los datos de la tarjeta de crédito o débito del usuario en cuestión. Cabe destacar que dicho formulario

está validado y controlado.

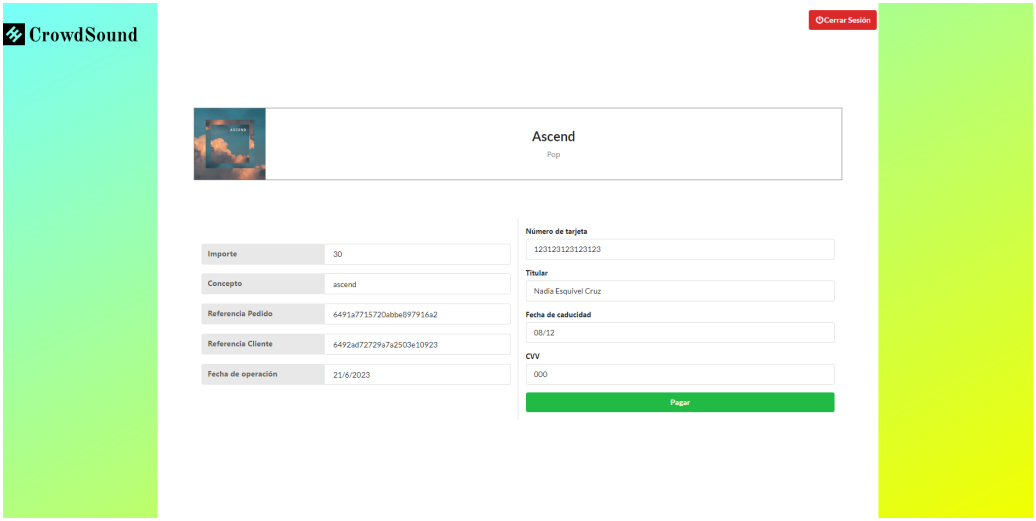



Figura 4.19: Plataforma de pago

Por otro lado, se ha implementado un modal que informa al usuario si la compra se ha realizado con éxito o no. Si el usuario realiza la compra, se mostrará un mensaje indicando que la transacción ha sido efectuada correctamente. Sin embargo, si el usuario intenta realizar la compra nuevamente o si el usuario que ha accedido no cuenta con los permisos necesarios para comprar, se mostrará un mensaje que impedirá la realización de la transacción.

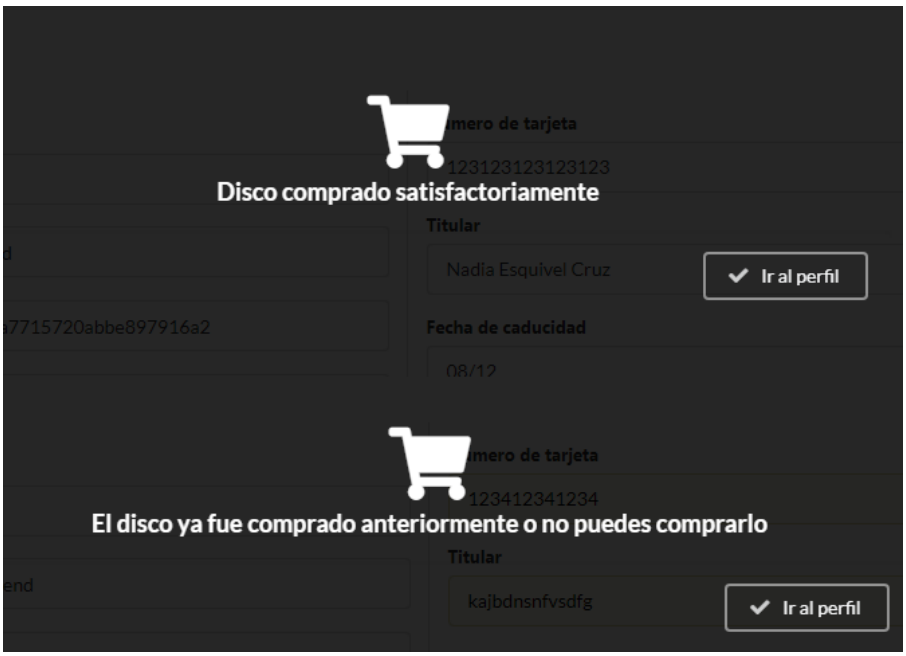



Figura 4.20: Modal de la plataforma de pago

### Newsletters

Para concluir, nos encontramos ante la página del boletín informativo, la cual brinda al usuario una explicación detallada sobre la naturaleza de los boletines de la aplicación.

Además, proporciona la logística necesaria para gestionar las suscripciones, permitiendo tanto el registro como la cancelación de las mismas. En este sentido, la propia página informa al usuario si se ha suscrito exitosamente o si se ha cancelado la suscripción.


Figura 4.21: Gestión de la suscripción del boletín

### 4.3.3. Zona Web

Esta zona es accesible por cualquier usuario, y al igual que la zona autenticada, todas las páginas comparten un mismo ”layout”, para el cual se realizaron los componentes del header y del footer.

El header conta del logotipo de la aplicación y posibilita la navegación entre las distintas páginas de la aplicación, las cuales comprenden la página de inicio u home, artistas, discos, conciertos, merchandising y datos. Asimismo, incluye un botón que facilita el acceso al registro o inicio de sesión. Si el usuario ya ha iniciado sesión, el contenido del botón se modifica para mostrar la imagen del usuario y el texto ”Tu perfil”. En lo que se refiere al pie de página, este contiene una concisa presentación acerca de lo que representa ”CrowdSound”, así como un mapa web y un enlace para acceder a las newsletters.


Figura 4.22: Diferencias del header al iniciar sesión

Esta área dispone de múltiples rutas para acceder a diferentes secciones de la aplicación. Podemos encontrar la opción de inicio (página principal), artistas, artistas específicos, discos, discos específicos, conciertos, conciertos específicos, merchandising, merchandise

específico, así como los datos de la aplicación.

```
return (
  <Routes>
    <Route path="/" element={loadLayout(ClientLayout, Home)} />
    <Route path="/artist/:id" element={loadLayout(ClientLayout, Artist)} />
    <Route path="/artists" element={loadLayout(ClientLayout, ArtistsList)} />
    <Route path="/concert/:id" element={loadLayout(ClientLayout, Concert)} />
    <Route
      path="/concerts"
      element={loadLayout(ClientLayout, ConcertsList)}
    />
    <Route path="/disc/:id" element={loadLayout(ClientLayout, Disc)} />
    <Route path="/discs" element={loadLayout(ClientLayout, DiscList)} />
    <Route
      path="/merchandise/:id"
      element={loadLayout(ClientLayout, Merchandising)}
    />
    <Route
      path="/merchandising"
      element={loadLayout(ClientLayout, MerchandisingList)}
    />
    <Route path="/web-data" element={loadLayout(ClientLayout, WebData)} />
  </Routes>
)
```

Figura 4.23: Rutas de la zona web

### Página Principal

El diseño de la aplicación se caracteriza por un encabezado que incluye un carrusel automático con tres imágenes, que proporciona una breve descripción de lo que se puede encontrar en la aplicación. Además, cuenta con la sección ”¿Quiénes somos?”, y un acceso directo a los diferentes proyectos disponibles. Estos accesos están representados por imágenes con animaciones que se agrandan al colocar el cursor sobre ellas, mostrando el contenido al que hacen referencia. Asimismo, se dispone de un botón que redirige al formulario de registro, y por último, se encuentra una sección de contacto que proporciona enlaces a diversas redes sociales.

### Buscador universal

Se ha diseñado una barra de búsqueda universal para las páginas de artistas, discos, conciertos y merchandising. Esta barra de búsqueda cuenta con un campo de entrada denominado ”término de búsqueda”, al cual se le pueden aplicar filtros, como por ejemplo, filtrar por nombre, géneros musicales, fecha de lanzamiento e incluso por tallas. Es importante tener en cuenta que no todos los filtros son aplicables en todos los elementos mencionados. Por lo tanto, se ha implementado un mecanismo que muestra y oculta los filtros según el contexto en el que se esté utilizando el componente de la barra de búsqueda.

El diseño que acompaña al buscador consiste en un conjunto de tres elementos que comparten un componente común. Este componente muestra la imagen del objeto correspondiente, y al pasar el cursor sobre él, se muestra el nombre del objeto, así como los géneros musicales a los que pertenece (o las tallas, en el caso de ser merchandising). Estos elementos se encuentran integrados en una paginación creada en el frontend, lo cual facilita la búsqueda.

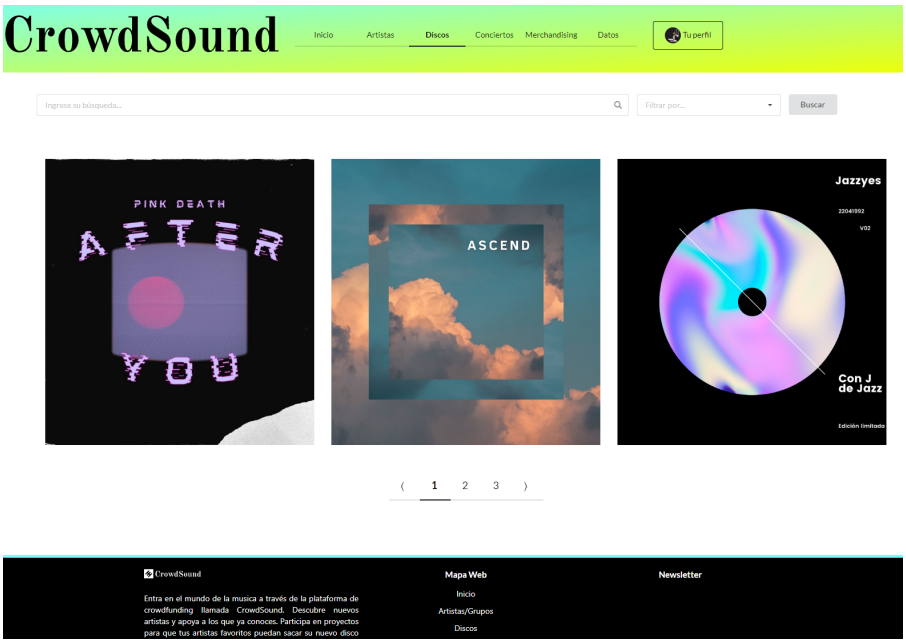



Figura 4.24: Ejemplo de página con barra de búsqueda

### Artistas

Al acceder a un artista específico, nos encontramos inicialmente con el avatar y la información relevante del artista, como su nombre, fecha de inicio y los géneros musicales en los que se desenvuelve. En esta página, tenemos la posibilidad de explorar todos los proyectos que el artista tiene dentro de ”CrowdSound”, incluyendo sus discos, conciertos y merchandising, en caso de tenerlos. Es importante destacar que se ha desarrollado un componente para listar los proyectos, presentando la información de manera legible y fácil de comprender. Además, al hacer clic en ellos, se puede acceder a una página específica que proporciona información adicional sobre cada proyecto.

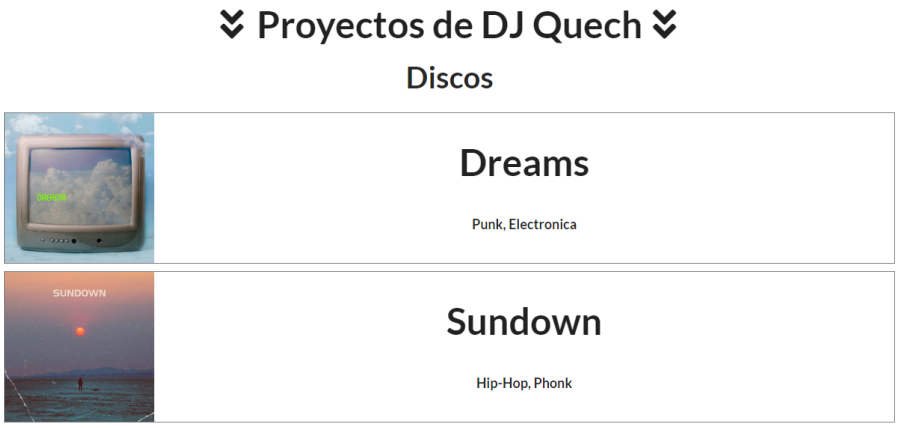



Figura 4.25: Proyectos de un artista

### Discos

En la página de discos, se han dispuesto tres columnas. La primera columna muestra la portada del disco, la segunda columna contiene información detallada, como el nombre del creador, el precio de participación, la fecha estimada de lanzamiento, así como los géneros musicales y las canciones incluidas en el disco. Por último, se presenta la imagen del artista, la cual al hacer clic en ella, nos redirige a su página, además de proporcionar información sobre la campaña de financiamiento colectivo, que incluye el objetivo de recaudación, la cantidad de dinero ya recaudada y una barra de progreso.

Adicionalmente, cada página cuenta con un botón que permite acceder a la pasarela de pago para adquirir el disco. Sin embargo, es importante destacar que dicho botón solo estará disponible si el proyecto está activo y si el usuario tiene la autorización necesaria para realizar la compra.


Figura 4.26: Información de un disco en concreto

### Conciertos

En la página de conciertos, se ha mantenido el mismo diseño que la página de discos, con la única diferencia en la columna central. En esta página, la información se ha adaptado al modelo de conciertos, presentando detalles adicionales como los participantes del evento y un mapa interactivo que muestra la ubicación del concierto.

Para la implementación del mapa, se ha optado por utilizar ”Leaflet”, una biblioteca de código abierto especializada en mapas interactivos y de fácil integración en aplicaciones web. Leaflet proporciona diversas funciones y herramientas para visualizar y manipular mapas en la página de conciertos.

Para obtener las coordenadas geográficas necesarias para el mapa, se ha utilizado la API gratuita de ”OpenCageData”. Esta API ofrece un servicio que permite traducir ubicaciones de texto en coordenadas de latitud y longitud. Mediante la integración de ”OpenCageData” en el sistema, se ha implementado una función que utiliza esta API para obtener de manera precisa las coordenadas correspondientes a la ubicación del concierto.


Figura 4.27: Información de un concierto en concreto

### Merchandising

De manera similar a las páginas de discos y conciertos, la página de merchandise sigue la misma estructura, adaptando la información al modelo específico de merchandise. Además, en esta página se incluye una descripción del producto, la cual es redactada por los artistas al momento de crearlo.


Figura 4.28: Información de un merchandise en concreto

### Página de Datos

En esta página se encuentran las estadísticas de la aplicación ÇrowdSound", donde se muestra el número de discos, conciertos y merchandising disponibles. Además, se ofrece

una visión general de los estados de los proyectos a través de un gráfico circular. En dicho gráfico se pueden visualizar los proyectos finalizados, los que están en curso y aquellos que han fracasado.

Por último, se destaca el proyecto que ha obtenido o está obteniendo el mayor respaldo por parte de los usuarios en la aplicación.

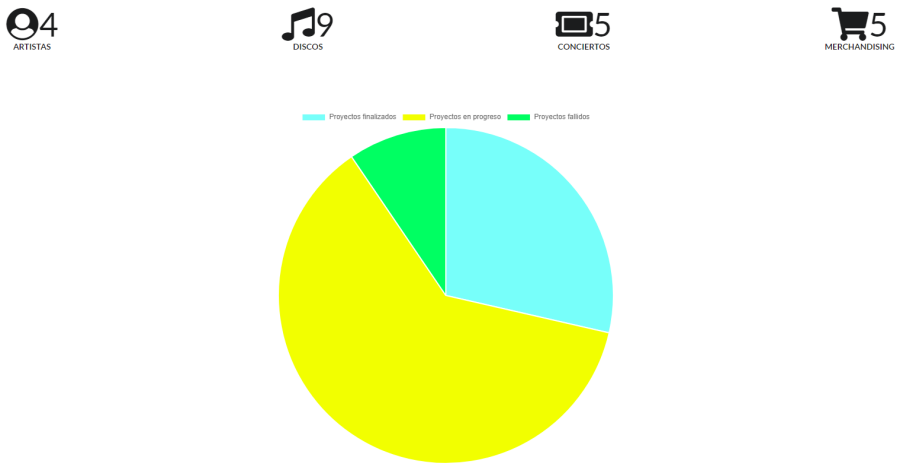



Figura 4.29: Datos de la aplicación

### Página 404

En la página 404, se muestra un mensaje que comunica claramente al usuario que el contenido buscado no ha sido encontrado. Además, se incluye un botón prominente que ofrece la opción de regresar a la página de inicio de la aplicación y que cuenta con una animación que, al poner el cursor encima, cambia su contenido a una flecha.

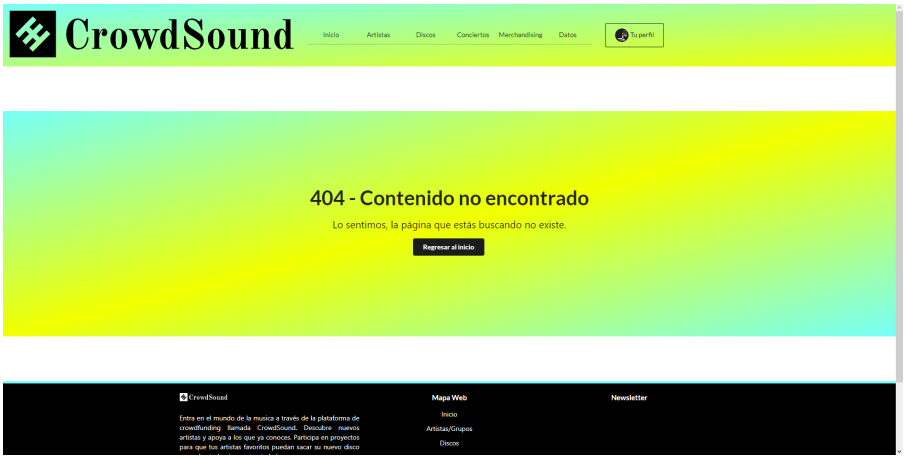



Figura 4.30: Página 404

# Capítulo 5

# Despliegue

## 5.1. Back End

Para llevar a cabo el despliegue del servidor, se evaluaron diversas alternativas, pero finalmente se decidió utilizar Render.

Render es una plataforma en la nube que brinda a los desarrolladores la posibilidad de desplegar y alojar sus aplicaciones y sitios web de manera fácil y eficiente. Ofrece una infraestructura en la nube administrada y optimizada, lo que significa que no es necesario preocuparse por la configuración y gestión de servidores.

El proceso de despliegue se llevó a cabo siguiendo los siguientes pasos: en primer lugar, se procedió a vincular el repositorio de GitHub donde se encuentra el servidor, para lo cual se modificó la configuración de visibilidad del repositorio a público. A continuación, se configuró en Render la carpeta del repositorio que alberga el Back End, y se establecieron los comandos necesarios para compilar y lanzar el servidor. Una vez completado el despliegue, se realizaron pruebas utilizando Postman para verificar la funcionalidad de todas las API del servidor.

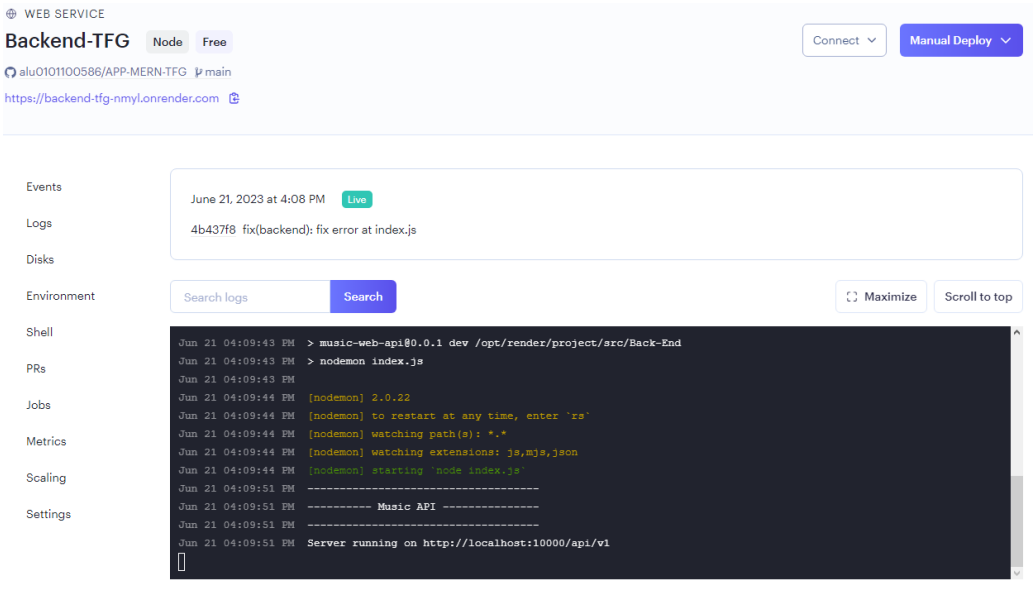



Figura 5.1: Despliege del Back End

## 5.2. Frontend

Para el despliegue del Front End se eligió utilizar Netlify, debido a la experiencia previa positiva que se había tenido con esta plataforma en proyectos anteriores.

Netlify es una plataforma de alojamiento y despliegue diseñada específicamente para sitios web estáticos. Su enfoque se basa en el uso de Git, automatización de despliegue y características adicionales que simplifican el proceso de publicación en la web y brindan una experiencia de desarrollo más fluida para los desarrolladores web.

El despliegue se llevó a cabo mediante la carga de la carpeta ”build”, que se genera como resultado de ejecutar el comando ”yarn run build”. Aunque el despliegue inicial se realizó sin problemas, se presentó un inconveniente al recargar la página, mostrando el error ”Page not found”. Para solucionar este problema, se realizaron algunas configuraciones en el Front End.

En el archivo correspondiente a la figura 3.14, se realizó un cambio sustituyendo el enrutador estándar ”BrowserRouter” por ”HashRouter”. La diferencia radica en cómo se manejan las rutas en la aplicación. Mientras que BrowserRouter utiliza rutas con la sintaxis estándar (”/ruta”), HashRouter utiliza rutas con un hash ”#” antes (”/#/ruta”). Esta variante es útil cuando se aloja la aplicación en un servidor que no está configurado para redireccionar todas las rutas al archivo principal.

Por otro lado, en el archivo ”redirects” ubicado en la carpeta ”public”, se agregó la siguiente línea: ”/* /index.html 200”. Esta configuración le indica al servidor que, cualquier solicitud que se realice, se redirija al archivo ”index.html” y se responda con el código de estado 200 (éxito). Esto asegura que todas las rutas sean manejadas por la aplicación y evita que se muestre el error de ”Page not found” al recargar la página.

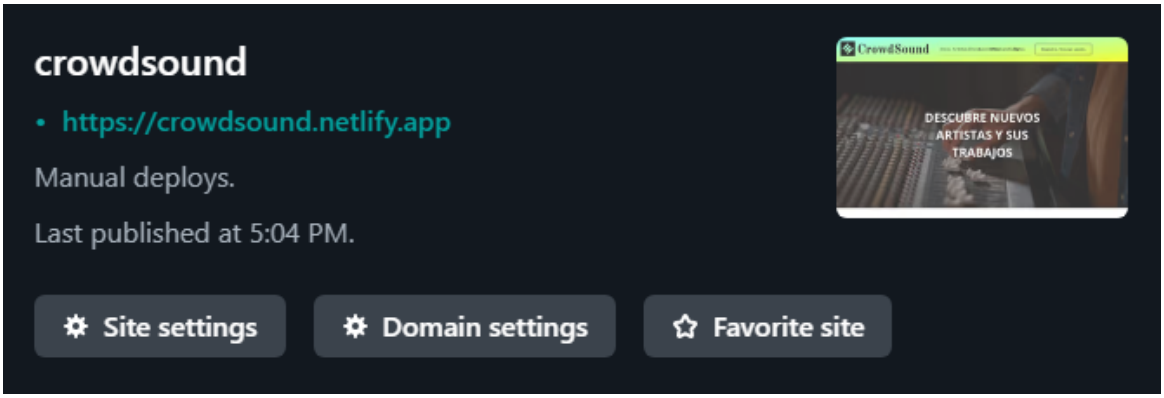



Figura 5.2: Despliege del Front End

# Capítulo 6

# Conclusiones y líneas futuras

## 6.1. Conclusiones

En conclusión, se ha creado una aplicación web que ofrece a los artistas la posibilidad de gestionar y crear proyectos basados en crowdfunding, al tiempo que permite a los usuarios descubrir géneros y artistas de su interés, así como participar en los proyectos de dichos artistas. En términos generales, se trata de una aplicación completa que facilita la gestión de proyectos de forma intuitiva y el acceso a la información de manera sencilla. La aplicación tiene un gran potencial y se ha llevado a cabo su desarrollo para que sea lo más escalable posible.

A pesar del tiempo, el uso de tecnologías y lenguajes de programación poco familiares, se logró cumplir con la mayoría de los objetivos establecidos, incluso se lograron incorporar objetivos secundarios, como la inclusión de merchandising y las estadísticas de la aplicación.

En lo personal, el desarrollo Full stack es un ámbito de gran interés, y trabajar en este proyecto ha permitido adquirir conocimientos sobre diversas tecnologías. Además, el mundo de la música es algo familiar para mí, por lo que dedicar mi proyecto de fin de carrera a estos dos aspectos me ha llevado a invertir mucho tiempo y pasión en el desarrollo de esta aplicación, a pesar de que algunos aspectos no hayan quedado completamente a mi gusto.

## 6.2. Líneas Futuras

La aplicación presenta un problema relacionado con el almacenamiento de imágenes que se suben, como fotos de avatares, portadas de discos y otros elementos. En la actualidad, todas las imágenes se guardan en el servidor y se vinculan a cada objeto a través de su ubicación almacenada en la base de datos. Para mejorar la aplicación en el futuro, se sugiere explorar la posibilidad de almacenar las imágenes en un servidor dedicado exclusivamente para este propósito o utilizar servicios especializados en este tipo de almacenamiento. Además, será necesario modificar cómo el backend gestiona la asignación de imágenes para asegurar un correcto funcionamiento.

Por otro lado, con relación a los avatares de artistas y usuarios comunes, actualmente

la aplicación permite subir imágenes de cualquier tamaño. Sin embargo, sería más óptimo establecer un mecanismo que defina un tamaño estándar para las imágenes. Esto evitará que las imágenes se muestren de forma distorsionada en la aplicación. Además, se podría implementar la opción de recortar las imágenes al tamaño establecido en caso de que sean demasiado grandes.

Otra posible mejora de la aplicación se refiere a la disposición y la información proporcionada en relación con los discos, conciertos y merchandising. Se sugiere mejorar la presentación y la cantidad de información ofrecida específicamente para estos elementos.

# Capítulo 7

# Summary and Conclusions

## 7.1. Conclusions

In conclusion, a web application has been created that offers artists the possibility to manage and create crowdfunding projects, while allowing users to discover genres and artists of interest and participate in their projects. Overall, it is a comprehensive application that facilitates intuitive project management and easy access to information. The application has great potential, and its development has been carried out to be as scalable as possible.

Despite the time constraints and the use of unfamiliar technologies and programming languages, most of the established objectives were achieved. Secondary objectives, such as the inclusion of merchandise and application statistics, were also successfully incorporated.

Personally, Full Stack development is an area of great interest to me, and working on this project has allowed me to acquire knowledge about various technologies. Additionally, the world of music is familiar to me, so dedicating my final project to these two aspects has led me to invest a lot of time and passion in the development of this application, despite some aspects not being entirely to my liking.

## 7.2. Furure Work

The application presents an issue related to the storage of uploaded images, such as avatar photos, album covers, and other elements. Currently, all the images are stored on the server and linked to each object through their location stored in the database. To improve this aspect, it is possible to store the images on a dedicated server solely for this purpose or utilize specialized services for this type of storage. Additionally, it will be necessary to modify how the backend manages image allocation to ensure proper functioning.

On the other hand, regarding the avatars of artists and regular users, the current application allows uploading images of any size. However, it would be more optimal to establish a mechanism that defines a standard size for the images. This will prevent the images from being displayed distorted in the application. Additionally, the option to crop

the images to the established size could be implemented if they are too large.

Another possible improvement to the application relates to the layout and information provided regarding albums, concerts, and merchandise. It is suggested to enhance the presentation and the amount of information specifically offered for these elements.

# Capítulo 8

# Presupuesto

## 8.1. Desarrollo del software

Durante el desarrollo del software se han contado las horas de las diferentes etapas de desarrollo, tomando en cuenta un valor de 36€/h.

| Tipos | Diseño (h) | Desarrollo(h) | Costes(€) |
|---|---|---|---|
| Base de datos | 3 | 5 | 288 |
| Back End | 8 | 60 | 2.448 |
| Front End | 10 | 70 | 2.880 |
| Total | 21 | 135 | 5.616 |

Tabla 8.1: Coste de desarrollo

## 8.2. Recursos empleados

Se utilizaron tecnologías, API y servicios de terceros aprovechando sus opciones gratuitas. Por consiguiente, hemos implementado la base de datos en la nube utilizando MongoDB Atlas y su versión gratuita, la cual ofrece un espacio de almacenamiento de poco más de 500 MB. Para el servidor, se utilizó el servicio web de Render, el cual nos permite desplegar el Back End. En cuanto al Front End, hemos utilizado la API de OpenCageData, la cual nos permite realizar hasta 3000 traducciones de direcciones diariamente sin costo alguno. Además, hemos utilizado Netlify para el despliegue del propio Front End, sin necesidad de realizar pagos.

También se utilizó el entorno de desarrollo Visual Studio Code (VsCode) para la programación y se aprovechó la plataforma de control de versiones GitHub para la gestión del código. Es importante mencionar que tanto VsCode como GitHub no implicaron ningún pago mensual u otro tipo de costo durante su utilización.

| Recurso | Coste (€/mes) |
|---|---|
| VsCode | 0 |
| GitHub | 0 |
| MongoDB Atlas | 0 |
| Render | 0 |
| Netlyfy | 0 |
| OpenCageData | 0 |

Tabla 8.2: Coste de recursos empleados

## 8.3. Total

Durante el desarrollo del sofware se han contado las horas de las diferentes etapas de desarrollo, tomando en cuenta un valor de 36€/h.

| Tipos | Coste plano (€) | Coste mensual (€) |
|---|---|---|
| Desarrollo Software | 5.616 | - |
| Recursos | - | 0 |

Tabla 8.3: Coste total del proyecto

# Bibliografía


[1] Cejudo, A., & Ramil, X. (2013). Crowdfunding: Financiación colectiva en clave de participación.

[2] NodeJs - https://nodejs.org/en/docs

[3] Express - https://expressjs.com/es/guide/routing.html

[4] Bcrypt - https://yarnpkg.com/package/bcrypt

[5] Multiparty - https://yarnpkg.com/package/connect-multiparty

[6] JWT - https://jwt.io

[7] Mongoose - https://mongoosejs.com

[8] Mongoose-Pagination - https://yarnpkg.com/package/mongoose-paginate

[9] React - https://es.react.dev/reference/react

[10] Semantic Ui React - https://react.semantic-ui.com

[11] React Router Dom - https://reactrouter.com/en/main

[12] React DropZone - https://react-dropzone.js.org

[13] React DatePicker - https://reactdatepicker.com

[14] React Chartjs 2 - https://react-chartjs-2.js.org

[15] Leaflet - https://leafletjs.com

[16] Formik - https://formik.org

[17] Yup - https://yarnpkg.com/?q=yup&p=1

[18] Sass - https://sass-lang.com