Desarrollo de una Aplicación Web para profesionales del sector musical

Memoria de Proyecto Final de Máster

Máster Universitario en Desarrollo de Sitios y Aplicaciones Web

Autor: Sara Paz Fernández

Consultor: Anna Ferry Mestres

Profesor: Juliá Minguillón Alfonso

Profesor: Cesar Pablo Córcoles Briongos

06/01/2020

Copyright

Abstract

Este proyecto consiste en el desarrollo de una aplicación web para profesionales del sector musical, con el objetivo de facilitar la búsqueda de empleo en este ámbito y la conexión entre distintos músicos para crear proyectos musicales. Surge de la necesidad de una plataforma en la que instrumentistas, cantantes, compositores y otros profesionales cubran sus necesidades, como encontrar compañeros para un proyecto musical o un empleo como profesor de música. Para su desarrollo se utiliza el *framework* de JavaScript Angular, aportando agilidad por tratarse de una SPA (*Single Page Application*). Se adapta a cualquier tamaño de pantalla y, al ser una PWA (*Progressive Web App*), se puede instalar en el dispositivo y utilizar algunas funciones sin conexión a Internet, lo que ofrece una mejor experiencia de usuario. El *backend* está desarrollado con TypeScript, y utiliza el *framework* Express de Node.js.

Keywords: PWA, Angular, Aplicación web, música, empleo para músicos

Abstract (english version)

This project consists in the development of a web application for professionals in the music sector, with the aim of facilitating the job search in this field and the connection between different musicians to create projects. It arises from the need for a platform where instrumentalists, singers, composers and other professionals cover their needs, such as finding partners for a musical project or a music teacher job. The Angular JavaScript framework is used for its development, providing agility because it is a SPA (Single Page Application). It adapts to any screen size and can be installed on the device and use some functions without an Internet connection as it is a PWA (Progressive Web App), which offers a better user experience. The backend is developed with TypeScript, and uses the Node.js framework Express.

Keywords: PWA, Angular, Web application, music, musician job

Índice

Figuras y tablas

Lista de imágenes, tablas, gráficos, diagramas, etc., numeradas, con títulos y las páginas en las que aparecen.

Índice de figuras

Índice de tablas

1. Introducción

1.1 Contexto y justificación del trabajo

Hoy en día la tecnología está presente en casi todos los ámbitos de nuestra vida. Sin embargo, en el sector de la música, a pesar de que existen varias páginas y aplicaciones web específicas para músicos, pocas contemplan buscadores para cubrir necesidades como, por ejemplo, encontrar a un músico cercano con unas características concretas o tener acceso a todas las ofertas para músicos publicadas en las principales páginas de empleo del país en una sola herramienta.

Mi experiencia en este sector es la que me motiva a desarrollar esta herramienta, ya que la música es una de mis aficiones y una pequeña parte de mi carrera profesional. Por lo tanto, este Trabajo de Fin de Máster (TFM) se involucra bastante en mi vida personal.

2. Descripción

Este Trabajo de Fin de Máster (TFM) trata de desarrollar una aplicación web progresiva destinada para músicos, y también para personas que quieren contratar a un músico.

La idea surge de la necesidad de conectar a los músicos entre ellos permitiendo, de esta manera, encontrar fácilmente compañeros para un proyecto musical o para una actuación en concreto. Creando una plataforma como punto de encuentro entre músicos de la misma región, se fomenta la colaboración y aumentan las posibilidades de promoción. Por otro lado, se unificarán muchas de las ofertas de empleo destinadas para músicos publicadas en distintas webs de empleo, lo que facilitará la búsqueda de empleo en este sector.

El objetivo principal de la aplicación es la conexión entre músicos para realizar proyectos en común, y facilitar la búsqueda de empleo en el sector musical.

A continuación, se describen las funcionalidades principales de la aplicación:

- Buscar y mostrar ofertas de empleo para profesionales del sector musical.
- Crear un perfil de usuario, con posibilidad de añadir foto de perfil, enlaces a redes sociales y a vídeos publicados en *Youtube* o *Vimeo*.
- Buscar profesionales entre todos los usuarios de la aplicación. Principalmente se crearán filtros por estilo, instrumento y ubicación.
- Registro y reseteo de contraseña con confirmación por correo electrónico.

Las tecnologías utilizadas para el desarrollo de la aplicación son Angular 8 para el *frontend*, y una API en Express para el *backend*. También se crea una base de datos en MySql. Para el *web scraping* se elige *Puppeteer*, una librería para Node.js de Google Chrome, y se utiliza en el *backend*, realizando una búsqueda por las distintas webs cada 6 horas.

3. Objetivos

A continuación, se enumeran los objetivos principales y secundarios de este trabajo.

3.1 Principales

- Desarrollar una aplicación que permita buscar profesionales del sector musical, filtrando por profesión, estilo musical, instrumento y ubicación, así como buscar ofertas de empleo para músicos.
- Desarrollar una aplicación web progresiva en Angular que sea escalable, adaptable a todos los dispositivos, intuitiva, y que ofrezca buena experiencia de usuario, poniendo en práctica los conocimientos adquiridos a lo largo del Máster.
- Adquirir conocimientos en Express para realizar una API que sirva de interfaz entre la aplicación y la base de datos.
- Obtener experiencia en *Web Scraping* con herramientas de JavaScript.

3.2 Secundarios

- Promocionar músicos.
- Facilitar la comunicación entre la comunidad de músicos para cubrir distintas necesidades.

4. Marco teórico

Se han encontrado varias aplicaciones similares a la que se desarrolla en este proyecto. A continuación se detallan las más relevantes:

- https://myjobmusic.es: Gestor de empleo para músicos y músicas, en la que empresas y particulares solicitan artistas para eventos.
- https://empleoparamusicos.com: Portal de empleo para músicos y músicas, en la que se muestran ofertas de trabajo publicadas en otras webs.
- https://www.reverbnation.com: Red social internacional en la que bandas y artistas crean sus perfiles y publican sus canciones. Los usuarios pueden escuchar canciones, comprarlas, seguir a los artistas, etc.
- http://www.miuseek.com: Red social para músicos y músicas en la que se visualizan los distintos perfiles.
- http://galiciantunes.com: Marca de la promoción internacional de la música gallega, ideada como plataforma conjunta de las instituciones, empresas y asociaciones profesionales del sector musical en Galicia. Grupos y artistas publican información como vídeos, discografía y datos de contratación.

Al comprobar que existen varios productos funcionando actualmente que cubren varias de las funcionalidades de la aplicación, se decide enfocarla para un ámbito más local, centrándose en la comunidad de músicos de Galicia. Dentro de este marco, *GalicianTunes* es la web que más se parece a este proyecto. Sin embargo, los objetivos principales que se describen en la propia web son contribuir al fortalecimiento de la industria musical gallega, lograr una mayor visibilidad social del sector y favorecer su proyección internacional. Los objetivos principales del proyecto no son ninguno de estos, sino que son facilitar la búsqueda de empleo para músicos y músicas, y conectar a los artistas entre ellos para crear nuevos proyectos.

Se ha descubierto un grupo de Facebook llamado “Se busca/ Se ofrece Músico Galicia”, en el que se publican ofertas y demandas de músicos y músicas. Principalmente hay publicaciones buscando un músico o música para tocar algún instrumento o cantar en bandas nuevas o ya existentes. Se ha comprobado que en los últimos días ha habido varias publicaciones con respuestas. Este grupo de Facebook con más de 1800 miembros y con una actividad relevante, demuestra que hay un número importante de posibles usuarios de la aplicación.

5. Contenidos

Los elementos que están presentes en todas las pantallas de la aplicación son el menú de navegación y el pie de página. El menú estará situado en la parte superior y será estático, es decir, aunque se haga *scroll* hacia abajo, siempre estará visible en la parte superior de la pantalla. El pie contendrá el mapa del sitio, permitiendo acceder a cualquier página desde él.

La aplicación está compuesta por las siguientes pantallas:

- **Inicio**: En esta página se mostrará una breve descripción de la aplicación, y algún enlace a las partes más relevantes. También contará con una sección con los últimos anuncios de empleo para músicos.
- **Nuevo usuario**: Formulario de registro para los nuevos usuarios.
- **Lista de profesionales**: Listado con todos los profesionales registrados en la aplicación. Se buscarán usuarios a través de filtros como especialidad (instrumentista, cantante, compositor/a …), estilo musical y ubicación. Al pulsar en buscar se mostrarán los resultados. Al pulsar sobre un registro, se entra en el perfil del usuario seleccionado.
- **Mi perfil**: Formulario para modificar los datos de perfil, accesible solamente si el usuario está autentificado.
- **Detalles del profesional**: Pantalla con los detalles del profesional seleccionado.
- **Ofertas de empleo**: Pantalla donde se visualizarán ofertas de empleo publicadas en otras aplicaciones como MilAnuncios o Indeed. Solamente es accesible para usuarios autentificados.
- **Detalle de la oferta**: Página con los detalles de la oferta, incluyendo un enlace donde se encontrará más información. En algún caso, este enlace descarga publicaciones del BOE u otros documentos oficiales. Solamente es accesible para usuarios autentificados.
- **Contacto**: Formulario para enviar comentarios o sugerencias sobre la aplicación.

6. Metodología

Antes del inicio del proyecto, hay una fase previa que consiste en la determinación de la temática de la aplicación y la investigación sobre las herramientas existentes actualmente en el mercado.

La fase inicial del trabajo consiste en el análisis completo de la aplicación. Se concretan las funcionalidades, los objetivos y los contenidos, las tecnologías a utilizar, los recursos necesarios para su desarrollo, se define la arquitectura de la aplicación y se planifica el desarrollo de la aplicación.

En la segunda fase se configura el entorno de trabajo y se inicia el desarrollo de la aplicación, documentando los procedimientos más relevantes. En este periodo se realiza un análisis de mercado, un estudio de usabilidad, se crean diagramas de flujo, árbol de navegación y casos de uso, y se diseñan los *wireframes* de las pantallas principales. Finalmente, se consolidan los fundamentos de la aplicación, revisando su arquitectura y se crea la base de datos.

En la siguiente fase se crea una primera versión de la aplicación con las funcionalidades más importantes, siendo esta fase la más completa en cuanto al desarrollo de la aplicación. Durante todo el proceso se documenta y justifica el desarrollo.

Por último, en la fase final se finaliza y se entrega el proyecto, finalizando su desarrollo y su documentación. Se realizan los test necesarios, se corrigen errores y se hace una revisión completa de la memoria. También se crean las presentaciones en vídeo y en diapositivas.

7. Arquitectura de la aplicación/sistema/servicio

Para desarrollar esta aplicación se decide utilizar Angular para el *frontend*, la parte con la que interactúa el usuario, MySql para la base de datos, y Express y Node.js para servir una REST API que recibirá las peticiones de la aplicación para enviar y obtener información de la base de datos. El cliente y el servidor estarán en proyectos diferentes.

7.1 *Frontend*

Angular será el *framework* que se utilizará para crear la parte del cliente de la aplicación, y se sigue la arquitectura estándar de una aplicación en Angular.

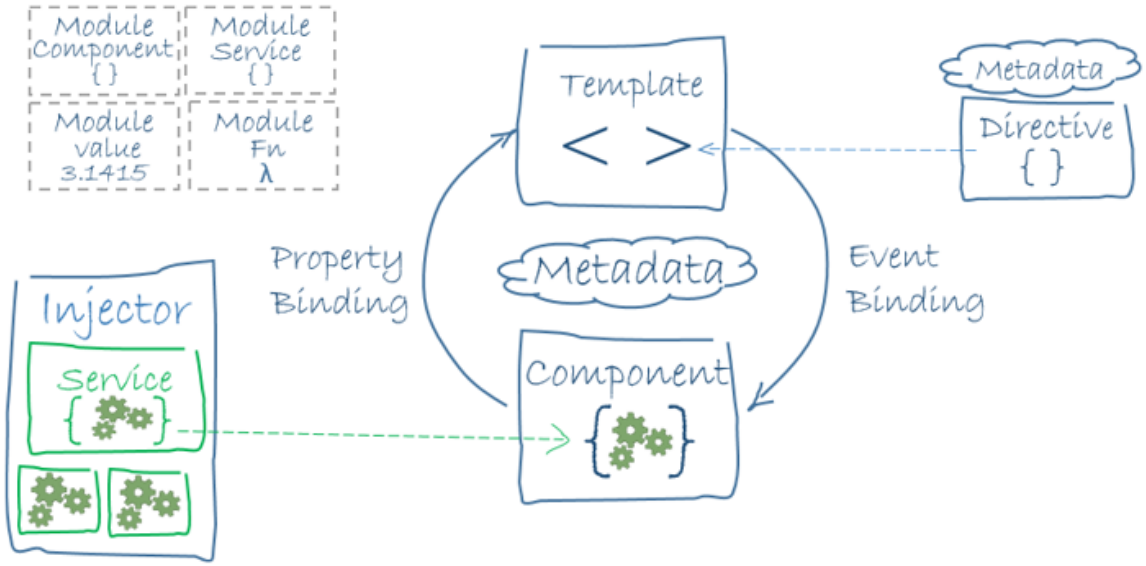

Ilustración 1: Arquitectura de Angular [1]

[1] https://angular.io/guide/architecture, 2019

La estructura de carpetas será la siguiente:

-src
    -app
        -helpers
        -home
        -models
        -pipes
        -services
        -…
    -assets
        -fonts
        -icons
        -img

La carpeta 'models' tendrá las clases que representan los objetos de la base de datos para facilitar el manejo de los datos.

En la carpeta 'services' estarán los servicios, encargados de realizar las llamadas a la API para cada acción.

Habrá una carpeta para cada componente de la aplicación dentro de 'src/app'.

7.2 *Backend*

Para el *backend* se crea una API REST en Express y se sirve con Node.js.

La estructura de carpetas será la siguiente:

-controllers
-routes

Los controladores, alojados en la carpeta 'controllers', implementarán funciones para cada acción en la que se necesite interactuar con la base de datos, tanto para obtener como para introducir o modificar información.

En el directorio 'routes' habrá un archivo por cada controlador, donde se definen los *endpoints* (URL) de la API y las funciones de los controladores para recibir las peticiones desde la aplicación. A mayores, en el archivo principal del *backend* (server/src/index.ts), se invoca a una función de una clase llamada 'webScrapping.ts'. Esta llamada se implementa dentro de la función de JavaScript *setInterval* con un intervalo de 6 horas entre cada llamada.

7.2 Base de datos

La base de datos se creará en *MySql* a través de *phpMyAdmin* en un servidor Wamp.

8. Plataforma de desarrollo

A continuación, se detallan los recursos tecnológicos necesarios para el desarrollo del proyecto:

- **Software**: Para desarrollar tanto el *frontend* como el *backend* se utiliza *Visual Studio Code*, y para crear la base de datos se ejecuta un servidor en local con *Wamp*, y se crea la base de datos en *PhpMyAdmin*. *Git* es el controlador de versiones.
- **Hardware**. El Hardware utilizado es un ordenador portátil Acer I5, con 8Gb de memoria RAM y sistema operativo Windows 10.
- **Web apps**: Se utiliza la interfaz web de *GitHub* para realizar algunas operaciones de control de versiones, aunque mayoritariamente se utiliza la consola.

9. Planificación

Para planificar el trabajo, se divide el proceso en cuatro fases principales. En la planificación se han distribuido los días contando que se dedican 4 horas diarias en la primera fase, 6 horas en las demás, y 8 horas diarias en los últimos 3 o 4 días, sin contar los fines de semana y festivos.

- **Análisis e investigación**: Se realiza una investigación sobre las aplicaciones similares a la idea del proyecto y un estudio de las herramientas adecuadas para el desarrollo de este tipo de aplicaciones. También se elabora la metodología de trabajo y la planificación, se definen las funcionalidades, objetivos y contenidos de la aplicación. Una vez que se deciden las herramientas a utilizar para el desarrollo, se analizan los recursos necesarios y se diseña la arquitectura.
- **Inicio del desarrollo y su documentación**: Se crea el diagrama de la base de datos, el árbol de navegación, el estudio de usabilidad y casos de uso. Posteriormente, se configura el entorno de desarrollo y se crea la base de datos para poder iniciar el desarrollo, comenzando por el *backend*. En esta fase se inicia el proceso de documentación de la aplicación, que estará presente hasta el fin del trabajo.
- **Desarrollo de la primera versión de la aplicación**: Esta fase se centra en el desarrollo de la aplicación, y en ella se comienza a desarrollar el *frontend*, creando los primeros servicios y componentes, y aplicando estilos a la web con herramientas como SCSS y Bootstrap. Finalmente, se publica una primera versión en un servidor.
- **Finalización del proyecto**: Se implementan los servicios y componentes restantes y el servicio de *web scraping*, además de las pruebas necesarias. Por último, se revisa la memoria y se elabora la presentación en diapositivas y en vídeo.

Fase	Tarea	Fecha inicio	Fecha fin	Horas	Días
Análisis e investigación	Primera fase: Todas	18/09/2019	01/10/2019	**44**	**11**
	Investigación sobre herramientas similares en el mercado	18/09/2019	19/09/2019	8	2
	Definición del problema e inicio de la memoria	20/09/2019	23/09/2019	6	1.5
	Definición de funcionalidades y contenidos de la aplicación	23/09/2019	25/09/2019	8	2
	Definición de objetivos	25/09/2019	26/09/2019	4	1
	Determinación de las tecnologías a utilizar y recursos necesarios para el desarrollo	26/09/2019	26/09/2019	2	0.5
	Diseño de la arquitectura	27/09/2019	27/09/2019	4	1
	Elaboración de la metodología y planificación del trabajo	28/09/2019	28/09/2019	4	1
	Revisión del índice y los primeros puntos de la memoria	30/09/2019	01/10/2019	8	2
Diseño e inicio del desarrollo	Segunda fase: Todas	02/10/2019	30/10/2019	**126**	**21**
	Diseño del árbol de navegación y casos de uso	02/10/2019	02/10/2019	6	1
	Perfiles de usuario	04/10/2019	07/10/2019	12	2
	Creación de prototipos	08/10/2019	18/10/2019	54	9
	Configuración del entorno de trabajo	21/10/2019	21/10/2019	6	1
	Diseño del diagrama de la base de datos	22/10/2019	22/10/2019	3	0.5
	Creación de la base de datos	22/10/2019	23/10/2019	9	1.5
	Revisión de memoria y desarrollo del *backend*	23/10/2019	30/10/2019	36	6
Desarrollo de la primera versión de la aplicación	Tercera fase: Todas	31/10/2019	08/12/2019	**156**	**26**
	Desarrollo de servicios y de componentes en Angular: Inicio, Nuevo Usuario, Iniciar sesión, Mi perfil, Detalles del profesional, Ofertas de Empleo	31/10/2019	02/12/2019	138	23
	Publicación de la primera versión en un servidor	04/12/2019	04/12/2019	6	1
	Corrección de la memoria	05/12/2019	05/12/2019	6	1
	Creación del vídeo de demostración	08/12/2019	08/12/2019	6	1
Finalización del proyecto	Cuarta fase: Todas	09/12/2019	06/01/2020	**114**	**18**
	Desarrollo del servicio de *web scraping*	09/12/2019	27/12/2019	78	13
	Pruebas y correcciones	30/12/2019	02/01/2020	12	2
	Revisión, corrección y finalización de la memoria	03/01/2020	03/01/2020	8	1
	Elaboración de presentaciones en diapositivas y vídeo	04/01/2020	06/01/2020	16	2

Tabla 1: Planificación del trabajo

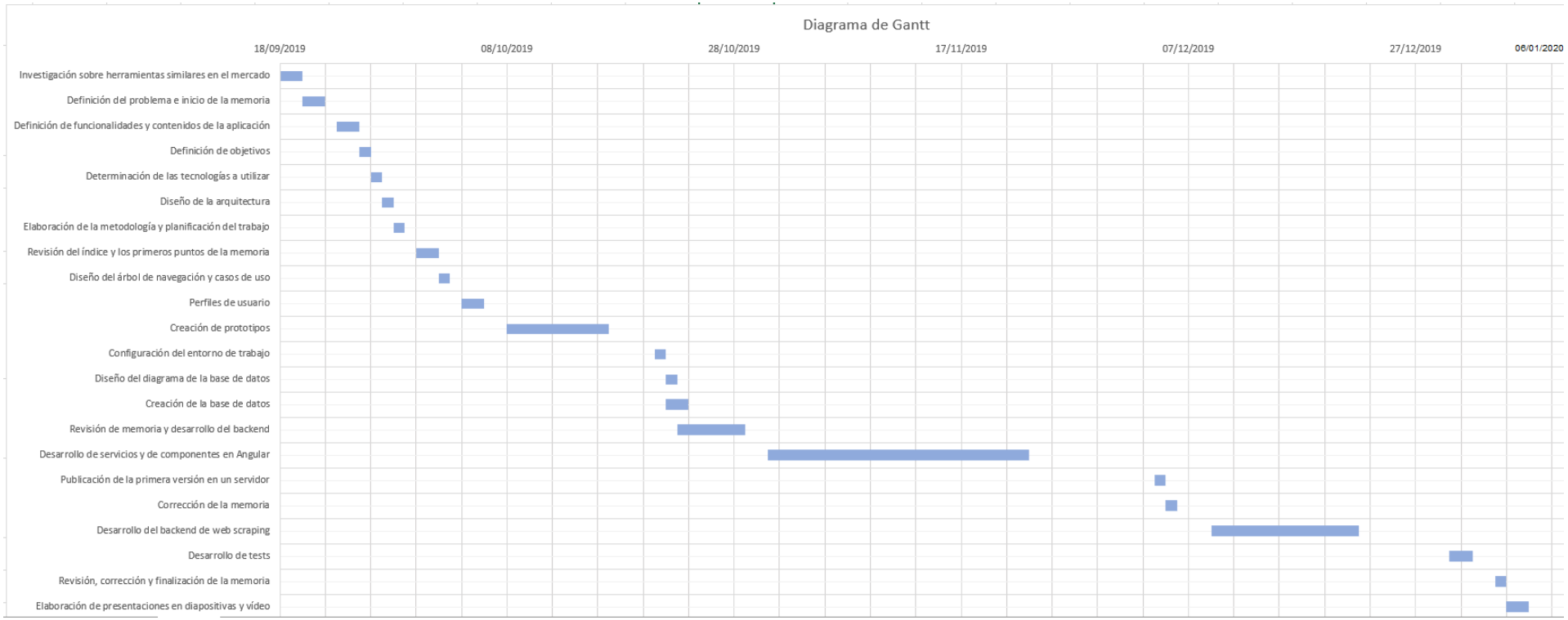

Ilustración 2 Diagrama de Gantt

10. Proceso de trabajo

El proceso de desarrollo para este proyecto se divide en cuatro etapas, marcadas por las fechas de entrega. Las dos primeras etapas se centran en la planificación, diseño de la arquitectura, estudio de los perfiles de usuarios tipo de la aplicación y realización de prototipos.

La tercera entrega consiste en la mayor parte del desarrollo de la aplicación. Para el desarrollo se comienza por la creación del *backend* con Express, y la implementación de los principales endpoints de la API REST. Se continúa creando el *frontend* con Angular CLI. Se crean los componentes, servicios, modelos y demás clases necesarias, y se comienza la implementación de las distintas funcionalidades. Al mismo tiempo se van añadiendo y modificando funciones del *backend*.
En la última etapa se desarrolla el *web scraping* de MilAnuncios y demás páginas, se corrigen errores, se refactoriza lo que sea posible y se realizan pruebas.

A continuación, se detallan los procesos del desarrollo que provocaron atascos o complicaciones.

- **Foto de perfil**: La funcionalidad de subir una foto de perfil ha tenido alguna complicación. En primer lugar, la librería 'multer' parecía una buena solución para subir los ficheros a una carpeta del mismo servidor donde se encuentra la API. Se ha comprobado que funciona en local, pero concretamente en 'Heroku' no es posible debido a un proceso que elimina todos los archivos cada poco tiempo. Finalmente se ha optado por utilizar la librería 'promise-ftp', además de 'multer'. Una vez subido el archivo, como no se borra instantáneamente, se realiza una subida a través de FTP al servidor donde se aloja el frontend. Los datos de conexión al FTP se indican en el archivo de configuración 'server/src/ftp.ts'. También se desarrolla la eliminación de la antigua foto del usuario si tenía alguna ruta en el campo foto de la tabla en base de datos. Finalmente se tuvo que cambiar la conexión FTP con una SFTP, ya que se decidió utilizar el servidor facilitado por la UOC para guardar las imágenes, y éste utiliza una conexión SFTP.
  Además de este problema, cuando se ha probado la aplicación en varios dispositivos móviles y el usuario saca la foto con la cámara, en algún dispositivo la foto se mostraba rotada lateralmente. Se ha añadido una pequeña funcionalidad en el *frontend* para solventar este problema. Cuando el usuario saca la foto, si después pincha en la foto, ésta se rota un ángulo de 90º.
- **Sección Vídeos del componente 'Mi perfil'**: En esta parte ha surgido un problema de renderizado de los *iframes* de *Youtube* y *Vimeo*. En un primer intento, al asociar el valor del atributo src del *iframe* a una variable del usuario a través del doble *data binding* de Angular ([src] = "video"), daba un error en la consola del navegador: "Unsafe value used in a resource URL context". Para solventar este error, se ha añadido en el constructor del componente un *DomSanitizer*, una clase de la librería '@angular/platform-browser', que hace que el componente ignore este problema de seguridad. Posteriormente, se ha observado que cualquier cambio en el formulario como un foco en un campo de texto, provocaba que el *iframe* volviese a ser cargado, lo que provocaba que los

vídeos desapareciesen y volviesen a aparecer continuamente de forma molesta. Para solucionar esto, se ha creado un componente concreto para los vídeos, que se muestra en cada interacción de la colección de vídeos del usuario. A este componente hijo se le indica en la directiva '@Component()' la propiedad ' `changeDetection: ChangeDetectionStrategy.OnPush`'

- **Actualizar usuario**: Aparte de las dificultades con la foto de perfil, fue necesario desarrollar en el *backend* un sistema para comparar todos los instrumentos, profesiones, estilos y vídeos asociados al usuario, eliminando de la base de datos los registros que no existan en el cuerpo de la petición, y añadiendo los nuevos.

A lo largo del desarrollo se han detectado y desarrollado optimizaciones. Por ejemplo, en lugar de hacer una llamada a la API cada vez que se cambia de provincia para obtener los ayuntamientos, se desarrolla una funcionalidad para almacenar los datos en *localStorage* (profesiones, instrumentos, estilos, provincias, últimos 3 profesionales registrados y ofertas de empleo). En el caso de los últimos profesionales y las ofertas de empleo, si han pasado más de dos horas desde la última sincronización con la base de datos, se vuelven a realizar las peticiones. De esta forma se ahorran muchas llamadas a la API, se ofrece mejor rendimiento al mostrar los resultados más rápido y se depende mucho menos de la conexión a Internet.
En el desarrollo del *web scraping*, se han aplicado filtros a las ofertas, ya que aparecían muchos anuncios que no iban destinados a músicos aunque apareciese la palabra "música" en la descripción, por ejemplo, anuncios de ofertas de empleo para camareros.

Se ha intentado obtener los datos de la web "InfoJobs", pero detecta el *scraping*, y muestra un mensaje de bloqueo en lugar del código fuente que se puede ver si se inspecciona la página en un navegador. En "MilAnuncios" se mostró el mismo mensaje de bloqueo después de realizar muchas pruebas, pero al volver a realizar las consultas unas horas más tarde se solventó el problema.
También fue necesario implementar en el servicio del *web scraping* un sistema que compara las ofertas existentes en la base de datos con las nuevas. Para ello se establece un identificador único (en cada web se extrae de campos diferentes), se eliminan de la base de datos las ofertas que ya no existan, se añaden las nuevas, y solamente se actualiza una oferta si cambia la fecha, ya que se ha detectado que, por ejemplo, en MilAnuncios, en alguna oferta se actualiza la fecha cada cierto tiempo.

Las webs que se han añadido en el servicio de *web scraping* son las siguientes:

- MilAnuncios: https://www.milanuncios.com/ofertas-de-empleo/musico.htm?fromSearch=1&dias=90&demanda=n
- Xunta de Galicia: https://emprego.xunta.gal/portal/index.php/gl/buscar-emprego.html?emprego=musi
- CanalOposiciones: https://www.canaloposiciones.com/buultconv.asp?busca=musica
- Indeed: https://www.indeed.es/jobs?q=musica&sort=date

11. Librerías utilizadas

A continuación, se detallan las librerías utilizadas en el trabajo, tanto en el *backend* como en el *frontend*.

- **@types/express**: Express es la librería que se utiliza para servir la API REST con el objetivo de comunicar el *frontend* con la base de datos. Se utiliza en el servidor, en el archivo 'server/src/index.ts'.
- **@types/cors**: Middleware para agregar funcionalidad CORS a la API. Se utiliza en el archivo 'server/src/index.ts' para evitar problemas de acceso CORS desde el *frontend*.
- **@types/mysql:** Librería que permite conectar la API con la base de datos en MySQL. Se utiliza en los controladores, y la configuración se encuentra en el archivo 'server/src/database.ts'.
- **@types/nodemailer:** Se utiliza en la funcionalidad de registro en el servidor, para enviar el correo electrónico de confirmación a los nuevos usuarios y para actualizar la contraseña.
- **@types/morgan:** facilita la depuración escribiendo logs en cada llamada y respuesta de la API.
- **multer:** se utiliza para subir archivos a una carpeta del servidor, en el fichero 'server/src/routes/usersRoutes.ts' para actualizar la foto de perfil del usuario.
- **@types/promise-ftp:** Al observar que una vez desplegado el *backend* en Heroku, las fotos se eliminaban pasado un corto período de tiempo, se decide añadir esta librería para subir las fotos a través de FTP a una carpeta del servidor donde se encuentra el *frontend*. Se utiliza en la funcionalidad de actualizar usuario, en el archivo 'server/src/controllers/usersControllers.ts'
- **@types/bcrypt:** Se utiliza para encriptar la contraseña del usuario, en el archivo 'server/src/controllers/usersControllers.ts', para las funcionalidades de autentificación, registro y actualización de contraseña.
- **@types/jsonwebtoken:** Librería para crear un *token* de acceso *JWT*. Se utiliza para añadir seguridad en algunos *endpoints* de la API.
- **@fortawesome/fontawesome-free:** Librería utilizada en el *frontend* para iconos de 'fontawesome'.
- **bootstrap:** se utiliza en el *frontend* para facilitar el diseño. Por ejemplo, se utiliza el componente 'carousel' en la página de inicio, y para los mensajes de éxito y error.
- **Google Fonts:** Se utilizan las tipografías de Google Fonts 'Roboto', 'Alice', 'Courguette' y 'Material Icons'.
- **Puppeteer:** Para el *web scraping* se utiliza esta librería de Google Chrome.

12. Diagramas UML

12.1 Diagrama de base de datos

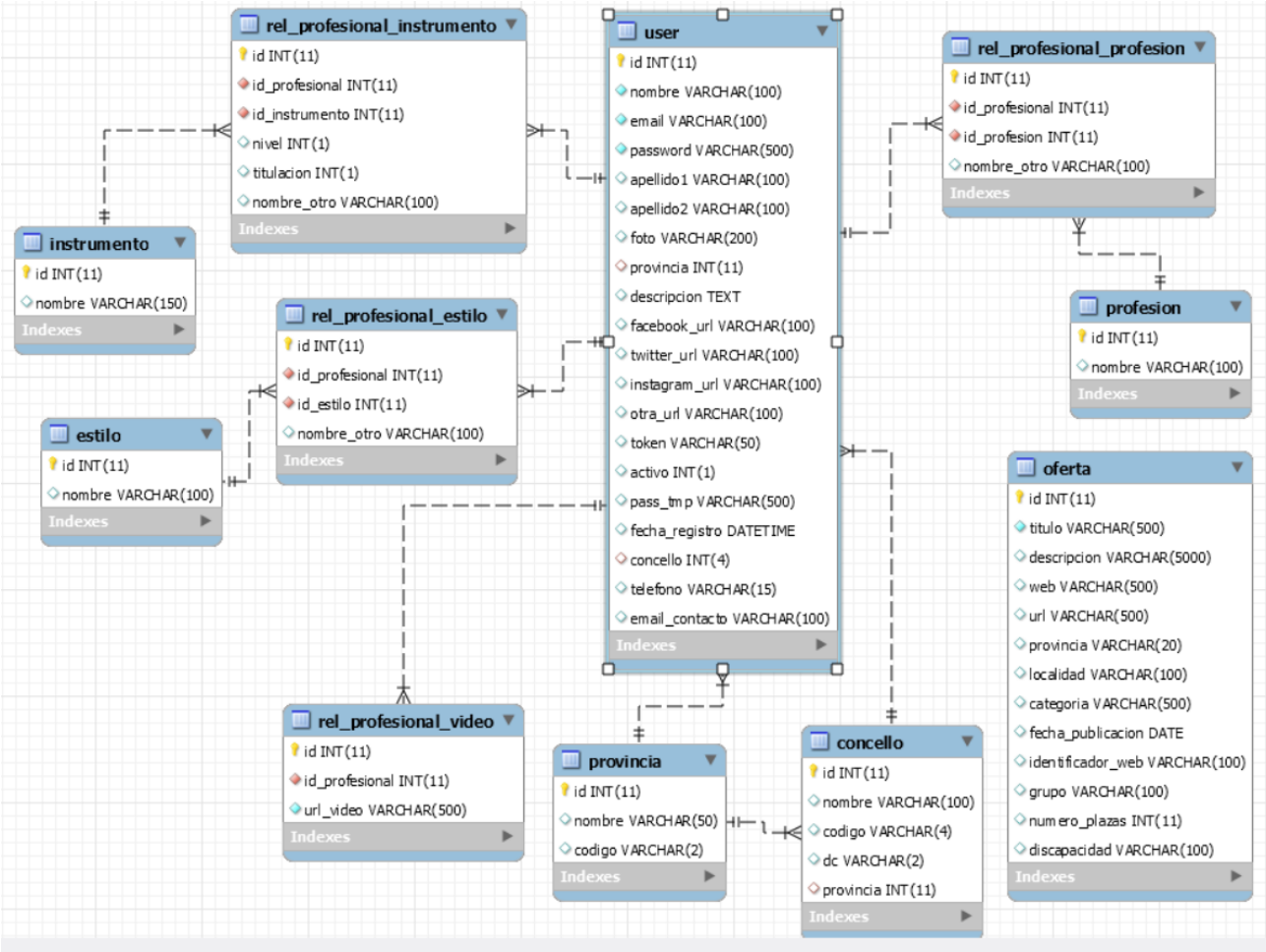

Ilustración 3 Diagrama ER

12.2 Árbol de navegación

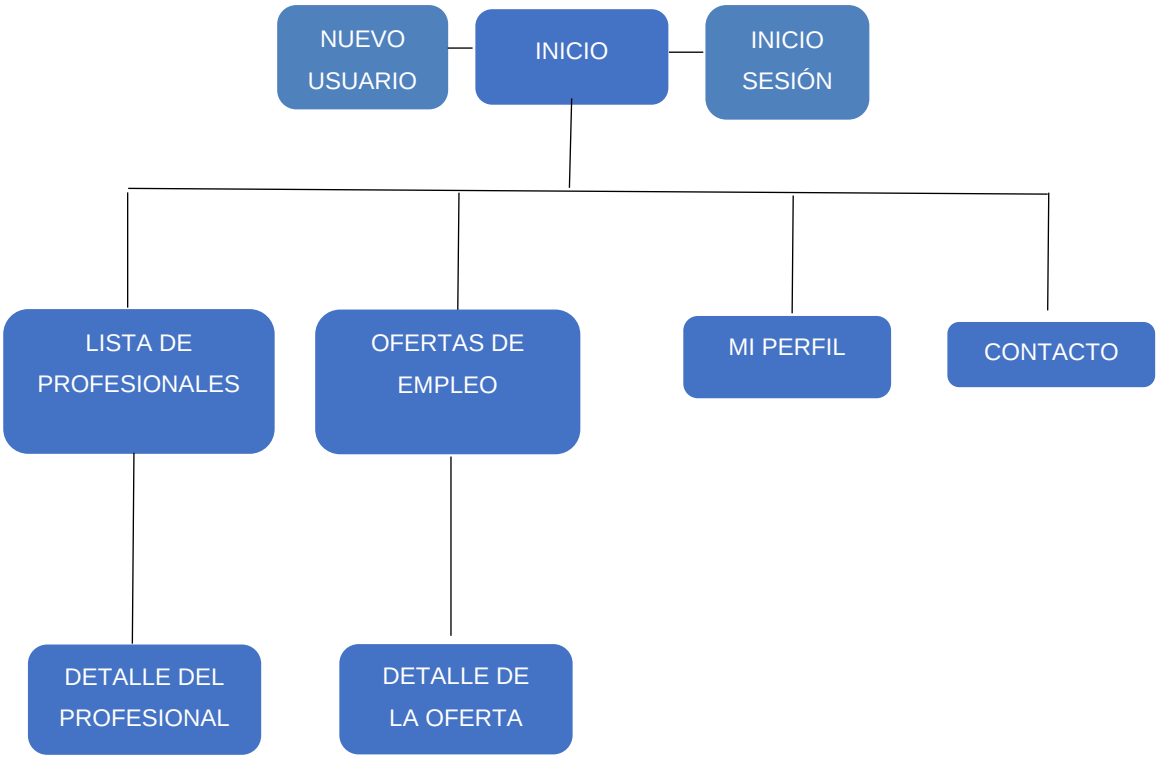

Ilustración 3: Árbol de navegación

12.3 Diagrama de casos de uso

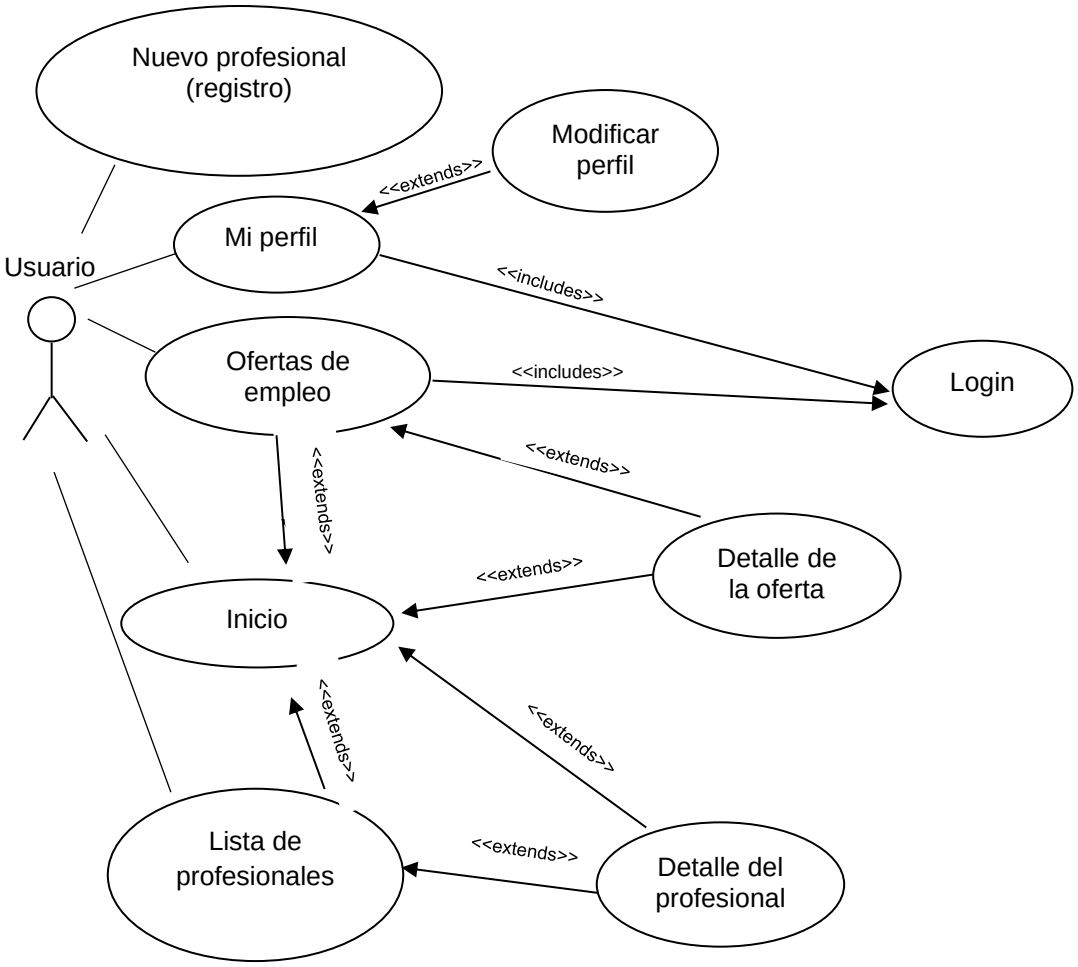

Ilustración 4 Diagrama de casos de uso

13. Prototipos

A continuación, se presentan los *wireframes* para los siguientes dispositivos:

- Escritorio y Tablet *Landscape*: Resolución de 1024px
- Smartphone vertical: Resolución de 360px

13.1 Lo-Fi

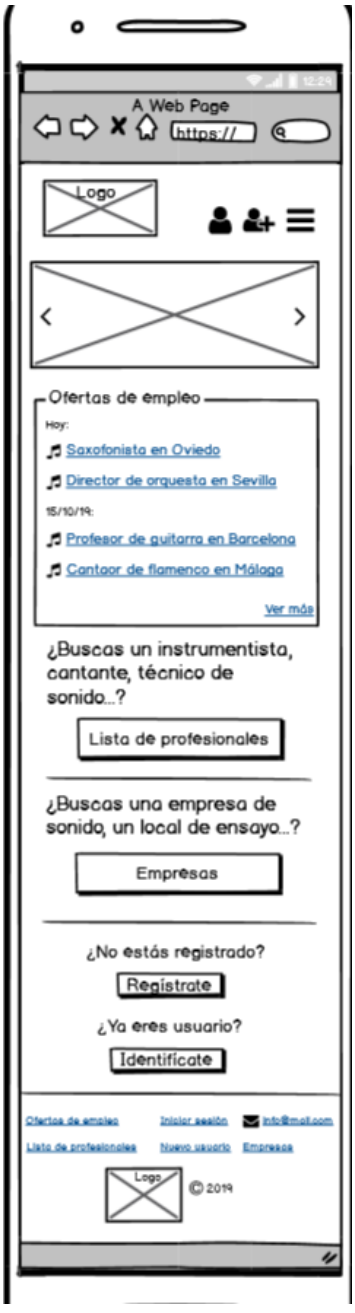

Ilustración 5 Wireframe Inicio sin login Smartphone

Ilustración 6. Wireframe Inicio con login Smartphone

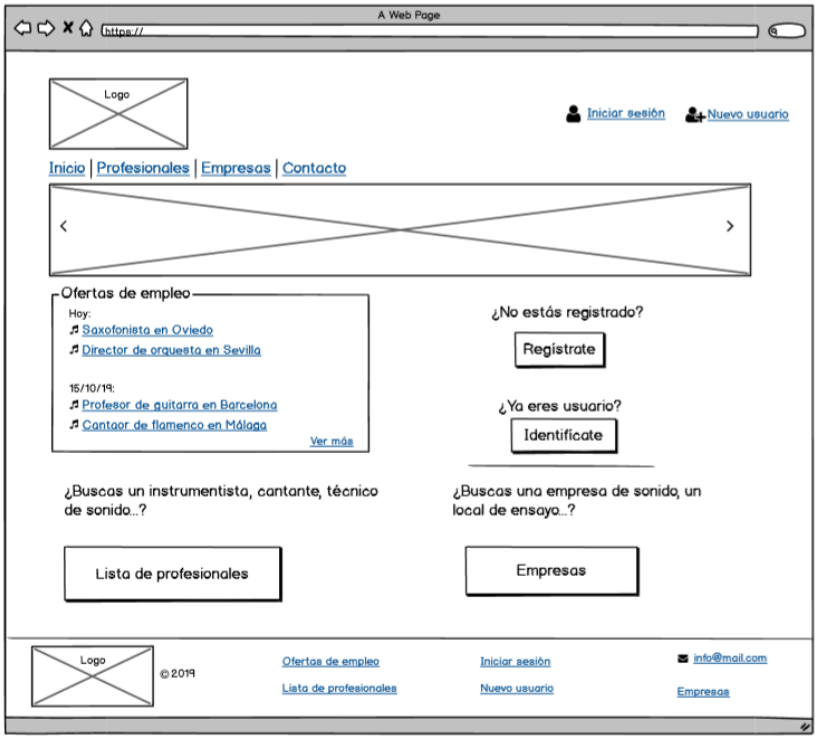

Ilustración 7. Wireframe Inicio sin loguin escritorio

Ilustración 8. Wireframe Buscar profesional Smartphone

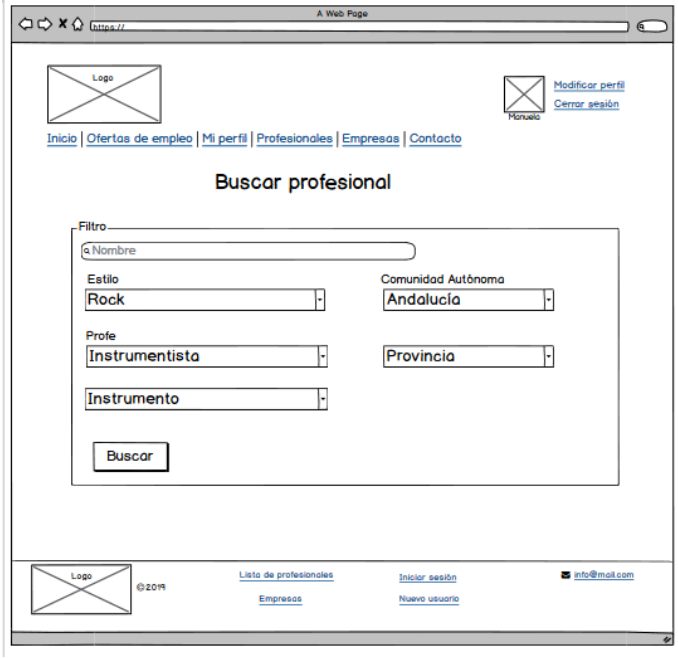

Ilustración 9: Wireframe Buscar profesional Escritorio

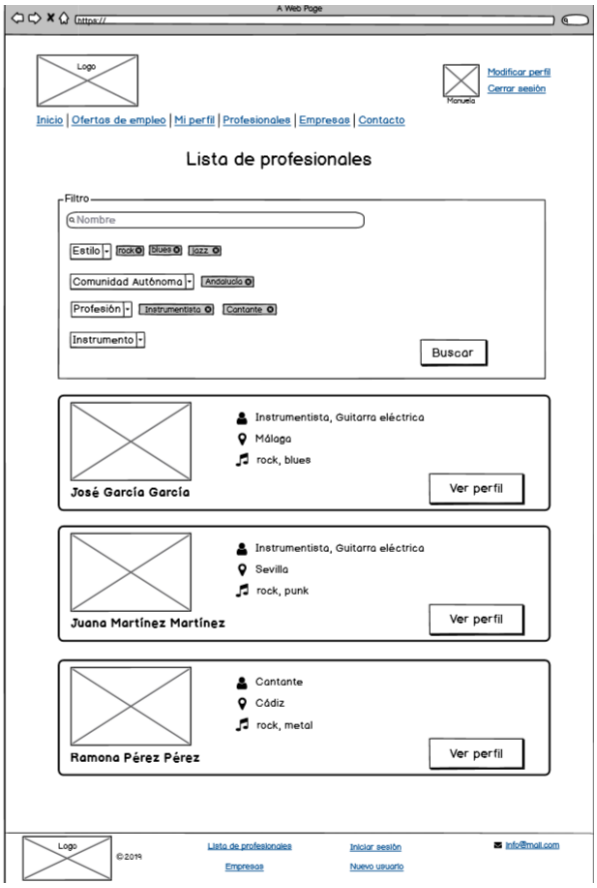

Ilustración 10. Wireframe Lista de profesionales escritorio

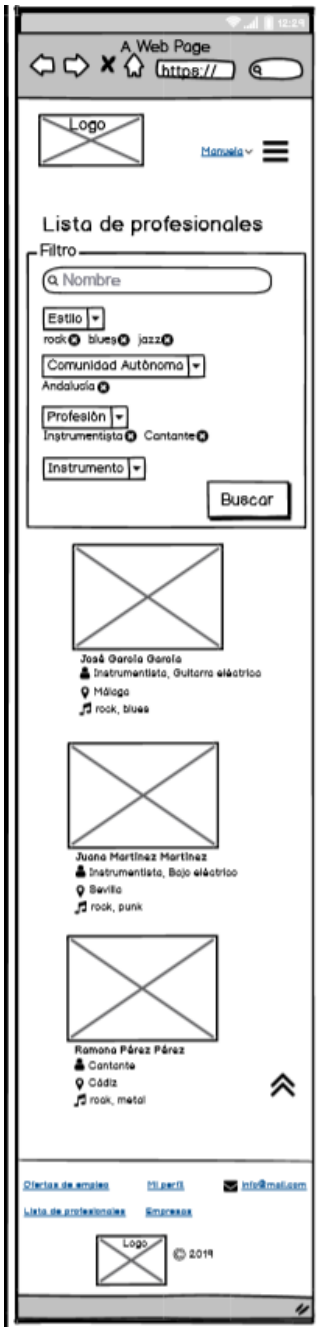

Ilustración 11. Wireframe Lista de profesionales Smartphone

Ilustración 12. Wireframe Ofertas de empleo Smartphone

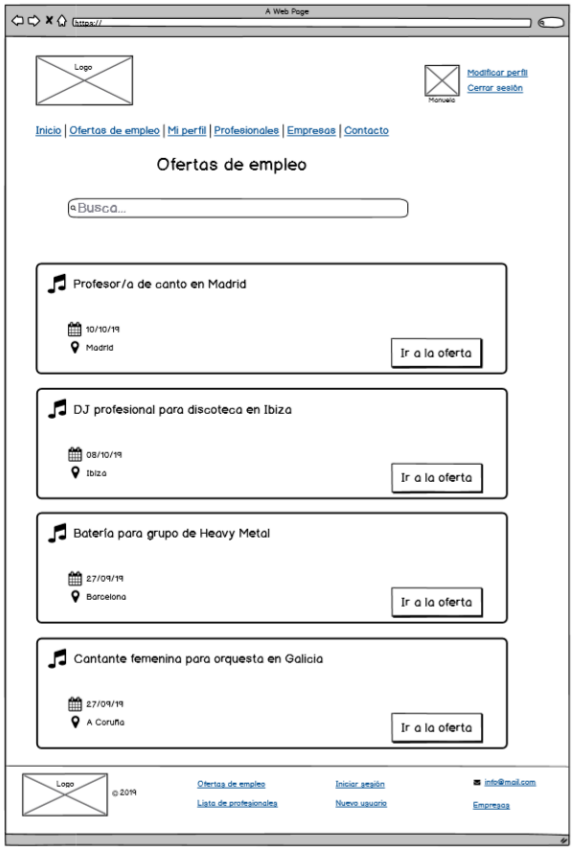

Ilustración 13. Wireframe Ofertas de empleo Escritorio

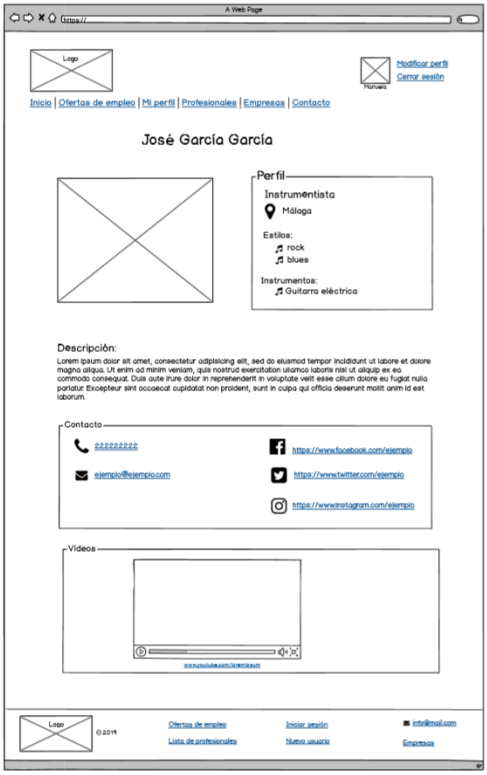

Ilustración 14. Wireframe Detalle del profesional Escritorio

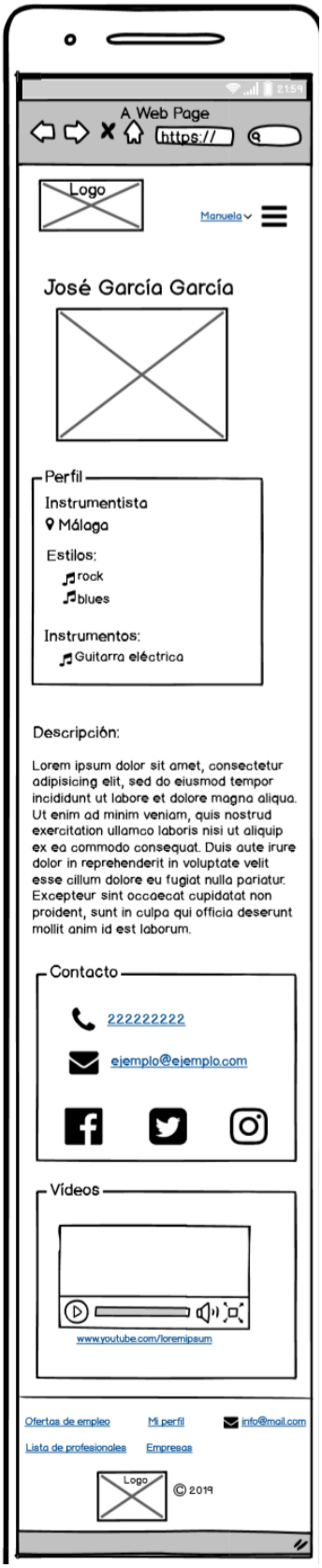

Ilustración 16. Wireframe Detalle del profesional Smartphone

Ilustración 15. Mi perfil Smartphone

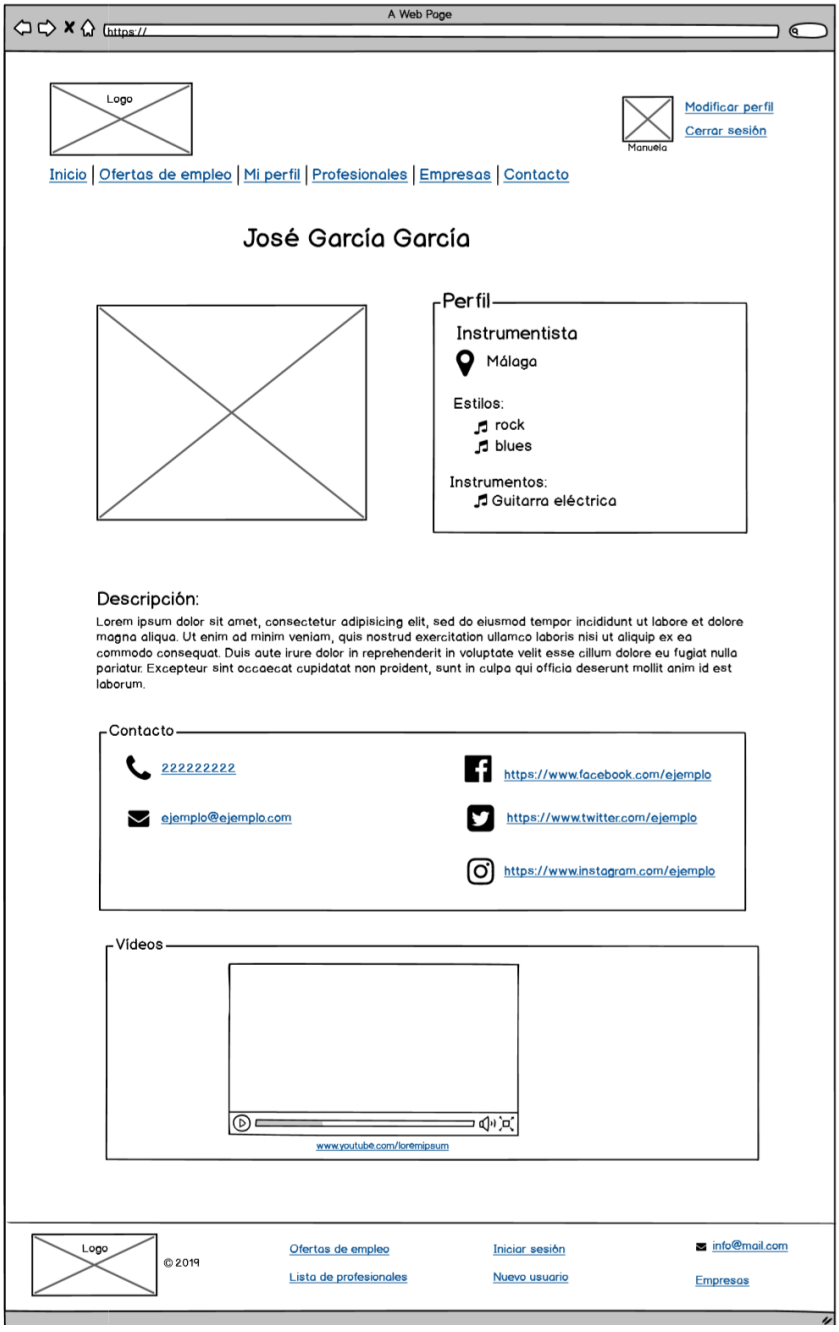

Ilustración 17. Wireframe Detalle del profesional Escritorio

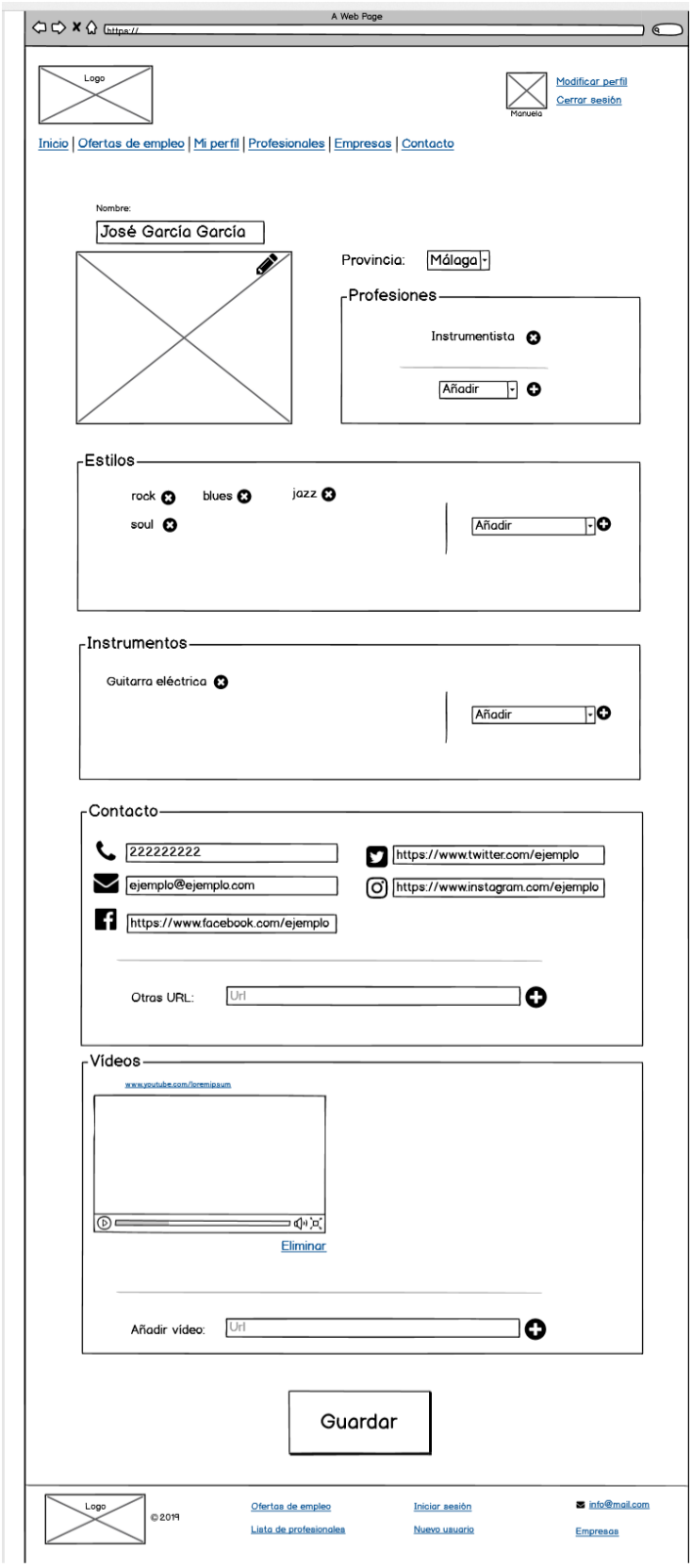

Ilustración 18. Wireframe Mi perfil Escritorio

3.2 Hi-Fi

- *Mockups*

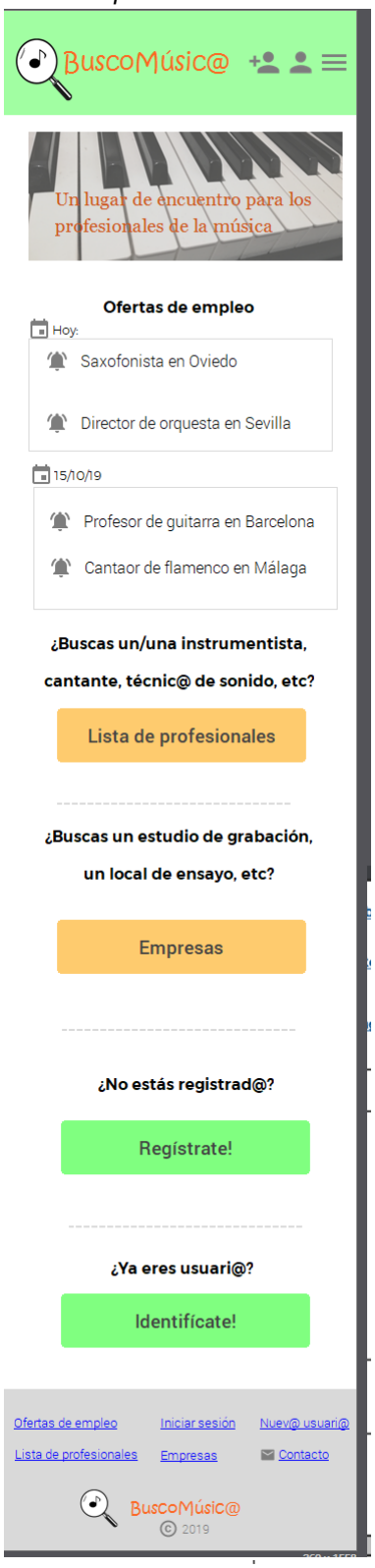

Ilustración 19. Mockup Inicio sin login Smartphone

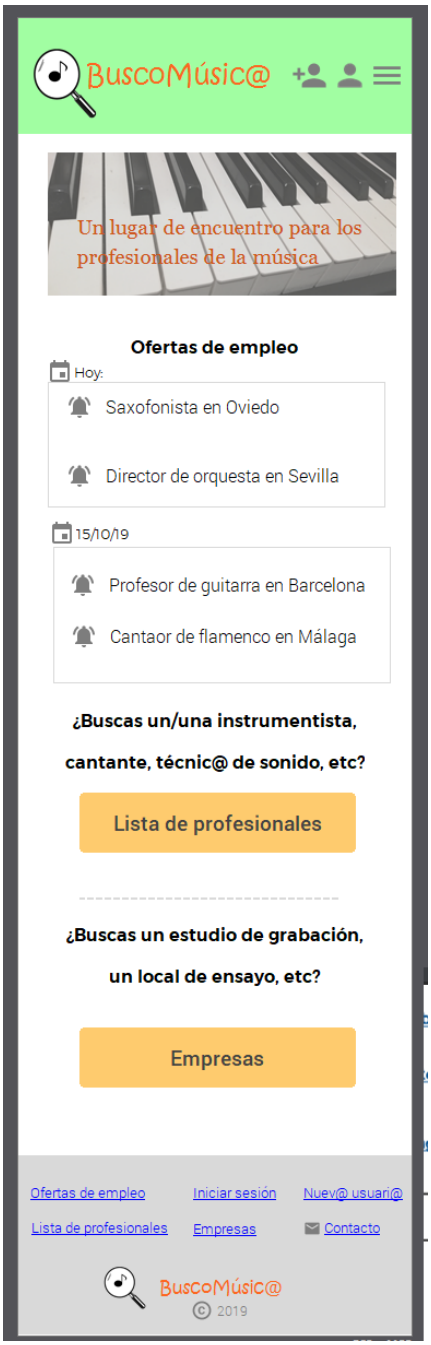

Ilustración 20. Mockup Inicio logueado Smartphone

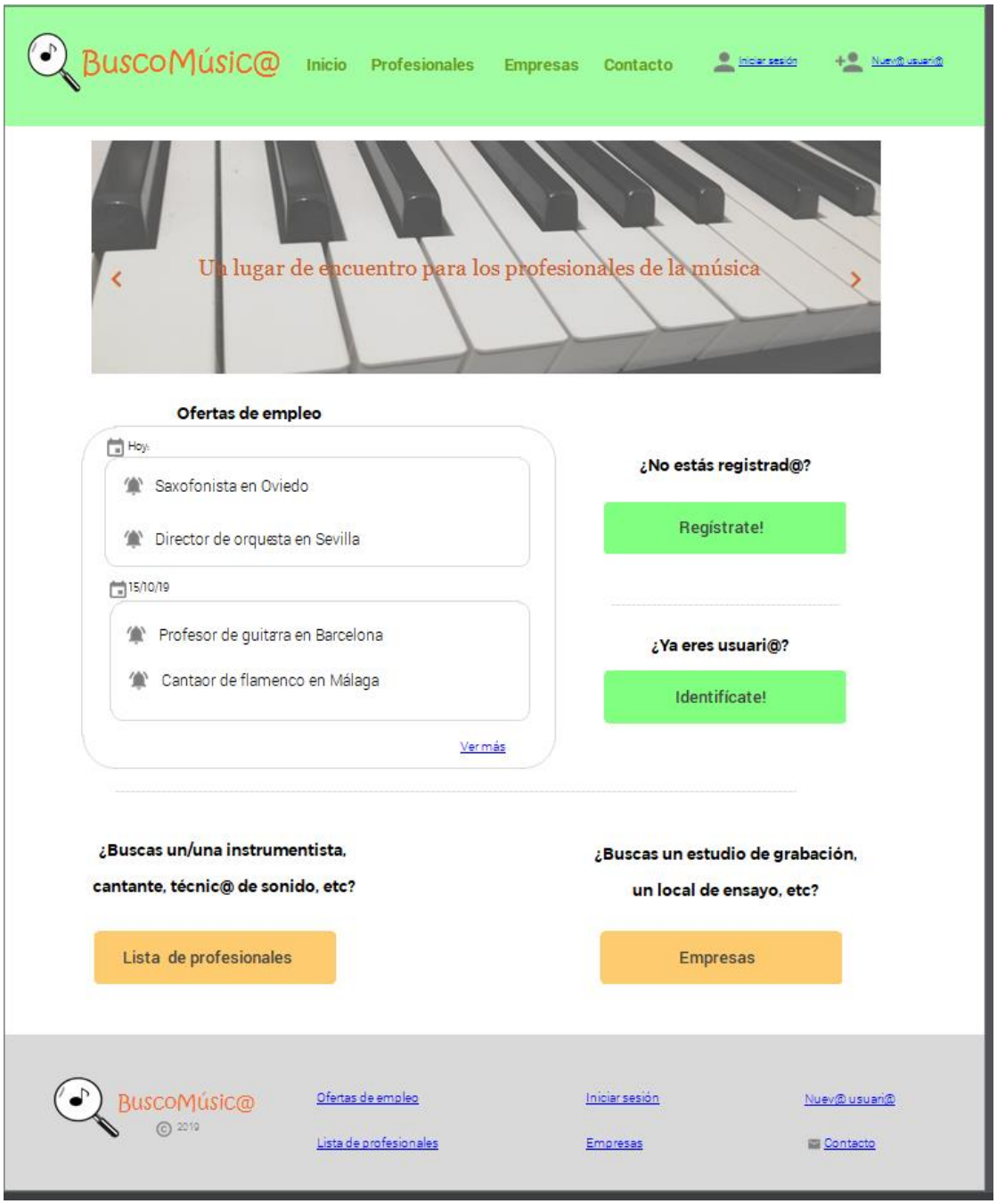

Ilustración 21. Mockup Inicio sin Loguin Escritorio

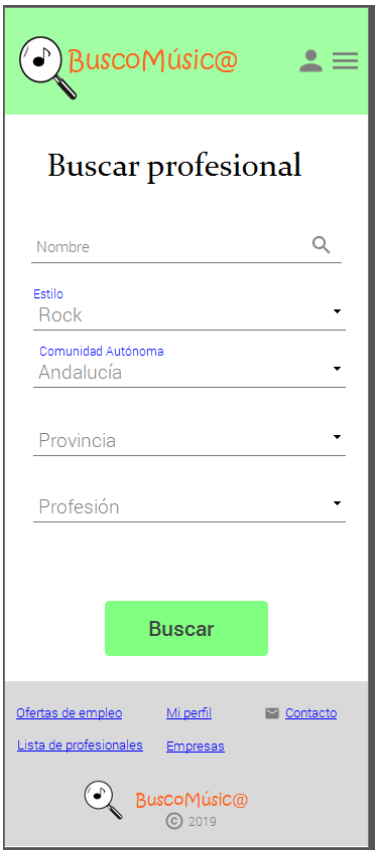

Ilustración 23. Mockup Buscar profesional Smartphone

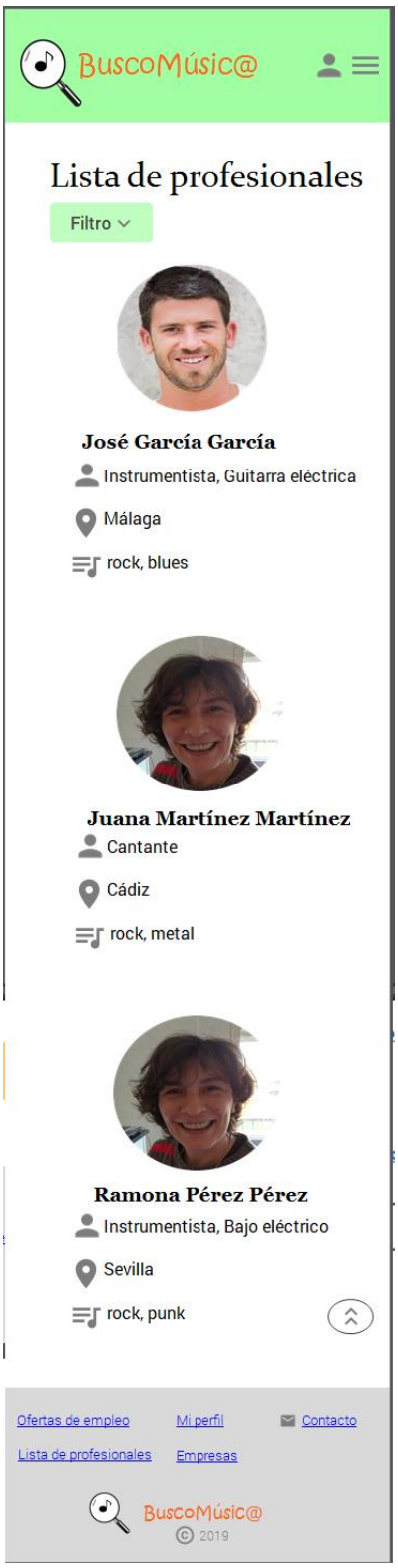

Ilustración 22. Mockup Detalle profesional Smartphone

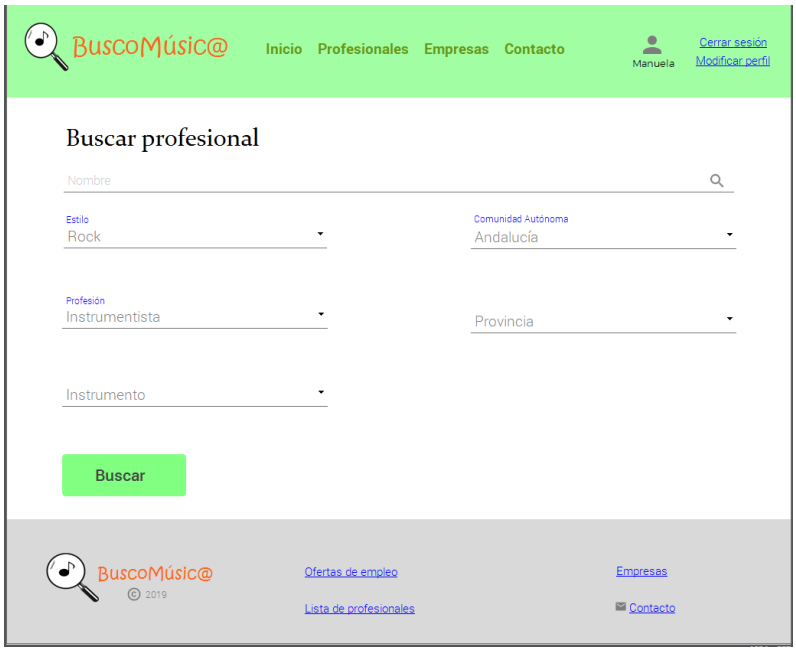

Ilustración 24. Mockup Buscar profesional Escritorio

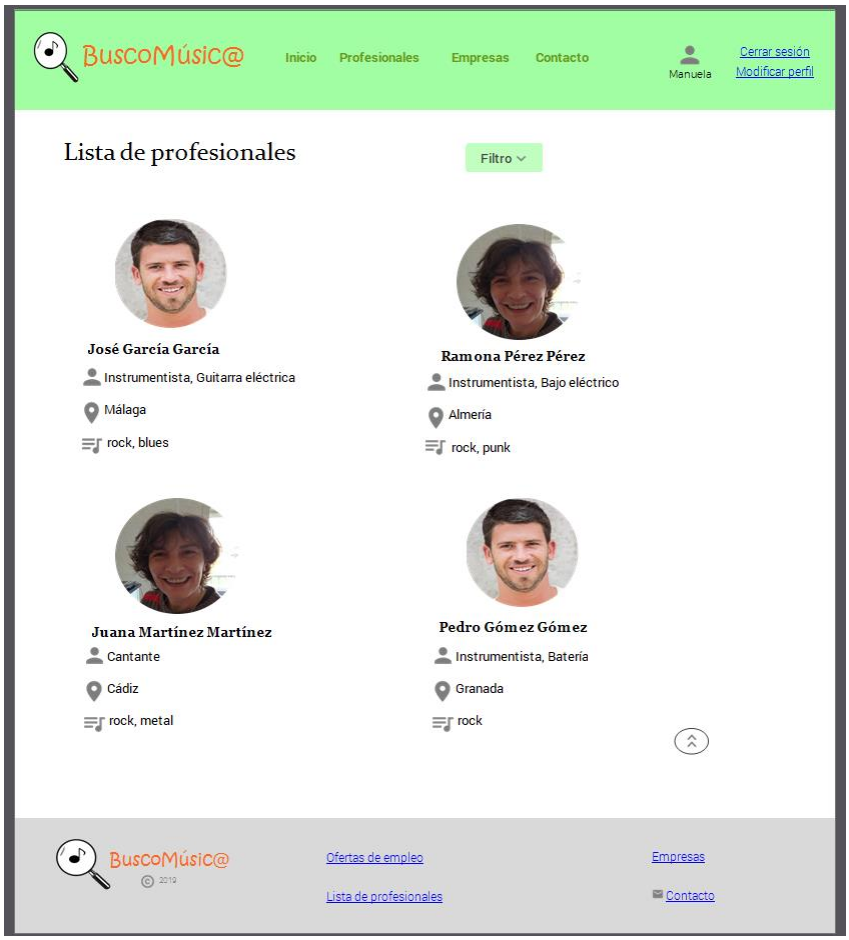

Ilustración 25. Mockup Lista de profesionales Escritorio

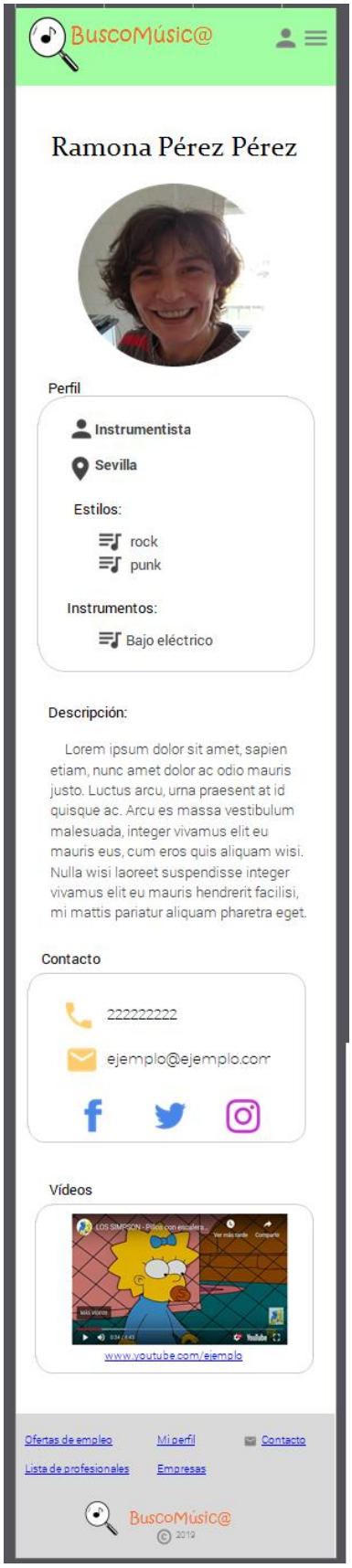

Ilustración 27. Mockup Detalles profesional Smartphone

Ilustración 26. Mockup Ofertas de empleo Smartphone

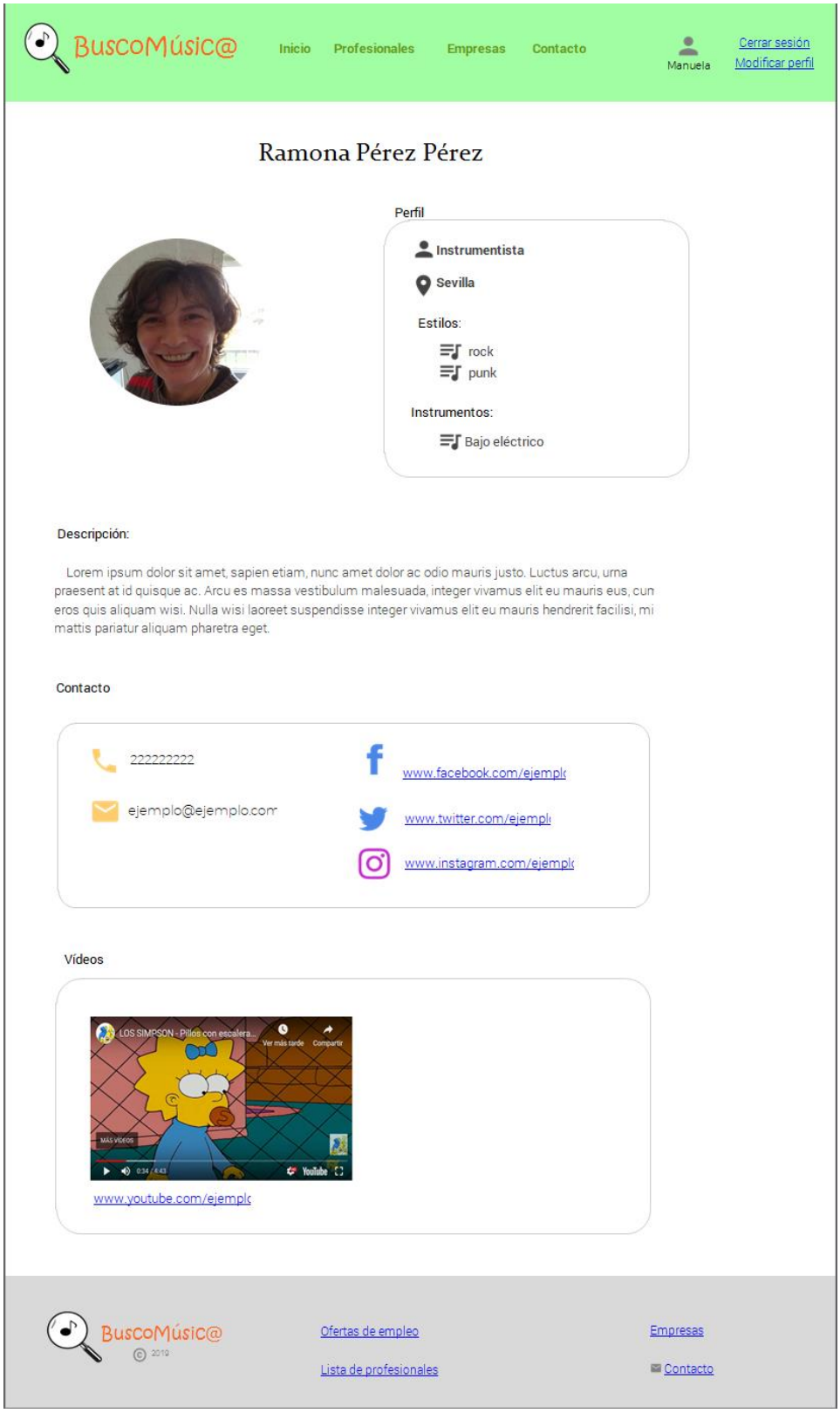

Ilustración 28. Mockup Detalle profesional Escritorio

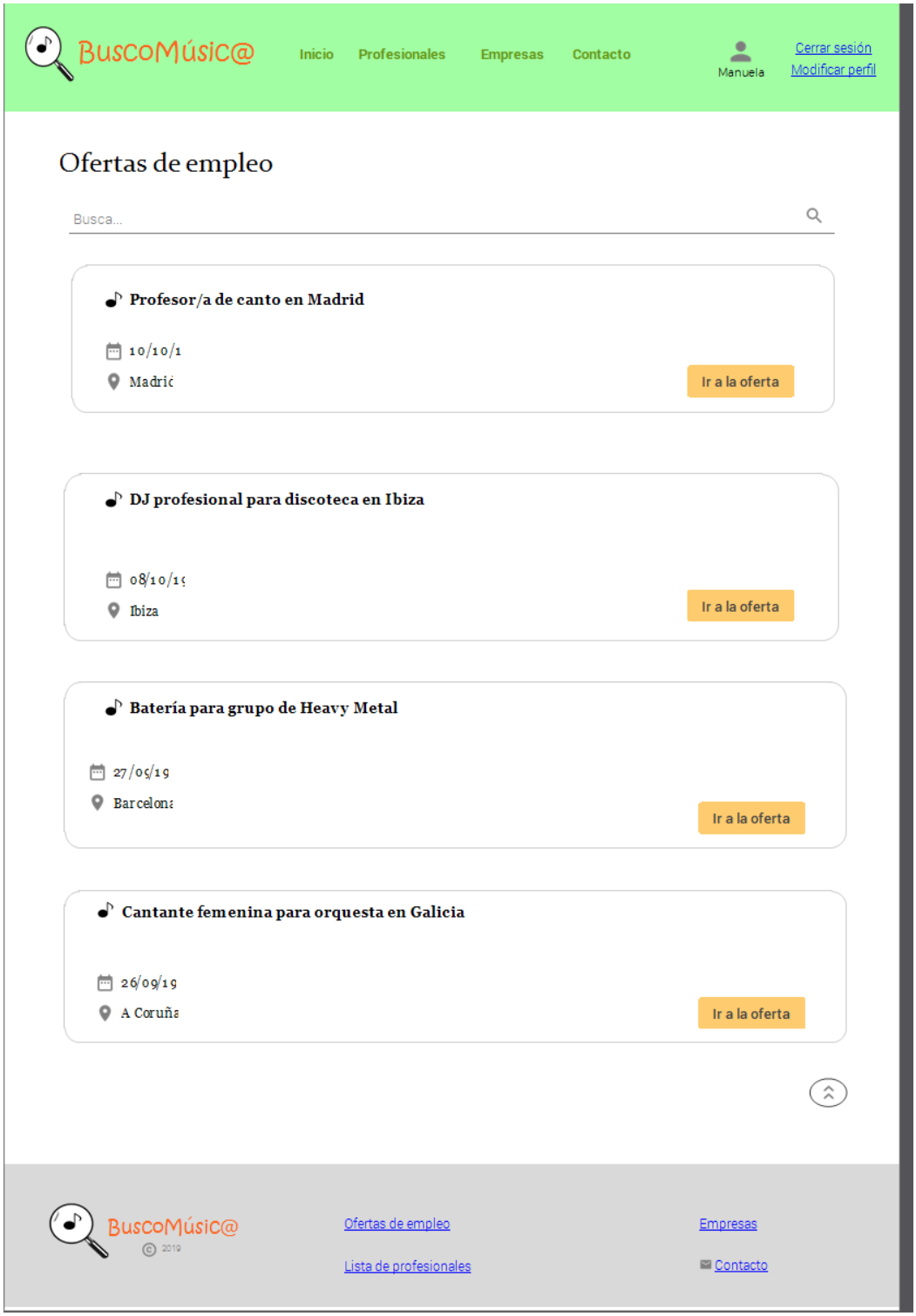

Ilustración 29. Mockup Ofertas de empleo Escritorio

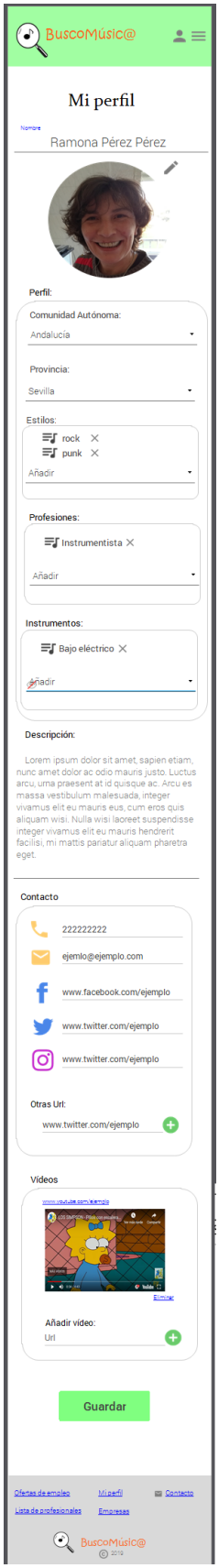

Ilustración 30. Mockup Mi perfil Smartphone

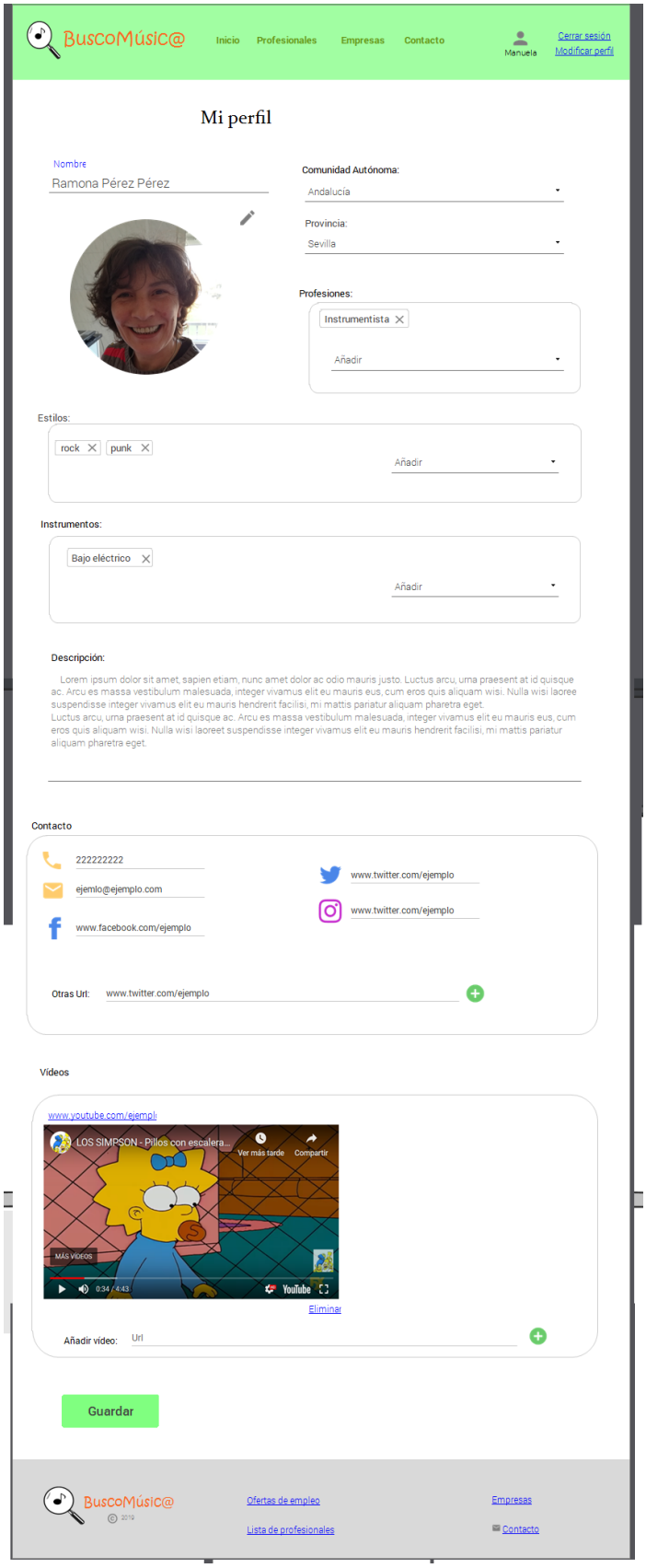

Ilustración 31. Mockup Mi perfil Escritorio

14. Perfiles de usuario

La aplicación está destinada a personas que se dedican a la música, principalmente instrumentistas y cantantes, pero también a técnicos, empresas, asociaciones y otras entidades que también están relacionadas con este sector, como por ejemplo salas de conciertos, estudios de grabación, productoras, bandas municipales, locales de ensayo, etc. Los usuarios secundarios de la aplicación serán los programadores y organizadores de eventos, que utilizarán la aplicación para buscar profesionales sin necesidad de estar registrados.

A continuación, se presenta un análisis sobre la actividad musical en España.

Según un informe publicado en *statista.com*, en España se acogieron, en total, más de 87.000 conciertos en 2017, siendo Madrid la comunidad autónoma en cabeza, seguido por Andalucía.

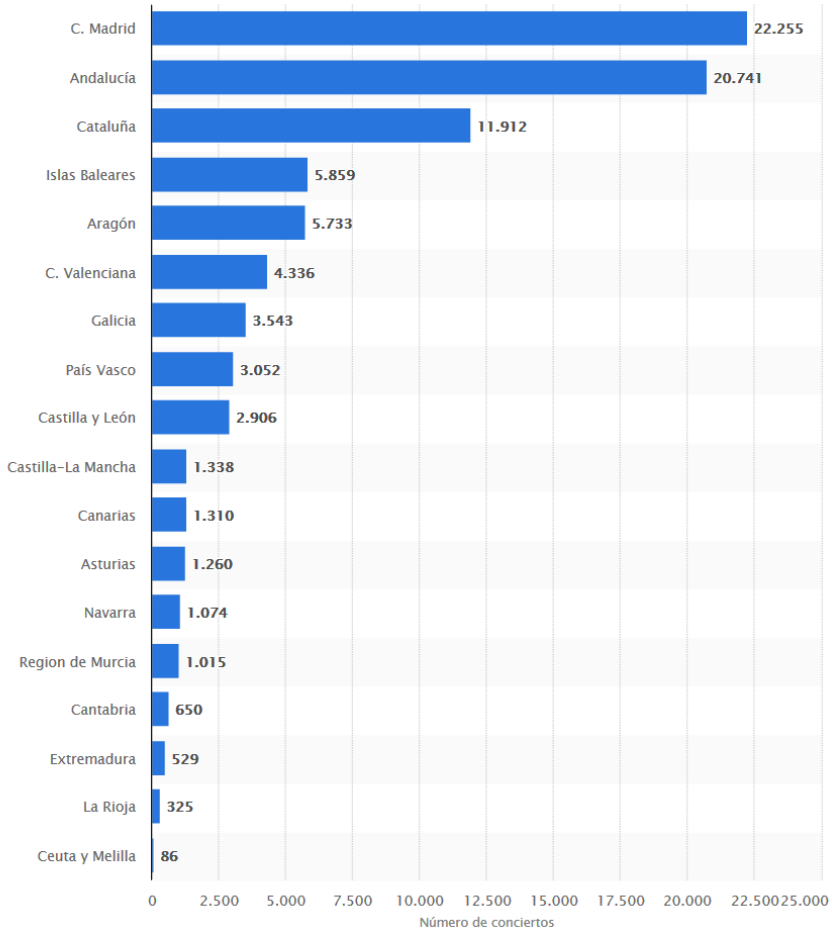

Ilustración 32. Número de conciertos de música popular en vivo en España en 2017, por comunidad autónoma [2]

[2] Fuente: https://es.statista.com/estadisticas/475086/numero-de-conciertos-de-musica-en-directo-espana-por-cc-aa/

El informe de la evolución anual de la facturación neta de la industria de la música en vivo en España muestra que se facturaron 333,9 millones de euros en música en vivo.

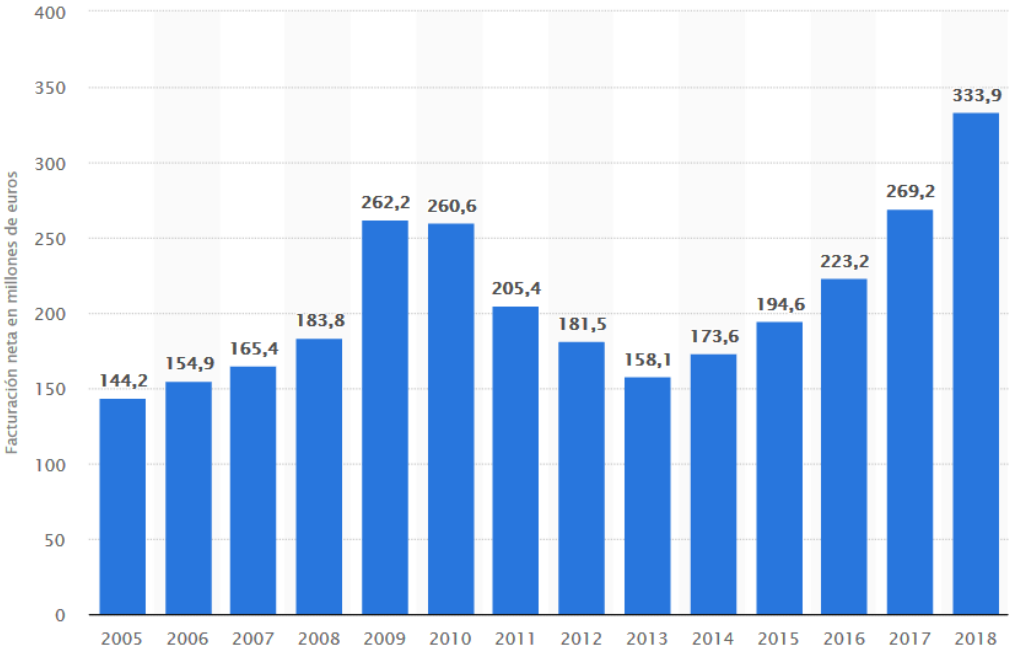

Ilustración 33: Evolución anual de la facturación neta de la industria de la música en vivo en España entre 2005 y 2018 (en millones de euros *[3]

Consultando las estadísticas de empleo por ramas de actividad en el INE, en el segundo trimestre de 2019 había 1.775.924 personas trabajando en actividades artísticas, recreativas y otros servicios, de un total de 20.437.437, lo que representa más del 8% de la población.

[3] Fuente: https://es.statista.com/estadisticas/472441/industria-de-la-musica-en-vivo-facturacion-en-espana/

* No incluye los datos de macrofestivales.

R-T Actividades artísticas, recreativas y otros servicios	
Dato base	1.775,924
Variación trimestral	1,9390
Variación anual	1,5950
Total Nacional	
Dato base	20.437,437
Variación trimestral	2,4518
Variación anual	2,1363

Ilustración 34. Empleo por ramas de actividad[4]

Según la Encuesta de Hábitos y Prácticas Culturales en España de la página web del Ministerio de Cultura y Deporte, el 4,9% de la población ha realizado actividades artísticas relacionadas con la danza en el 2015, el 7,8% ha tocado un instrumento musical, y el 2,4% ha cantado en un coro. Las mujeres se decantan más por la danza y cantar en coros, y los hombres por tocar instrumentos. Analizando las estadísticas por grupos de edades, las personas de 15 a 24 años realizan más actividades relacionadas con la música.

9.35. Personas que realizaron actividades artísticas en el último año según sexo por tipo de actividad

(En porcentaje de la población de cada colectivo)

	TOTAL			Hombres			Mujeres		
	2007	2011	2015	2007	2011	2015	2007	2011	2015
Escribir	7,5	7,1	7,8	6,8	6,1	7,0	8,1	8,1	8,6
Pintar o dibujar	9,2	13,2	13,7	7,5	11,2	11,6	10,8	15,1	15,6
Otras artes plásticas	4,5	7,7	8,3	3,0	5,4	5,7	6,0	9,9	10,8
Hacer fotografía	16,6	29,1	28,9	18,0	30,9	29,7	15,3	27,3	28,1
Hacer vídeo	5,7	12,8	15,0	6,6	14,5	16,0	4,9	11,1	14,0
Diseño de páginas web	2,2	2,6	3,0	2,8	3,9	4,1	1,7	1,4	2,0
Hacer teatro	2,1	2,1	2,2	1,7	1,7	1,6	2,4	2,4	2,8
Danza, ballet, baile	3,8	3,9	4,9	2,0	2,1	2,6	5,4	5,7	7,1
Flamenco, baile español	-	-	1,7	-	-	1,1	-	-	2,2
Tocar un instrumento musical	5,9	8,0	7,8	7,5	10,2	9,7	4,4	5,9	6,0
Cantar en un coro	2,8	2,4	2,4	2,3	1,8	1,8	3,2	2,9	3,0

Fuente: MCUD. Encuesta de Hábitos y Prácticas Culturales en España

Ilustración 35. Personas que realizaron actividades artísticas en el último año según sexo por tipo de actividad[5]

[4] Fuente: https://www.ine.es/jaxiT3/Tabla.htm?t=28607&L=0

[5] Fuente: https://www.culturaydeporte.gob.es/dam/jcr:eb5b8140-e039-42ab-8e24-500fddc5b2a4/anuario-de-estadisticas-culturales-2018.pdf

9.36. Personas que realizaron actividades artísticas en el último año según edad por tipo de actividad

(En porcentaje de la población de cada colectivo)

	TOTAL			De 15 a 24 años			De 25 a 54 años			De 55 y más años		
	2007	2011	2015	2007	2011	2015	2007	2011	2015	2007	2011	2015
Escribir	7,5	7,1	7,8	13,9	13,4	16,1	7,3	7,0	7,4	5,0	4,8	5,8
Pintar o dibujar	9,2	13,2	13,7	17,7	28,5	28,5	9,1	13,4	13,6	5,6	7,1	8,9
Otras artes plásticas	4,5	7,7	8,3	5,7	8,7	10,2	4,7	8,3	9,4	3,6	6,2	6,0
Hacer fotografía	16,6	29,1	28,9	22,2	43,0	43,0	20,8	35,4	34,5	7,1	13,2	15,6
Hacer vídeo	5,7	12,8	15,0	9,3	24,0	28,7	7,0	16,0	18,4	2,1	3,1	5,4
Diseño de páginas web	2,2	2,6	3,0	5,3	6,8	8,3	2,3	3,0	3,6	0,9	0,4	0,4
Hacer teatro	2,1	2,1	2,2	3,6	4,7	4,4	2,1	2,1	2,3	1,4	1,1	1,4
Danza, ballet, baile	3,8	3,9	4,9	7,1	8,5	9,9	3,6	3,9	4,9	2,6	2,3	3,3
Flamenco, baile español	-	-	1,7	-	-	2,6	-	-	1,6	-	-	1,5
Tocar un instrumento musical	5,9	8,0	7,8	12,2	17,2	18,2	6,3	8,6	8,3	2,6	3,4	3,7
Cantar en un coro	2,8	2,4	2,4	4,2	3,7	3,6	2,5	2,0	1,8	2,7	2,5	3,1

Fuente: MCUD. Encuesta de Hábitos y Prácticas Culturales en España

Ilustración 36. Personas que realizaron actividades artísticas en el último año según edad por tipo de actividad[6]

Como consecuencia de la Ley de Protección de datos, los datos de profesionales de música no están disponibles a partir de la fecha de 2005, por lo que a continuación se muestran los datos de 2004.

En 2004 había 3.471 instrumentistas y 2.402 docentes de música trabajando en España. También se puede ver que se contaba con 1.142 compositores, 770 cantantes, 202 críticos, 622 directores y 268 investigadores. Estos perfiles serán usuarios tipo de la aplicación.

	2004	2003	2002	2001	2000
Danza: bailarines	2.572	2.260	1.274	593	568
Danza: bailarines coreógrafos	340	250	249	77	70
Danza: bailarines docentes	168	114	151	94	92
Danza: bailarines docentes coreógrafos	103	98	93	56	56
Danza: coreógrafos	115	146	70	19	18
Danza: docentes	604	463	518	304	299
Danza: docentes coreógrafos	94	109	86	45	42
Música: compositores	1.142	1.107	1.026	980	889
Música: cantantes	770	746	736	734	442
Música: críticos	202	202	203	202	89
Música: directores (orquesta, banda, coro)	622	601	578	563	442
Música: docentes	2.402	2.340	2.133	1.976	1.504
Música: instrumentistas	3.471	3.444	3.305	3.144	1.750
Música: investigadores	268	265	244	243	219

Ilustración 37. Profesionales de la música y danza por tipo[7]

[6] Fuente: https://www.culturaydeporte.gob.es/dam/jcr:eb5b8140-e039-42ab-8e24-500fddc5b2a4/anuario-de-estadisticas-culturales-2018.pdf

[7] Fuente: http://estadisticas.mecd.gob.es/CulturaJaxiPx/Tabla.htm?path=/t18/p18/a2005//l0/&file=T1801005.px&type=pcaxis&L=0

Finalmente, se muestra el informe de la población que ha usado internet de manera frecuente en los últimos tres meses.

Tabla | Gráfico

	Hombres	Mujeres	Brecha de género
Total de 16 a 74 años	83,0	82,0	1,0
De 16 a 24	96,7	97,2	-0,5
De 25 a 34	95,5	96,6	-1,1
De 35 a 44	94,6	95,4	-0,8
De 45 a 54	85,1	88,9	-3,8
De 55 a 64	71,6	67,8	3,8
De 65 a 74	45,3	40,0	5,3

Ilustración 38. Población que ha usado Internet de manera frecuente en los últimos tres meses por grupos de edad y sexo. 2018.[8]

Considerando los datos, los usuarios tipo de la aplicación serán personas entre 18 y 45 años.

[8] Fuente: https://www.ine.es/jaxi/Datos.htm?path=/t00/mujeres_hombres/tablas_1/l0/&file=c05001.px

	**Cristina Crujeiras** Pianista, apasionada por la música clásica ● Acaba de terminar la carrera de piano ● Le gustaría conseguir un trabajo como pianista o teclista, pero no sabe cómo encontrarlo.
X.com quiere que Cristina:	● Encuentre trabajo como pianista. ● Recomiende la web a otros compañeros del conservatorio.

Perfil personal
Cristina acaba de finalizar sus estudios de piano en el conservatorio superior de música. Sigue viviendo con sus padres, y desde que terminó el instituto con 18 años, además de seguir estudiando en el conservatorio, trabaja de camarera en un bar de tapas de su barrio en Barcelona.
Aunque está cómoda con su trabajo, su pasión es la música, y le encantaría trabajar de lo que más le gusta: tocar el piano. Cristina siempre ha oído que era imposible vivir de la música, pero ella no pierde la esperanza y busca a diario ofertas de empleo en internet, esperando que algún día aparezca su primer trabajo como músico.

Objetivos y Motivaciones
Cristina necesita...
- Encontrar un trabajo de pianista o profesora de piano.
- Promocionar su carrera.

Cristina teme...
- No poder trabajar nunca como músico.
- Que la música clásica tenga poca salida laboral.

Cristina visita x.com para…
- Encontrar empleo como pianista o teclista.
- Conocer gente para crear proyectos musicales.

Cristina presta atención a…
- Aplicaciones, sitios web y redes sociales donde publiquen ofertas de empleo.

Scenario:
Cristina llega a su casa de trabajar a las 7 de la tarde. Mientras toma un descanso, busca ofertas de empleo para músicos en su teléfono móvil. Abre la aplicación, y ve varias ofertas para pianistas y profesores de música. Le interesa una de las ofertas, que es para profesora de clases extraescolares de música en Barcelona, por lo que pincha en el enlace para acceder a la oferta de trabajo.

Demografía:
Edad: 21 años (15/07/1998)
Estudios: Bachiller y Conservatorio superior de piano
Trabajo: Camarera
Sueldo: 18000 €/año
Estado civil: Soltera
Hobbies: Tocar el piano, asistir a conciertos, hacer deporte y participar en redes sociales.
Personalidad: Estudiosa, organizada.

Capacidades tecnológicas
Ordenador: lo utiliza básicamente para:
● Leer el correo
● Redes sociales, tanto para amistades como para buscar grupos musicales interesantes
● Buscar partituras, audios y vídeos para practicar con el piano
● Consultar páginas de ofertas de empleo
● Compras online

Smartphone:
● Leer el correo
● Redes sociales, tanto para amistades como para publicitar su página de perfil como pianista.
● Consultar páginas de ofertas de empleo
● Buscar conciertos

15. Usabilidad/UX

A continuación, se describe cómo se han aplicado distintos principios de usabilidad y técnicas para mejorar la experiencia de usuario.

- ***Scroll* top**. Cada vez que el usuario navega por la aplicación, automáticamente se desplazará hacia la parte superior en la siguiente 'página'. Para conseguir esta funcionalidad de una forma simple, se añade el evento 'onActivate()' en el fichero 'client/src/app/app.component.ts' dentro de la etiqueta '<router-outlet>' del fichero 'client/src/app/app.component.html':

  ```
  <router-outlet (activate)="onActivate($event)"></router-outlet>

  onActivate(event) {
      window.scroll(0,0);
    }
  ```

  También se añade un botón fijo para volver a la parte superior de la pantalla en los resultados de la búsqueda de profesionales.
- **Diálogos de confirmación**. Para evitar descuidos o clicks accidentales, en algunas funcionalidades se muestra un mensaje de confirmación al usuario. Por ejemplo, si elimina un vídeo en la página de 'Mi perfil', aparece un diálogo preguntando si está seguro de eliminarlo, y solamente se eliminará si pulsa 'Aceptar'. En la misma página de perfil, si el usuario realiza algún cambio en el formulario y después navega hacia otra página, también se muestra un diálogo para que confirme que quiere salir sin guardar los cambios.
- **Autocompletar**. Los combos que tienen un número importante de elementos, como los ayuntamientos, estilos musicales e instrumentos, cuentan con un campo de texto en el que se van mostrando los resultados que contienen el texto a medida que se va escribiendo. Esto ayuda a encontrar un elemento fácilmente. Se muestran antes los resultados cuyo nombre empieza por el texto buscado, y debajo de éstos se muestran los que contienen el texto, pero comienzan de otra forma. La funcionalidad de autocompletar se ha obtenido de la documentación de Angular Material[9]
- **Paginación**. En el componente de ofertas de empleo se muestran los resultados de forma paginada. En el 'paginador' se pueden elegir el número de elementos a mostrar. Este componente también se ha obtenido de Angular Material.[10]
- **Botones atrás y guardar/buscar** en la parte superior de la aplicación.
- **Navegación**: El usuario puede navegar a cualquier parte de la aplicación a través del menú de navegación situado en la parte superior, o a través de los enlaces del *footer*. Si el usuario ha navegado por la aplicación, se muestra en la parte superior un botón "Atrás" para poder prescindir de los botones del navegador o dispositivo para volver a la página anterior. Además, si se encuentra en la página de perfil, en la misma sección que el botón de retroceso, aparece un botón para guardar, con el objetivo de facilitar la experiencia del usuario y que no sea necesario desplazarse

[9] https://material.angular.io/components/autocomplete/overview
[10] https://material.angular.io/components/paginator/overview

hasta la parte inferior de la página para guardar los datos, ya que estos botones están siempre visibles. Por ejemplo, si el usuario solamente quiere cambiar la foto de perfil, que está en la parte superior de la página, no tendría que hacer *scroll* hasta el final para poder guardar los cambios. Si el usuario se encuentra en el buscador de profesionales, hay un botón para buscar profesionales en lugar del de guardar.

- **Guardado de datos en caché**. Se mejora la experiencia del usuario guardando datos que se obtienen de la API en caché con una fecha de caducidad temprana (entre 2 y 4 horas). Por ejemplo, al entrar en la página de inicio, si en la caché no hay datos o han "caducado", se hace la petición a la API para obtener los elementos de la base de datos. La próxima vez que el usuario requiera de esa información, como esos datos están disponibles en caché se cargarán mucho más rápido. Con esta funcionalidad, además de mejorar la experiencia de usuario, se reducen las peticiones a la API (puede suponer un ahorro económico según el plan de alojamiento) y las conexiones a la base de datos.
- **Visibilidad del estado del sistema**. En todos los botones que realizan peticiones a la API, por lo que tardan más tiempo en responder, se añade un *spinner* de la librería Bootstrap, desde que el usuario pulsa el botón hasta que se obtiene el resultado.
  También se añaden validaciones en los campos del formulario, de forma que cuando el usuario está escribiendo algo incorrecto se muestra un mensaje de error, informando del formato que se espera.
  Se añaden mensajes de aviso, tanto de error, como de éxito y alerta. Por ejemplo, si el usuario realiza el registro, se redirige a la página de acceso con un mensaje en la parte superior de la pantalla indicando que el registro se ha realizado correctamente y debe revisar su correo para confirmarlo, o que ha ocurrido un error.
- **Relación entre el sistema y el mundo real**. Se utilizan iconos para los distintos campos del perfil de un profesional, o de los detalles de una oferta. Por ejemplo, el icono de un teléfono en el campo teléfono. También se utiliza el icono de usuario y nuevo usuario en el menú de navegación.
- **Libertad y control por parte del usuario**. El usuario siempre puede deshacer o rehacer cualquier acción. Por ejemplo, puede modificar sus datos de perfil y cambiar la contraseña. También puede eliminar el usuario en la página de perfil. Para esto se añade una confirmación
- **Ayuda a los usuarios a** reconocer, diagnosticar y **recuperarse de los errores**. Se modifican los mensajes de error de la API para que el usuario lo entienda y pueda solucionarlo. Por ejemplo, si el usuario intenta iniciar sesión y no ha realizado la confirmación por correo electrónico, se muestra un aviso informando de que debe hacerlo, y si tiene algún problema puede pinchar en 'Olvidé mi contraseña'. También se muestra un diálogo si el usuario realiza alguna modificación en el formulario de su perfil y abandona la página sin guardar, preguntando si está seguro de descartar los cambios.

16. Seguridad

Se han desarrollado las siguientes funcionalidades para prevenir riesgos de seguridad en la aplicación:

- **Encriptación de contraseña**: Se utiliza la librería '`bcrypt`'[11] para enviar la contraseña del usuario a la base de datos encriptada. Esta librería no permite desencriptar una contraseña, sino que solamente se puede comparar el texto plano con el encriptado, ya que siempre genera un resultado diferente para el mismo texto. Se puede configurar la complejidad de la encriptación mediante el parámetro `saltOrRounds`, siendo más compleja la encriptación cuanto más alto es el valor del parámetro. En este caso se establece este valor a 12, siendo 10 el mínimo recomendado, y teniendo en cuenta que cuanto mayor sea el `saltOrRounds`, más tarda en encriptarse o en comparar un texto plano con otro encriptado.
- **JWT** (*JSON Web Token*): Se utiliza la librería '`jsonwebtoken`'[12] para gestionar la sesión de los usuarios. En este caso, se añade el *token* al usuario devuelto en la función de autentificación de la API, y desde el *frontend* se gestiona la caducidad del *token* (24 horas) mediante `localStorage`. En el *backend* se añade la comprobación del *token* en el *endpoint* `getOfertas`, que es añadido mediante un interceptor http.

[11] https://www.npmjs.com/package/bcrypt
[12] https://www.npmjs.com/package/jsonwebtoken

17. Requisitos de instalación/implantación/uso

A continuación, se enumeran los requisitos de implantación para el cliente y el servidor de la aplicación.

Cliente:

- Cuenta en *Firebase*

Servidor:

- Cuenta en *Heroku* con los siguientes *BuildPacks*:
    - https://buildpack-registry.s3.amazonaws.com/buildpacks/jontewks/puppeteer.tgz
    - heroku/nodejs
- Cuenta y repositorio en *GitHub* enlazado a la aplicación de *Heroku*.

Para el uso óptimo de la aplicación es necesario un dispositivo con un navegador actual como Google Chrome o Mozilla Firefox y con conexión a internet.

18. Instrucciones de instalación/implantación

A continuación, se detallan las instrucciones de implantación de la aplicación web.

18.1 Producción

Servidor

En este caso, para publicar la API en Express se utiliza la Plataforma como Servicio (PaaS) Heroku. Para ello se crea una cuenta y se compra un contenedor (dyno). También se puede crear una aplicación gratuita. Hay que tener el código del servidor en un repositorio de GitHub y asociar este repositorio a la aplicación de Heroku. Para publicar la aplicación se siguen los pasos que se indican en la pestaña 'Deploy' de la página de la aplicación en Heroku. En este caso, la aplicación se llama 'buscomusicoserver' y, una vez añadido el repositorio de Git en la carpeta raíz del código del servidor (server), se ejecutan los siguientes comandos:

```
heroku login
heroku git:clone -a buscomusicoserver
cd buscomusicoserver
git add .
git commit -am "make it better"
git push heroku master
```

Al estar desarrollado en TypeScript, para poder ejecutar la API, se configuran los scripts del fichero 'package.json' de la siguiente forma:

```
"scripts": {
    "build-ts": "tsc",
    "preinstall": "npm install pm2 -g",
    "postinstall": "npm run build-ts",
    "start": "pm2-runtime start build/index.js --env production",
    "serve": "node build/index.js",
    "watch": "tsc -w",
    "dev": "nodemon ./build"
  },
```

Se añade pm2 al script 'start' para obligar a reiniciar el *backend* en caso de un error bloqueante.
En el fichero 'server/src/keys.ts' se configuran las variables relativas a la base de datos, ftp, correo electrónico y la URL del *frontend*.

Cliente

Antes de nada, hay que configurar la variable API_ENDPOINT_PROD en el fichero client/src/app/app.settings.ts con la url de la aplicación que se ha creado en Heroku para el *backend*. En este caso:

```
public static readonly API_ENDPOINT_PROD =
'https://buscomusicoserver.herokuapp.com/api';
```

El contenido de la carpeta 'client/dist/' se ha generado a través de Angular-Cli con el comando '`npm run build:ssr`'. Para publicar el *frontend* hay que desplegar el contenido de la carpeta 'client/dist/browser' en Firebase, siguiendo los siguientes pasos:

- Asociar una cuenta de Google en Firebase (https://firebase.google.com), añadir un proyecto y seguir los pasos indicados.
- En la carpeta client, ejecutar el comando `firebase login` e introducir las credenciales.
- Ejecutar comando `firebase init` y seguir las instrucciones añadiendo Hosting.
- Copiar contenido de la carpeta 'client/dist/browser' a la carpeta public (la que se ha seleccionado en el comando anterior).
- Ejecutar `firebase deploy`.

Base de datos

Se debe importar el archivo 'musicjobs.sql' en una base de datos MySql de un servidor. Los datos de configuración se deben de introducir en 'server/src/keys.ts', en la variable 'database'.

Cuenta de correo

Es necesaria una cuenta de correo para poder enviar los emails de confirmación y reseteo de contraseña a los usuarios. Para este proyecto se ha contratado un servicio de Hosting que incluye cuentas de correo. La configuración de la cuenta (dirección, contraseña, servidor y nombre de usuario) se incluye en el archivo 'server/src/keys.ts', en la variable 'mail'.

18.2 Desarrollo

Servidor

Para ejecutar la API en un servidor local hay que tener instalado Node.js y npm. En la carpeta 'server', se ejecuta '`npm install`', seguidamente '`npm run build-ts`' y '`npm run dev`' para servir el *backend* en el puerto 8081 de localhost. Para que funcione el *web scraping* es necesario tener instalado Google Chrome.

Cliente

Para poder ejecutar el *frontend* en desarrollo, hay que tener instalado Node.js, npm y Angular-Cli.
Para compilar y abrir la aplicación en un navegador hay que modificar el archivo 'client\src\app\app.settings.ts', y reemplazar la variable 'API_ENDPOINT' por la ruta en localhost, que está definida en la variable 'API_ENDPOINT_DEV', quedando de la siguiente forma:

```
public static readonly API_ENDPOINT_DEV = 'http://localhost:8081/api';
public static readonly API_ENDPOINT_PROD =
'https://buscomusicoserver.herokuapp.com/api';

public static readonly API_ENDPOINT = AppSettings.API_ENDPOINT_DEV;
```

Después de modificar el fichero ‘app.settings.ts’, hay que abrir una consola en la carpeta ‘client’, y ejecutar el comando `npm install`, y después `ng serve --open`.

19. Instrucciones de uso

La aplicación se puede probar en https://musicjobs.es.

Se puede acceder a los componentes de la aplicación a través del menú de navegación, de los enlaces del *footer*, o desde los botones de la página de inicio. En dispositivos pequeños y medianos, el menú de navegación está disponible al pulsar el botón representativo de menú con tres rayas horizontales.
La página que se muestra por defecto es la de inicio. Desde esta página se pueden ver las 3 últimas ofertas de empleo y los 3 últimos profesionales registrados. Si el usuario está autentificado, podrá acceder al detalle de cada oferta, así como a la lista completa de ofertas. Si no está autentificado, al pulsar sobre el detalle de una oferta o sobre el botón “Más ofertas”, se redirigirá a la página de inicio de sesión con un aviso de que para poder acceder a esa funcionalidad debe estar registrado.

Se puede acceder al detalle de un profesional a través de la pantalla de inicio si es uno de los tres últimos que se han registrado, y desde el buscador de profesionales. En el buscador se puede filtrar por provincia, ayuntamiento, nombre, estilo musical, instrumento y especialidad, seleccionando uno o más campos para la búsqueda. Al pulsar ‘Buscar’ se mostrarán los resultados, ocultándose el formulario de búsqueda, y mostrando el botón ‘Filtro’ para volver a mostrarlo. Si no hay resultados, se muestra un mensaje informando de que no se han encontrado profesionales con esas características. Se pueden limpiar los campos de búsqueda pinchando el botón ‘Limpiar búsqueda’.

Pulsando el enlace ‘Ofertas de empleo’ del menú de navegación, del *footer*, o el botón ‘Más ofertas’ de la pantalla de inicio, se mostrará un listado de todas las ofertas de empleo. En la parte superior hay un campo de búsqueda que filtra por título y descripción a medida que se va escribiendo. Al pulsar en el botón ‘Detalles’ de una oferta, se accede al detalle de la oferta, mostrando la descripción completa y otros campos.

Se puede modificar el perfil de un usuario si está autentificado, pulsando en el icono que aparece en la parte derecha del menú de navegación con el avatar y el nombre del usuario. En esta pantalla se puede modificar la foto de perfil, datos personales, añadir y eliminar estilos musicales, instrumentos, especialidades, vídeos y enlaces a redes sociales. Se guardarán los datos al pulsar ‘Guardar’.

Al pulsar sobre el enlace ‘Contacto’ del menú de navegación o del *footer*, se accede a un formulario donde el usuario puede introducir su correo electrónico y un comentario, duda o sugerencia a las personas administradoras de la aplicación.

20. Bugs

A continuación, se detallan los errores detectados en la funcionalidad de la aplicación.

- Al buscar profesionales con muchos filtros tarda demasiado la búsqueda y se reproduce un error de *timeout*, ya que se realizan varias consultas en la base de datos. Para solventar este error, en la próxima versión se valorará optimizar las consultas e incrementar el valor del timeout, o limitar los filtros en el formulario, mostrando un aviso si se rellenan más campos de los permitidos.
- En algunos navegadores se ha observado que no se visualizan correctamente todas las pantallas. Se ha probado en un IPhone con Safari y con Chrome y se muestran los elementos descolocados en los dos navegadores, pero de forma diferente.
- En algunos dispositivos, al sacar la foto de perfil con la cámara, ésta se muestra con una rotación de 90 grados. Para dar una solución, al hacer click sobre la foto, se rotará 90 grados.
- Al actualizar la foto de perfil no se muestra la foto en el menú de navegación. Hay que actualizar la página después de guardar para que se muestre.

21. Proyección a futuro

A continuación, se detallan posibles mejoras en futuras versiones de la aplicación:

- Optimización de la imagen de perfil del usuario para mejorar el rendimiento.
- Añadir filtros en las entradas de los formularios para evitar faltas de respeto y palabras obscenas.
- Añadir más webs donde se publiquen ofertas de empleo en la función de *scraping*, y también más filtros a la hora de guardar las ofertas en la base de datos.
- Proporcionar mejoras en el servicio de *web scraping*, como contemplar si falla alguna web para no eliminar los registros de la base de datos, e incluir un sistema de log de errores.
- Alarmas personalizadas de ofertas de empleo con unas características concretas, como la provincia o el tipo de oferta. Estas notificaciones saldrían en el dispositivo si se añade la aplicación a la pantalla de inicio, y también se enviarían por correo electrónico.
- Envío de mensajes entre los usuarios de la aplicación.
- Posibilidad de añadir una oferta a una lista de favoritos, de forma que no se elimine de la base de datos hasta que ningún usuario la tenga en favoritos.
- Posibilidad de “seguir” un profesional para facilitar el acceso a su perfil o enviarle un mensaje.
- Ofrecer la aplicación en distintos idiomas, seleccionando gallego, español, catalán, euskera o inglés, así como añadir todas las provincias y ayuntamientos españoles.
- Mejorar el diseño, ofreciendo estilos más agradables.
- Diálogos personalizados para algún aviso importante y para confirmar acciones.

22. Presupuesto

A continuación, se muestra el presupuesto total del proyecto.

- Equipo humano:

Fase	Horas	Días	Coste/hora (€)	Coste (€)
Análisis e investigación	44	11	20	**880**
Diseño e inicio del desarrollo	126	21	20	**2.520**
Desarrollo de la primera versión de la aplicación	156	26	20	**3.120**
Finalización del proyecto	114	18	20	**2.280**
Total	**440**	**76**	**20**	**8.800**

- Equipamiento técnico:

Recurso	Unidades	Coste/unidad (€)	Coste anual (€)
Heroku Dyno	12	7	**84**
Dominio .es	1		**12**
Total			**96**

Entre el equipo humano y técnico, el coste total de la aplicación sería de **8.896€**.

23. Marketing y Ventas

En la próxima versión de la aplicación se incluirían dos planes de suscripción en el registro:

- Plan gratuito: Incluye publicidad.
- Plan premium: No incluye publicidad y tendría un coste de 5€ anuales.

Se promocionaría la web en redes sociales.

24. Conclusiones

A continuación, se enumeran las conclusiones personales acerca del proyecto realizado, el proceso de trabajo y los resultados obtenidos.

- *MusicJobs* facilita la búsqueda de empleo en el sector musical, con lo que se ha cumplido el principal objetivo propuesto para el trabajo.
- Se ha comprobado que el hecho de ser una SPA ofrece una mejor experiencia de usuario al ser muy ágil en la navegación.
- Utilizar el mismo lenguaje para el frontend y el backend ha facilitado y agilizado el desarrollo.
- He aprendido a desarrollar y desplegar una aplicación web en todas sus fases, poniendo en práctica los conocimientos adquiridos durante el Máster y otros que he adquirido por cuenta propia, como el *web scraping* o la creación de una API en Express.
- El resultado del producto final ha sido satisfactorio, aunque no se han podido completar todas las funcionalidades de la idea inicial.
- La aplicación se podría adaptar fácilmente a otra rama profesional, como las artes escénicas, haciendo pequeñas modificaciones.

Anexo 1. Entregables del proyecto

- **proyecto /client**: Directorio del código del *frontend* en Angular
- **proyecto /server**: Directorio del *backend* en Node.js (API y *web scraping*)
- **proyecto /musicjobs.sql**: Fichero de importación de la base de datos
- **proyecto /client/dist/browser**: Directorio de publicación del *frontend*
- **proyecto/server/build:** Directorio de publicación del backend
- **documentación/PAC_FINAL_mem_PazFernandez_Sara.pdf**: Memoria del proyecto
- **documentación/PAC_FINAL_prs_PazFernandez_Sara.pdf**: Presentación escrita-visual
- **documentación/PAC_FINAL_vid_PazFernandez_Sara.pdf**: Vídeo de presentación publicado en el espacio *Present@*

Anexo 2. Código fuente (extractos)

A continuación, se añaden y describen partes relevantes del código de la aplicación.

En primer lugar, se describe la funcionalidad de registro del *backend*:

- server/src/controllers/usersController.ts:

```
public async register(req: Request, res: Response): Promise<void> {
        const { email, password, nombre } = req.body;
        if (email != null && email != undefined && password != null && password != 
undefined) {
            let url = req.protocol + '://' + req.headers.host;
            const users = await pool.query('SELECT * FROM user WHERE email = ?', [r
eq.body.email]);

            let id_user = 0;
            const token = Math.random().toString(36).substring(2, 15) + Math.random
().toString(36).substring(2, 15);
            if (users.length > 0) {
                res.status(500).json({ message: 'Usuario xa rexistrado' });
            }
            else {
                const saltRounds = 10;

                const passEnc = await new Promise((resolve, reject) => {
                    bcrypt.hash(password, saltRounds, function (err, hash) {
                        if (err) reject(err);
                        resolve(hash);
                    });
                });
                const d = new Date();

                let mes = '' + (d.getMonth() + 1);
                let dia = '' + d.getDate();
                const ano = d.getFullYear();
                let hora = '' + d.getHours();
                let minuto = '' + d.getMinutes();
                let segundo = '' + d.getSeconds();
                if (mes.length < 2)
                    mes = '0' + mes;
                if (dia.length < 2)
                    dia = '0' + dia;
                if (hora.length < 2)
                    hora = '0' + hora;
                if (minuto.length < 2)
```

```
                minuto = '0' + minuto;
            if (segundo.length < 2)
                segundo = '0' + segundo;

            const dataString = [ano, mes, dia].join('-
') + ' ' + [hora, minuto, segundo].join(':');
            const result = await pool
                .query('INSERT INTO user set email=?, password=?, nombre=?,  to
ken=?, fecha_registro=?',
                    [email, passEnc, nombre, token, dataString]);

            id_user = result.insertId;
            const from = mailOptions.username;

            console.log('enviando mail');

            let cuenta = { user: mailOptions.userMail, pass: mailOptions.passwo
rd }

            let transporter = nodemailer.createTransport({
                host: mailOptions.service,
                service: mailOptions.service,
                port: 587,
                secure: false,
                auth: {
                    user: cuenta.user,
                    pass: cuenta.pass
                },
                tls: {
                    rejectUnauthorized: false
                }
            });

            const link = url + "/api/confirmaUsuario/" + id_user + "/" + token;
            let info = await transporter.sendMail({
                from: from,
                to: email,
                subject: 'Benvido@, ' + nombre + '! Confirma a túa conta en Mus
icJobs',
                html: "<html><body style=\"font-
size:large;color:#383838; \">¡Hola!<br /><br />Pulsa <a href='" + link + "'>aquí</a
> para verificar a conta.<br /><br /></body></html>"

            });
            res.status(200).json({ message: 'Usuario gardado' });
            console.log('mail enviado');
        }
```

```
        }
        else {
            res.status(500).json({ message: 'Sin datos' });

        }
    }

    public async confirmaUsuario(req: Request, res: Response): Promise<void> {
        const { id, token } = req.params;
        const users = await pool.query('SELECT * FROM user WHERE id = ?', [id]);
        if (users && users.length) {
            let url = frontendOptions.url;
            const user = users[0];
            if (user.activo) {
                res.redirect(url);

            }
            else if (user.token && token != undefined && token != '' && user.token
=== token) {
                await pool.query('UPDATE user set activo=1, token=null WHERE id = ?
', [id]);
                res.redirect(url);


            }
            else {
                res.statusMessage = "Error, token non válido";
                res.status(401);
                res.json({ message: res.statusMessage });

            }
        }
        else {
            res.statusMessage = "Error. O usuario non existe";
            res.status(404);
            res.json({ message: res.statusMessage });
        }
    }
```

La primera función (`register(…)`), empieza definiendo las variables que vienen en el cuerpo de la petición. Si el correo o la contraseña están vacíos se devuelve error. Se comprueba que no exista el correo electrónico, y si existe también se devuelve error. El siguiente paso es encriptar la contraseña, y para ello se utiliza la librería 'bcrypt' con un *saltRounds* de 10, y se asigna el resultado encriptado a la variable 'passEnc'. Además de encriptar la contraseña, se calcula un *token* aleatorio para asignarlo al usuario en la

base de datos con la finalidad de confirmar la cuenta en el correo electrónico que se enviará. También se guarda la fecha de registro del usuario, por lo que es necesario guardar la fecha actual en un formato fecha de MySql ('yyyy-MM-dd hh:mm:ss').
Una vez se asigna la fecha y la contraseña encriptada a unas variables, se inserta el nuevo usuario en la base de datos. Por defecto, el campo 'activo' de la tabla 'user' se establece a 0, por lo que no podrá registrarse hasta que se cambie el valor a 1.
Por último, se envía un correo electrónico al usuario con la librería 'nodemailer' para comprobar que el correo electrónico existe y es el mismo que se ha registrado en la aplicación. Se utilizan los datos de configuración del fichero 'server/src/keys.ts'.

En la segunda función (`confirmaUsuario(…)`) se consulta en la base de datos el usuario filtrando por la id que viene como parámetro en la solicitud. Si no existe el usuario se devuelve error. Si existe, se comprueba que el *token* recibido en la solicitud es el mismo que el que está guardado como campo 'token' en la tabla 'user'. Si el *token* es el mismo, se actualiza el campo 'activo' a 1, y se redirige a la URL obtenida en el fichero 'server/src/keys.ts' en la variable frontendOptions.url. Si el *token* no coincide, se devuelve error.

A continuación, se analiza la función de actualizar usuario.

- server/src/controllers/usersController.ts

```
  public async update(req: Request, res: Response): Promise<void> {
        const { nombre, provincia, concello, descripcion, facebook_url, twitter_url, instagram_ur
l, otra_url, telefono, email_contacto } = req.body;
        const { id } = req.params;

        if (nombre !== null && nombre !== undefined && nombre !== '') {

            let fotoUrl = null;
            if (req.file != null && req.file != undefined) {
                let filename = '';
                if (req.file && req.file != undefined) {
                    const file = req.file;
                    filename = file.filename.replace(/ /g, '_').replace(/,/g, '')
                        .replace(/\//g, '').replace(/\\/g, '');
                }
                const fotoOld = await pool.query('select foto from user where id = ?', [id]);
                let nombreFotoOld = '';
                var c = new client();


                c.on('ready', function () {

                    if (req.file != null && req.file != undefined) {
                        c.put('./public/' + filename, 'public_html/fotos/' + filename, function (
err: any) {
                            if (err) {
                                throw err;
                            }
                            c.end();
                        });
                    }
                    if (fotoOld != null && fotoOld != undefined && fotoOld[0] != null) {
```

```
                    let foto1: string = fotoOld[0].foto;
                    try {
                        console.log(foto1);
                        console.log(req.body.foto);

                        if (foto1 != null) {
                            if (foto1 != req.body.foto) {
                                nombreFotoOld = foto1.split('/')[foto1.split('/').length - 1]
;



                                c.delete('public_html/fotos/' + nombreFotoOld, function (err:
 any) {
                                    if (err) {

                                        throw err;
                                    }
                                    c.end();
                                });
                            }
                        }
                    }
                    catch (e) {
                        console.log(e);
                    }
                }
            });
            try {
                var options = {
                    host: ftpOptions.host,
                    user: ftpOptions.user,
                    password: ftpOptions.password
                };
                c.connect(options, { debug: true });
            }
            catch (err) {
                console.log('FTP connection error catched: ', err);
            }
            if (req.file != null && req.file != undefined) {
                fotoUrl = frontendOptions.fotosUrl + "/" + filename;
            }
            else if (req.body.foto != undefined && req.body.foto != null) {
                fotoUrl = req.body.foto;
            }
        }
        else {
            const foto = req.body.foto;
            if (foto != undefined)
                fotoUrl = foto;
        }
        console.log('update user');


        const descripcionAux = descripcion != null && descripcion != undefined && descripcion
 != 'null' ? descripcion : null;
        const facebook_urlAux = facebook_url != null && facebook_url != undefined && facebook
_url != 'null' ? facebook_url : null;
        const twitter_urlAux = twitter_url != null && twitter_url != undefined && twitter_url
 != 'null' ? twitter_url : null;
        const instagram_urlAux = instagram_url != null && instagram_url != undefined && insta
gram_url != 'null' ? instagram_url : null;
```

```
            const otra_urlAux = otra_url != null && otra_url != undefined && otra_url != 'null' ?
 otra_url : null;
            const telefonoAux = telefono != null && telefono != undefined && telefono != 'null' ?
 telefono : null;
            const emailContactoAux = email_contacto != null && email_contacto != undefined && ema
il_contacto != 'null' ? email_contacto : null;
            const id_provincia = provincia != null && provincia != undefined && provincia && prov
incia > 0 ? provincia : null;
            const id_concello = concello != null && concello != undefined && concello && concello
 > 0 ? concello : null;

            if (fotoUrl != null) {
                await pool.query
                    ('UPDATE user set nombre=?, foto=?, descripcion=?, facebook_url=?, twitter_ur
l=?, instagram_url=?, otra_url=?, telefono=?, email_contacto=?, provincia=?, concello=? WHERE id=
?',
                        [nombre, fotoUrl, descripcionAux, facebook_urlAux, twitter_urlAux, instag
ram_urlAux, otra_urlAux, telefonoAux, emailContactoAux, id_provincia, id_concello, id]);
            }
            else {
                await pool.query
                    ('UPDATE user set nombre=?, descripcion=?, facebook_url=?, twitter_url=?, ins
tagram_url=?, otra_url=?, telefono=?, email_contacto=?, provincia=?, concello=? WHERE id=?',
                        [nombre, descripcionAux, facebook_urlAux, twitter_urlAux, instagram_urlAu
x, otra_urlAux, telefonoAux, emailContactoAux, id_provincia, id_concello, id]);
            }

            const tablas_rel = ['profesion', 'estilo', 'instrumento'];
            for (let tabla of tablas_rel) {
                //console.log(tabla);
                let nomeTablaPlural = '';
                if (tabla === 'profesion') {
                    nomeTablaPlural = 'profesiones';
                }
                if (tabla === 'estilo') {
                    nomeTablaPlural = 'estilos';
                }
                if (tabla === 'instrumento') {
                    nomeTablaPlural = 'instrumentos';
                }
                if (req.body[nomeTablaPlural]) {
                    const sqlTabla = 'select ' + tabla + '.id as id, ' + tabla + '.nombre as nomb
re from user, ' + tabla + ' join rel_profesional_' + tabla +
                        ' on rel_profesional_' + tabla + '.id_' + tabla + ' = ' + tabla +
                        '.id where user.id = rel_profesional_' + tabla + '.id_profesional and rel
_profesional_' + tabla + '.id_profesional = ?';
                    const resultsTabla = await pool.query(sqlTabla, [id]);
                    const registrosNew = JSON.parse(req.body[nomeTablaPlural]);
                    let registrosAnadir: any[] = [];


                    if (resultsTabla && resultsTabla.length > 0) {
                        let registrosBorrar: any[] = [];
                        for (let elem of resultsTabla) {
                            if (!registrosNew.map((x: { id: number, nombre: string }) => x.id).in
cludes(elem.id)) {
                                registrosBorrar.push(elem);
                            }
                        }

                        if (registrosBorrar.length > 0) {
```

```
                            const ids = registrosBorrar.map((x: { id: number, nombre: string }) =
> x.id).join(',');
                            await pool
                                .query('DELETE FROM rel_profesional_' + tabla + ' where id_profes
ional = ? and id_' + tabla + ' in (' + ids + ')',
                                    id);
                        }
                    }
                    //se añaden los nuevos estilos al usuario
                    if (registrosNew && registrosNew.length > 0) {

                        for (let elem of registrosNew) {
                            if (!resultsTabla || (!resultsTabla.map((x: { id: number, nombre: str
ing }) => x.id).includes(elem.id))) {
                                registrosAnadir.push(elem);
                            }
                        }
                        if (registrosAnadir.length > 0) {

                            const values_sql = registrosAnadir.map((x: { id: number, nombre: stri
ng }) => '(' + id + ', ' + x.id + ') ').join(',');

                            const sql = 'INSERT INTO rel_profesional_' + tabla + ' (id_profesiona
l, id_' + tabla + ') VALUES ' + values_sql;
                            await pool
                                .query(sql);
                        }
                    }
                }
            }
            if (req.body.videos) {
                const sqlVideos =
                    'SELECT * FROM `rel_profesional_video` where id_profesional =?';
                const resultVideos = await pool.query(sqlVideos, [id]);
                const videosNew = JSON.parse(req.body.videos);

                let videosAnadir: any[] = [];
            //se eliminan los videos asociados al usuario que ha eliminado (no están en req.body)
                if (resultVideos && resultVideos.length > 0) {
                    let videosBorrar: any[] = [];

                    for (let elem of resultVideos) {
                        if (!videosNew.includes(elem.url_video)) {
                            videosBorrar.push(elem.id);
                        }
                    }

                    if (videosBorrar.length > 0) {
                        const urls = videosBorrar.join(',');
                        // console.log(urls);
                        await pool
                            .query('DELETE FROM rel_profesional_video where id_profesional = ? an
d id in (' + urls + ')',
                                id);
                    }
                }
                //se añaden los nuevos videos al usuario
                if (videosNew && videosNew.length > 0) {
                    for (let elem of videosNew) {
                        if (!resultVideos.map((x: { id: number, id_profesional: number, url_video
: string }) => x.url_video).includes(elem)) {
                            videosAnadir.push(elem);
                        }
```

```
                }
                if (videosAnadir.length > 0) {
                    const values_sql = videosAnadir.map((x: string) => '(' + id + ', ' + "'"
+ x + "'" + ') ').join(',');
                    const sql = 'INSERT INTO rel_profesional_video (id_profesional, url_video
) VALUES ' + values_sql;
                    await pool
                        .query(sql);
                }
            }
            res.json({ message: "Usuario actualizado" });
        }
    }
    else {
        res.status(403).json({ message: "Error. Faltan datos." });

    }
}
```

- server/src/routes/usersRoutes.ts

```
const DIR = './public/';
    const storage = multer.diskStorage({
      destination: (req, file, cb) => {
        cb(null, DIR);
      },
      filename: (req, file, cb) => {
        const fileName = file.originalname.toLowerCase().split(' ').join('-');
        cb(null, fileName)
      }
    });
    var upload = multer({ storage: storage });
    this.router.put('/user/:id', upload.single('fotoUsuario'), usersController.update);
```

En primer lugar, se comprueba que el nombre del usuario recibido en el cuerpo de la solicitud no viene vacío, ya que es un campo obligatorio. El siguiente paso es obtener la foto de perfil. Se utiliza la librería 'multer' para guardar un archivo en el servidor donde está ejecutada la API, y se añade a la *request*. De esta forma, en la función 'update' de 'usersController', se tiene acceso a este archivo con '`req.file`'. Se comprueba si el usuario ya tenía foto de perfil, y si es así se elimina de la carpeta a través de FTP. Se utiliza la librería 'ftp' para eliminar y subir la foto de perfil.

El siguiente paso es actualizar los campos de la tabla 'user' como la descripción, los enlaces a redes sociales, la URL de la foto de perfil, provincia, etc. Para cada propiedad, se comprueba que no sean indefinidos ni tengan el valor 'null'.

Por último, se revisan las tablas asociadas al usuario en caso de que el usuario obtenido en la solicitud tenga alguno de estos datos. Se implementa un bucle para no repetir código casi idéntico tres veces, y para que sea fácil de refactorizar en caso de querer añadir algún campo más al usuario. Para cada tabla (estilo, profesión e instrumento) se consultan en la base de datos los registros asociados al usuario, y se añaden a una lista. Esta lista se compara con la lista que viene como propiedad del cuerpo de la solicitud, se eliminan los registros que ya no existen, y se añaden los nuevos. Se hace lo mismo para los vídeos, eliminando los

enlaces que había en la base de datos y no hay en la solicitud, y se añaden los que no existían en la base de datos.

Por último, se muestra el código del componente 'add-elements' en Angular, utilizado en el componente 'perfil' para añadir especialidades, estilos musicales o instrumentos.

- client/src/app/perfil/add-elements/add-elements.component.html

```
<form [formGroup]="rForm" fxFlex fxLayout="column">
<div class="resultados-elementos" class="p-4">
    <ul>
        <li *ngFor="let elem of elementos">
            <div class="contenedor-elemento">
                <div class="nombre-elemento">{{elem.nombre}}</div>
                <div class="eliminar-
elemento"><button type="button" (click)="quitaElemento(elem)">x</button></div>
            </div>
        </li>
    </ul>
</div>

<mat-form-field class="buscarElemento"  class="p-2">
    <input type="text" placeholder="Engadir" [value]="elemento.nombre" matInput
        [formControl]="elementoControl" [matAutocomplete]="auto">
    <mat-
autocomplete required #auto="matAutocomplete" (optionSelected)='anadeElemento($event.option.value
)'>
        <mat-option *ngFor="let option of filteredOptions | async" [value]="option.nombre">
            {{option.nombre}}
        </mat-option>
    </mat-autocomplete>
</mat-form-field>
</form>
```

- client/src/app/perfil/add-elements/add-elements.component.ts

```
import { Component, OnInit, Input, SimpleChanges } from '@angular/core';
import { FormControl, FormGroup } from '@angular/forms';
import { Observable } from 'rxjs';
import { startWith, map } from 'rxjs/operators';

@Component({
  selector: 'app-add-elements',
  templateUrl: './add-elements.component.html',
  styleUrls: ['./add-elements.component.scss']
})
export class AddElementsComponent implements OnInit {
  rForm: FormGroup;
  elementoControl = new FormControl();
  elemento: any = { id: 0, nombre: '' };
  @Input() elementos: any[] = [{}];
  @Input() totalElementos: any[] = [{}];
  filteredOptions: Observable<any[]>;

  constructor() { }
  ngOnInit() {
    this.rForm = new FormGroup(
      {
        nombre: new FormControl(this.elemento.nombre)
```

```
      }
    );
    this.filteredOptions = this.elementoControl.valueChanges
      .pipe(
        startWith(''),
        map(value => this._filter(value))
      );
  }
  private _filter(value): any[] {
    const filterValue = value.toLowerCase();
    let elementosAux = this.totalElementos.filter(option => option.nombre.toLowerCase().startsWit
h(filterValue));
    let elementosResto = this.totalElementos.filter(option => option.nombre.toLowerCase().include
s(filterValue) && ! option.nombre.toLowerCase().startsWith(filterValue) );
    return elementosAux.concat(elementosResto);
  }

  anadeElemento(event) {
    if (event && event != '') {
      const eventAux = this.totalElementos.filter(x => x.nombre === event)[0];
      this.elementos.push(eventAux);
      console.log(eventAux);

      this.elemento = { id: 0, nombre: '' };
      console.log(this.elementos);
      this.elementoControl.setValue('');
      this.totalElementos.splice(this.totalElementos.indexOf(eventAux), 1);
    }
  }
  quitaElemento(event) {
    if (event.nombre && event.nombre !== "") {
      this.elementos.splice(this.elementos.indexOf(event), 1);
      this.totalElementos.push(event);
    }
  }
}
```

Como las tablas relacionadas al usuario tienen las mismas propiedades (id y nombre), se reutiliza el mismo componente para añadir y eliminar profesiones, estilos musicales e instrumentos.

En el *html*, en primer lugar, se muestran los elementos asociados al usuario en un bucle '*ngFor'. Debajo de esta sección se muestra un input con el componente de Angular Material 'mat-autocomplete'. Se obtiene la colección de elementos del componente padre 'perfil'.

La función '_filter', que se ejecuta cada vez que se escribe algo en el input 'autocomplete', devuelve la colección de elementos filtrada por lo que escribe el usuario, mostrando primero las opciones cuyo nombre empieza por el valor del input, y debajo las que contienen ese valor, pero no al principio. La función 'quitaElemento' se ejecuta al hacer click en el botón representado por 'x' presente en cada elemento asociado al usuario situado al lado del nombre del elemento, y 'anadeElemento' se ejecuta al seleccionar una opción de la colección 'filteredOptions' en el 'autocomplete'.

El componente 'add-elements' se muestra en el componente 'perfil' tres veces, asignando el total de elementos a la propiedad 'totalElementos', y los elementos asociados al usuario a la propiedad 'elementos':

```
<div class="titulo-elementos">Estilos musicais</div>
<app-add-elements [totalElementos]='totalEstilos' [elementos]='estilosUser'></app-add-elements>
```

Anexo 3. Capturas de pantalla

A continuación, se añaden capturas de pantalla de la aplicación final en un navegador Google Chrome.

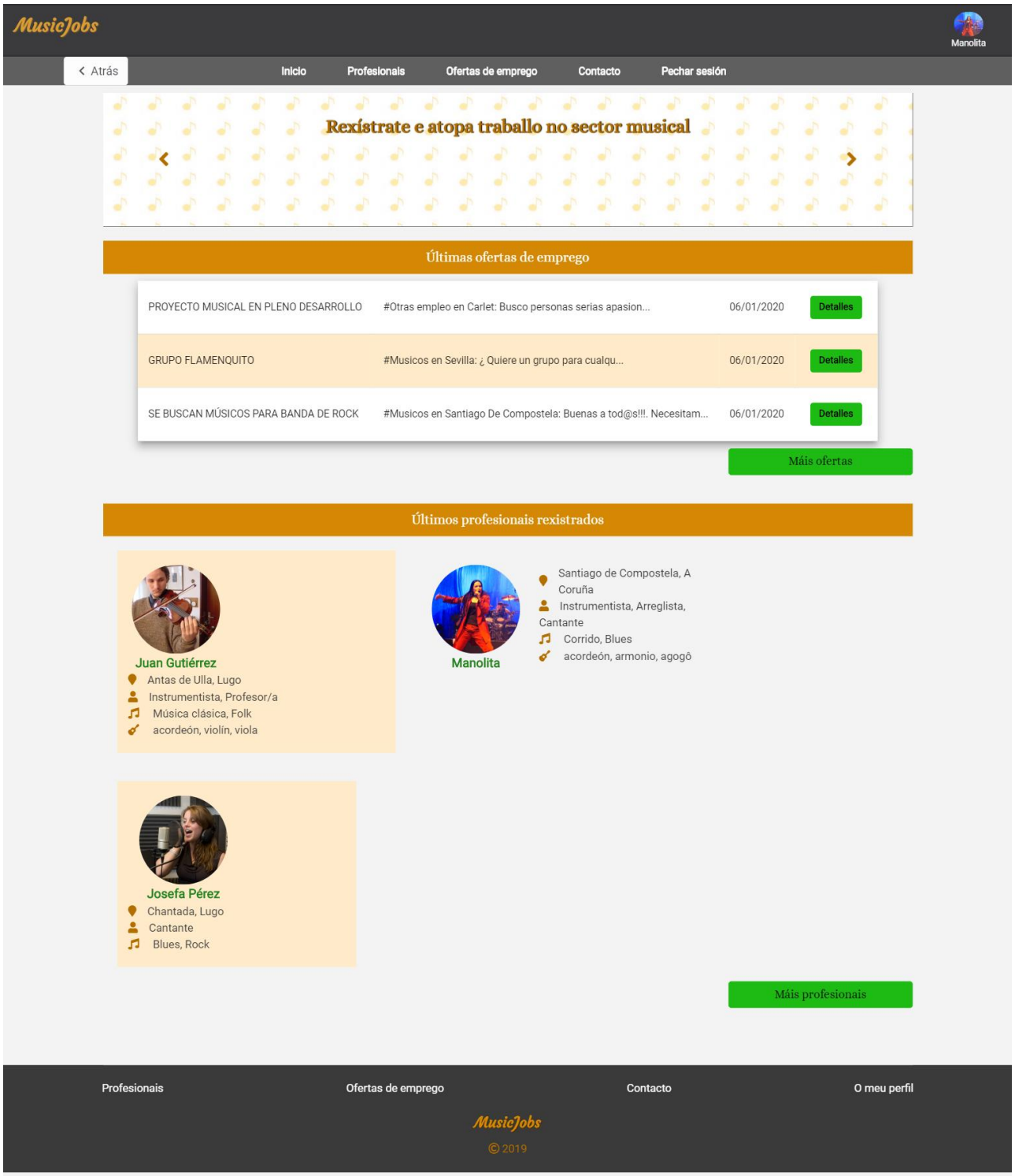

Ilustración 39. Captura Inicio Escritorio

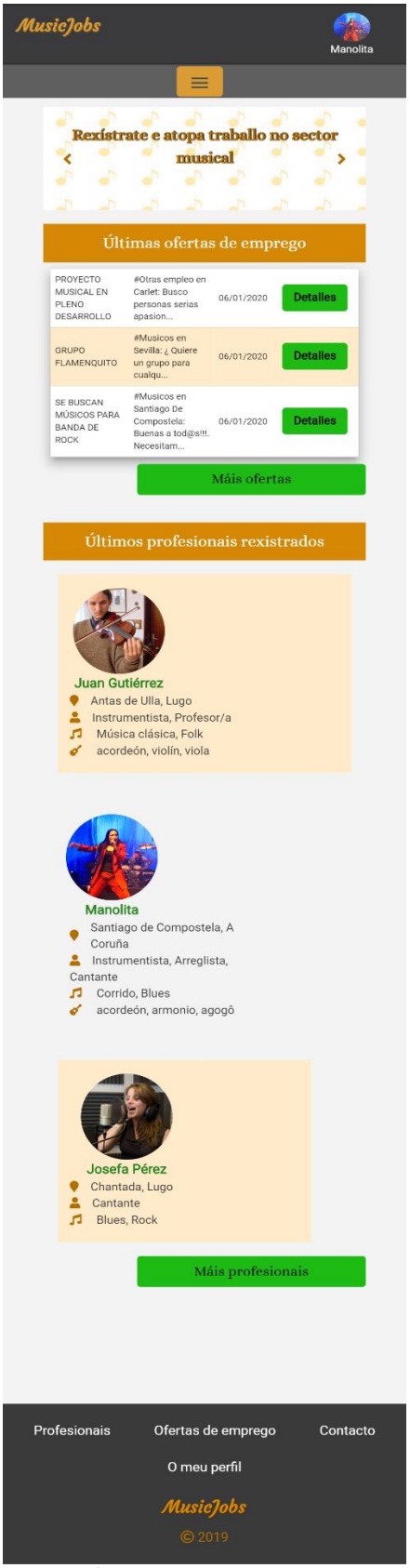

Ilustración 40. Captura inicio Smartphone

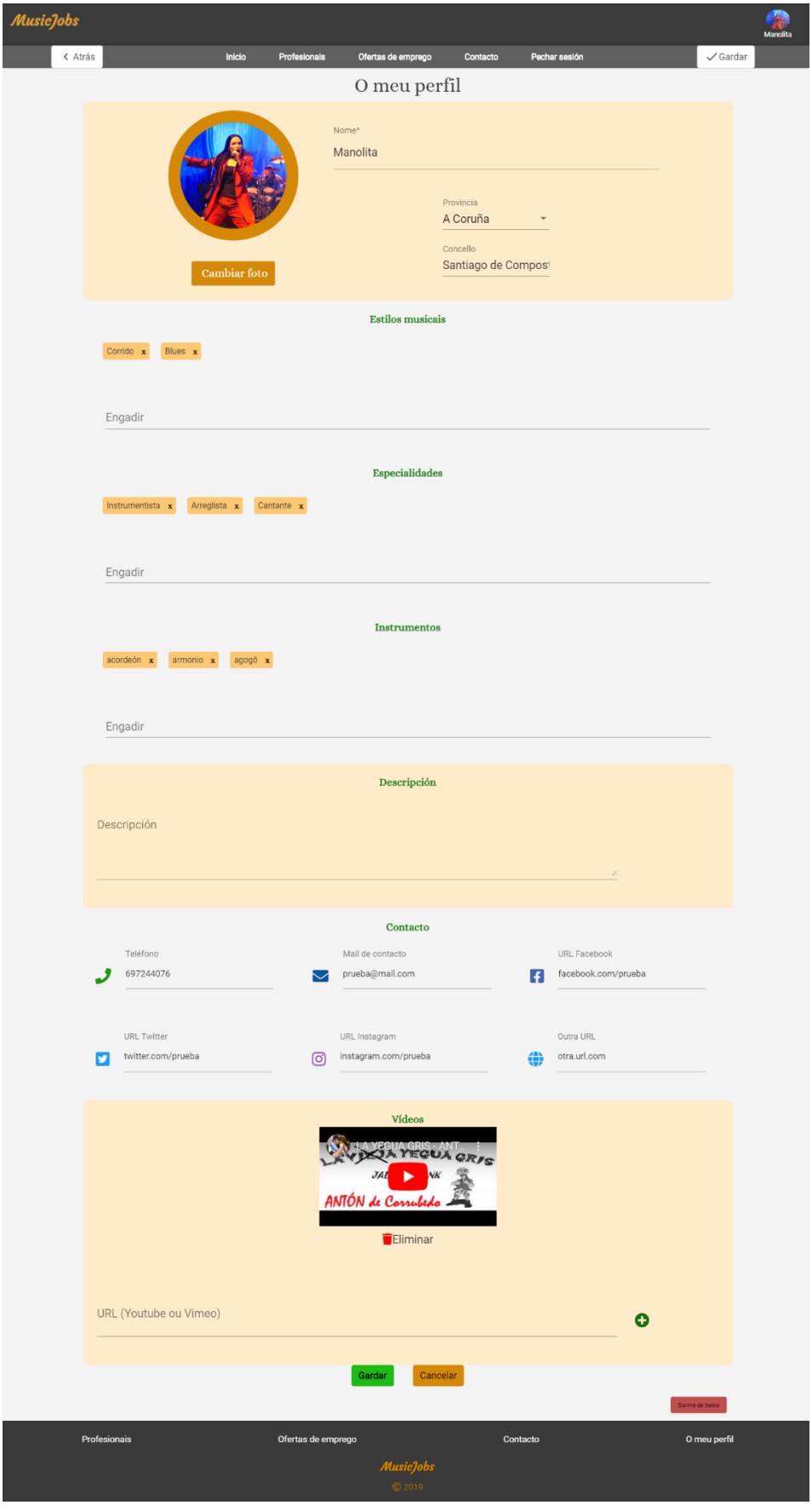

Ilustración 41. Captura Mi perfil Escritorio

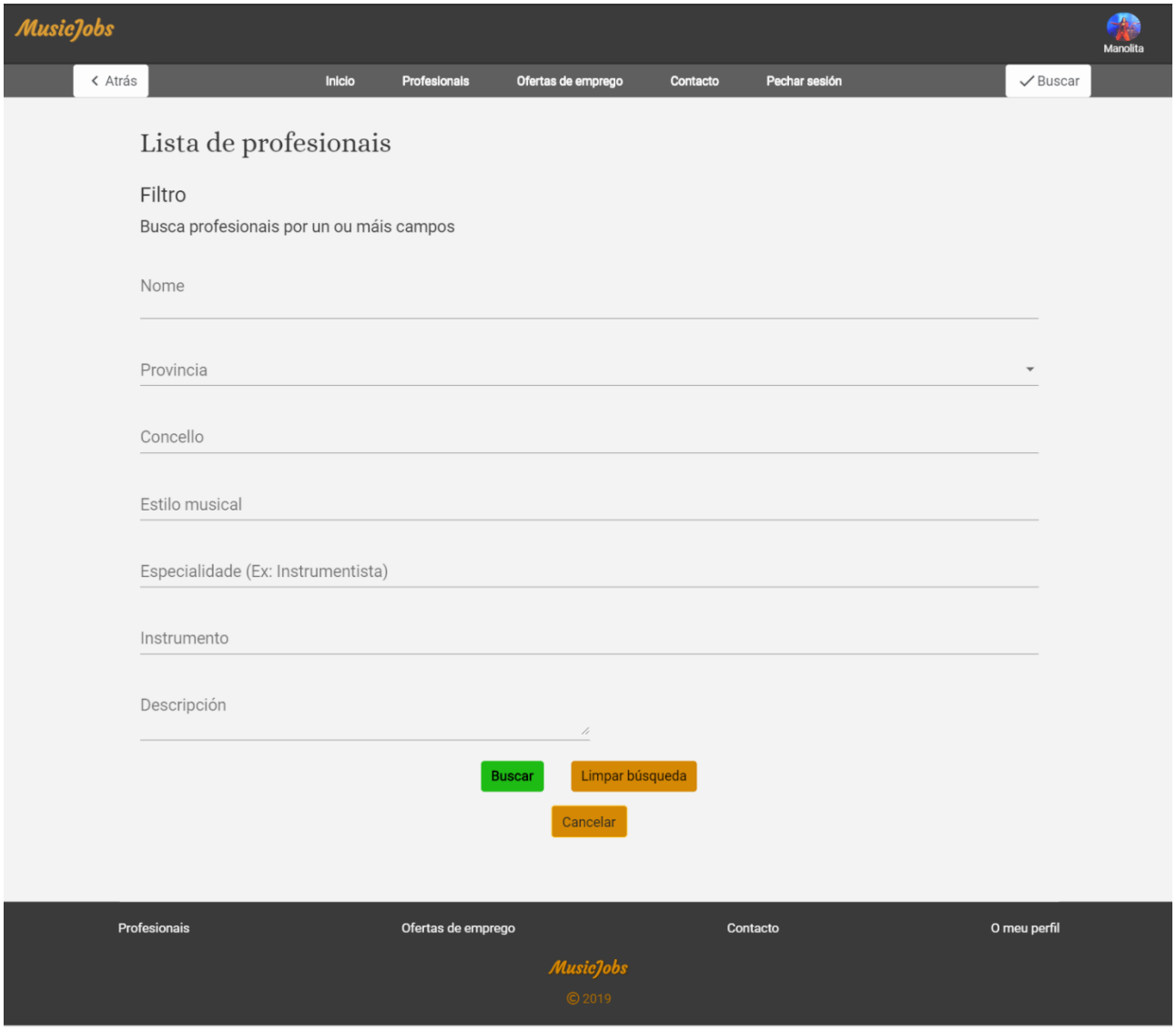

Ilustración 42. Captura buscador profesionales Escritorio

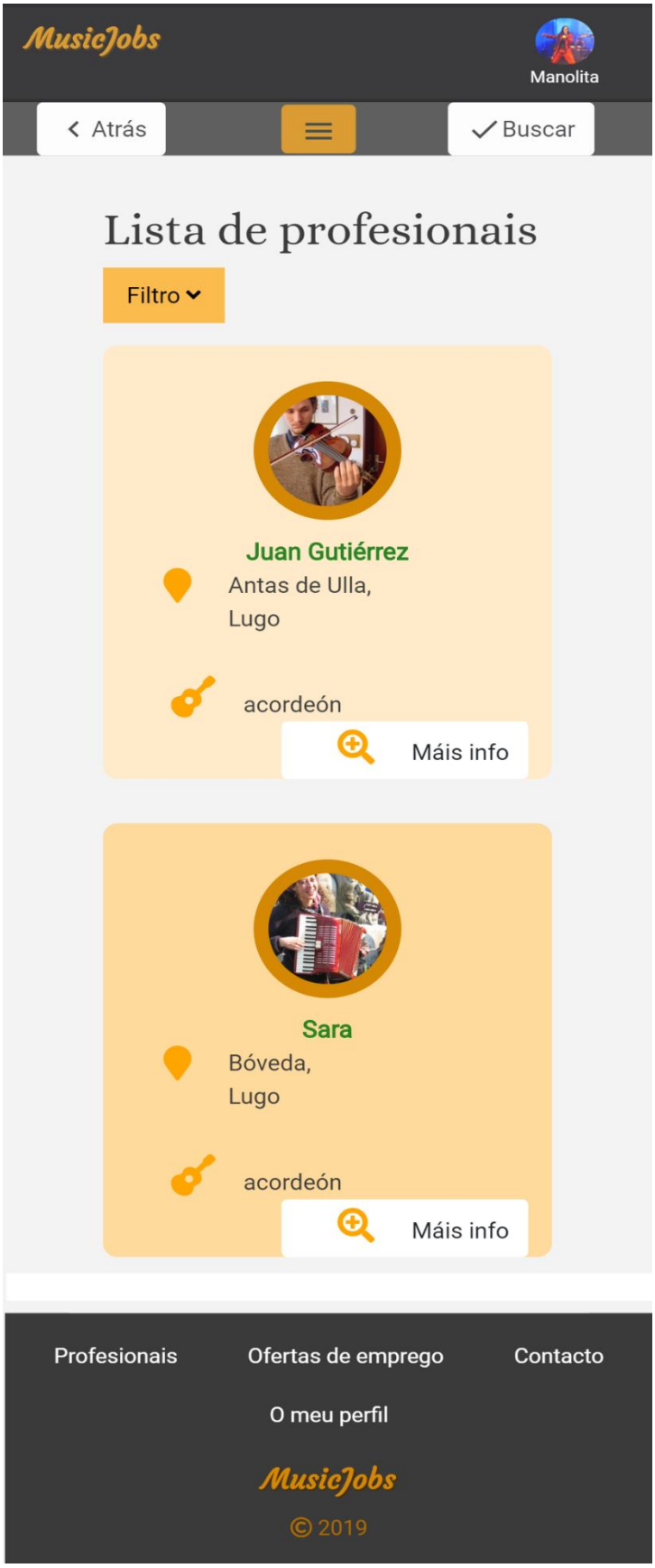

Ilustración 43. Captura resultados profesionales Smartphone

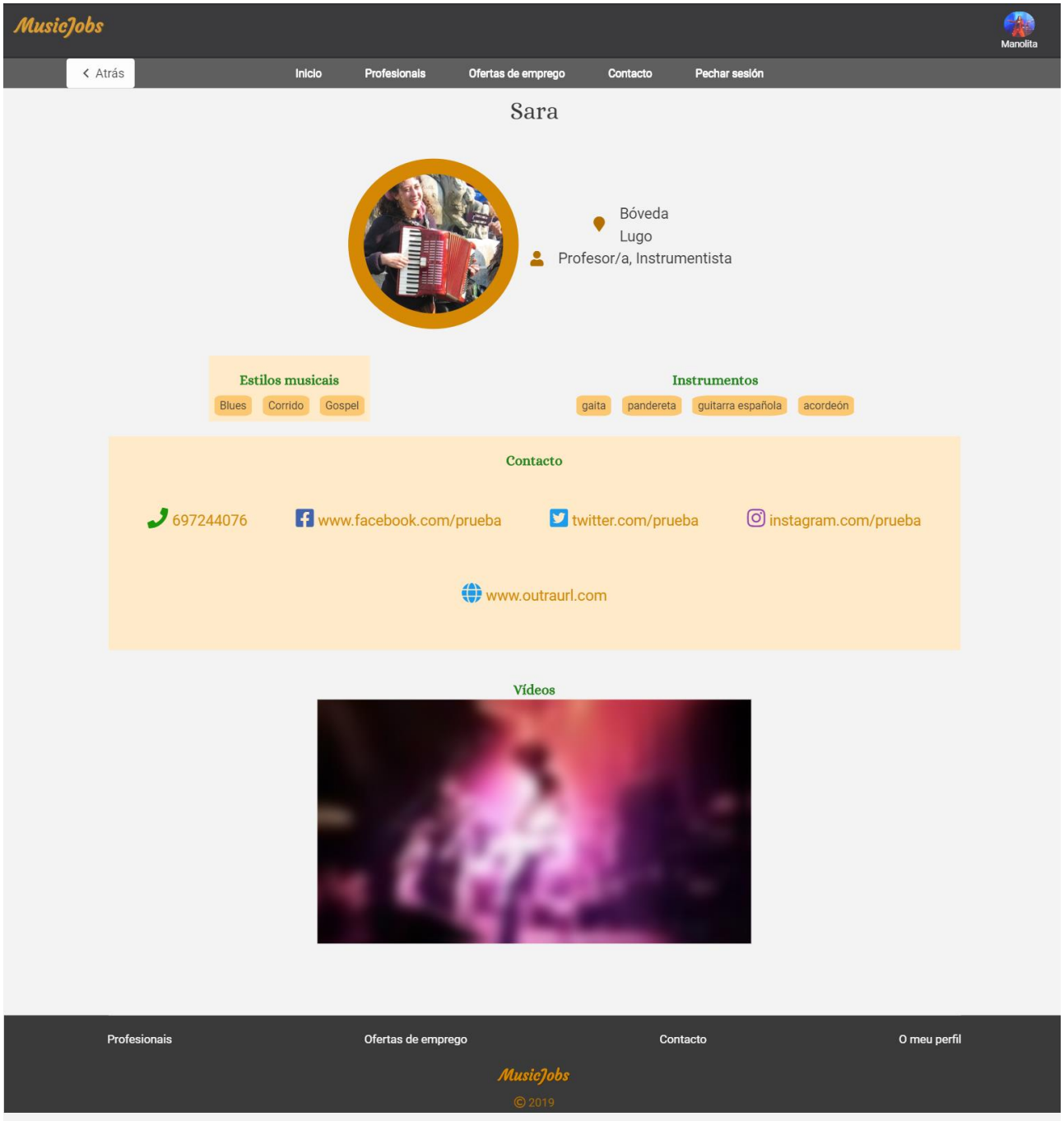

Ilustración 44. Captura Detalle Profesional Escritorio

Ilustración 45. Captura Detalle Profesional Smartphone

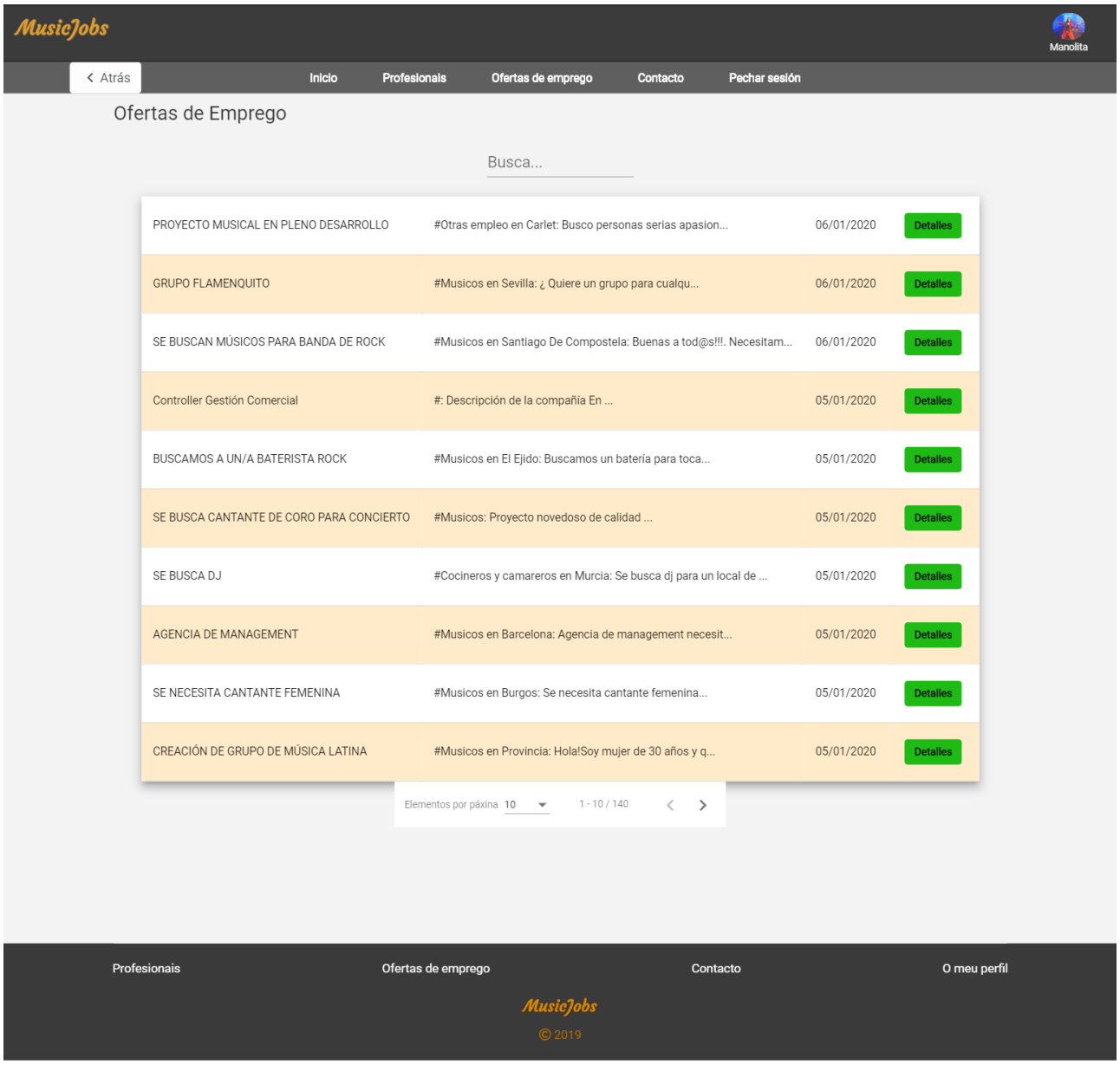

Ilustración 46. Captura Ofertas de Empleo Escritorio

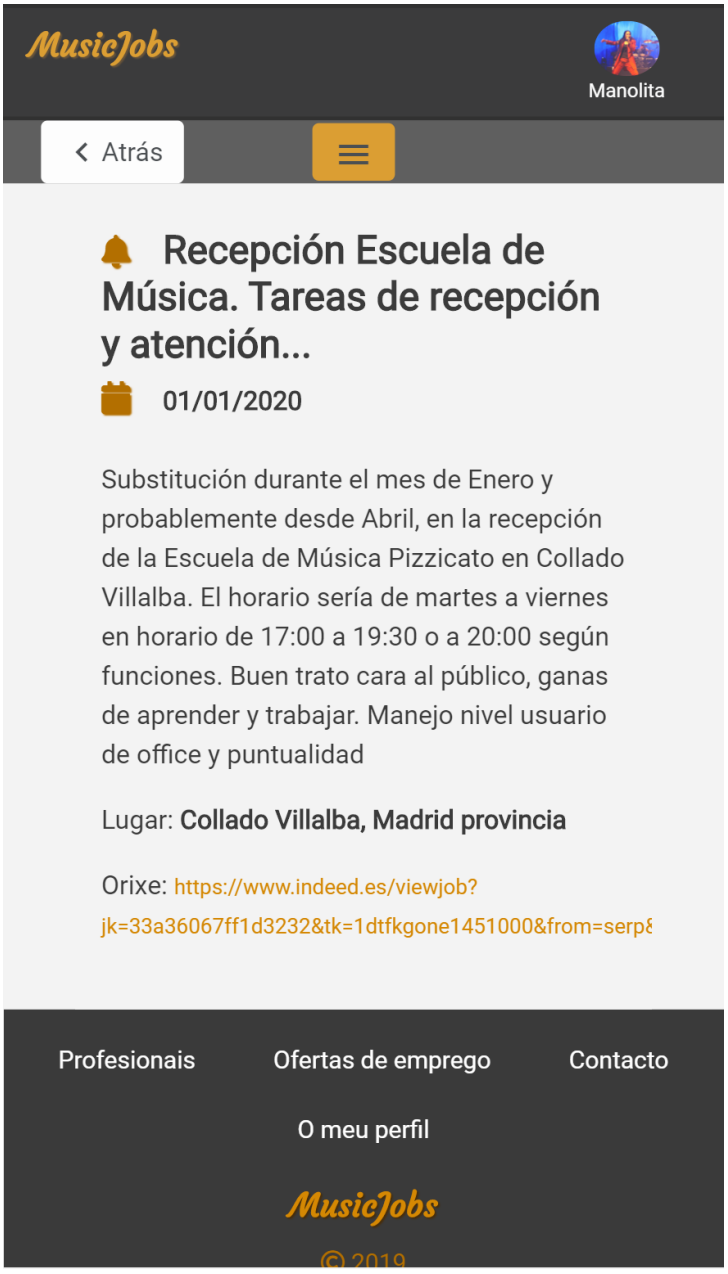

Ilustración 47. Captura Detalle Oferta Smartphone

Anexo 4. Libro de estilo

A continuación, se describe la línea gráfica del trabajo.

Logotipo: Se ha creado el icono y logotipo de la aplicación en Paint, ya que es una corchea formada por un círculo, una línea recta y otra curva.

Ilustración 48. Icono *MusicJobs*

Colores: Como colores principales se utilizan el naranja y el verde en distintas tonalidades. Para el menú de navegación y el *footer* se utilizan grises oscuros casi negros.

Fuentes: Se utilizan 3 fuentes obtenidas de *Google Fonts*. La fuente por defecto es 'Roboto'. Para el título de la aplicación en el menú de navegación y en el *footer* se utiliza 'Courguette', y para algún texto en títulos o botones se utiliza 'Alice'.

En los mensajes de alerta se utilizan el rojo en caso de error, el verde en caso de éxito y el naranja para los avisos, con las clases de 'Bootstrap' 'alert alert-danger', 'alert alert-success' y 'alert alert-warning'.

Iconos: Se utilizan iconos de 'font-awesome' y de 'material-icons' en el menú de navegación en dispositivos pequeños, y para la información de los profesionales y de las ofertas.

Anexo 5. Bibliografía

<http://llengua.gencat.cat/ca/serveis/informacio_i_difusio/publicacions_en_linia/btpl_col/citaciobibliografica/>
Relación de municipios y códigos por provincia (2019) [en línea]. Instituto Nacional de Estadística. [Fecha de consulta: 5 de noviembre de 2019]
<https://www.ine.es/daco/daco42/codmun/codmun19/19codmun.xlsx>
Nombres de instrumentos musicales (2018) [en línea]. Promoción Musical.
<https://promocionmusical.es/instrumentos-musicales/nombres-de-instrumentos-musicales/>
Número de conciertos de música popular en vivo en España en 2017, por comunidad autónoma (2019). [en línea] Statista. [Fecha de consulta: 12 de diciembre de 2019]
https://es.statista.com/estadisticas/475086/numero-de-conciertos-de-musica-en-directo-espana-por-cc-aa/
Evolución anual de la facturación neta de la industria de la música en vivo en España entre 2005 y 2018 (en millones de euros) (2019) [en línea]. Statista. [Fecha de consulta: 14 de noviembre de 2019].
https://es.statista.com/estadisticas/472441/industria-de-la-musica-en-vivo-facturacion-en-espana/
Empleo por ramas de actividad (2019) [en línea]. Instituto Nacional de Estadística. [Fecha de consulta: 14 de noviembre de 2019]
https://www.ine.es/jaxiT3/Tabla.htm?t=28607&L=0
Anuario de Estadísticas Culturales 2018 (2018) [en línea]. Ministerio de Cultura y Deporte. [Fecha de consulta: 14 de noviembre de 2019]
https://www.culturaydeporte.gob.es/dam/jcr:eb5b8140-e039-42ab-8e24-500fddc5b2a4/anuario-de-estadisticas-culturales-2018.pdf
Profesionales de la música y danza por tipo (2017). Ministerio de Cultura y Deporte. [Fecha de consulta: 14 de noviembre de 2019]
http://estadisticas.mecd.gob.es/CulturaJaxiPx/Tabla.htm?path=/t18/p18/a2005//l0/&file=T1801005.px&type=pcaxis&L=0