TRABAJO DE FIN DE GRADO

MUSICIN

FECHA: JUNIO 2018

AUTOR: IGNACIO GARCÍA-CANO LAVAZZA

TUTOR: ABRAHAM RODRÍGUEZ RODRÍGUEZ

UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA
Escuela de Ingeniería Informática

TFT04

SOLICITUD DE DEFENSA DE TRABAJO DE FIN DE TÍTULO

D/Dª **Ignacio García-Cano Lavazza**, autor del Trabajo de Fin de Título **MusicIn**, correspondiente a la titulación **Grado Ingeniería Informática**

S O L I C I T A

que se inicie el procedimiento de defensa del mismo, para lo que se adjunta la documentación requerida.

Asimismo, con respecto al registro de la propiedad intelectual/industrial del TFT, declara que:

[ ] Se ha iniciado o hay intención de iniciarlo (defensa no pública).
[X] No está previsto.

Y para que así conste firma la presente.

Las Palmas de Gran Canaria, a 23 de mayo de 2018.

El estudiante

Fdo.: GARCIA-CANO LAVAZZA IGNACIO - 44747095C

Firmado digitalmente por GARCIA-CANO LAVAZZA IGNACIO - 44747095C
Fecha: 2018.05.23 13:25:14 +01'00'

A rellenar y firmar **obligatoriamente** por el/los tutor/es

En relación a la presente solicitud, se informa:

[x] Positivamente

[ ] Negativamente
(la justificación en caso de informe negativo deberá incluirse en el TFT05)

Fdo.: RODRIGUEZ RODRIGUEZ ABRAHAM - 42832764W

Firmado digitalmente por RODRIGUEZ RODRIGUEZ ABRAHAM - 42832764W
Nombre de reconocimiento (DN): c=ES, serialNumber=42832764W, sn=RODRIGUEZ RODRIGUEZ, givenName=ABRAHAM, cn=RODRIGUEZ RODRIGUEZ ABRAHAM - 42832764W
Fecha: 2018.05.24 13:13:46 -01'00'

DIRECTOR DE LA ESCUELA DE INGENIERÍA INFORMÁTICA

Lista de comprobación de documentación adjunta (resumida, detalles en el Manual Operativo)

[X] Memoria (máximo 100 pág. ~30000 palabras) en formato pdf con logos e identificación del Trabajo en la portada y, tras ella, la primera página de este TFT04 insertada (el pdf resultante debe firmarse digitalmente, ubicando las firmas en el espacio previsto para ello en el TFT04).

[X] Resumen en formato txt en español e inglés (un único fichero txt con límite de 120 palabras por idioma).

[X] Código (en caso de que el TFT lo incluya, un directorio, o bien fichero comprimido, o bien un txt con instrucciones para acceder al repositorio).

- Sólo en el caso de que en este impreso TFT04 se haya indicado que NO está previsto iniciar trámite de registro de la propiedad intelectual/industrial …

[ ] Impreso relativo a la Difusión en Abierto del TFT en Biblioteca ULPGC como pdf firmado de forma digital.

NOTA: si se marca el registro de la propiedad intelectual sólo los tutores y el tribunal pueden asistir a la defensa.

Procedimiento de entrega: se enviarán instrucciones concretas a través del Moodle.

Tabla de contenido

1 Introducción

1.1 Contexto

Hoy en día usamos las redes sociales para todo, desde compartir contenido hasta buscar opiniones sobre algún restaurante de la zona. El potencial y alcance de las mismas es tan grande que nos satura de la cantidad de posibilidades e información disponible. Los músicos actuales utilizan bastante "las grandes", como Facebook o Twitter para compartir gustos musicales, avisar de conciertos a los que van a asistir o que van a realizar, conseguir nuevos miembros para sus bandas, etc. El problema, es que esta información se mezcla con todo el contenido y muchas veces el alcance esperado no es el mejor.

Por lo tanto, se propone la concepción de una plataforma social (MusicIn) diseñada específicamente para músicos, donde pueden ver y compartir opiniones, información sobre eventos, curiosidades, etc. Tanto a nivel global como a nivel local. Además, se utilizará no sólo como una red social, sino como una herramienta de búsqueda de "empleo" para músicos y promotores.

1.2 Objetivos

El desarrollo del proyecto implica la realización de un análisis de aplicaciones similares, la definición de los usuarios potenciales, la definición y el diseño detallado del producto final y la implementación de un prototipo de aplicación móvil que, al menos incluya las siguientes características o funcionalidades:

- Gestión de usuarios
- Envío de notificaciones
- Gestión de publicaciones
- Geolocalización de eventos y publicaciones
- Buscar y agregar como amigos a otros usuarios
- Chat individual
- Información adicional sobre músicos y álbumes en las publicaciones, aprovechándose de la Web Semántica
- Conexión entre músicos y promotores para conseguir empleo

1.3 Metodología

Se utilizará una metodología iterativa en la que se definirán distintos ciclos hasta obtener un prototipo funcional con las principales características implementadas. Se ha estimado que el desarrollo del proyecto se realizará en cinco iteraciones, las cuáles constan de tres fases cada una, divididas en definición, implementación y validación y pruebas. En cada una de las iteraciones los conceptos a tratar son:

- El rol del músico dentro de la aplicación

- Los eventos y tipos de publicaciones que se pueden realizar
- La gestión de usuarios y el rol de promotor
- El sistema de mensajería entre músicos y promotores
- La aplicación de la web semántica para ofrecer información adicional dentro de la app

1.4 Competencias específicas cubiertas

Las competencias específicas cubiertas en el desarrollo de este TFT son la CII02 y la TI06.

1.5 Aportaciones al entorno socio-económico, técnico o científico

Como se explicó en el contexto, la gran mayoría de los músicos utilizan las redes sociales de propósito general para no sólo compartir contenido, sino que además buscan trabajo y conexión con otros músicos/productores. El problema radica en que esto se entremezcla con muchas publicaciones que poco o nada tienen que ver con estos temas. **MusicIn** quiere convertirse en un producto de nicho y trasladar el concepto de una red social como LinkedIn y orientarse específicamente al mundo de la música. Esto se ve en los dos aspectos principales que se ofrecen, los cuales son:

- Ponerte en contacto directo con otras personas de diferentes perfiles para buscar componentes para tu banda, conseguir contratos musicales, conseguir actuaciones en locales o buscar empresas organizadoras de conciertos entre otras cosas.
- Poder conocer de forma rápida y sencilla los diferentes eventos que suceden tanto a nivel local y a nivel global.

1.6 Estructura de la memoria

Para el desarrollo de esta memoria se ha elegido una estructura donde se hable de los siguientes puntos:

Análisis previo
En este apartado se incluye todo lo estudiado previamente al desarrollo de la aplicación y que además, se explican los diferentes motivos por los cuales el producto final tiene sentido en el mercado. Para ello, se incluyen entre otras cosas comparaciones con otras aplicaciones similares, tablas de dummies, análisis DAFO, modelo de negocio, etc.

Iteraciones
Para cada iteración se le dedicará un apartado explicando cada una de las tres fases por las que se desglosa cada iteración. Algunos de los aspectos a incluir entre otros son el diseño arquitectónico, los diagramas de caso de uso y los mockups.

Tecnologías
Para este punto se describirán las diferentes tecnologías utilizadas para el desarrollo de la aplicación, explicando de forma genérica qué son y cuál es su cometido dentro del proyecto.

Acceso al código y despliegue
En este apartado de la memoria se incluirán los enlaces para acceder al código, al igual que los aspectos más relevantes para el despliegue del proyecto.

Conclusiones
Aquí se incluyen las valoraciones personales sobre todo el proceso de creación del producto, al igual que la explicación de por qué la implementación valida el análisis realizado.

Fuentes de información
Todas las fuentes y recursos consultados serán incluidos en este apartado.

Anexos
Donde se incluye otra información y el manual de usuario.

2 Análisis Previo

Antes de desarrollar cualquier producto, es necesario conocer el entorno donde nos desenvolvemos y realizar un previo estudio del mismo. De ese modo aumentaremos las probabilidades de triunfar y realizar un proyecto útil y rentable.

2.1 Estudio de aplicaciones similares

ReverbNation

Desarrollador: eMinor Inc.
Fecha de inicio: 31 de octubre de 2006
Usuarios: 4 millones
Reverbnation es una plataforma que provee herramientas a los músicos para gestionar sus carreras profesionales a través de una app y a través de un portal web. Las funcionalidades más destacadas que presenta Reverbnation son:

- Subir temas propios a tu perfil, en formato de audio o vídeo, para compartirlos con el resto de usuarios.
- Un apartado de "Oportunidades" lanzado recientemente, donde se presentan diferentes eventos que pueden filtrarse por cercanía o fecha para conseguir una actuación o contrato con alguna discográfica.
- Gestionar y acceder a "Shows", incluyendo los precios de los mismos con posibilidad a pagar a través de la plataforma.
- Realizar campañas de email marketing a tus seguidores.
- Acceder a estadísticas sobre tu perfil, tus reproducciones y sobre tus listas de correo.

Por otra parte, Reverbnation plantea un enfoque de enlace sólo entre músicos y discográficas. El diseño de la interfaz y el planteamiento, dejan de lado a aquellos músicos cuya especialidad es el mundo audiovisual y buscan conexión con directores, programadores, productores, etc.

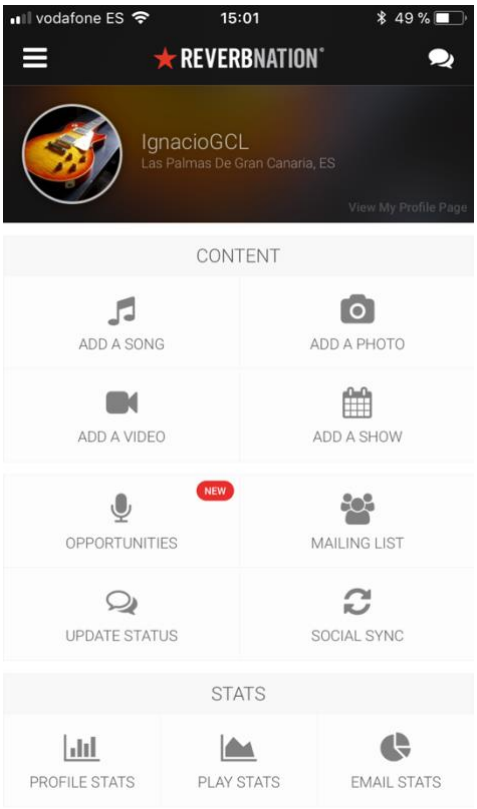


Figura 1.1. Interfaz de la app Reverbnation y sus diferentes opciones.

Reverbnation utiliza un modelo de ingresos basado en una suscripción. Hay tres modalidades, siendo la primera gratuita. La principal diferencia entre las dos modalidades de pago radica en la cantidad de opciones posibles.

Kompoz

Kompoz es una red social que permite a los músicos componer música de forma colaborativa sin importar la distancia a la que se encuentren. Todo esto se realiza a través del portal web www.kompoz.com. Esta herramienta cuenta con diferentes tipos de cuentas, todas ellas orientadas a músicos. La principal diferencia entre ellas es la cantidad de opciones disponibles, que incluyen poder colaborar con más artistas, subir tus canciones en diferentes formatos, crear colaboraciones privadas, obtener beneficios en la venta de tus composiciones y poder crear grupos. Además, cuenta con las opciones características de una red social que son, seguir a otros músicos que te gusten y compartir proyectos y oportunidades. Por último, incorpora un panel sencillo de estadísticas con información básica sobre el número de colaboraciones en las que uno ha participado y que ha visitado últimamente y el número de notificaciones recibidas.

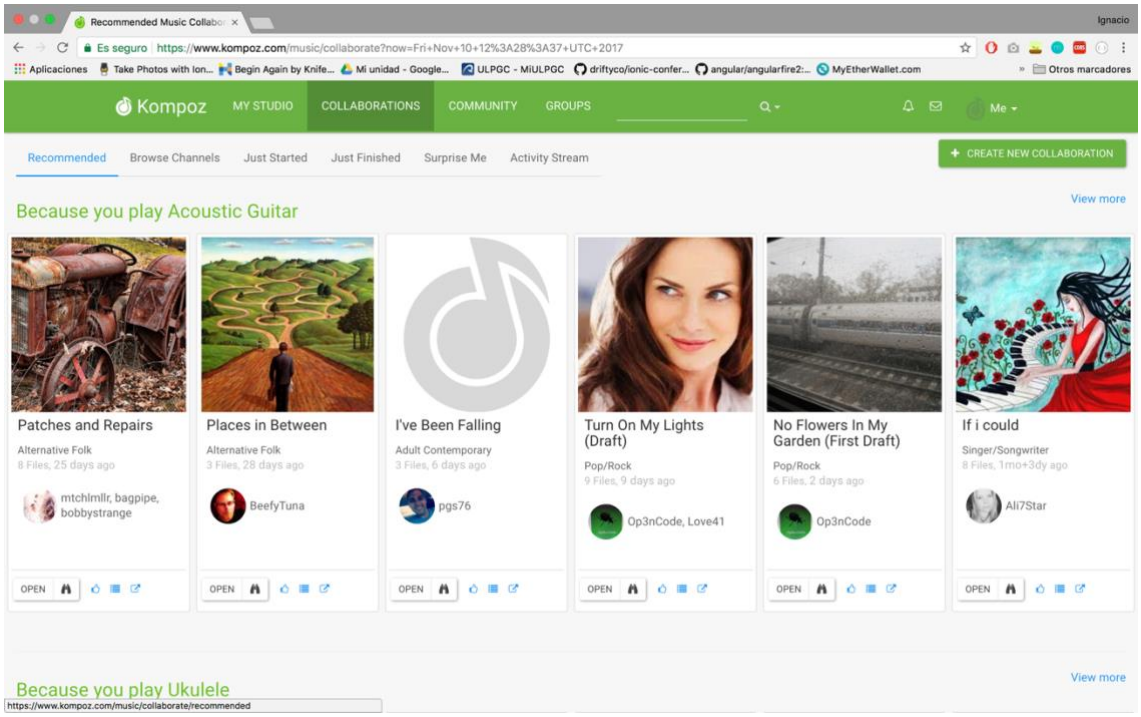


Figura 1.2. Interfaz principal del portal web de Kompoz

Al igual que Reverbantion, Kompoz cuenta con un modelo de ingresos basado en una suscripción. Ésta, cuenta con una opción gratuita y tres de pago, cuya principal diferencia, como se explicó en párrafos anteriores, es la cantidad de opciones disponibles.

Miuseek

"*Miles de músicos para formar y completar tu banda, encontrar trabajo, promocionar tu música y tus conciertos, comprar y vender instrumentos, los mejores estudios y locales de ensayo y muchas cosas más.*" Así es como se definen ellos mismos en su portal web www.miuseek.com. Además de lo incluido en la descripción, Miuseek también te da la posibilidad de acceder a la agenda musical de conciertos de tu ubicación y de poder escuchar música de otros artistas que sean afines a tus estilos musicales. Cuenta con una interfaz de aspecto un poco antiguo y con demasiadas opciones que al principio pueden empeorar la experiencia de usuario.

A diferencia de las dos anteriores, Miuseek opta por utilizar un sistema de pago por anuncios publicados. Esto cobra sentido ya que podemos incluir a Miuseek en el sector de los marketplaces. La mecánica de precios es similar a la de Facebook, ya que éste varía en función del tipo de objetivo al que se pretende llegar.

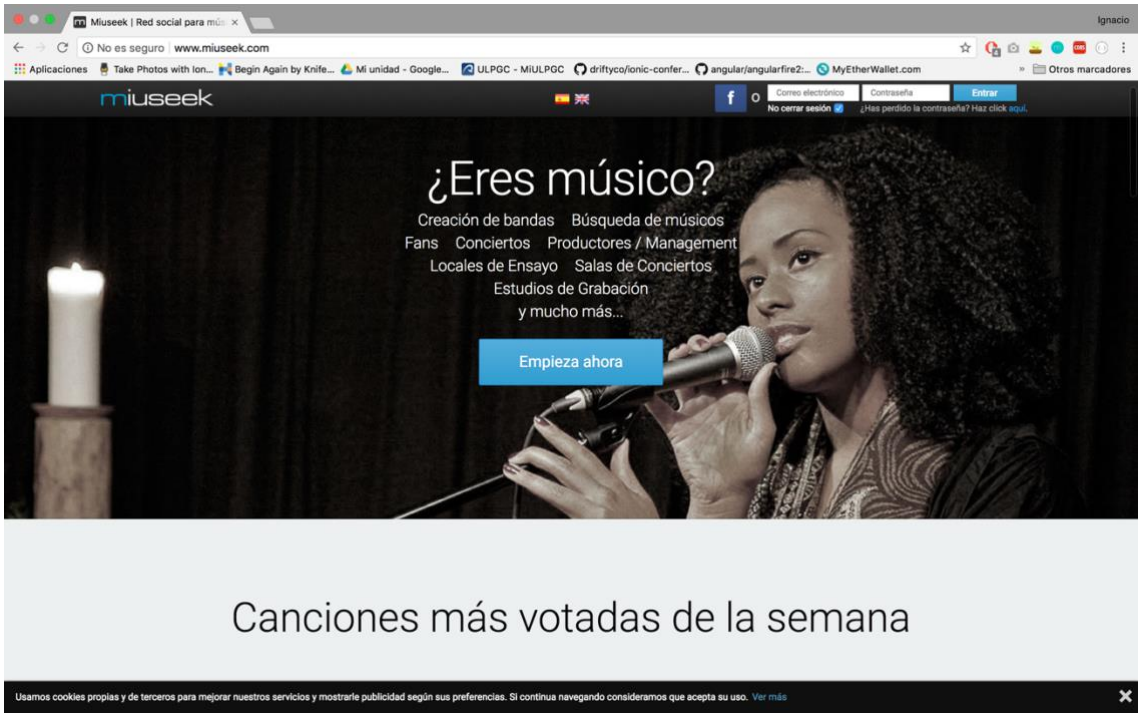


Figura 1.3. Interfaz del portal web de Miuseek

Comparación con MusicIn

Finalmente podemos comparar MusicIn de forma sencilla, con el resto de aplicaciones analizadas en una tabla resumen que se muestra a continuación.

	ReverbNation	Kompoz	Museek	MusicIn
Colaborar con otros músicos en composiciones		✓		
Contactar con otros músicos			✓	✓
Campañas de email marketing	✓			
Promoción de empleos de diferentes sectores del mundo audiovisual				✓
Compra venta de instrumentos			✓	

Escuchar música de otros artistas	✓	✓	✓	
Poner en contacto a músicos con discográficas	✓			✓
Promocionar eventos	✓			✓
Saber que eventos están realizándose cerca de ti en este momento				✓

2.2 Requisitos

Público objetivo

Aunque cualquiera puede acceder a esta aplicación, ésta se orienta al mundo musical. A priori podemos diferenciar dos perfiles genéricos de usuarios dentro de MusicIn. El primer tipo de perfil es el de **músico**. Definimos como músico a todas aquellas personas cuya actividad principal consista en tocar algún instrumento, estos además pueden ser de cualquier edad y sexo. Su motivación para usar esta app será el de compartir sus gustos musicales con el resto de usuarios, encontrar oportunidades en las que poder participar como músicos y crear y enterarse de eventos a los que asistir.

Por otra parte, tenemos el segundo tipo de perfil que denominamos como **promotor**. Aquí podemos englobar a todas aquellas personas que están involucradas en el mundo musical, pero desde la parte de la gestión, es decir profesiones tales como un productor, promotor o director. Al igual que en el perfil de músico, el sexo y la edad son aspectos indiferentes. Su principal interés para usar MusicIn será el de encontrar a músicos que se adapten a sus necesidades y poder contratar sus servicios. Como aspecto secundario, la utilizarán para compartir sus gustos musicales y crear y encontrar eventos.

Requisitos funcionales

A continuación, se describen los requisitos funcionales de la aplicación:

- El usuario podrá crear una cuenta en la aplicación para poder acceder a su contenido. Si el usuario utiliza Facebook o Google para crearse una cuenta, se le cogerá la información dada por estos proveedores y se le pedirá de forma adicional el rol de usuario.
- El sistema permitirá acceder a los usuarios previamente registrados a través de su correo electrónico, su cuenta de Facebook o su cuenta de Gmail.
- El usuario podrá generar comentarios que se integrarán en el timeline de mensajes general y de su perfil. Estos mensajes podrán mostrar contenido enriquecido aprovechando las funcionalidades de la web semántica. Sólo se permitirá agregar texto.
- Se podrá buscar y enviar solicitudes de amistad a usuarios de la aplicación. La solicitud llegará en forma de notificación push.
- Al finalizar de crear un evento, éste se agregará a la lista de eventos. Para poder completar la creación de un evento será necesario rellenar todos los campos del formulario.
- Al crear una oferta, ésta se agregará al listado de ofertas. Para poder completar la creación de una oferta será necesario rellenar todos los campos del formulario.
- Al visualizar los eventos cerca, el sistema localizará al usuario mediante el GPS de su móvil y le mostrará el listado de los eventos que se encuentren a una distancia de 500 metros.

Requisitos no funcionales

A continuación, se describen los requisitos no funcionales de la aplicación:

- Toda funcionalidad del sistema debe responder al usuario en menos de 3 segundos.
- El sistema debe tener una disponibilidad de más del 90%.
- El desarrollo del Frontend del producto se realizará utilizando el framework Ionic 2.
- El desarrollo del Backend del producto se realizará utilizando Firebase.
- La aplicación será compatible con dispositivos móviles Android e iOS.
- El sistema debe asegurar que los datos estén protegidos del acceso no autorizado.

Mockups de la aplicación

A continuación, se incluyen los diferentes mockups de MusicIn:

Figura 1.4 Login

Figura 1.5 Crear Cuenta

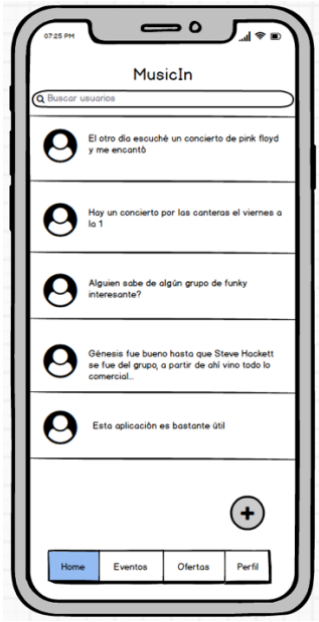


Figura 1.6 Pestaña Principal

Figura 1.7 Pestaña de eventos

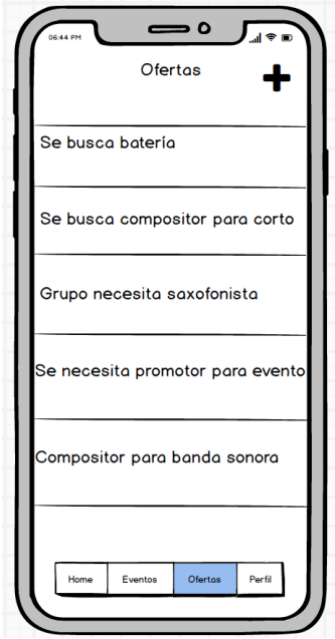


Figura 1.8 Pestaña de ofertas

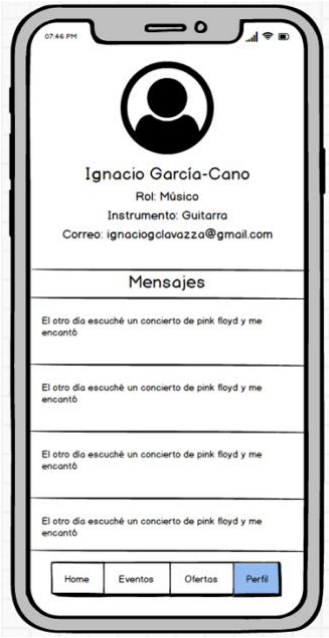


Figura 1.9 Pestaña del perfil

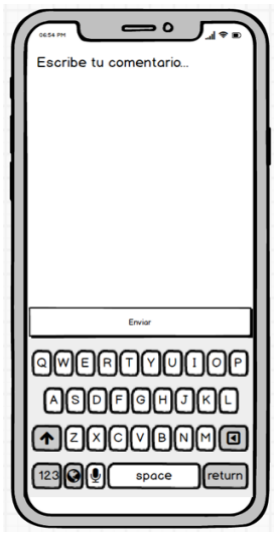


Figura 1.10 Crear comentario

Figura 1.11 Crear oferta

Figura 1.12 Crear evento

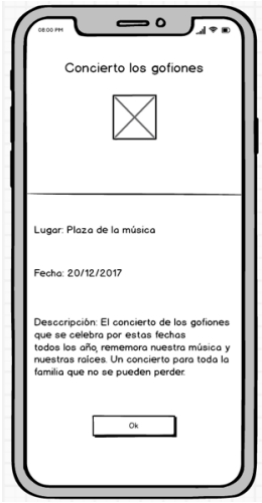


Figura 1.13 Información de un evento

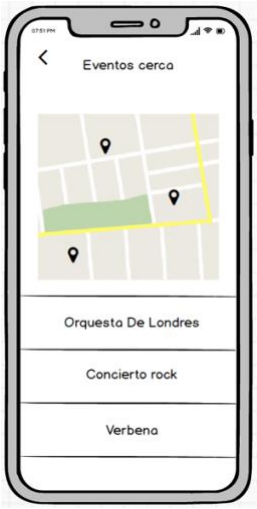


Figura 1.14 Eventos cerca

Figura 1.15 Listado de usuarios

Aunque sólo se han incluido los mockups, la paleta de colores que se utilizará será de colores análogos.

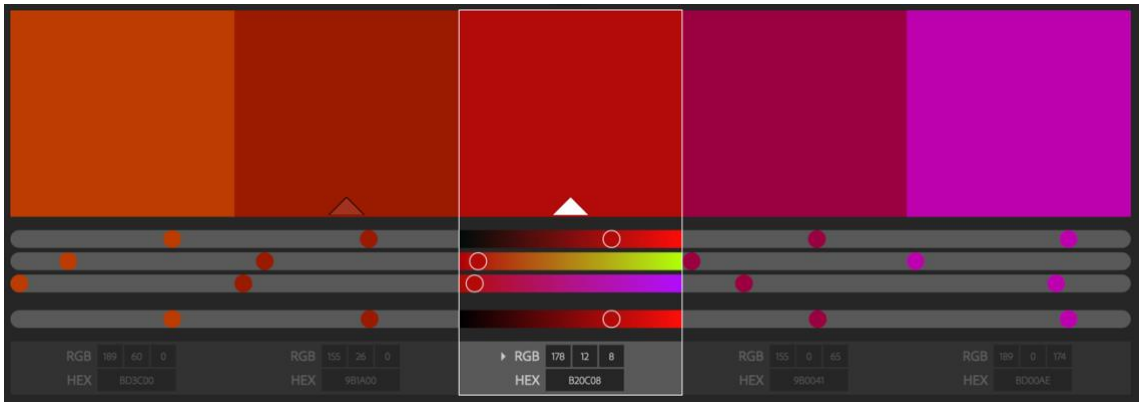


Figura 1.16 Paleta de colores análogos

Como se muestra en la figura 1.16, para el diseño de la aplicación se utilizará una paleta de colores análogos teniendo como base el color rojo, en concreto el color FF2A25 en hexadecimal. Las aplicaciones analizadas en apartados anteriores no muestran ninguna preferencia por un color específico, que se identifique en mayor medida con los músicos. La elección del color rojo como base, viene dado por su significado. El color rojo representa una personalidad enérgica e independiente que prefiere optar por trabajos que le permitan ser autónomo. Además, el rojo no se encuentra entre los colores que menos gustan tanto a hombres como mujeres. Aunque el color azul gane en cuestión de porcentaje de gusto y sea una buena opción, se evita parecerse a otras redes sociales generales que lo utilizan, tales como LinkedIn o Facebook y optar por algo más atrevido.

2.3 Modelo de negocio

Aunque el producto en su prototipo no contempla ningún tipo de ingresos, en siguientes fases cuando se encuentre en explotación, se agregarán nuevas funciones acompañadas de un modelo de negocio detallado a continuación.

Estudiando el modelo de ingresos de las aplicaciones descritas en esta memoria, podemos aplicar para MusicIn un sistema de suscripciones y un sistema de pagos por anuncios. Como se ha mencionado anteriormente, los usuarios pueden crear eventos y ofertas dentro de la aplicación, en un principio serán gratuitas y no habrá ningún tipo de limitación a la hora de publicarlas. Posteriormente se cobrará individualmente por cada **oferta** publicada dentro de la app que poseerá un precio variable. Éste se calculará en función del alcance que se quiere llegar, teniendo en cuenta la edad, el sexo y el rol del usuario. Por otra parte, en un futuro se añadirán opciones para que todos sean capaces de subir archivos multimedia a la plataforma. Para ello intervendrá un tipo de suscripción que permitirá subir tanto audio y vídeo en cualquier tipo de formato, para compartirlo posteriormente en MusicIn. El precio de la suscripción se calculará en base a un estudio que se hará de la aplicación una vez tenga tracción en el mercado.

2.4 Normativa y legislación

Debido al tipo de contenido de la aplicación, será necesario seguir dos tipos de normativa:

- Para la información básica de cada usuario, que en este caso son el correo y el nombre del usuario en MusicIn se seguirá la normativa vigente para el almacenamiento y la protección de datos, es decir, lo establecido por la ley orgánica 15/1999 de 13 de diciembre de Protección de Datos de Carácter Personal o L.O.P.D.
- Para el contenido musical disponible en la aplicación se seguirá el Real Decreto Legislativo 1/1996, de 12 de abril, por el que se aprueba el texto refundido de la Ley de Propiedad Intelectual, regularizando, aclarando y armonizando las disposiciones legales vigentes sobre la materia.

Para versiones posteriores del producto, se incluirá la posibilidad de subir archivos de sonido con obras propias, por lo que la legislación en referencia a ese tema afecta a estos archivos.

En caso de que algún usuario considere que existe material en MusicIn, que infringe la ley anteriormente mencionada, podrá denunciar dicho contenido a través de la aplicación para que se estudie su caso y sea resuelto con la mayor antelación posible.

Como el principal modelo de MusicIn no se basa en la recopilación de datos, en versiones posteriores se implementarán sistemas de eliminación de contenido para aquellos usuarios que quieran eliminar tanto su información personal como todo el contenido que hayan compartido a través de la aplicación a lo largo de su uso.

3 Iteraciones

3.1 Alcance de la implementación

Para la validación de las ideas del análisis y de diseño de MusicIn, se realizará un producto mínimo viable. Además, esto permitirá adquirir un mayor conocimiento en las tecnologías utilizadas para el desarrollo de aplicaciones híbridas, en cuyo caso son Ionic 3, que hace uso de Angular 4 para el Frontend y Firebase para el Backend.

Todas las características implementadas en este prototipo se listan a continuación.

- Registrar a un usuario.
- Loguear a un usuario.
- Buscar amigos.
- Ver mi listado de amigos.
- Agregar amigos.
- Ver mis solicitudes pendientes.
- Aceptar o eliminar solicitudes de amigos.
- Ver las publicaciones de amigos.
- Escribir publicaciones.
- Comentar publicaciones.
- Añadir y quitar "me gusta" a las publicaciones.
- Incluir vídeos de YouTube en las publicaciones.

3.2 Iteración 1

En esta iteración se establecerán las bases de la aplicación y por lo tanto se incluirán las características que cualquier usuario puede realizar, indistintamente del rol que éstos posean.

Requisitos

Para esta iteración la lista de características a implementar es la siguiente:

- Escribir publicaciones.
- Incluir vídeos de YouTube en las publicaciones.
- Comentar publicaciones.
- Añadir y quitar "me gusta" a las publicaciones.
- Buscar amigos.
- Ver mi listado de amigos.
- Agregar amigos.

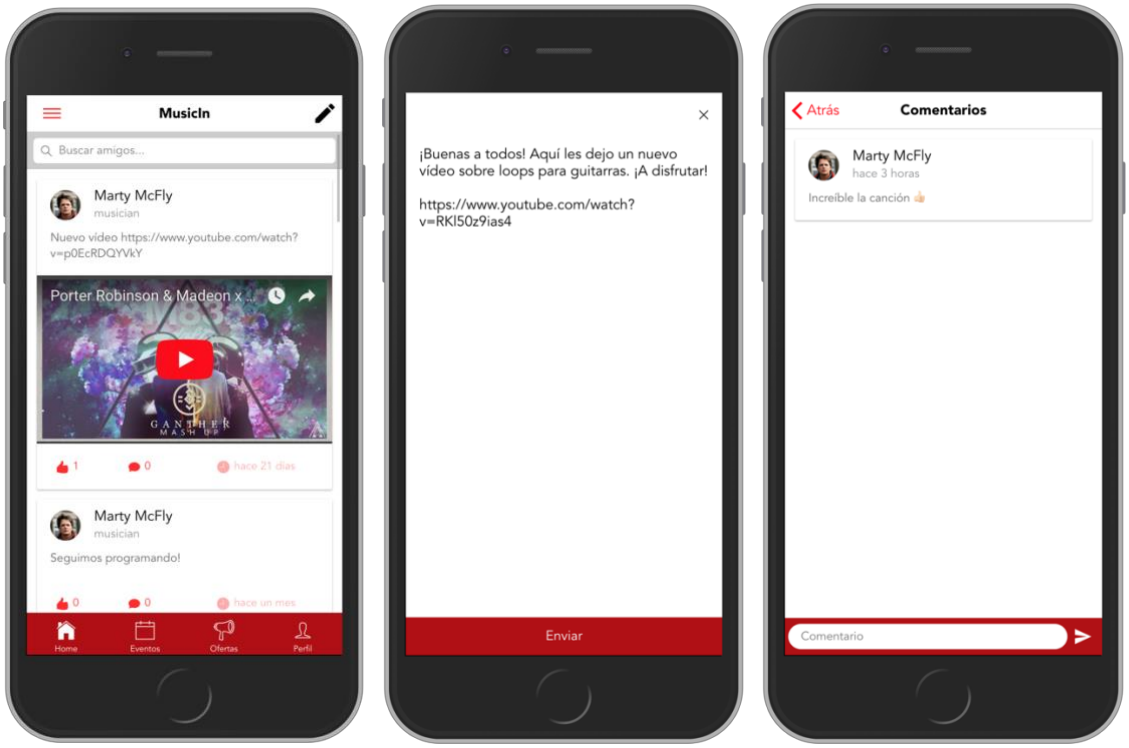
- Ver mis solicitudes pendientes.
- Aceptar o eliminar solicitudes de amigos.
- Ver las publicaciones de amigos.

Diagrama de casos de uso

Figura 2.0 Diagrama de casos de uso de un usuario en MusicIn

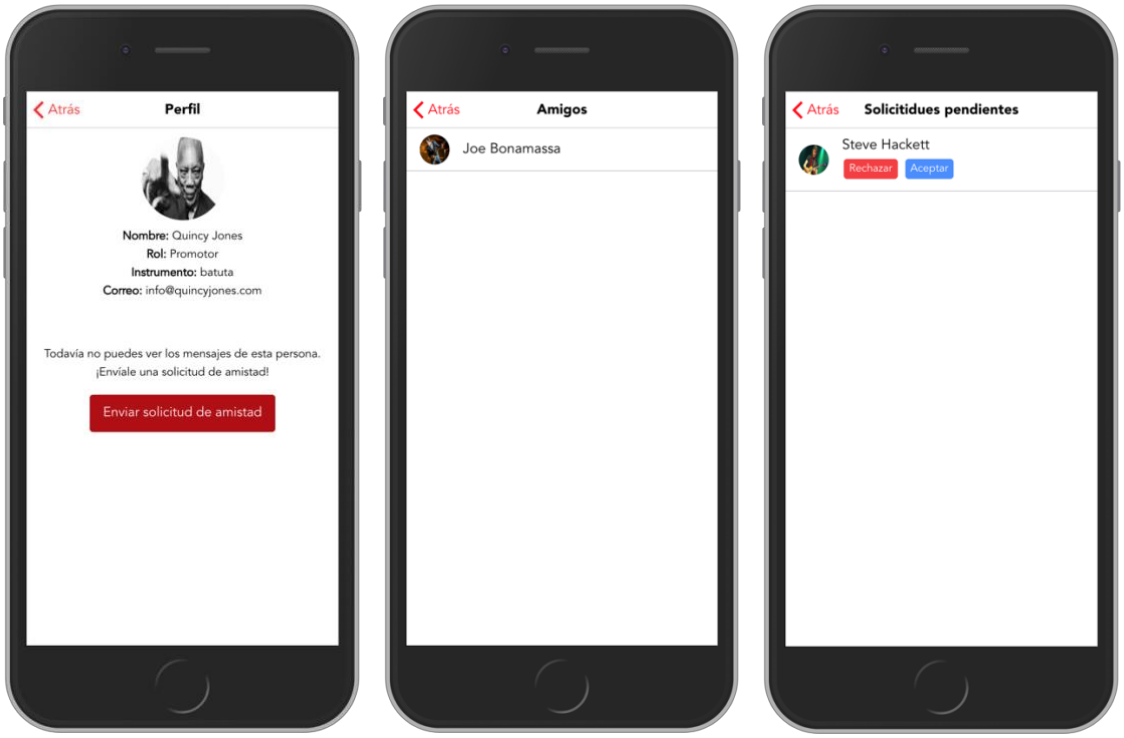
Diseño

En este apartado se incluyen los diseños con las implementaciones mencionadas en apartados anteriores.



Figuras 2.1, 2.2 y 2.3 Donde se pueden ver la pantalla del home, la pantalla de escribir publicación y la de comentarios de un mensaje respectivamente.

Como se ve en la figura 2.1, se pueden ver las publicaciones que han hecho mis amigos dentro de la aplicación, al igual que los vídeos extraídos de YouTube cuando se incluye una url del mismo en una publicación. También podemos añadir o quitar "me gusta" y acceder a los comentarios del mensaje. Por último, en el buscador incluido en la misma figura poder buscar a gente para agregarlos como amigos.



Figuras 2.4, 2.5 y 2.6 Donde podemos ver las pantallas de perfil, listado de amigos y listado de solicitudes pendientes respectivamente.

Diseño arquitectónico

Dado que el framework utilizado para el desarrollo de esta aplicación utiliza un **patrón de arquitectura basado en componentes**, todas las iteraciones incluirán explicaciones similares en este apartado. En angular 2, se pasó de utilizar una arquitectura MVC (modelo-vista-controlador) a una arquitectura basada en componentes. Éstos son clases de TypeScript que incluyen tanto la lógica como la vista. Por lo tanto, para la realización de esta iteración se ha tenido que crear al menos un componente por cada vista reflejada en el apartado de diseño. Además, la lógica del componente trabaja con datos que vienen de diferentes **proveedores** existentes en la aplicación. Los creados para esta iteración son:

- **ProfileProvider**: encargado de pedir y gestionar toda la información que tiene que ver con el usuario que actualmente está usando la aplicación.
- **ToastProvider**: encargado de generar mensajes tanto de confirmación como de error en formato de 'toast'.
- **MessagesProvider**: encargado de gestionar todo lo relacionado con los mensajes y comentarios de la aplicación.
- **LikesProvider**: encargado de tratar todo lo referente a los "me gusta" de las publicaciones.
- **FriendsProvider**: encargado de administrar las solicitudes de amistad y de gestionar los amigos de un usuario.

Los principales componentes que hacen uso de los proveedores anteriores son:

- **HomePage**: componente que hace uso de los cuatro primeros proveedores descritos en el punto anterior y donde se encuentran el buscador de amigos y el listado de todas las publicaciones de los amigos del usuario actual, con todas las posibles interacciones que éstas tienen.
- **WriteMessageComponent**: componente que se encarga de recoger un mensaje y de pasárselo al MessagesProvider para generar una nueva publicación.
- **MessageCommentsPage**: componente que muestra el listado de comentarios de una publicación obtenidos del MessagesProvider.
- **ProfilePage** y **FriendProfilePage**: componentes que muestran el perfil de un usuario con sus mensajes. La principal diferencia radica en que el segundo evita que un usuario vea las publicaciones si no es amigo. Dicha acción puede hacerse en el mismo componente.
- **FriendsPage**: componente que muestra el listado de amigos actual.
- **PendingRequestsPage**: componente que obtiene y muestra el listado de solicitudes de amistad pendientes a través del proveedor FriendsProvider.

Estructura de la base de datos

Como se comentó en puntos anteriores, para el desarrollo del backend se utiliza RealtimeDatabase, que es una base de datos NoSQL alojada en la nube de Google. Los datos se almacenan en formato JSON y se sincronizan en tiempo real con cada cliente conectado.

Para un usuario se ha planteado una estructura como la que se muestra en la figura 2.7

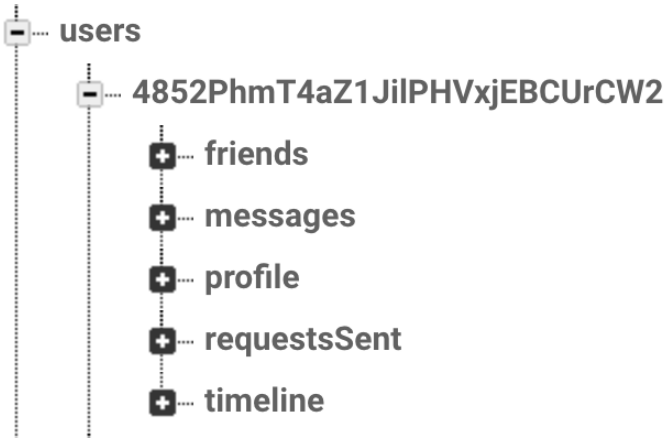


Figura 2.7 Representación de la estructura de un usuario.

En la rama **friends** se incluye una lista de objetos con el id y el nombre de los usuarios que se tienen agregados como amigos.

En la rama **messages** se incluye una lista de mensajes que cumplen con la estructura mostrada en la siguiente imagen.

Figura 2.8 Representación de la estructura de un mensaje

Cada mensaje contiene tres ramas, que incluyen:

- Los comentarios realizados dentro del mensaje.
- El contenido publicado en el mensaje y toda la información que se muestra en una tarjeta.
- Una lista con los ids de los usuarios que han dado "me gusta" al mensaje.

En la rama **profile** se almacena la información que se muestra en la vista de perfil, cuyos datos son, la url de la imagen de perfil, el nombre, el rol, el instrumento favorito, el email y el id del usuario.

En la rama **requestSent** se incluye una lista con los ids de los diferentes usuarios a los que se les ha enviado una solicitud de amistad.

Por último, se encuentra la rama **timeline**, donde se almacena una lista de objetos que incluyen los ids tanto de los usuarios como de los mensajes que figuran en el timeline. Esto se realiza así para evitar duplicar el objeto que ya figura en la rama de **messages** y aumentar de forma innecesaria el tamaño de la base de datos.

Test

Todas las pruebas realizadas para esta iteración se han realizado compilando la aplicación en dispositivos Android, simulando el flujo normal que haría un usuario. Para ello se han realizado las siguientes acciones para verificar el correcto funcionamiento:

- Crear nuevas publicaciones, con y sin vídeos.
- Comentar en publicaciones realizadas.
- Simular la llegada de peticiones de amistad, visualizarlas y aceptarlas o rechazarlas.
- Dar y quitar "me gusta" de un mismo mensaje.
- Buscar a otros usuarios en la app y enviarles peticiones de amistad, que quedan registradas en la base de datos.

Por último, cabe decir que todas las acciones descritas se realizaron más de una vez, para probar que el matar el proceso de la aplicación y reiniciarlo no implicase errores.

3.3 Iteración 2

En esta iteración se establecen los modelos de las ofertas y los eventos que se pueden encontrar en MusicIn.

Requisitos

Los requisitos definidos para esta iteración son:

- Crear eventos
- Visualizar eventos
- Crear ofertas
- Visualizar ofertas
- Visualizar eventos en un mapa

Diagrama de casos de uso

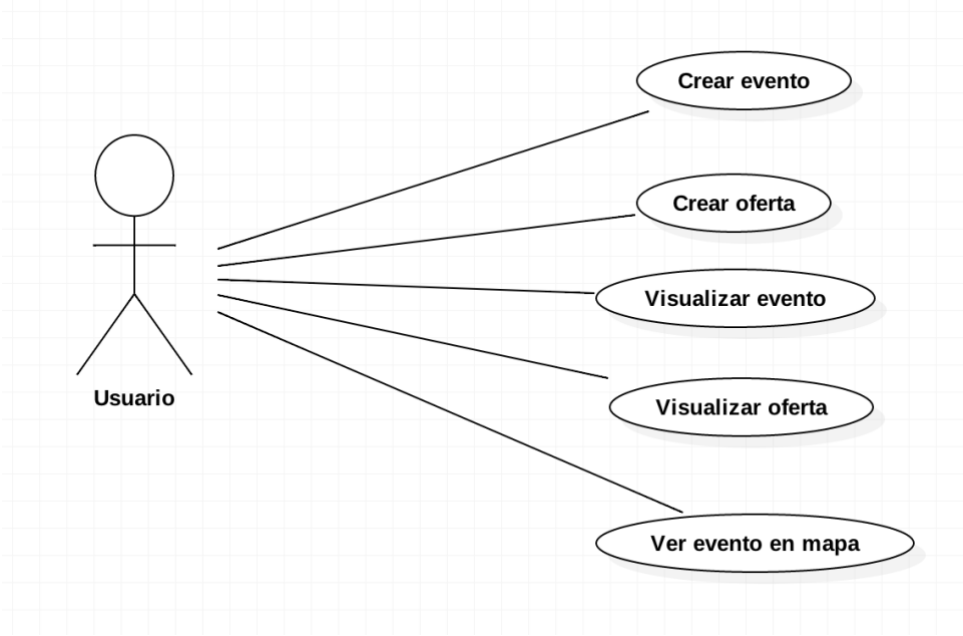
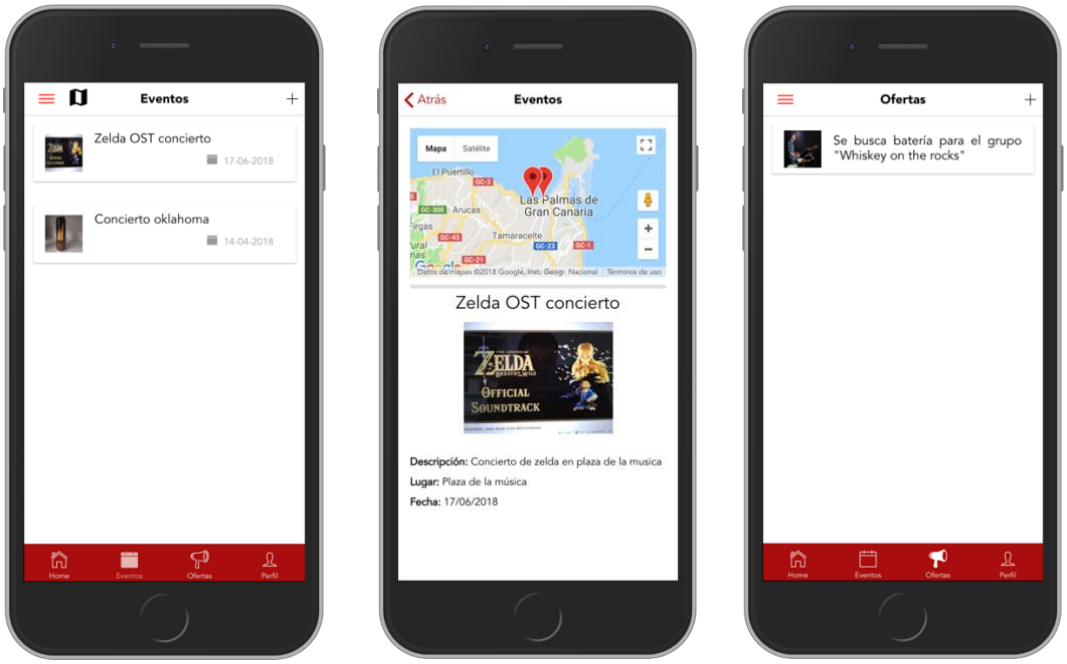


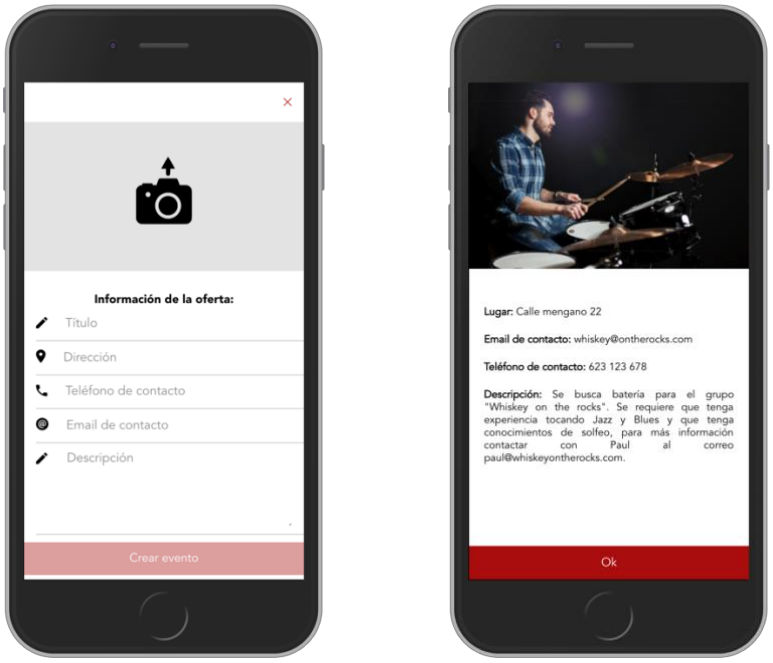
Figura 2.9 Diagrama de casos de uso de ofertas y eventos

Diseño

En este apartado se incluyen las pantallas de la implementación de los casos de uso planteados en el apartado anterior.



Figuras 2.10, 2.11 y 2.12 Donde se muestran el listado de eventos, la visualización de los eventos cerca de mi zona y el listado de ofertas disponible.



Figuras 2.13 y 2.14 Vista de la creación de una oferta y la información detallada.

Aunque no se hayan incluido capturas de pantalla, el formato de creación de un evento y la visualización de los detalles son iguales que en las ofertas, salvo por algunos campos de información.

Diseño arquitectónico

Como ya se ha explicado en párrafos anteriores, la estructura utilizada en la aplicación es una **basada en componentes** y, por lo tanto, al igual que en anteriores iteraciones, se utilizan componentes para la lógica y renderización de la vista y proveedores para la obtención de los datos. Para esta iteración, los componentes y proveedores más importantes han sido los siguientes:

- **EventsPage** y **OffersPage**: componentes encargados de mostrar los eventos y ofertas que hay actualmente en la aplicación.
- **EventDetailsComponent** y **OfferDetailsComponent**: componentes que muestran los detalles de un evento y de una oferta.
- **CreateEventComponent** y **CreateOfferComponent**: componentes que se encargan de crear un evento y una oferta.
- **EventsMapPage**: componente que se encarga de mostrar en un mapa todos los eventos que hay cerca de mi, además de su información.
- **EventsManagerProvider** y **OffersManagerProvider**: proveedores que se encargan de la gestión de la creación y obtención de los eventos y ofertas.

Estructura de la base de datos

Para esta iteración se ha planteado una estructura para los eventos y otra para las ofertas, tal y como se ve en las figuras 2.15 y 2.16.

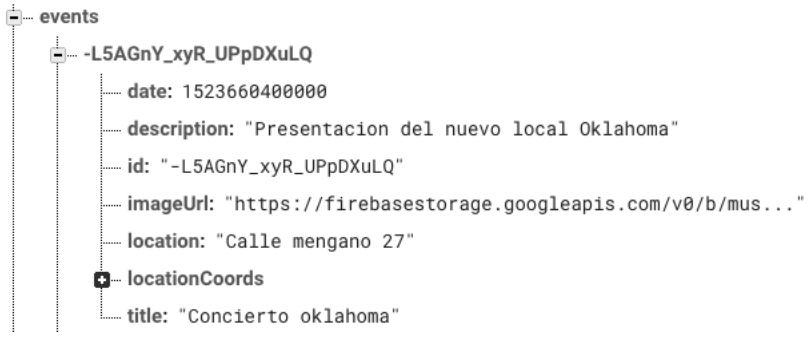


Figura 2.15 Representación de la estructura de un evento en la base de datos

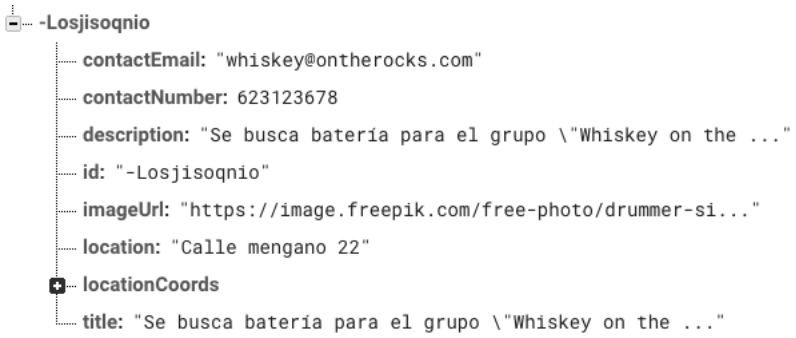


Figura 2.16 Representación de la estructura de una oferta en la base de datos

Tanto los eventos como las ofertas se encuentran en ramas separadas dentro de la base de datos y cada una se identifica con una clave única generada por firebase. El objeto **locationCoords** contiene dos atributos, que son la latitud y la longitud.

Test

Para probar que la implementación de los casos de uso es correcta, se ha compilado la aplicación en dispositivos móviles y se ha realizado el flujo que haría un usuario normal, probando las siguientes acciones:

- Crear un evento y una oferta.
- Acceder a las vistas de eventos y ofertas visualizar los mismos.
- Ver la información detallada de una oferta y un evento.
- Visualizar el mapa correctamente con los indicadores de los eventos.
- Pulsar en uno de los iconos de evento y ver la información detallada.

3.4 Iteración 3

En esta iteración se define e implementa la gestión de usuarios y el rol del promotor en MusicIn.

Requisitos

Los requisitos necesarios por implementar en esta iteración son:

- Crear cuenta.
- Acceder a mi cuenta.

Diagrama de casos de uso

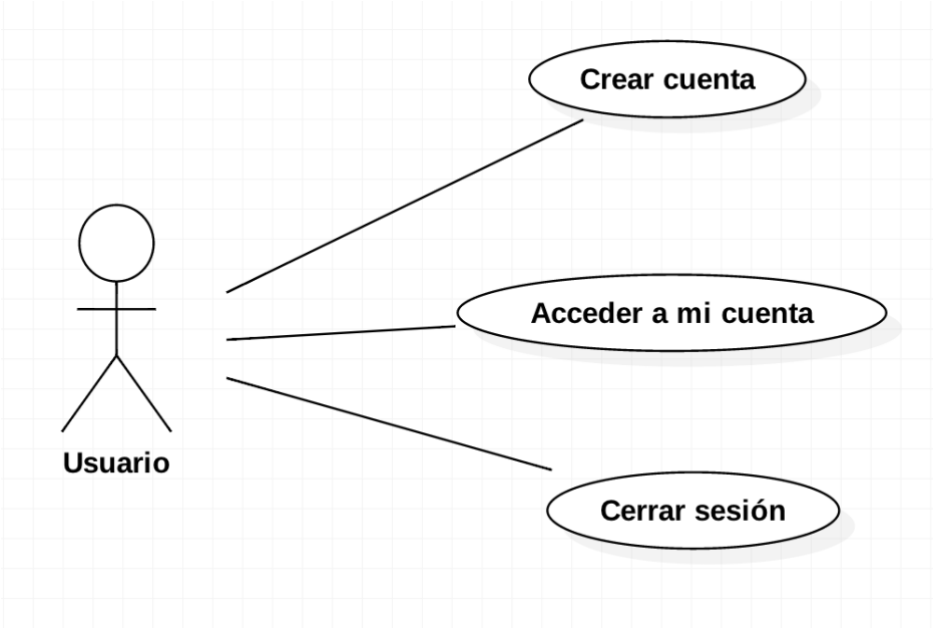
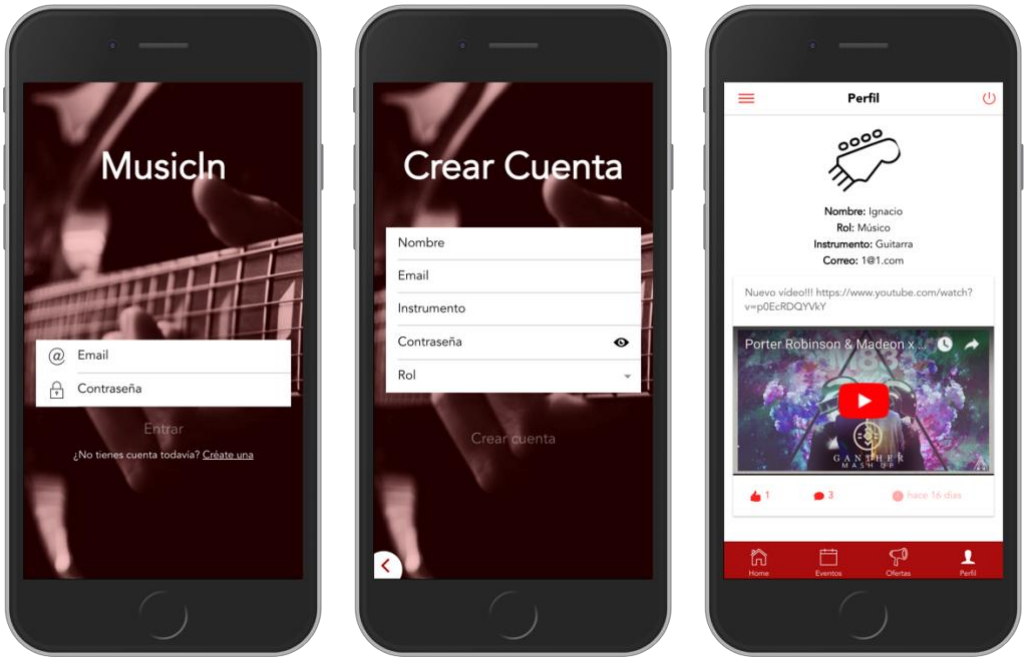


Figura 2.17 Diagrama de casos de uso de la gestión de usuarios.

Diseño

En este apartado se incluyen las pantallas de la implementación de los casos de uso planteados en el apartado anterior.



Figuras 2.18, 2.19 y 2.20 Pantallas de login, creación de cuenta y de perfil, en donde puedes salir de tu cuenta.

Diseño arquitectónico

Como ya se ha explicado en párrafos anteriores, la estructura utilizada en la aplicación es una **basada en componentes** y, por lo tanto, al igual que en anteriores iteraciones, se utilizan componentes para la lógica y renderización de la vista y proveedores para la obtención de los datos. Para esta iteración, los componentes y proveedores más importantes han sido los siguientes:

- **LoginPage**: componente que se encarga de mostrar la pantalla del login y de enviar la información al proveedor para loguear un usuario.
- **CreateAccountPage**: componente que muestra el formulario de registro y envía la información al proveedor para crear un usuario.
- **ProfileProvider**: se han implementado nuevos métodos para el login, el logout y la creación de usuarios.

Estructura de la base de datos

La estructura de los usuarios para este apartado es la que se planteó para la primera iteración, pero en esta ocasión, se dispone además de una tabla de autenticación de usuarios provista por firebase.

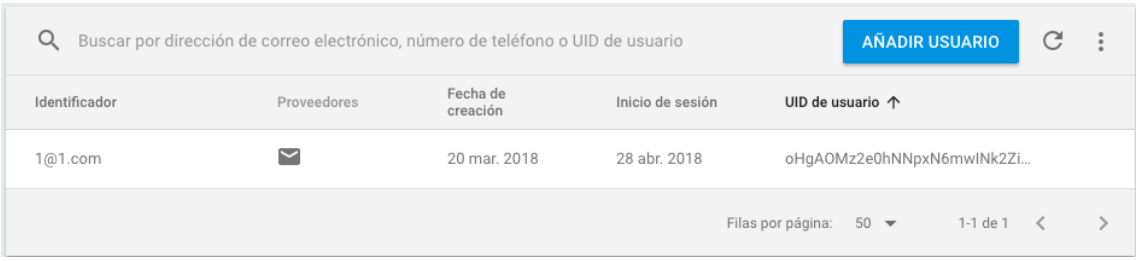


Figura 2.21 Tabla de autenticación de usuarios.

Como se muestra en la figura 2.21, se indica para cada usuario el método de acceso utilizado, las fechas del último inicio de sesión y de la creación de la cuenta y el UID del mismo. Éste último dato es el que se utiliza posteriormente para acceder a toda la información del usuario.

Test

Para probar que la implementación de los casos de uso es correcta, se ha compilado la aplicación en dispositivos móviles y se ha realizado el flujo que haría un usuario normal, probando las siguientes acciones:

- Acceder con una cuenta existente.
- Crear una cuenta nueva y acceder a la aplicación.
- Cerrar sesión.

3.5 Iteración 4

En esta iteración se define e implementa el sistema de mensajería instantánea entre usuarios que sean amigos de MusicIn.

Requisitos

Los requisitos definidos para esta iteración son:

- Iniciar una conversación con otro usuario.
- Eliminar conversación.
- Enviar mensajes a otro usuario.
- Recibir mensajes de otro usuario.

Diagrama de casos de uso

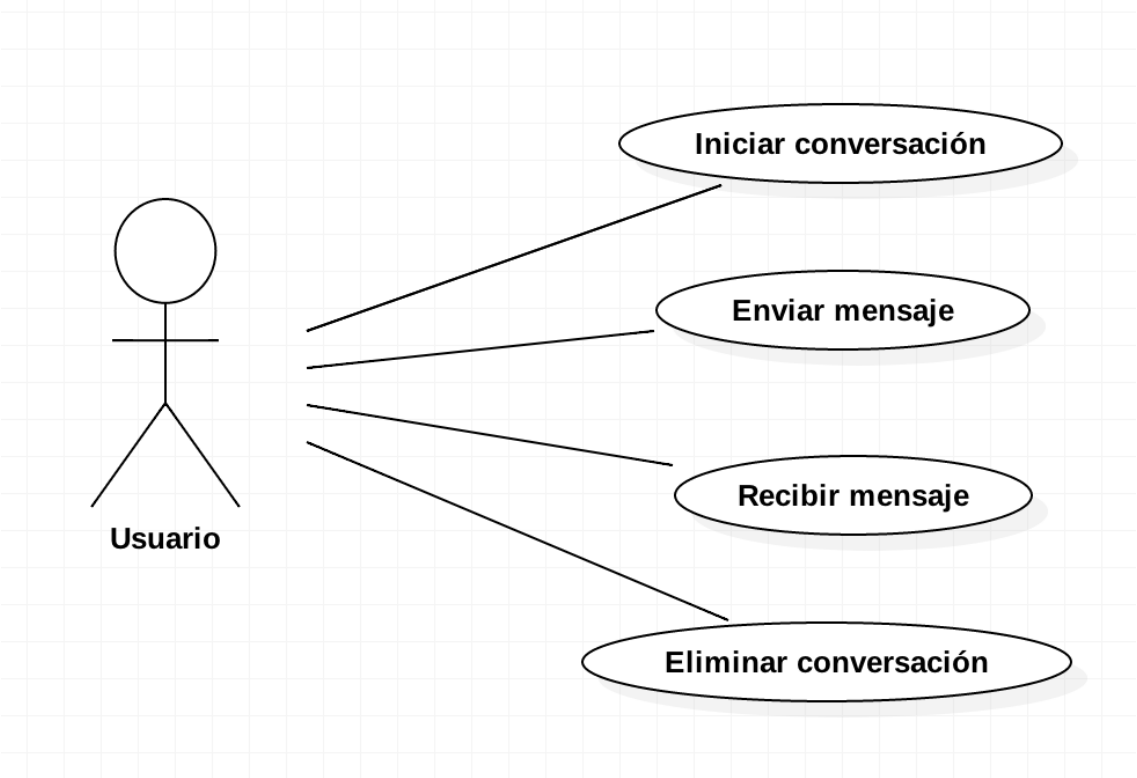
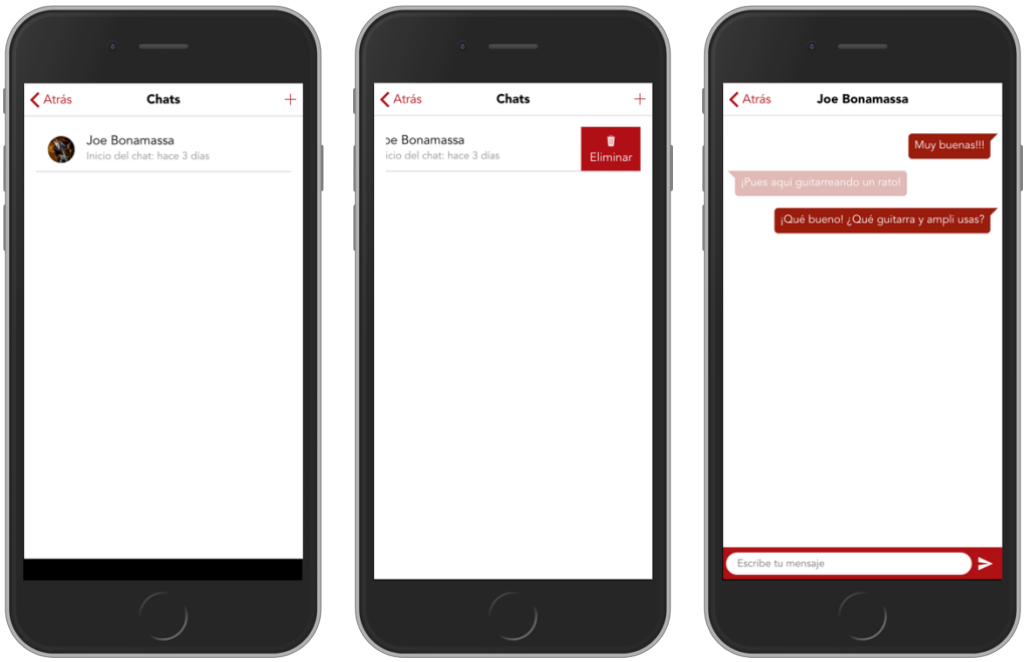


Figura 2.22 Diagrama de casos de uso del servicio de mensajería instantánea.

Diseño

En este apartado se incluyen las pantallas de la implementación de los casos de uso planteados en el apartado anterior.



Figuras 2.23, 2.24 y 2.25 Pantallas del listado de chats, opción de eliminar del chat y la vista con los mensajes de un chat.

Diseño arquitectónico

Como ya se ha explicado en párrafos anteriores, la estructura utilizada en la aplicación es una **basada en componentes** y, por lo tanto, al igual que en anteriores iteraciones, se utilizan componentes para la lógica y renderización de la vista y proveedores para la obtención de los datos. Para esta iteración, los componentes y proveedores más importantes han sido los siguientes:

- **ChatsPage**: componente que se encarga de mostrar el listado de chats a través de la información que le llega por el proveedor.
- **ChatPage**: componente que se encarga de visualizar la conversación entre dos usuarios y de enviar la información al proveedor para que la mensajería sea instantánea.
- **ChatsManagerProvider**: proveedor que se encarga de gestionar la información de los chats de la aplicación.

Estructura de la base de datos

Para la implementación de la estructura en esta iteración, se ha pensado en el desarrollo de "salas de chats", donde se almacenan los mensajes de la conversación y los usuarios poseen las claves para acceder a dichas salas.

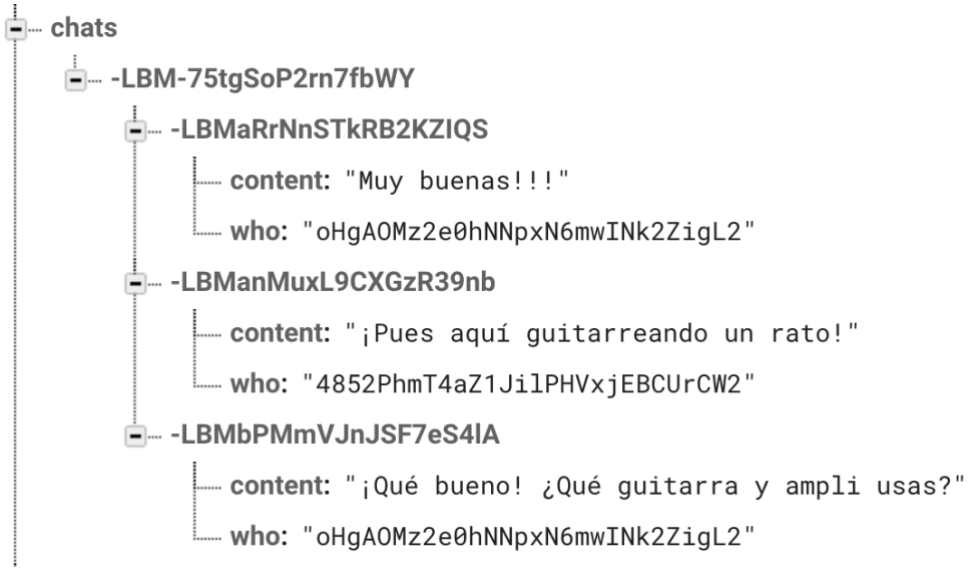


Figura 2.26 Esquema en la base de datos de un chat.

Como se muestra en la figura 2.26, existe un árbol de chats que solamente incluye los mensajes con el contenido y el id de la persona que lo escribió.

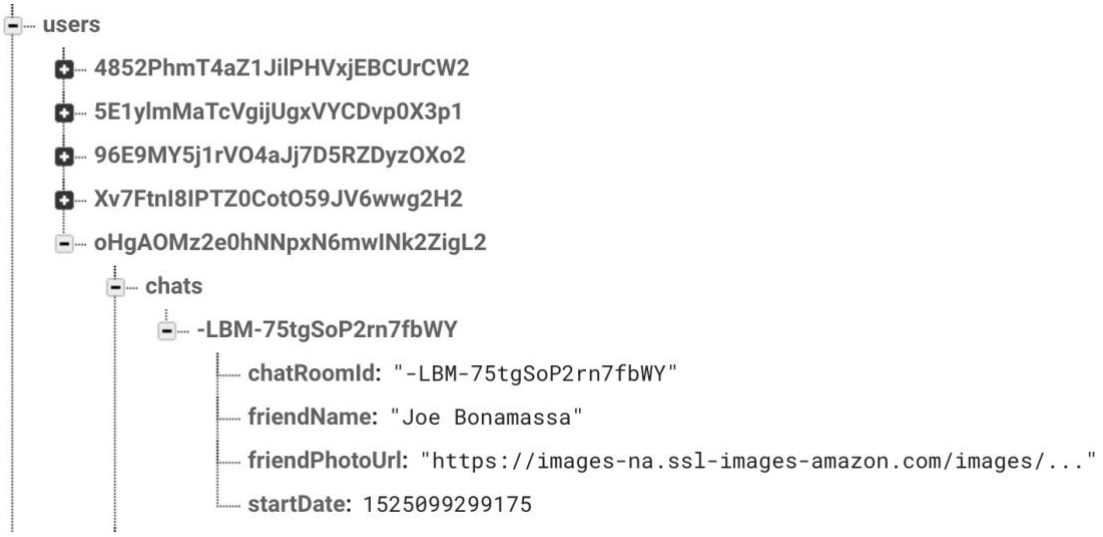


Figura 2.27 Representación de la información de una sala de chat en un usuario

Por otra parte, y como se muestra en la figura 2.27, a cada usuario se le asigna una lista con los ids de cada sala de chat y la información básica de cada una.

Test

Para probar que la implementación de los casos de uso es correcta, se ha compilado la aplicación en dos dispositivos móviles, se ha accedido a dos cuentas diferentes y se ha realizado el flujo que haría un usuario normal, probando las siguientes acciones:

- Ambos usuarios han iniciado una conversación.
- Ambos usuarios han enviado mensajes.
- Ambos usuarios han recibido mensajes.
- Ambos usuarios han eliminado una conversación.

3.6 Iteración 5

En esta iteración, se define la utilización de la web semántica para el contenido etiquetado en cada mensaje de la aplicación, con la finalidad de obtener información sobre diferentes artistas sin necesidad de disponer de dicha información en una base de datos propia. Para este proyecto se ha utilizado la información disponible en la ***DBPEDIA*** y la forma de acceder a su información es a través del lenguaje de consultas ***SPARQL***.

SPARQL hace consultas a grafos ***RDF***, que es un modelo de datos de grafos formado por una combinación de tres elementos (sujeto, predicado y objeto). Estos tres elementos son representados por URIs los cuales pueden ser abreviados como nombres prefijados. Además, los objetos pueden ser literales, es decir, enteros, strings, etc.

Como se mencionó anteriormente, las consultas de SPAQRL se hacen contra RDF datasets los cuales están formados por grafos RDF. Un endpoint de SPARQL acepta consultas y las devuelve a través del protocolo HTTP. Aquí podemos diferenciar entre dos tipos de endpoint:

- **Genéricos**: éstos preguntarán a cualquier "Web RDF"
- **Específicos**: éstos están unidos para preguntar por conjuntos de datos particulares

Los resultados de las consultados realizadas a un endpoint SPARQL pueden ser devueltas o renderizadas en diferentes formatos como son:

- **XML**: SPARQL especifica un vocabulario XML para devolver tablas de resultados.
- **JSON**: Una conversión del vocabulario XML a JSON.
- **CSV/TSV**: Representaciones textuales simples, ideales para importarlas a hojas de cálculo.
- **RDF**: Ciertos resultados de consultas SPARQL generan respuestas RDF, las cuales pueden ser serializadas de diferentes formas (RDF/XML, N-Triples, Turtle, etc.)
- **HTML**: Cuando se usa una forma interactiva de trabajar con consultas SPARQL. A menudo implementado aplicando transformaciones XSL a resultados XML.

Ejemplos de sintaxis:

URIs:

- <http://example.com/resource>
- prefix:name

Literales:

- "string plana"
- "13.4"^^xsd:float
- "string con idioma"@en

Tupla de tres elementos:

- prefix:subject other_prefix:predicate "object"

Estructura de una consulta SPARQL:

```
# prefix declarations
PREFIX foo: <http://example.com/resources/>
...
# dataset definition
FROM ...
# result clause
SELECT ...
# query pattern
WHERE {
    ...
}
# query modifiers
ORDER BY ...
```

Figura 2.28 Representación de la estructura de una consulta de SPARQL

Requisitos

Los requisitos para esta iteración son:

- Etiquetar un músico en un mensaje del timeline
- Buscar información sobre un músico a través de la web semántica

Diagramas de casos de uso

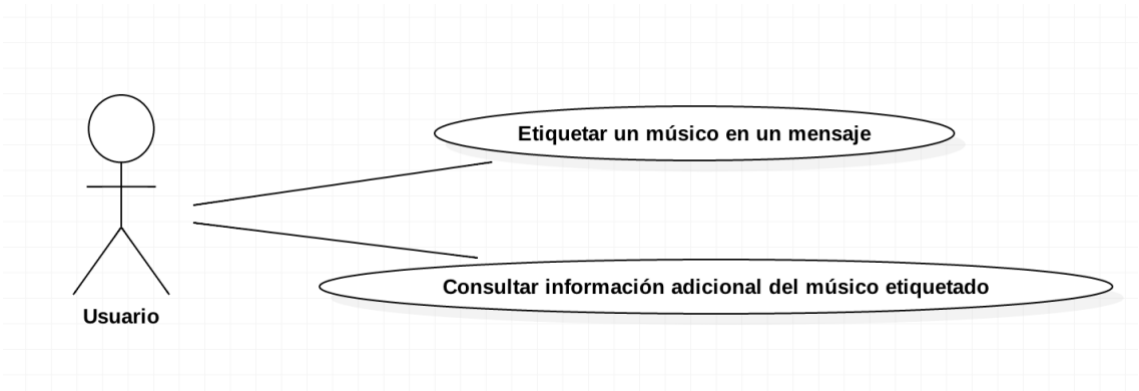
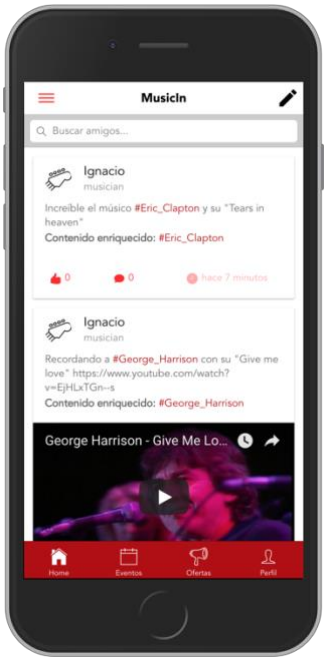


Figura 2.29 Diagrama de casos de uso de la consulta de información utilizando la web semántica.

Diseño

En este apartado se incluyen las pantallas de la implementación de los casos de uso planteados en el apartado anterior.



Figuras 2.30 y 2.31 Pantallas donde se muestran a un artista etiquetado con un "#" en un mensaje del timeline y su biografía utilizando las ventajas de la web semántica.

Diseño arquitectónico

Como ya se ha explicado en párrafos anteriores, la estructura utilizada en la aplicación es una **basada en componentes** y, por lo tanto, al igual que en anteriores iteraciones, se utilizan componentes para la lógica y renderización de la vista y proveedores para la obtención de los datos. Para esta iteración, los componentes y proveedores más importantes han sido los siguientes:

- **HomePage**: componente donde se encuentran el buscador de amigos y el listado de todas las publicaciones de los amigos del usuario actual. En este caso se ha añadido la posibilidad de etiquetar a músicos en los mensajes con el fin de mostrar información adicional.
- **RichContentComponent**: componente que se encarga de recibir el nombre del músico a buscar y de pasárselo a un proveedor para obtener la información de este. Además renderiza los datos recibidos.
- **MessagesProvider**: proveedor encargado de realizar la gestión de los mensajes de la aplicación y de buscar información adicional sobre músicos etiquetados en los mensajes. Además, se encarga de realizar el etiquetado de los músicos en cada mensaje.

Estructura de la base de datos

Para esta iteración, se han aprovechado las ventajas que aporta la web semántica y poder prescindir de una estructura propia. En este caso se ha utilizado la dbpedia, en la que cada entidad de información conforma un grafo que a su vez está relacionado con multitud de grafos. La forma de consultar la información disponible en estos grafos es a través del lenguaje de consultas Sparql, explicado al inicio de esta iteración.

Test

Para probar que la implementación de los casos de uso es correcta, se ha compilado la aplicación en un dispositivo móvil y se han probando las siguientes acciones:

- Crear un mensaje con el nombre de un músico etiquetado.
- Seleccionar el contenido etiquetado del mensaje.
- Visualizar la biografía del músico etiquetado.

4 Tecnologías

Para la realización de este proyecto se han utilizado diferentes tecnologías, cada una centrada en diferentes aspectos de la aplicación. Para el frontend se han utilizado:

- Angular 5
- TypeScript
- Ionic 3

Angular 5

Según la documentación oficial, Angular se define como una plataforma que facilita el desarrollo de aplicaciones web. Angular empezó como un framework MVC de JavaScript en su primera versión de salida en 2010. Desde ese entonces se ha mantenido por Google y el código fuente se encuentra disponible en github. En 2016 Angular pasó de la versión 1 a la 2 y eso no sólo implicó un cambio de número. El código del framework se rescribió desde cero y se pasó a utilizar TypeScript en vez de JavaScript. Además, se dejó de lado la arquitectura MVC para pasar a una arquitectura basada en componentes, donde la lógica, la vista y el estilo se encapsulan en componentes. Este framework suele utilizarse en las llamadas SPA (single page application).

TypeScript

TypeScript es un lenguaje de programación de código abierto desarrollado y mantenido por Microsoft. TypeScript se define como un superset de Javascript, al que le añade tipado y objetos basados en clases. Cabe decir que éste extiende la sintaxis de JavaScript, por lo que admite cualquier código JavaScript.

Ionic 3

Ionic es un sdk de código abierto para el desarrollo de aplicaciones híbridas. Éste se construye sobre Angular y Apache Cordova. La primera versión fue lanzada en 2013 y al igual que Angular, sufrió una transformación significativa en su segunda versión. La gran ventaja de las aplicaciones híbridas es que el código sólo se realiza una vez y sirve tanto para iOS como para Android. Cabe destacar que posee una documentación sobresaliente, debido a la cantidad y la calidad de los ejemplos y explicaciones.

Para el desarrollo del backend se ha utilizado firebase.

Firebase

Firebase se describe como "La plataforma móvil de Google que te ayuda a desarrollar apps de alta calidad con rapidez y hacer crecer tu empresa". Firebase en sus inicios comenzó como una empresa independiente, que ofrecía un servicio de base de datos en la nube en tiempo real. En 2014 fue adquirida por Google y convertida en una plataforma completa, que además de su base de datos en tiempo real, ofrece múltiples servicios, desde notificaciones push o Google analytics a test a/b, reporte de errores, funciones en la nube, etc. Para este proyecto se utilizan:

- Realtime Database
- Firebase Authentication
- Notificaciones push

Firebase Authentication permite autentificar a tus usuarios a través del método tradicional utilizando correo y contraseña y a través de proveedores de terceros como Facebook, Google, Twitter y Github. Todo ello de forma muy sencilla, sin necesidad de realizar la programación de la gestión de los usuarios.

Como éste, es un proyecto que utiliza tecnologías web, se maneja un gran número de dependencias y por lo tanto es necesaria la utilización de un gestor de paquetes. Para ello se ha utilizado npm.

NPM

NPM es el gestor de paquetes y dependencias de NodeJS, el cual se encarga de la instalación de nuevos módulos o librerías que se quieran incluir en el proyecto. Nosotros sólo tenemos que indicar que queremos incluir y el se encarga de todo el resto, ya que algunos casos, un módulo puede requerir de unos cuantos más. Tarea que realizada manualmente puede ser muy engorrosa.

Por último, se ha utilizado un lenguaje de consultas para darle valor añadido a la aplicación y enriquecer su contenido. En este caso se ha utilizado Sparql

Sparql

Sparql es un lenguaje de consultas de grafos RDF utilizado para la web semántica. En este caso se ha utilizado para enriquecer el contenido de los mensajes que se comparten en MusicIn, aportando información adicional sobre músicos, discos, etc. Utilizando la información existente en la dbpedia, que a su vez se nutre de la wikipedia.

5 Acceso al código y despliegue

El código de MusicIn se encuentra disponible en un repositorio público de GitHub con la siguiente url: https://github.com/IgnacioGCL/MusicIn

Para el despliegue del mismo, es necesario disponer de NodeJS e Ionic, un proyecto propio de Firebase y las credenciales de Google Maps para JavaScript desde la consola del proyecto en Google Cloud. Por ello los pasos a seguir son:

1. Acceder a la web de NodeJS y descargar la última versión estable. En caso de tener problemas a la hora de lanzar el código, utilizar la versión 8.7.0 de NodeJS, que es la utilizada para el desarrollo del proyecto.
2. Instalar como dependencia global Ionic con el comando ***npm i -g ionic***. En caso de tener problemas a la hora de lanzar el código, utilizar la versión 3.20.0, que es la utilizada para el desarrollo del proyecto.
3. Acceder a la consola de firebase, crear un proyecto y habilitar una instancia de RealtimeDatabase. Por último, es necesario conseguir las credenciales del proyecto para inyectarlas posteriormente en el código y poder utilizar MusicIn.
4. Acceder a la consola del proyecto en Google Cloud y generar la credencial para poder utilizar Google Maps en MusicIn.

5. Por último, clonar el repositorio en un equipo y ejecutar el comando ***npm i*** para instalar dependencias necesarias para hacer funcionar MusicIn.

6 Conclusiones

El desarrollo y planteamiento de MusicIn se ha hecho desde un primer momento desde el punto de vista y las necesidades reales de un músico. El mundo de la música es bastante complejo y plantea muchas barreras de entrada para todos aquellos que quieren vivir de él, sobre todo porque se mueve en base a círculos cerrados de contactos, que dificultan enormemente que cualquiera pueda conseguir un buen empleo. En base al breve estudio de mercado realizado al inicio de esta memoria, los proyectos más "conocidos", plantean un acercamiento por parte de los músicos, como por ejemplo, crear música de forma colaborativa, poner a las discográficas en contacto con los músicos o crear un Marketplace para vender tus instrumentos y además compartir contenido musical, pero ninguna aborda el problema que se intenta resolver con MusicIn. Esto es conseguir la comunicación directa entre los de "arriba" y los de "abajo" de forma fácil y sencilla, imitando el concepto de LinkedIn adaptado al mundo de la música.

Además de realizar un análisis de la competencia, se ha validado la idea con músicos profesionales, que ven de primera mano todos los bloqueos que este mundo plantea y han colaborado en la definición del producto. Como resultado de este proyecto, existe un prototipo implementado listo para su salida al mercado, que en una primera instancia resuelve los problemas planteados con anterioridad. Además, debido a las tecnologías implicadas para el desarrollo de MusicIn, se agiliza el proceso de modificación y actualización tanto de las aplicaciones móviles como la posible versión web que tenga en un futuro.

Después de todo este tiempo de desarrollo, investigando sobre las tecnologías empleadas para este proyecto, uno se va dando cuenta que éstas avanzan a pasos agigantados.

Cuando uno utiliza herramientas como por ejemplo, frameworks pertenecientes a grandes empresas, se sitúa en una situación de dependencia con dicha empresa que no siempre es favorable. Ya sucedió con AngularJS, que al reescribir todo el core desde cero de dicho framework, Google cambió radicalmente la estructura y sintaxis y dejó tirados a todos los que dominábamos la primera versión de este framework.

Al hablar de tecnologías web, no sólo debemos tener en cuenta a estas enormes empresas, sino que además contamos con los estándares que regularizan y calman este avance tan brusco, aunque a veces no consigan llegar al mejor resultado, como el caso de la web semántica. No hay que

indagar mucho para ver que aunque todavía existe una comunidad grande detrás, no ha podido alcanzar su máximo potencial. Los motivos son desconocidos, pero lo que parecía ser una estructura estándar y legible para todo el mundo con el fin de intercambiar información de forma fácil y sencilla, ya no suena con tanta fuerza como antes.

Por otra parte, la creciente potencia de los dispositivos móviles y de los navegadores, ha permitido que el término aplicación híbrida coja bastante fuerza. Antes era impensable que alguien desarrollase una aplicación móvil utilizando este tipo de tecnologías, tanto por lo verdes que estaban, como por todos los problemas que podían acarrear. Ahora, lo más común es que para desarrollar aplicaciones móviles, se programe con la siguiente filosofía: "código una sola vez y funcionamiento en cualquier dispositivo" y frameworks como Angular y SDKs como Ionic agilizan la velocidad de desarrollo y abaratan enormemente los costes.

7 Fuentes de información

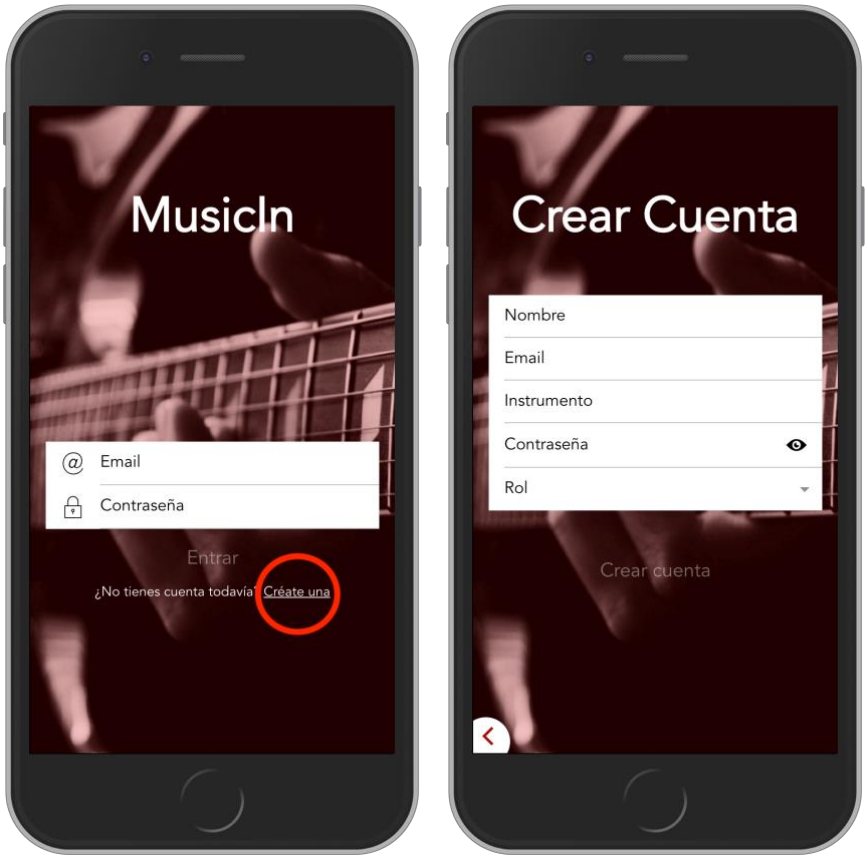
1. Reverbnation: Red social para músicos, www.reverbnation.com. Última fecha de acceso: 08/11/2017
2. Wikipedia: Reverbnation, https://en.wikipedia.org/wiki/ReverbNation. Última fecha de acceso: 08/11/2017
3. Kompoz: Red social para músicos, www.kompoz.com. Última fecha de acceso: 10/11/2017
4. Miuseek: Red social para músicos, www.miuseek.com. Última fecha de acceso: 10/11/2017
5. Significado de los colores: http://significadodeloscolores.net. Última fecha de acceso: 29/11/2017
6. Documentación de Angular: https://angular.io/docs. Última fecha de acceso: 12/04/2018
7. Documentación de TypeScript: https://www.typescriptlang.org/docs/home.html. Última fecha de acceso: 12/04/2018
8. Documentación de Ionic: https://ionicframework.com/. Última fecha de acceso: 12/04/2018
9. Documentación de Firebase: https://firebase.google.com/docs/?hl=es-419. Última fecha de acceso 12/04/2018
10. Web de NPM: https://www.npmjs.com/. Última fecha de acceso: 12/04/2018
11. Wikipedia: Sparql: https://es.wikipedia.org/wiki/SPARQL. Última fecha de acceso: 12/04/2018
12. W3C: Sparql: https://www.w3.org/TR/rdf-sparql-query/ Última fecha de acceso: 12/04/2018
13. Como consultar a Wikipedia: http://tinysubversions.com/notes/how-to-query-wikipedia/. Última fecha de acceso 03/05/2018

8 Anexo

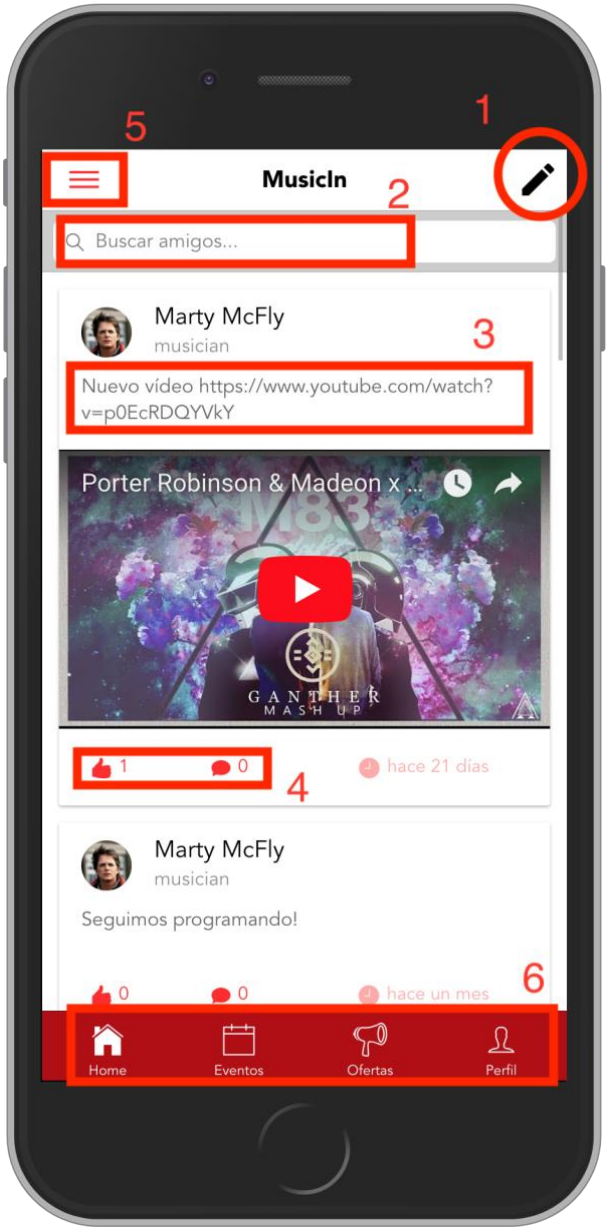
Manual de uso

Login y Home

Para poder empezar a utilizar la aplicación, es necesario realizarse una cuenta en MusicIn, para ello accedemos a la pantalla de crear cuenta a través del login y rellenamos los campos solicitados.



Una vez hayamos accedido a la aplicación, podemos escribir un mensaje a través del icono del lápiz arriba a la derecha [1], además, podemos etiquetar un músico para obtener información adicional sobre él, etiquetándolo con una '#' y escribiendo '_' en vez de espacios si el nombre tiene espacios de por medio. Ej: #George_Harrison. También se aceptan urls de youtube que luego se incluirán como vídeos.



Además, podemos utilizar la barra de búsqueda para agregar y ver perfiles de amigos [2]. Si queremos ver los usuarios a los que les ha gustado el mensaje, solo tendremos que picar en el texto del mensaje [3]. Por último podemos dar me gusta a un mensaje y acceder a la vista de comentarios de un mensaje [4]. Para acceder a otras vistas principales de la aplicación tenemos el menú [5] y los tabs [6]. Los tabs nos muestran las opciones de:

- "Chats", donde podemos ver los chats que tenemos pendientes.
- "Amigos", donde vemos el listado actual de amigos.
- "Solicitudes", donde vemos las solicitudes de amistad pendientes.

Los tabs nos dan acceso a las vistas:

- Ofertas: para ver las ofertas disponibles además de crear nuevas ofertas.
- Eventos: donde se ven los eventos disponibles además de crear nuevos eventos.
- Perfil: donde puedes cerrar sesión y ver los mensajes escritos.

Ofertas y eventos

Si queremos crear una oferta, debemos picar en el icóno '+' [1] y rellenar los campos solicitados. Para acceder a la información de un evento basta con seleccionar el evento a visualizar [2] y para ver en el mapa los diferentes eventos, debemos seleccionar el icono de mapa en la esquina superior izquierda, al lado del menú. El funcionamiento de la vista de ofertas es igual al planteado aquí para los eventos a excepción de la funcionalidad del mapa.