Carlos Licha de la Encarnación |Manuel González Saavedra

Escuela Politécnica Curso 2018/2019

Entrega seguimiento Asee

FestivApp

Contenido

[Descripción de la idea 2](#_Toc533934660)

[Público objetivo 2](#_Toc533934661)

[Propuesta inicial. 2](#_Toc533934662)

[Análisis 2](#_Toc533934663)

[Requisitos funcionales 2](#_Toc533934664)

[Requisitos no funcionales 3](#_Toc533934665)

[Diseño de la IU 4](#_Toc533934666)

[Mapa de navegación 4](#_Toc533934667)

[Mockup/Wireframe de la aplicación 4](#_Toc533934668)

[Estado del proyecto en la reunión 6](#_Toc533934669)

[Proyecto final 6](#_Toc533934670)

[Características 6](#_Toc533934671)

[Arquitectura 6](#_Toc533934672)

[Conclusión 8](#_Toc533934673)

# Descripción de la idea

El proyecto que estamos desarrollando para la asignatura de Arquitecturas Software para Entornos Empresariales es una aplicación para entornos móviles Android que incluya funcionalidad completa de acceso a base de datos, *networking*, tareas asíncronas y patrones de diseño singulares en Android. Además, también pretende que nos desenvolvamos en el desarrollo de interfaces gráficas (*Activities*) valiéndonos de principios como *Material Design 2.0*

Nuestra app contiene diferentes categorías en las que se puede clasificar un evento, no excluyentes entre sí, es decir, puede existir en varias a la vez, o en ninguna. Estos eventos se obtienen buscándolos a través de una API que se conecta a un servidor remoto y se ofrecen en vista de rejilla. Cuando el usuario pulse un evento, verá la información asociada, proporcionada por dicha API. En la interfaz del evento se encuentran los botones para clasificarlo en las diferentes categorías

# Público objetivo

Corresponde a personas, generalmente jóvenes o aficionados a la música, que suelan con frecuencia asistir a conciertos de grupos en su provincia o en otras provincias de España, de manera que con esta aplicación tengan más fácil y rápido el acceso a la información: dónde es, cuándo y quiénes participan.

# Propuesta inicial.

La funcionalidad principal de la aplicación consiste en la consulta de eventos, que pueden ser festivales y conciertos, a partir de una configuración (Nombre, provincia, fecha de inicio, etc.) Además, estos eventos pueden almacenarse localmente guardándose en categorías como *Favoritos*, *Eventos próximos* o *Asistidos* dentro de una base de datos en la propia app. Podrá verse toda la información de un evento, que será proporcionada por la API de *SongKick.* La base de datos SQLite se ha construido como ORM y el *parsing* de los objetos en la API se ha realizado manualmente.

## Análisis

### Requisitos funcionales

#### Buscador:

* R1- Buscar eventos(Festival/Concierto) por nombre.
* R2- Buscar eventos(Festival/Concierto) por provincia.
* R3- Buscar eventos(Festival/Concierto) por fecha de inicio y/o finalización
* R4- Poder elegir si se quiere buscar, Festival, Concierto o ambos.

#### Menú Buscador:

* R5- Acceder a la lista de eventos Favoritos
* R6- Acceder a la lista de eventos Pendientes
* R7- Acceder a la lista de eventos Asistidos

#### Lista de eventos:

* R8- Acceder a un evento de la lista pulsandolo.
* R9- Alternar entre páginas de la lista en caso de que las haya.

#### Detalles del evento:

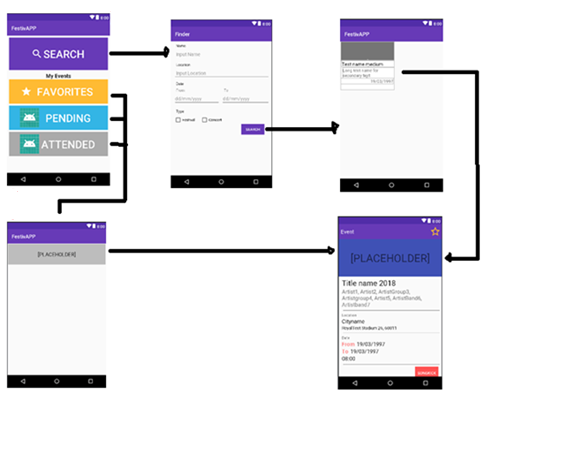
* R10- Operaciones *CRUD* para gestionar lista de Favoritos.
* R11- Operaciones *CRUD* para gestionar lista de Pendientes.
* R12- Operaciones *CRUD* para gestionar lista de Asistidos.
* R13- Consultar información relevante al evento.
* R14- Acceder a la página de *SongKick* del evento en cuestión mediante un botón.
* R15- Poder cambiar el idioma de la aplicación.
* R16- Recibir notificaciones cuando un evento de interés se acerque.

### Requisitos no funcionales

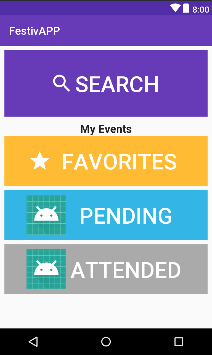
* Aplicación accesible e intuitiva mediante el uso de *Material Design*.
* Uso de base de datos mediante la *API SQLite*.
* Obtención de los datos mediante llamadas *REST* a la API *SongKick*.
* Uso fluido de la aplicación, sin pantallas de carga ni retrasos.
* Programación de la lógica en *Java* y en *Kotlin*, haciendo uso de la interoperabilidad.
* Patrones de diseño para mejorar la calidad del código.

## Diseño de la IU

### Mapa de navegación

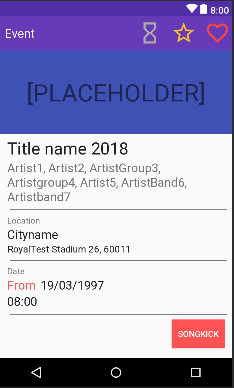


### Mockup/Wireframe de la aplicación

 MainActivity

Este es el Activity principal que se cargará cuando se inicie la aplicación. Permite navegar hacia los eventos marcados como Favoritos, eventos que aún no próximos (Pending) y eventos a los que ya se haya asistido (Attended).

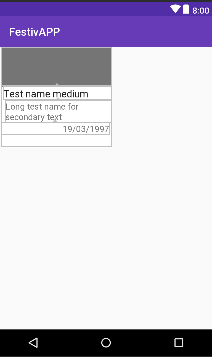
También se puede acceder a una búsqueda por diferentes cambios en Search

EventActivity

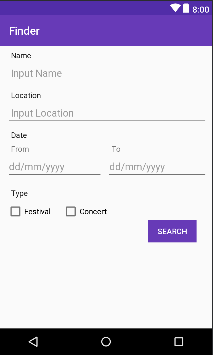
Esta Activity permite mostrar la información de un evento concreto: su nombre, los artistas que participan, la ciudad en la que se desarrolla, el lugar exacto en esa ciudad y la fecha y hora de comienzo.

En la AppBar se implementan las opciones de añadir el evento a Favoritos

También se implementa la posibilidad de acceder a su página en SongKick

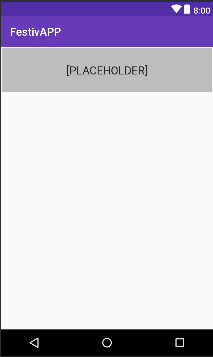
EventListActivity

Este Activity muestra todos los eventos según la búsqueda que se haya hecho en el FinderActivity. Si se presiona sobre uno, llevará a los detalles del evento.

FinderActivity

Este Activity permite buscar un evento accediendo a la API de SongKick. Permite hacer una búsqueda según el nombre del evento, localización (campo obligatorio), y fecha de comienzo o cierre.

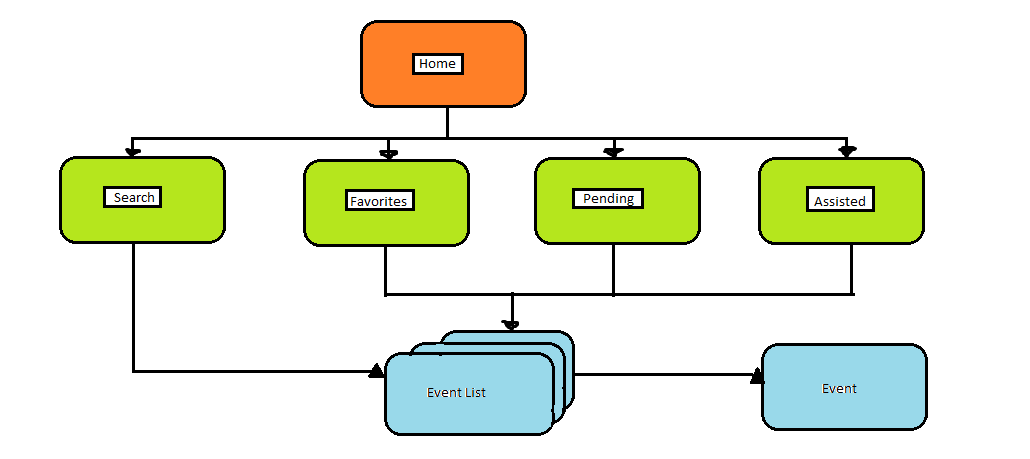
También permite discriminar si el evento es un festival, concierto o ambos.



EventListActivity

La lista de eventos es accesible desde el buscador o desde las opciones de Favoritos, Asistidos o Pendientes en ella se mostrarán las distintas previews de los eventos que coinciden con los criterios de búsqueda.

# Mapa de navegación



# Estado del proyecto en la reunión

En la reunión de seguimiento vimos el estado del proyecto con la implementación inicial. Se nos indicó principalmente aspectos de la documentación. En cuanto a diseño se destacaron aspectos como que el activity inicial debe contener información más directa al usuario, pero que al final por falta de tiempo se dejará con el menú existente. La implementación cumple con los requisitos para la entrega.

Ahora debe reimplementarse el proyecto con los patrones de arquitectura de Android, que describiremos a continuación.

# Proyecto final

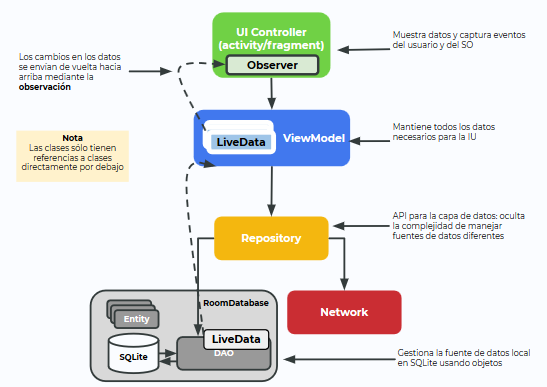
## Características

La funcionalidad del proyecto sigue siendo la misma que en la versión inicial: podemos buscar eventos a través de una fuente externa para mostrar su información. Estos eventos pueden ser guardados localmente en diferentes listas como son *Favoritos, Asistidos o Pendientes*.

No obstante, el proyecto ahora sigue los patrones de arquitectura recomendados en Android

## Arquitectura

La estructura de una aplicación Android debe seguir la siguiente arquitectura



En nuestro proyecto se ha aplicado como sigue:

* UI Controller: Los controladores de la interfaz son las Activities que mostrarán la interfaz al usuario. Se conservan en aspecto igual que en la entrega anterior. La diferencia está en la lógica interna, ahora los datos **no** se obtienen desde la base de datos, sino desde **ViewModel**, que controla los datos, sustituyendo todas las llamadas realizadas de la base de datos por llamadas a ViewModel.

Además, se ha implementado el patrón de implementación **DataBinding**, que permite vincular cada View con el layout de forma directa.

Los Activities fundamentales que contienen datos son ***EventActivity***y ***EventListActivity****.*

* ViewModel: El ViewModel es el controlador de los datos que se mostrarán a la interfaz. Contiene una instancia de **LiveData**, de tipo Evento, que es donde se almacenará la información. Las clases ViewModel existen para gestionar los datos de la lista de eventos como a un evento de forma individual (EventActivityViewModel y EventListActivityViewModel)

**LiveData** es un componente consciente del ciclo de vida, y en este caso son los eventos de Songkick. Android se encarga de gestionar su ciclo de vida y con este patrón se nos garantiza datos actualizados, sin pérdidas de memoria y robustez.

Nos ha permitido utilizar el patrón **Observer**: un evento notificará a su observador (activity) sobre sus cambios para que actualice su interfaz.

Su misión es ofrecer de forma simple la información a la interfaz, valiéndose de ello de llamadas únicamente al Repository:

* Repository: El repositorio unifica las llamadas de la fuente externa como es la API de Songkick y la fuente interna como es la base de datos. Lleva toda la lógica necesaria para simplificar sus métodos a ViewModel. La función de repositorio la lleva la clase **EventRepository**. Esta clase está implementada bajo el patrón **Singleton**
* RoomDataBase: Esta clase hace la función de base de datos. Está implementada con el patrón Data Mapper, que nos ofrece la librería **Room**. Las clases que hacen función de base de datos son:
  + AppDatabase: La propia base de datos se conserva casi igual, con las etiquetas añadidas de Room
  + DBContract
  + EventItem: Son las llamadas entities; se conserva casi igual, con las etiquetas en el nombre de la clase y sus atributos para poder usar Room.
  + EventDAO: Esta clase ha cambiado radicalmente a ser una interfaz con los equivalentes métodos CRUD, pero realizados únicamente como consultas SQL escritas en etiquetas. Room se encarga de generar su implementación.

Esta clase está implementada como patrón **Singleton.**

* Network: Esta es la fuente de datos externa. Es la implementación de la API de Songkick, que consta de dos métodos principalmente: buscar un evento por localización y otro por nombre y localización, tratando los datos como JSON

# Conclusión

Crear este proyecto nos ha resultado difícil en esta última parte con patrones de arquitectura, ya que no entendíamos bien el funcionamiento de LiveData y hemos tenido problemas con llamadas asíncronas de la base de datos, pero consideramos que ha sido bastante útil y nos da pie a crear aplicaciones robustas siguiendo un estándar con estos patrones.