NUEVOS PARADIGMAS DE INTERACCIÓN

Práctica 1: Sensores

Memoria Técnica

ALBA CASILLAS RODRÍGUEZ
MIGUEL MOLINO MORENO
ANDREA NAVARRO JIMÉNEZ
JOSE MANUEL OSUNA LUQUE

Índice

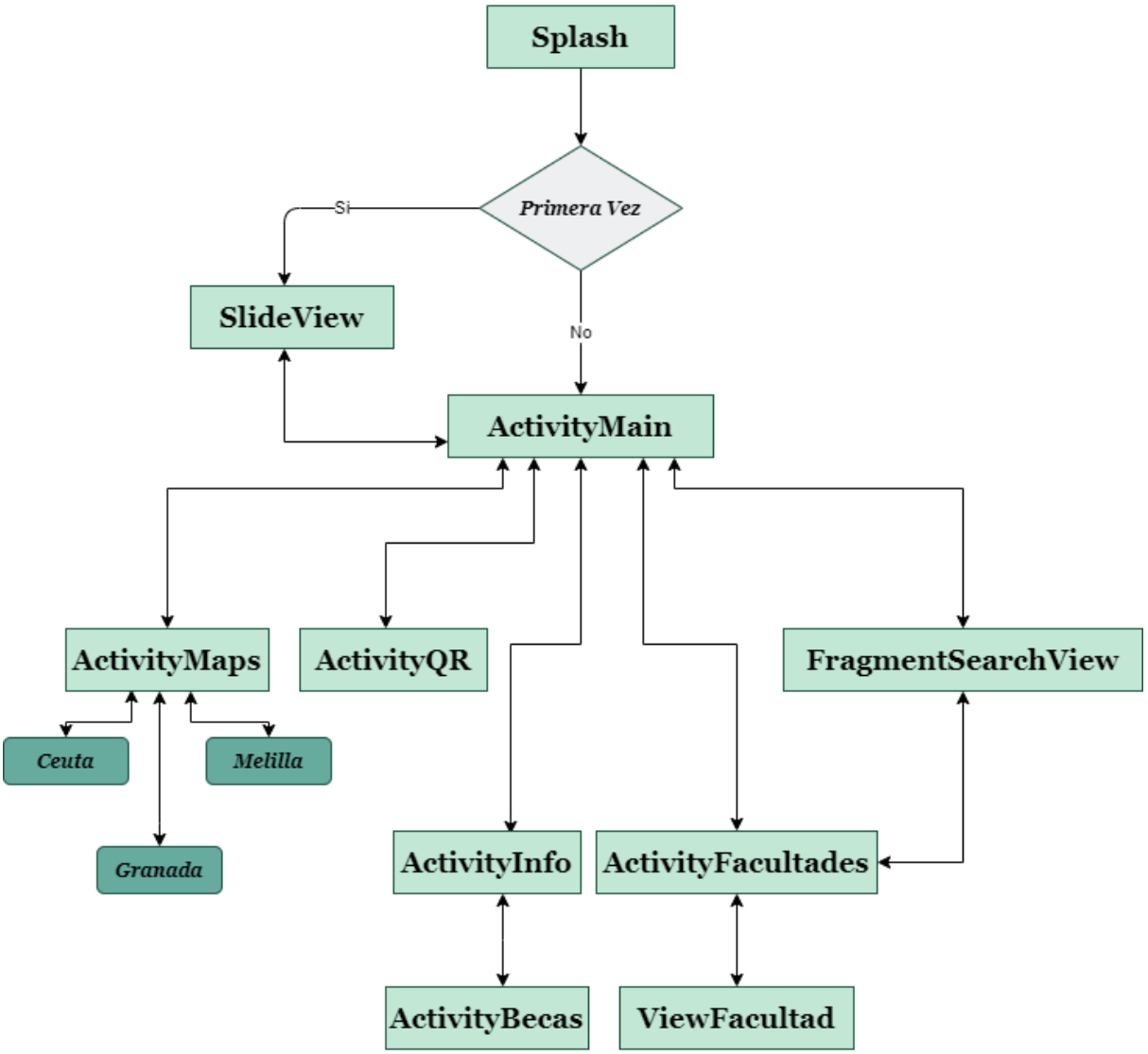
1. Introducción

En este proyecto vamos a explicar a *Between Students (BEST)*, una aplicación enfocada a los estudiantes de la Universidad de Granada (incluyendo Ceuta y Melilla), guiándose por las facultades de una manera dinámica y productiva y pudiendo obtener acceso a información de interés.

En esta primera versión del proyecto se proporciona una interfaz sencilla que dará acceso a la información de una manera ordenada y directa mediante el uso de algunos botones, y basada en sensores con los que se podrá manejar el sistema mediante un dispositivo móvil.

El logo representativo de nuestra aplicación y la paleta de colores usada será:

2. Diagrama de Flujo



3. Funcionalidades

Vamos a proceder a comentar brevemente qué es lo que hace cada una de las funcionalidades (clases) de la aplicación.

Cabe destacar que cada clase es usada como una "*View*" de la aplicación, es decir, cargada o llamada, ya sea por el propio sistema android o a través de un redirección desde otra clase. Por ello, es necesario que sean declaradas en el archivo *AndroidManifest.xml*, ya que de lo contrario, si se quiere cargar el contenido de una clase y mostrar su visualización en el dispositivo, lanzará un error y se cerrará la aplicación.

3.1. Splash

Es la pantalla inicial, la cual se muestra durante un tiempo de carga al iniciar la aplicación. Actúa a modo de presentación, puesto que únicamente presenta el nombre y logo de la misma, simulando un *preloader*.

Para lograr este efecto, se ha implementado una clase *Splash*. En dicha clase, se establece un *DELAY* en microsegundos para simular ese retardo típico en las pantallas de carga de una aplicación. Ese retardo se le pasa a un objeto de tipo *TimeTask* que permite la creación de hilos. La función del hilo generado es "dormir" al programa durante el tiempo establecido en el *DELAY* y acto seguido ejecutar la acción pertinente. En este caso, navegar hacia la siguiente Actividad de la aplicación.

3.2. SlideView

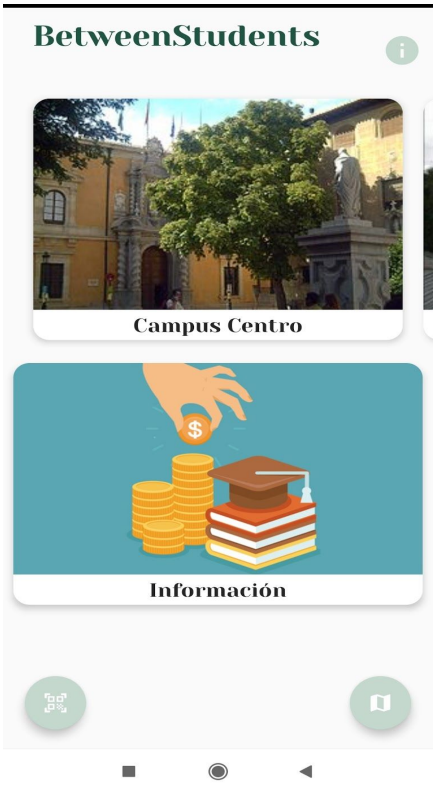
Esta funcionalidad se ha implementado para hacer la presentación de la app y dar la bienvenida a los usuarios en la primera entrada a la aplicación. Actúa como transiciones de una pantalla completa a otra.

Consta de vistas, que en primer lugar son creadas en un archivo de diseño llamado *slide_screem.xml*, y que serán utilizadas en la clase *AdapterSlideViewPager*. En dicha clase se inicializan todos los elementos de cada vista en función del número "*count*" a mostrar. Para que esta funcionalidad se realice correctamente y solo se inicie en la primera apertura de la aplicación se ha utilizado la clase *Intent* para crear una instancia de la clase anteriormente nombrada. Esta instancia se inicializa en una clase implementada llamada *ActivitySlide*, encargada de una única inicialización del tutorial.

La información que se visualiza en el tutorial puede ser vista de nuevo pulsando el botón de información del *ActivityMain*.

3.3 ActivityMain

Tras la visualización del tutorial (la primera vez), o en su defecto, el Splash, se visualiza la pantalla principal (*ActivityMain*). Éste es el layout y la clase desde la cual parten todas las demás funcionalidades de la aplicación creada:



3.3.1. Header

Consta del nombre de la propia aplicación y un botón cuya finalidad es mostrar de nuevo el tutorial de la aplicación, a modo de ayuda en caso de que el usuario necesitase consultar de nuevo información sobre cómo usar la aplicación.

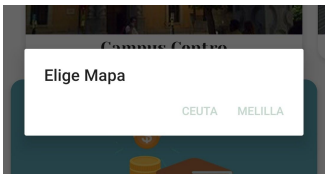
Este botón se ha programado usando una imagen a la que se le ha dado la funcionalidad de ejecutar una acción de ser tocada. Esto se consigue mediante el método *setOnClickListener*, de la clase *ImageView*. Este método recoge la imagen mediante la función *findViewById(RECURSO)* y se le otorga la función de recoger el tutorial de inicio.

3.3.2. Footer

El footer de la aplicación está compuesto por dos botones flotantes, generados con la clase *FloatingActionButton*.

El botón situado a la izquierda permite abrir el lector QR mientras que a la derecha se encuentra aquel que activa la clase del Mapa. Este último consta de dos acciones programadas:

- ***Pulsación simple***: Abre el mapa de la ciudad de Granada.
- ***Pulsación de larga duración***: Proporciona la opción de elegir el mapa de Ceuta o de Melilla.



Estas acciones se implementan gracias a dos métodos disponibles de la clase *FloatingActionButton:* ***setOnClickListener()***, donde se generará un *Intent* asociado a la clase que gestiona los mapas (*ActivityMaps*) de nuestro proyecto e indicando mediante el método *putExtra()* el mapa que deseamos abrir (el de Granada, en este caso); y ***setOnLongClickListener()*** donde, a parte del *Intent* asociado a la clase *ActivityMaps* , se implementa el método *onLongClick()* que proporciona un Dialog encargado de proporcionar las opciones de los mapas de Ceuta y Melilla y de generar el parámetro pasado en *putExtra()*.

3.3.3. Interfaz principal

La parte central de la interfaz está formada por dos elementos principales: un *RecyclerView* y una única *CardView*.

- El ***RecyclerView*** se rellena con todos los campus pertenecientes a la Universidad de Granada. Cada campus tiene asociado el nombre del mismo y una imagen representativa de este. Esta información, junto a la información referente a las Facultades, es almacenada en una **Base de Datos SQLite**, una base de datos local dentro de la propia aplicación; de manera que se facilita la actualización o modificación de los datos.

Para añadir cada Campus a la lista del RecyclerView se debe hacer uso de un *Adapter*, que además ajusta la información y la muestra en una vista predefinida. Esta vista está desarrollada en un archivo XML usado como template, llamado *item_campus.xml*, construido bajo un elemento

CardView. El Adapter es una clase JAVA, que lo que hace es recibir una cantidad de elementos, en este caso de tipo Campus y los añade la RecyclerView como si de una lista se tratase.

Para cada elemento recibido en el Adapter se crea un nuevo objeto View modificando los elementos base del XML, y cambiandolos por el nombre e imagen del campus objeto recibido. Tras ello, lo carga al RecyclerView, proporcionando una interfaz deslizable con la que encontrar la información de manera rápida y directa.

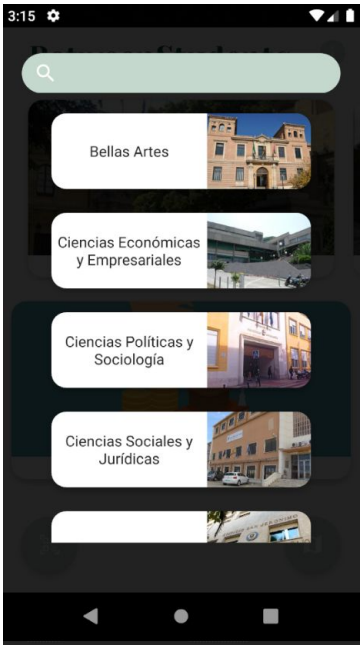
Al hacer click en un elemento, para poder realizar una acción en función del elemento clicado se ha gestionado el método *setOnClickListener()*, pero esta vez del objeto *AdapterCampus* creado, de forma que se coge la posición del elemento. Cuando se pulsa sobre un elemento del RecyclerView, la aplicación navega a una nueva Activity donde se mostrarán las Facultades que pertenecen al Campus seleccionado. Esta nueva ventana que aparece es gestionada en la *ActivityFacultades* [Apartado 2.4].

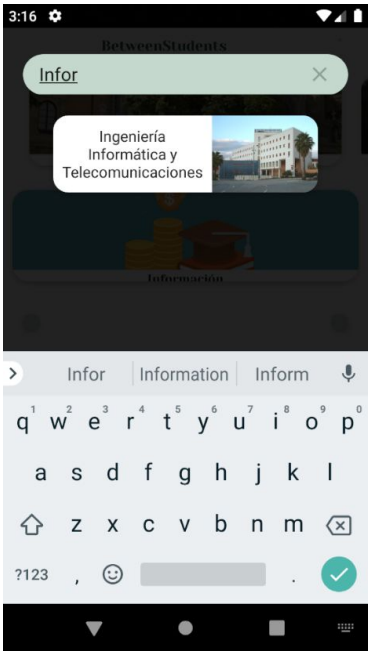
- El ***CardView*** es un único elemento definido manualmente y es usado como botón para navegar al apartado de información de interés para un alumno. Este apartado de información nos llevará a otra pantalla compuesta de cuatro CardView: becas, transporte, bibliotecas y comedores.

En esta primera versión del proyecto, únicamente se le ha otorgado funcionalidad al apartado de Becas, recogido en *ActivityBecas* [Apartado 2.6].

3.4. FragmentSearchView

El usuario tiene la posibilidad de realizar una búsqueda rápida de una facultad, ya que cabe la posibilidad de que desconozca el campus en el que se encuentra. Por ello, mediante un deslizamiento hacia arriba *(swipe up)* desde la parte inferior de la pantalla principal (*ActivityMain*), se desplegará una barra de búsqueda, como se puede observar en las siguientes imágenes:





Para generar este efecto, se ha hecho uso de un *FragmentActivity* al que se le ha llamado *FragmentSearchView*.

La llamada a este Fragment se realiza en un método del ActivityMain llamado *onFling()*, el cual se encuentra disponible cuando la clase implementa la interfaz *OnGestureListener* (*GestureDetector.OnGestureListener*). Este es el método por excelencia cuando se desea realizar un gesto de Swipe Up.

Para convertir esta clase en un buscador, dentro de *FragmentSearchView* se declararán dos objetos, uno de tipo *SearchView* y otro de tipo *SearchManager*. Estos objetos permiten implementar una búsqueda dentro de unas listas ya definidas, que se irán llenando o vaciando a razón del contenido introducido en la barra de búsqueda.

Para controlar esas listas de datos, ha sido necesario crear un nuevo Adapter, *AdapterFiltro*, que permitirá el filtrado de información introducida. Este Adapter será aplicado a un RecyclerView para mostrar los resultado de la búsqueda justo debajo de la barra, tal y como se aprecia en la imagen.

Por último para que el RecyclerView vaya cambiando mientras se van introduciendo caracteres en la barra de búsqueda, es necesario sobreescribir el método *setOnQueryTextListener()*, el cual controla qué ocurre cuando se pulsa el botón de buscar tras introducir un texto, o bien qué ocurre en tiempo real mientras va cambiando el texto. En este caso, solo se ha implementado *onQueryTextChange()*, para que cambie los resultados filtrando conforme se va escribiendo. Para mostrar los resultados, se llama al método *getFilter().filter(String)* del AdapterFiltro creado.

El *AdapterFiltro* creado para el buscador es distinto a los creados para las Facultades y los Campus (mencionados más adelante), pues este implementa una interfaz extra, denominada *Filterable* que permite trabajar los datos que recibe el filtro (Adapter) y realizar búsquedas a través de un patrón de texto (los caracteres introducidos en la barra de búsqueda) para devolver una resultado filtrado, mostrando solo aquel contenido que coincide en alguna parte (o completa) con los datos existentes. Esto se consigue creando un objeto de tipo *Filter* dentro de la propia clase *AdapterFiltro*, y al inicializarlo, sobreescribir, dotando así de la funcionalidad deseada, los métodos *performFiltering()* y *publishResults()*. En la clase *FragmentSearchView* simplemente se llama a ese objeto Filter creado al iniciar el Adapter, mediante el método *getFilter()* para usarlo.

3.5. ActivityFacultades

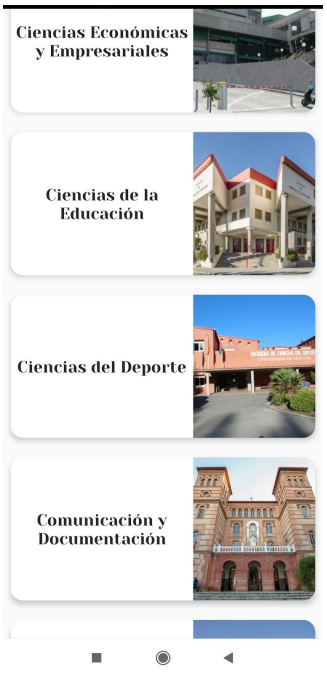
Esta clase controla las facultades que se deben de mostrar a razón del Campus seleccionado. Para ello, es necesario que esta Activity reciba de alguna forma información del elemento seleccionado en la actividad anterior. Esto es posible de realizar gracias a los *Intent*, que además de permitir navegar entre Activities dan la posibilidad de mandar con ellos datos, ya sean variables de tipo primitivo, u objetos creados por el programador.

Esta clase al ejecutarse tras ser llamada por un Intent en la clase Main, recibe el nombre del Campus que fue clicado, a través de un *pushExtra()*. Una vez en la clase, se recoge ese valor a través del método *getStringExtra()*, recibiendo como parámetro la etiqueta usada para el dato. Una vez es recogido, se realiza una lectura en la Base de Datos, con la salvedad de que, en esta ocasión, la lectura tendrá una condición, y es que solo devolverá aquellas Facultades que en su campo “Campus” tengan como valor el nombre recibido. Esta lectura se hace usando el objeto estático creado en el *ActivityMain* e inicializado en el momento de la creación de la app: *database = new BaseDatos(this)*.

Cuando este objeto de tipo BaseDatos se crea por primera vez (al ejecutar la aplicación) carga una pequeña base de datos local con toda la información necesaria sobre las facultades. Para poder trabajar con sus datos se han creado varios métodos: *ReadCAMPUS()*, *ReadFACULTADES()*

y *ReadFACULTADESWHERE(String)*, siendo este último el que usa una sentencia SQL definida con una condición de búsqueda *WHERE*, por el campo de la tabla Facultades.

Cuando este último método devuelve la lista con las facultades que pertenecen al Campus seleccionado, se lleva a cabo el mismo proceso que hacía el *AdapterCampus* para rellenar el RecyclerView implementado en el layout XML usado para visualizar las facultades, *activity_facultades.xml*, con la salvedad de que se usa otro template, *item_facultad.xml*.

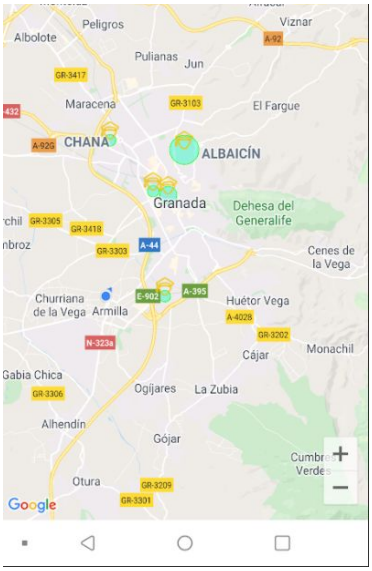


Sobreescribiendo el método *setOnClickListener()* se permite de nuevo recoger una función a la hora de realizar un clic sobre alguno de los elementos, para realizar una nueva acción. En este caso, pulsar sobre alguna de las facultades mostradas, lleva al usuario a una nueva ventana donde se muestra la información referente a la Facultad elegida. La forma de pasar a la nueva Activity que mostrará los datos de la facultad es la misma usada en el *ActivityMain* al pulsar sobre un campus, con la ayuda de los *Intent* y los *pushExtra()*.

Hay una diferencia respecto el RecyclerView de los Campus del *ActivityMain* y del RecyclerView de las Facultades, y es *la orientación*. Esto se controla con la ayuda de un objeto de tipo *LinearLayoutManager*, donde se indica la orientación mostrada a través de una macro predefinida **`LinearLayoutManager.VERTICAL`** o **`LinearLayoutManager.HORIZONTAL`**.

3.6. ActivityMaps

En la interfaz de los mapas nos encontramos un mapa con diferentes vistas: una de ellas, el mapa de Granada con sus cinco campus; y otras que incluyen los campus de Ceuta y Melilla. Estos campus están representados con simples Markers con una zona de incidencia imaginaria creada con *GoogleMap.addCircle()*.

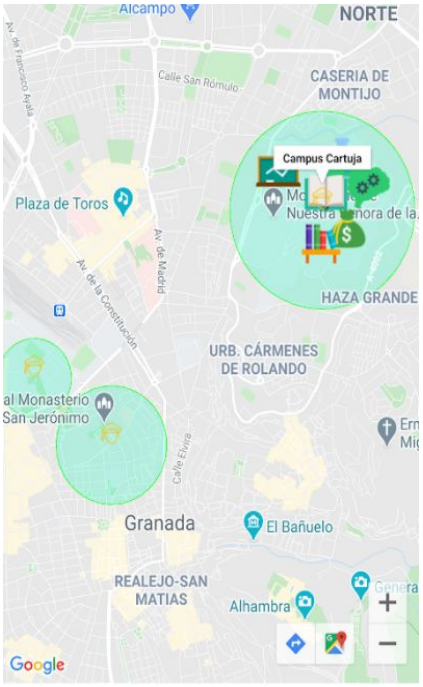


Para la creación de esta interfaz se ha hecho falta dar acceso a permisos para abrir el mapa de Google (***PERMISSIONS_REQUEST_MAPS***) y permisos de ubicación del usuario (**Manifest.permission.*ACCESS_COARSE_LOCATION***) y activar para una API de Google con los siguientes servicios:

- ***Geocoding API*** : Para obtener información de lugares de interés y a tiempo real.
- ***Geolocation API*** : Obtener la ubicación del usuario.
- ***Places API*** : Similar al servicio Geocoding.
- ***Directions API*** : Obtener direcciones entre dos puntos mediante una petición http.
- ***Maps SDK for Android***

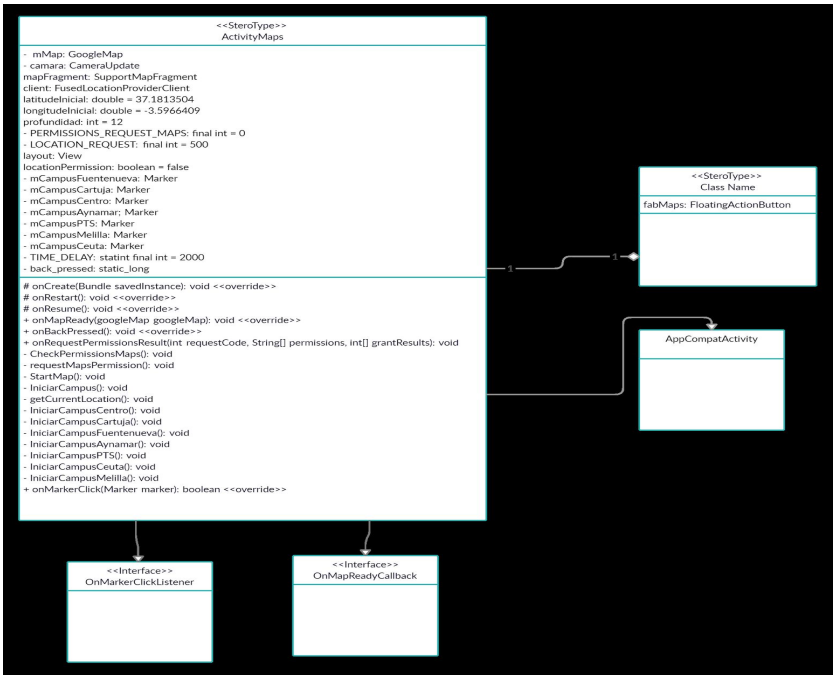
Para obtener la ubicación del usuario a tiempo real basta con darle al botón situado en la parte superior derecha de la interfaz. Esta pulsación realiza una llamada al método *getMyCurrentLocation()*, el cual funciona de la siguiente manera: Se piden los permisos necesarios y se obtiene, mediante el método *getLastLocation(),* la última posición del objeto *client* de la clase *FusedLocationProviderClient*, para después sincronizarlo con el mapa y centrar la cámara a la última ubicación obtenida por *getLastLocation()*. A su vez, el icono que representa la ubicación actual del usuario vendrá acompañado de una brújula [Apartado 3.4].

Si se realiza click en el Marker de un campus, se hace una llamada al método *onMarkerClick(Marker)*, configurado para que aparezcan los iconos representativos de las facultades pertenecientes a dicho campus.



Para borrar los markers si lo desea el usuario es hacer click hacia atrás una vez, apareciendo solo los campus. Para volver a la interfaz principal basta con darle dos veces seguidas. Esto es posible gracias al método *onBackPressed()* sobreescrito.

A continuación se mostrará el diagrama uml enfocado a entender solo la parte de los mapas.



3.7. ActivityBecas

El acceso al apartado de Becas se controla en la clase *ActivityBecas*. En ella se cargan las *CardView* propias de cada tipo de beca disponible. A su vez, se creará una instancia de la clase *WebView* con la que se podrá visualizar páginas web dentro de la propia aplicación mostrando la correspondiente a cada tipo de becas, según se haya elegido.

Para poder visualizar correctamente una página web dentro de la propia aplicación usando el WebView, es necesario recoger las opciones de este objeto en otro de tipo *WebSettings*, y tras ello activar *JavaScript*, con el método del objeto *WebSetting*: *setJavaScriptEnabled(true)*.

Las CardView definidas son las de las Becas del Ministerio, las de la Junta de Andalucía y la propia de la U*GR;* la cual cada una realizará una llamada a la página web correspondiente al pulsar sobre ella *(setOnClickListener()*).

Para poder hacer uso de un navegador interno dentro de la aplicación, es necesario conceder permisos de Internet, **`android.permission.INTERNET`**.

4. Sensores

Una vez detalladas las clases que componen la aplicación,se explicará con detalle los sensores implementados en ella, así como el uso que se le ha proporcionado.

Para hacer uso de algunos de los sensores, es necesario dejar definido su uso dentro del archivo *AndroidManifest.xml*, e indicar qué sensores o permisos deberá tener la aplicación para proceder a su uso. Los que han sido necesario indicar su uso son los siguientes:

(Necesario para poder usar el Lector QR, el permiso de la cámara y la vibración)

- ***android.permission.CAMERA***
- **android.permission.VIBRATE**

(Necesario para poder saber la ubicación actual del usuario)

- **android.permission.ACCESS_COARSE_LOCATION**
- **android.permission.ACCESS_FINE_LOCATION**

(A partir de la API 29, es necesario para seguir sabiendo la localización en segundo plano)

- **android.permission.ACCESS_BACKGROUND_LOCATION**

(Necesario para poder usar el navegador interno en la aplicación)

- **android.permission.INTERNET**

(Necesario para poder comprobar si el dispositivo tiene conexión a Internet)

- **android.permission.ACCESS_WIFI_STATE**
- **android.permission.ACCESS_NETWORK_STATE**

Para poder aceptar estos permisos, en nuestra versión (y a partir de la versión 6 de Android), es necesario pedir consentimiento explícito para activar y usar estas funcionalidades. Esta parte será necesaria programarla dentro de los Activity que hagan uso de dichos sensores.

A la hora de preguntar por los permisos mediante un *Dialog*, cuando el usuario los acepta, la Activity se quedará sin funcionar ya que en el momento de ser llamada no tenía permisos para poder usar los sensores. La forma de arreglar eso es llamar al método *onRestart()* previamente sobrescrito, que recargue de nuevo la Activity.

4.1. Lector QR

En la aplicación se ha implementando el sensor de la cámara dándole la funcionalidad de Lector QR. Dicha funcionalidad se recoge en la clase *ActivityQR* cuyo archivo layout asociado es *activity_qr.xml*.

Empezando por el código XML, este archivo solo contiene un elemento de tipo *SurfaceView* que será usado para implementar la cámara. Dentro del archivo JAVA es necesario declarar un objeto de tipo *SurfaceView*, ya que para tomar una foto se suele requerir que los usuarios vean una vista previa de su tema antes de hacer clic en el obturador; *SurfaceView* permite obtener vistas previas de lo que capta el sensor de la cámara. En este caso no se quiere realizar una foto, pero igualmente será necesario capturar de alguna forma lo que el lector recoge para comprobar si se trata de un código QR. Por ello, para poder reconocer si lo leído es un Código QR o no, este SurfaceView se apoyará en un objeto de tipo *BarcodeDetector*, el cual permite reconocerlos.

Como la cámara va recogiendo información del exterior, es necesario mantenerla activa de alguna manera esperando una petición, en este caso un código QR. Para conseguir que la aplicación se mantenga activa esperando la lectura de un código, se formula un *Thread*.

Cuando el lector QR lee un código aceptado por el objeto *BarcodeDetector*, comprueba si es una URL válida y si es posible abrirlo en alguna aplicación externa que el usuario tenga activa, y que quizás, por ende, sea más compatible abrirlo con ella. Esta funcionalidad se define en un método privado creado denominado *setQRDetector()*, que también controla el thread que mantiene en espera a la actividad.

4.2. Sensor de Giroscopio

La implementación de este sensor se realiza en la actividad principal de la aplicación (*ActivityMain*). La funcionalidad de este sensor reside en que el usuario, mediante un giro del dispositivo móvil hacia la izquierda, tendrá la posibilidad de cerrar la aplicación.

El giroscopio mide la velocidad de rotación en rad/s alrededor de los ejes x, y y z del dispositivo; por ello, los datos del sensor serán tres valores float, que especifican la velocidad angular del dispositivo sobre cada uno de dichos ejes. Se deberá tener en cuenta que se obtendrá una rotación positiva si el dispositivo gira en sentido antihorario (y viceversa).

Para ello, se ha creado una instancia de la clase *SensorManager* que hace uso del método *getSystemService()* y el argumento *SENSOR_SERVICE*. De esta manera, se podrá usar cualquier sensor del dispositivo. Para crear una instancia del giroscopio, se deberá llamar a la función *getDefaultSensor()* e indicar como parámetro el sensor deseado, en este caso: ***`Sensor.TYPE_GYROSCOPE`***.

Una vez declarado el sensor, le daremos una funcionalidad a través de un *Listener* (método que espera una acción para ser ejecutado). Como estamos interesados en una rotación de eje Z del dispositivo, solo manipularemos este valor accediendo a la sentencia: *sensorEvent.values[2]*, de manera que si este valor supera los *"2.5f"*, se llamará al método *onBackPressed()* que, a su vez, realiza una llamada al método *salirAplicacion()*, el cual activará un *Dialog* antes de cerrar automáticamente la aplicación, de manera que se evitará el cierre de la misma de manera accidental.

El valor tope de la velocidad de rotación ha sido elegido arbitrariamente tras realizar pruebas manuales, de manera que el sensor no se activará con el más mínimo movimiento, sino que el usuario deberá girar considerablemente el móvil para activarlo.

4.3. Sensor de Proximidad

Al igual que en el giroscopio; se creará una instancia de la clase *SensorManager*. Sin embargo, en esta ocasión se indicará que el tipo de sensor es: ***`Sensor.TYPE_PROXIMITY`***.

La finalidad de este sensor se basa en mostrar información de cada facultad. Cuando seleccionamos una universidad, el usuario verá a modo de portada una foto de la misma, y al acercar la mano por encima de la cámara frontal del dispositivo, se hará visible información del edificio. Internamente, se está aplicando un cambio de la vista mostrada. Es por ello que este sensor ha sido implementado en la clase *ViewFacultad*.

Los datos recibidos por el sensor están conformados por un único valor que especifica la distancia (en centímetros) entre el sensor y un objeto cercano (la mano del usuario, en nuestro caso). Si este valor, guardado en un array values, es igual al alcance máximo del sensor (el cual se puede consultar mediante el método *getMaximumRange()*), se puede asumir con seguridad que no hay nada cerca. De esta forma, cuando el usuario acerque la mano al sensor, se realizará una llamada al método *CambiarVista()* , el cual recibe un parámetro facultad, de manera que la información que se muestre (tanto la fotografía del edificio como los datos generales) sea la correspondiente a la de la universidad elegida.

4.4. Sensor Magnetómetro

El sensor magnetómetro mide la fuerza o dirección de una señal magnética. Para nuestro proyecto, se ha implementado este sensor a modo de brújula, detectando el polo norte magnético de nuestro planeta para definir dónde se encuentra el polo norte geográfico.

Implementamos el sensor en la clase *ActivityMaps* para combinar la posibilidad de que el usuario conozca su ubicación actual, con la de conocer también su orientación. Se usa el método *GoogleMap.getUiSettings().setCompassEnabled(true);* de la clase *UiSetings()* para hacer uso del sensor magnetómetro y que así funcione correctamente al girar el móvil.

5. Bibliografía

- https://trello.com/
- https://developer.android.com/
- https://stackoverflow.com/
- https://developers.google.com/maps/documentation?hl=es-419
- https://www.tutorialspoint.com/android/android_sqlite_database.htm
- https://sodocumentation.net/android/topic/871/sqlite
- https://www.desarrollolibre.net/blog/android/como-crear-un-lector-de-codigos-qr-en-android-con-android-studio#.X7LCo-Wg9hE
- https://code.tutsplus.com/es/tutorials/android-sensors-in-depth-proximity-and-gyroscope--cms-28084