Doble Grado de Ingeniería Informática y Matemáticas
Curso 2021/22
Grupo 1

Memoria Técnica Android

Práctica 1: Sensores

Nuevos Paradigmas de Interacción. Grupo 1.

UNIVERSIDAD DE GRANADA

Silvia Barroso Moreno
Elena Merelo Molina
Adrián Rodríguez Montero
Carlota Valdivia Manzano

ÍNDICE

1. Introducción

En esta memoria se describirá el proyecto android que se ha desarrollado. El nombre de la aplicación es ***EventUp*** y está principalmente diseñada para los estudiantes de la Universidad de Granada, con la finalidad de facilitar al estudiantado su actividad universitaria diaria.

Asimismo, se explicará brevemente tanto la idea inicial de la aplicación como su posterior implementación, incluyendo una descripción más exhaustiva de la estructura del proyecto junto con las clases necesarias para su implementación.

El logo representativo de la aplicación es el siguiente:

2. Descripción de la aplicación

EventUp permite a cada estudiante de la UGR tener acceso a múltiples servicios que le serán de gran utilidad para el transcurso de su experiencia universitaria. Solo deberá iniciar sesión con su correo de la UGR para acceder a todas estas funcionalidades.

Un usuario podrá visualizar los distintos eventos que se realizarán en la facultad y podrá filtrar su búsqueda. Además, podrá crear y editar eventos así como rechazar eventos que se hayan aceptado previamente.

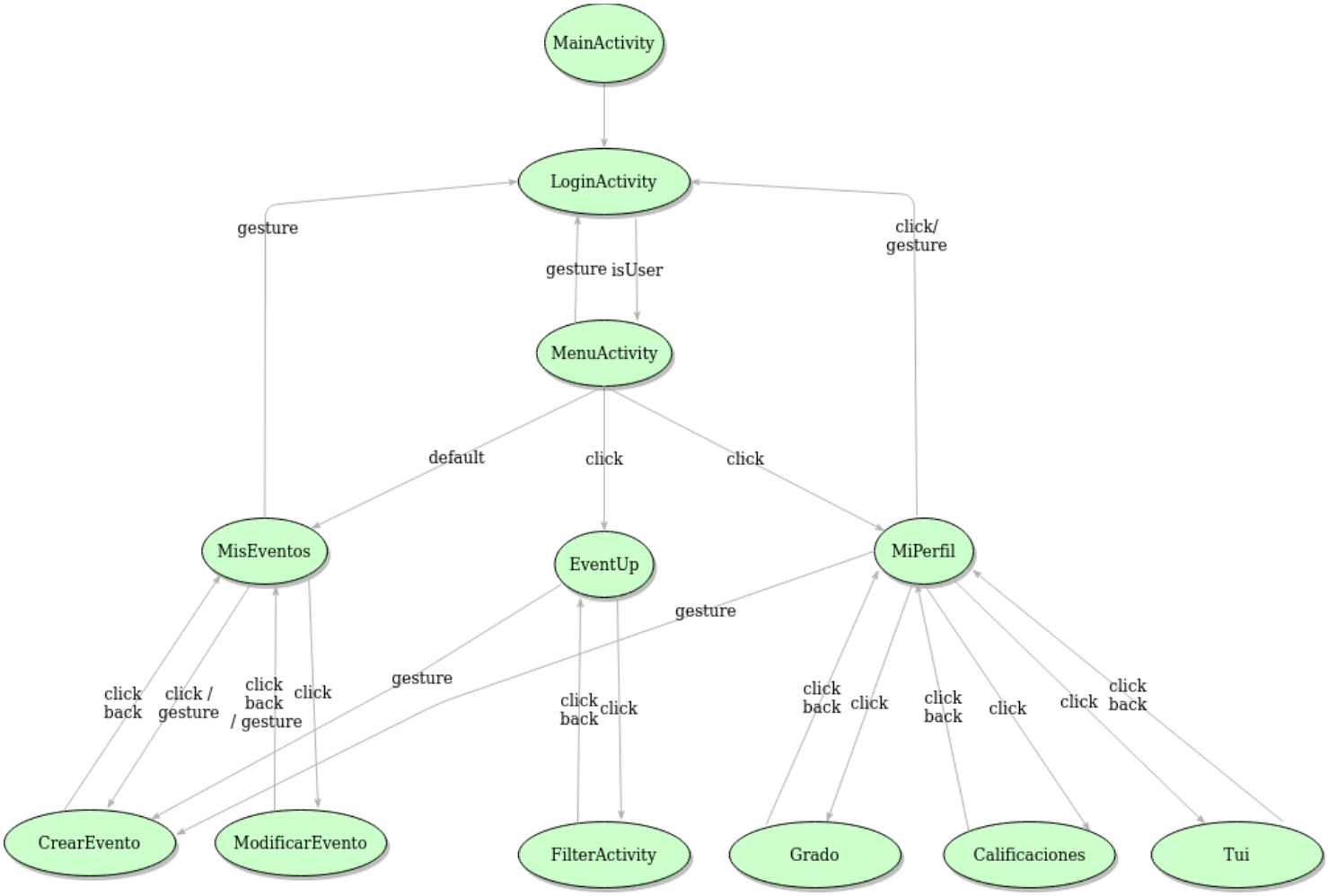
Los eventos se distinguirán por su nombre y se programarán para una fecha y hora determinada. Tendrán un aforo limitado y se podrá especificar el idioma en el que se llevará a cabo. También podremos llegar al destino gracias a su ubicación. Asimismo, como cada evento se adecúa a determinados estudiantes, entonces tendrá asociado un grado.

Por otro lado, cada usuario tendrá su perfil donde se puede apreciar su carrera, sus calificaciones y su TUI virtual.

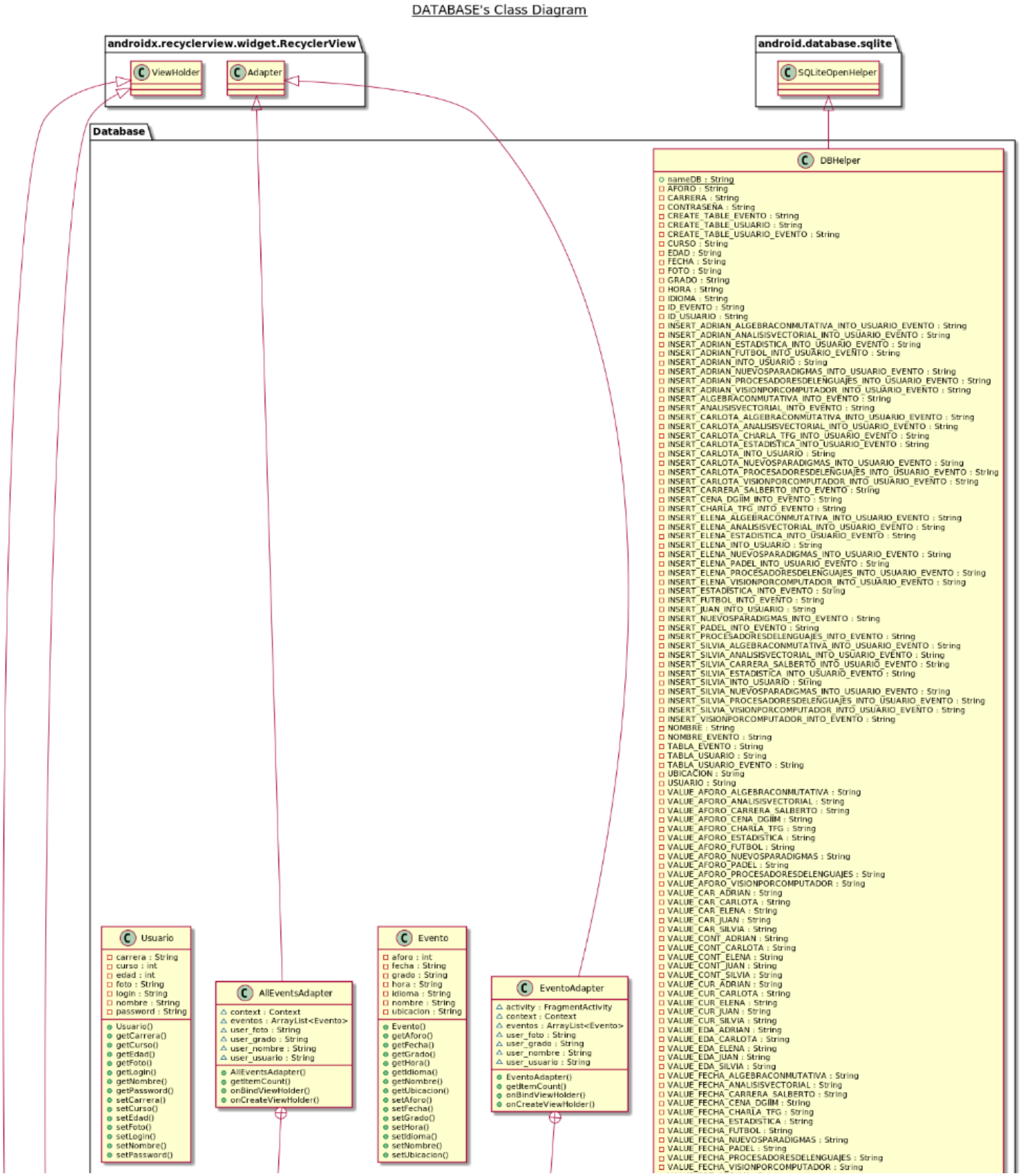
Una vez tengamos nuestros eventos seleccionados podremos localizarlos haciendo uso del GPS y poder trasladarnos de forma cómoda hacia allí.

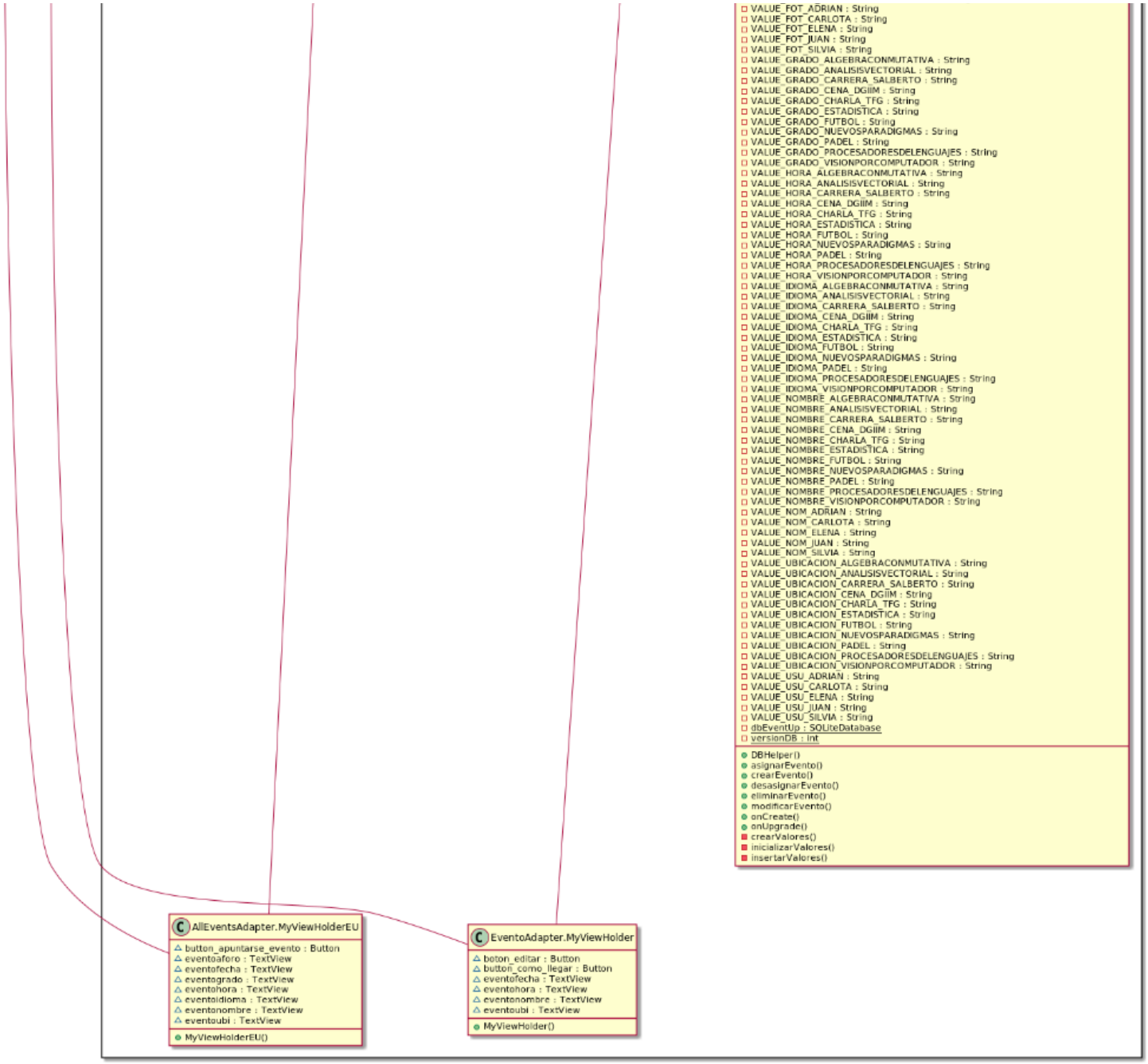
3. Diagramas

3.1 Diagrama de flujo

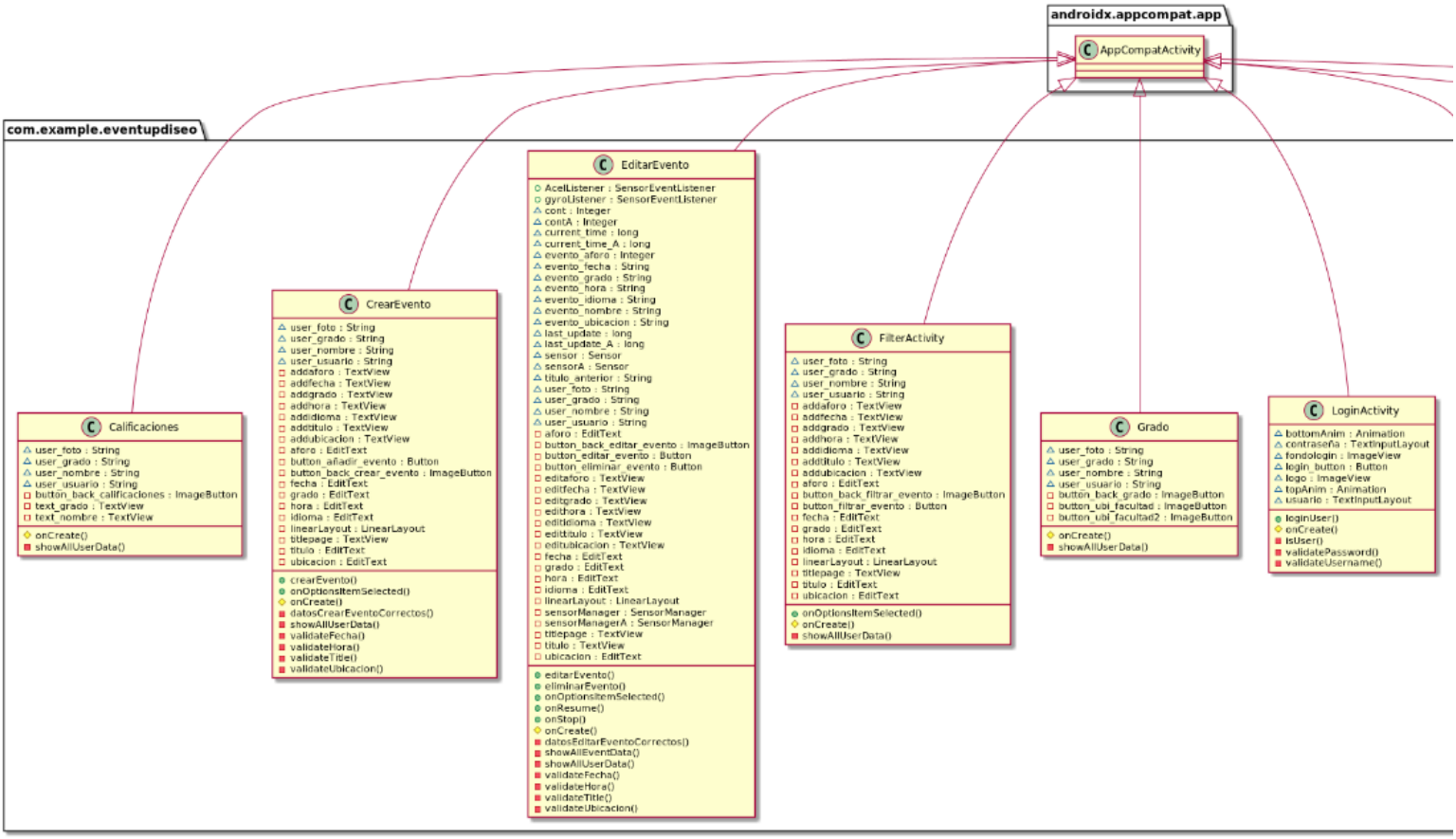


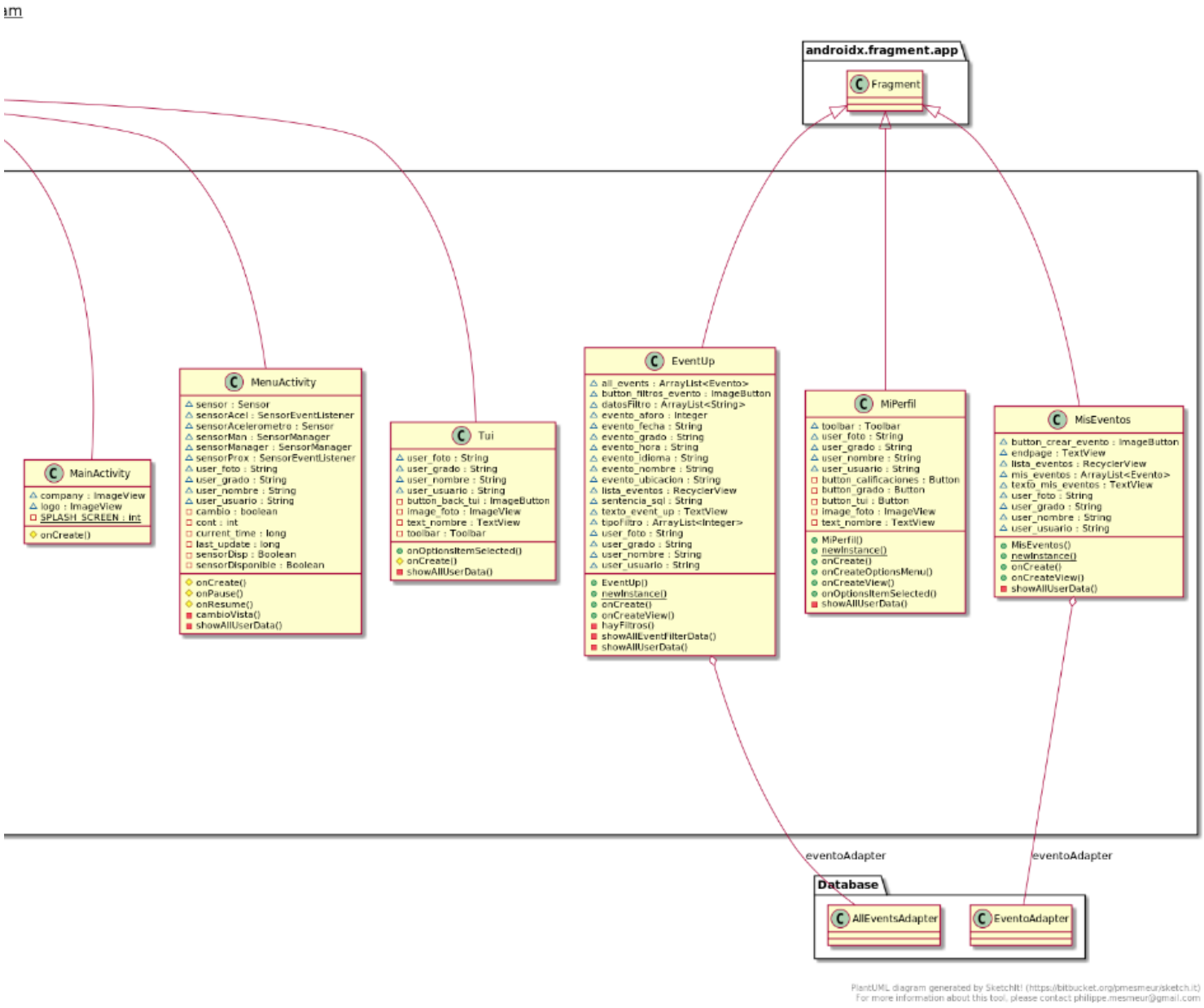
3.2 Diagramas de clases





EVENTUPDISEO's Class Diagrar




androidx.fragment.app
Fragment
MainActivity
company : ImageView
logo : ImageView
SPLASH_SCREEN : int
onCreate()
MenuActivity
sensor : Sensor
sensorAcel : SensorEventListener
sensorAcelerometro : Sensor
sensorMan : SensorManager
sensorManager : SensorManager
sensorProx : SensorEventListener
user_foto : String
user_grado : String
user_nombre : String
user_usuario : String
cambio : boolean
cont : int
current_time : long
last_update : long
sensorDisp : Boolean
sensorDisponible : Boolean
onCreate()
onPause()
onResume()
cambioVista()
showAllUserData()
Tui
user_foto : String
user_grado : String
user_nombre : String
user_usuario : String
button_back_tui : ImageButton
image_foto : ImageView
text_nombre : TextView
toolbar : Toolbar
onOptionsItemSelected()
onCreate()
showAllUserData()
EventUp
all_events : ArrayList<Evento>
button_filtros_evento : ImageButton
datosFiltro : ArrayList<String>
evento_aforo : Integer
evento_fecha : String
evento_grado : String
evento_hora : String
evento_Idioma : String
evento_nombre : String
evento_ubicacion : String
lista_eventos : RecyclerView
sentencia_sql : String
texto_event_up : TextView
tipoFiltro : ArrayList<Integer>
user_foto : String
user_grado : String
user_nombre : String
user_usuario : String
EventUp()
newInstance()
onCreate()
onCreateView()
hayFiltros()
showAllEventFilterData()
showAllUserData()
MiPerfil
toolbar : Toolbar
user_foto : String
user_grado : String
user_nombre : String
user_usuario : String
button_calificaciones : Button
button_grado : Button
button_tui : Button
image_foto : ImageView
text_nombre : TextView
MiPerfil()
newInstance()
onCreate()
onCreateOptionsMenu()
onCreateView()
onOptionsItemSelected()
showAllUserData()
MisEventos
button_crear_evento : ImageButton
endpage : TextView
lista_eventos : RecyclerView
mis_eventos : ArrayList<Evento>
texto_mis_eventos : TextView
user_foto : String
user_grado : String
user_nombre : String
user_usuario : String
MisEventos()
newInstance()
onCreate()
onCreateView()
showAllUserData()
eventoAdapter
eventoAdapter
Database
AllEventsAdapter
EventoAdapter
PlantUML diagram generated by SketchIt! (https://bitbucket.org/pmesmeur/sketch.it)
For more information about this tool, please contact philippe.mesmeur@gmail.com

4. Estructura del proyecto

En este apartado describiremos la estructura que se ha empleado para desarrollar la aplicación. Esto incluye la descripción del esquema de representación de los eventos y usuarios, así como de la base de datos utilizada, la descripción de los activities y fragments que han sido necesarios para el diseño de la aplicación y la descripción de los sensores implementados.

En primer lugar, cabe mencionar que la aplicación ***EventUp*** se ha desarrollado mediante el software **Android-studio** con el lenguaje de programación **Java**. Se ha hecho uso de algunas librerías con funcionalidades externas como:

implementation 'androidx.recyclerview:recyclerview:1.1.0.

En segundo lugar, para desarrollar la aplicación ha sido necesario implementar diversas clases y hacer uso de una base de datos. En nuestro caso hemos utilizado **SQLite**, creando una clase que hereda de **SQLiteOpenHelper**.

Por otro lado, ha sido necesario la inserción de algunos archivos de tipo "***drawable***" para personalizar el diseño de algunos elementos de las vistas, así como algunos botones de tipo ImageButton, o algunos items como los de los eventos.

Del mismo modo, se han implementado unas pequeñas ***animaciones*** para la apertura de la aplicación, al igual que ha sido necesario modificar archivos como colors.xml, strings.xml y themes.xml para realizar más modificaciones de diseño.

Por último, se le han añadido diversos ***sensores*** que realizan unas funcionalidades concretas y que serán explicados más detalladamente en esta sección.

4.1 Clases empleadas

Para el desarrollo de la aplicación se han implementado siete clases: ***Evento, Usuario, MyViewHolder, MyViewHolderEU, EventoAdapter, AllEventsAdapter*** y ***DBHelper.*** Todas ellas pertenecen al package **Database**.

4.1.1 Clase Evento

La clase Evento está formada por los siguientes atributos: seis strings con el nombre, la fecha, la hora, el grado, la ubicación y el idioma del evento, y un dato de tipo entero con el aforo.

Dentro de la clase contamos con un constructor con argumentos y diversos getters y setters para cada uno de sus respectivos atributos.

4.1.2 Clase Usuario

La clase Usuario está formada por los siguientes atributos: cinco strings con el nombre de usuario, la contraseña, el nombre, la carrera y la ubicación de la foto de perfil del usuario, y dos enteros con la edad y el curso actual.

Dentro de la clase tenemos un constructor con argumentos y varios getters y setters para cada uno de sus atributos.

4.1.3 Clase DBHelper

La clase DBHelper es una clase que hereda de la clase SQLiteOpenHelper y se encuentra en la carpeta ***Database***. En esta clase hemos implementado nuestra base de datos con ***SQLite*** la cual se llama ***EventUp.db***.

La base de datos consta de 3 tablas, una llamada ***Usuario*** la cual simula la base de datos de la UGR con los alumnos matriculados, otra es la tabla de ***Evento*** para saber los eventos que han sido creados y por último la tabla ***Usuario-Evento*** en la cual se recogen los eventos que ha aceptado y a los que va a ir un usuario.

4.1.3.1 TABLA USUARIO

La tabla ***Usuario*** tiene 7 atributos, los cuales son, el correo electrónico, la contraseña, asociados con el correo de la UGR, el nombre del usuario, el nombre de la carrera, el curso, la edad y una foto.

Para simular la base de datos hemos añadido 5 usuarios a esta.

4.1.3.2 TABLA EVENTO

La tabla ***Evento*** posee también 7 atributos, el nombre del evento, la fecha, la hora, el aforo permitido, la carrera universitaria si se quiere precisar, la ubicación y el idioma en el que se va a hablar.

Para simular la base de datos y poder manejarla hemos añadido una serie de eventos.

4.1.3.3 TABLA USUARIO-EVENTO

La tabla ***Usuario-Evento*** tiene 2 atributos, los cuales son, el nombre del usuario y el del evento. Esta tabla sirve para saber qué usuarios van a ir a un evento concreto y poder asociar el usuario con el evento.

4.1.3.4 Métodos

La clase tiene una serie de métodos:

- ***iniciarValores***: inicializa los valores de los atributos que añadiremos a las tablas.
- ***crearValores***: crea las tablas con sus claves primarias y atributos y los valores que toman estos
- ***insertarValores***: inserta en las tablas las diferentes sentencias
- ***onCreate***: en este método creamos la base de datos, llamando a los métodos ***crearValores*** e ***insertarValores*** creando las tablas y realizando las inserciones
- ***onUpgrade***: sirve para manejar nuevos cambios si se produce un cambio de versión
- ***crearEvento***: creamos un nuevo evento
- ***eliminarEvento***: eliminamos el evento con nombre específico
- ***modificarEvento***: modificamos un evento concreto
- ***asignarEvento***: añadimos a la tabla ***Usuario-Evento*** el evento y el usuario que acaba de aceptar dicho evento
- ***desasignarEvento***: eliminamos de la tabla ***Usuario-Evento*** el evento asociado al usuario que acaba de rechazar el evento

4.1.4 Clase MyViewHolder

La clase MyViewHolder es una clase que hereda de ***RecyclerView.ViewHolder.*** Esta clase ha sido implementada para visualizar un evento en la vista de "***Mis Eventos***". Es por ello que está formada por cuatro TextView y un botón de tipo Button.

Los cuatro TextView hacen referencia a los datos necesarios para representar un evento en la vista de "***Mis Eventos":*** el nombre del evento, la fecha, la hora y la ubicación.

El atributo Button hace referencia al botón de "***Go***", que al pulsarlo nos mostraría cómo llegar a la ubicación del evento.

La clase además cuenta con un constructor con argumentos, que se encarga de relacionar los atributos con la parte visual mediante sus identificadores.

4.1.5 Clase EventoAdapter

La clase EventoAdapter es una clase que hereda de ***RecyclerView.Adapter<EventoAdapter.MyViewHolder>***. Esta clase ha sido implementada para poder visualizar una lista de eventos en la vista de "***Mis Eventos***".

Por ello la clase está formada por un atributo de la clase "***Context***", un array de elementos de la clase "***Evento***", y cuatro string adicionales con los datos del usuario que necesitamos fijos: el nombre del usuario, el grado, el correo y su foto.

La clase cuenta con un constructor con argumentos y diversos métodos: ***onCreateViewHolder, onBindViewHolder*** y ***getItemCount***.

El método llamado ***onCreateViewHolder*** está implementado por defecto, y el método ***getItemCount*** devuelve el número de eventos que tiene el array de eventos.

Por otro lado, el método llamado ***onBindViewHolder*** se encarga de recoger la información de todos los eventos que tiene un usuario e ir introduciendolos en el array de eventos.

Además, se han implementado dos funcionalidades adicionales: una para que al cliquear en un evento de dicha lista de eventos en la vista "***Mis Eventos***" se cambie a la actividad en la que puedes modificar dicho evento, y otra para que al pulsar al botón de "***Go***" te redirija al Google Maps y te indique la ruta a seguir hasta la ubicación del evento.

4.1.6 Clase MyViewHolderEU

La clase MyViewHolderEU es una clase que hereda de ***RecyclerView.ViewHolder.*** Esta clase ha sido implementada para visualizar un evento en la vista de "***EventUp***". Es por ello que está formada por siete TextView y un botón de tipo Button.

Los siete TextView hacen referencia a los datos necesarios para representar un evento en la vista de "***EventUp***": el nombre del evento, la fecha, la hora, la ubicación, el grado, el idioma y el aforo.

El atributo Button hace referencia al botón de "***Añadir Evento"***, que al pulsarlo nos agregaría el evento en cuestión a la lista de eventos propios.

La clase además cuenta con un constructor con argumentos, que se encarga de relacionar los atributos con la parte visual mediante sus identificadores.

4.1.5 Clase AllEventsAdapter

La clase AllEventsAdapter es una clase que hereda de ***RecyclerView.Adapter <AllEventsAdapter.MyViewHolderEU>***. Esta clase ha sido implementada para poder visualizar una lista de eventos en la vista de "***EventUp***".

Por ello la clase está formada por un atributo de la clase "***Context***", un array de elementos de la clase "***Evento***", y cuatro string adicionales con los datos del usuario que necesitamos fijos: el nombre del usuario, el grado, el correo y su foto.

La clase cuenta con un constructor con argumentos y diversos métodos: ***onCreateViewHolder***, ***onBindViewHolder*** y ***getItemCount***.

El método **onCreateViewHolder** está implementado por defecto, y el método ***getItemCount*** devuelve el número de eventos que tiene el array de eventos.

Por otro lado, el método llamado ***onBindViewHolder*** se encarga de recoger la información de todos los eventos que tiene un usuario e ir introduciendolos en el array de eventos.

Además, se ha implementado una funcionalidad adicional con la cual al pulsar al botón de "***Añadir Evento***" se añade el evento en cuestión a la lista de eventos propios.

4.2 Activities y fragments

La aplicación cuenta con nueve **activities** distintas. Estas son:

- ➔ MainActivity
- ➔ LoginActivity
- ➔ MenuActivity
- ➔ Grado
- ➔ Calificaciones
- ➔ Tui
- ➔ CrearEvento
- ➔ EditarEvento
- ➔ FilterActivity

Asimismo dispone de tres **fragments**, los cuales pertenecen a un menú situado en la parte inferior de la pantalla. Estos son:

- ➔ MisEventos
- ➔ EventUp
- ➔ MiPerfil

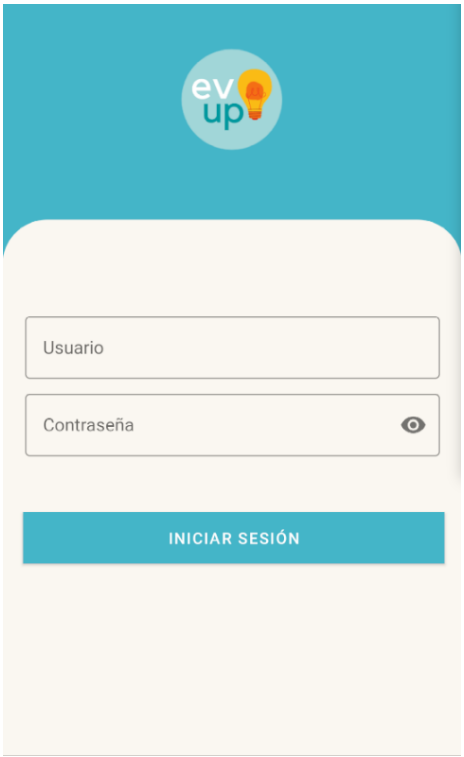
4.2.1 Activity MainActivity

Es la actividad principal que se dispara al abrir la aplicación. En ella se ha implementado una "***Splash Screen***" con el logo de la aplicación simulando un *preloader.* Se ha empleado una animación para hacer su aparición y desaparición.

Para ello, se ha hecho uso de las funciones ***animate***, añadiendoles una traslación, una duración y un tiempo de retardo o *DELAY*.

Posteriormente, se introduce automáticamente en la siguiente pantalla con la vista para introducir las credenciales de acceso.

4.2.2 Activity LoginActivity



En esta actividad el usuario introduce sus credenciales de acceso y se comprueba que los datos introducidos son correctos.

Para ello consta de un método denominado "***loginUser***". En él se comprueba primero que se ha introducido información tanto en el campo de usuario como de contraseña. Si no es así, se envía un error por pantalla. Para ello se usan dos métodos "***validateUsername***" y "***validatePassword***" que comprueban si los campos son vacíos.

A continuación, si ninguno de los dos campos es vacío llamamos a la función "***isUser***" y comprobamos mediante una consulta a la base de datos si las credenciales introducidas corresponden a los datos de un usuario de la Universidad de Granada.

Así pues, si los datos son correctos se pasará a la siguiente pantalla, que es la del menú principal, y sino permanecerá en la misma hasta que se introduzcan datos correctos.

4.2.3 Activity MenuActivity

En esta actividad se muestra un menú en la parte inferior de la pantalla con tres fragments: **Mis Eventos**, **EventUp** y **MiPerfil.**

Para ello hemos creado un archivo tipo ***navigation*** llamado ***my_nav.xml*** donde hemos creado los tres fragments que necesitamos.

Posteriormente, creamos un archivo tipo ***menú*** llamado ***bottom_menu.xml*** en el cual hemos añadido tres ítems, uno para cada fragment, asignándole el icono correspondiente.

Por último, en el layout correspondiente al activity ***MenuActivity***, añadimos un ***BottonNavigationView*** y le asignamos el menú que hemos creado. Dentro de este agregamos un ***fragment*** y le añadimos el grafo de navegación mencionado anteriormente.

4.2.3.1 Fragment Mis Eventos

En este fragment se muestra la lista de eventos que tiene el usuario que ha accedido a la aplicación. Para ello se hace uso de las clases mencionadas anteriormente: ***EventoAdapter*** y ***RecyclerView***.

Además en dicha vista hay también un botón en cuál puedes crear un nuevo evento, y al clickearlo te llevará a una nueva actividad denominada "***CrearEvento***".

Por otro lado, si clickeas en un evento de la lista de evento, te llevará a una nueva actividad denominada "***EditarEvento***" en la cual puedes tanto modificar algún dato del evento como eliminarlo.

Asimismo, puedes clickear solamente el botón que pone "***Go***" para ir a una nueva actividad de Google Maps.

En cuanto a los elementos empleados para visualizar el fragment, hemos utilizado un ***Layout*** para el título, en el cual le hemos insertado un ImageButton, y para los eventos hemos usado un ***RecyclerView*** dentro de un ***ScrollView*** con orientación vertical para poder ir bajando y ver los distintos eventos.

4.2.3.2 Fragment EventUp

En este fragment se muestra la lista de eventos disponibles en los que no está todavía apuntado el usuario. Para ello se hace uso de las clases mencionadas anteriormente: ***AllEventsAdapter*** y ***RecyclerView***.

Además en dicha vista hay también un botón en cuál puedes crear seleccionar filtros para los distintos campos de un evento. Al clickearlo te llevará a una nueva actividad denominada "***FilterActivity***".

De esta forma, cada vez que abras el fragment "***EventUp***" se mostrarán los eventos en los últimos filtros que añadiste. Esto es posible ya que al crear la vista llamamos a una función llamada "***hayFiltros***" que comprueba si había filtros anteriormente y realizamos la búsqueda de eventos en la base de datos según esos filtros.

Por otro lado, si clickeas en el botón "***Añadir Evento***" dentro de cada evento disponible, te añadirá dicho evento a la lista de tus eventos.

En cuanto a los elementos empleados para visualizar el fragment, hemos usado un ***Layout*** para el título, en el cual le hemos añadido un ImageButton, y para los eventos disponibles hemos usado un ***RecyclerView*** dentro de un ***LinearLayout*** con orientación horizontal.

4.2.3.1 Fragment Mi Perfil

En este fragment se muestra la información personal del usuario: el nombre, el grado universitario y una foto.

Este fragment cuenta a su vez con tres botones con distintas funcionalidades:

1. El primer botón te dirige a otra actividad con información acerca de la facultad del estudiante: "***Grado***".
2. El segundo te dirige a una actividad con las calificaciones del mismo: "***Calificaciones***".

3. El tercer botón te dirige a otra actividad con la TUI virtual: "***Tui***".

Además, hay otro menú en la esquina superior derecha con la opción de cerrar sesión. Al clickear en dicha opción te redirige a la actividad "**LoginActivity**".

En cuanto a los elementos empleados para visualizar dicho fragment, hemos utilizado un ***AppBarLayout*** para la cabecera. En él, hemos añadido un ***Toolbar*** al cual le hemos adjuntado un menú personalizado en la esquina superior derecha, con el ítem de "***Cerrar Sesión***", y le hemos añadido además el título del fragment.

A continuación, tenemos un ***CardView*** con la imagen del estudiante y distintos TextViews e ImageViews con información sobre el mismo. Por último, hay tres botones del tipo ***Button*** para realizar las funcionalidades mencionadas anteriormente.

4.2.4 Activity Grado

En esta actividad se muestra información sobre la facultad del alumno, y dispone de un botón para poder acceder a Google Maps y que te muestre el camino hasta tu respectiva facultad.

Esta actividad cuenta con un botón para volver al fragment "***Mi Perfil***".

4.2.5 Activity Calificaciones

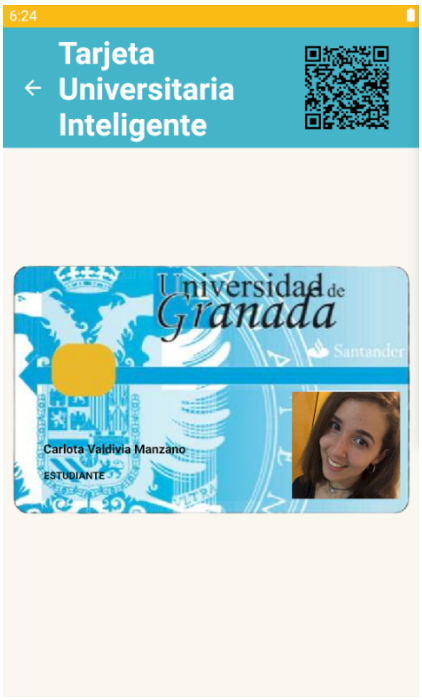
En esta actividad se muestran tus calificaciones del cuatrimestre que estás cursando. Además de la media ponderada de las notas obtenidas.

Esta actividad cuenta con un botón para volver al fragment "Mi Perfil".

4.2.6 Activity Tui

En esta actividad se muestra la Tarjeta Universitaria Inteligente del estudiante.

Esta actividad cuenta con un botón para volver al fragment "Mi Perfil".



4.2.7 Activity CrearEvento

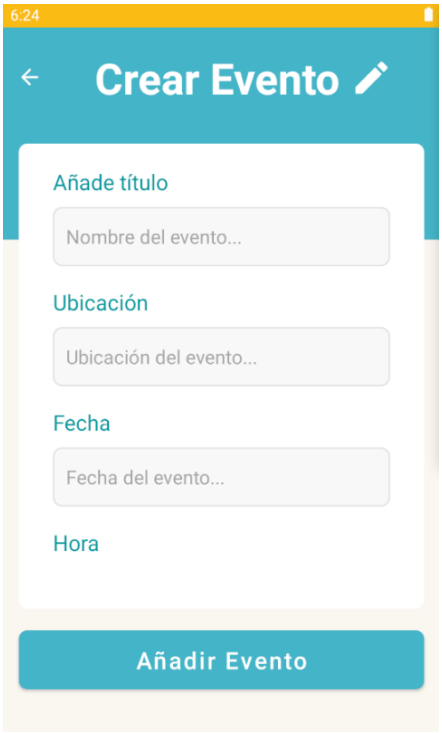
En esta actividad se muestran los distintos campos para añadir la información del evento que se quiere crear. Hay cuatro campos que son obligatorios para poder crear el evento: el nombre, la fecha, la hora y la ubicación.

Cuenta con dos botones: uno para crear el evento y otro para volver atrás. Ambos botones vuelven al fragment "**Mis Eventos**".

En cuanto a los elementos empleados para visualizar el activity, hemos usado un ***LinealLayout*** para el título, en el cual le hemos añadido un ***ImageButton*** para volver hacia atrás.

Para los campos para crear el evento hemos usado una serie de ***LinealLayouts***, compuestos por un ***TextView*** y un ***EditText***, dentro de un ***ScrollView*** con orientación vertical.

Por último hemos usado un ***Button*** para el botón de "***Añadir Evento".***



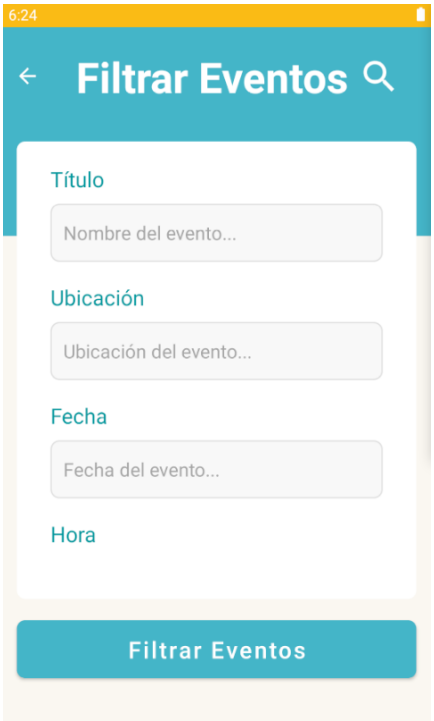
4.2.8 Activity EditarEvento

En esta actividad se muestran los distintos campos para modificar la información del evento que se quiere editar. El nombre del evento no puede ser modificado.

Cuenta con tres botones: uno para modificar el evento, otro para eliminarlo y otro para volver atrás. Todos ellos regresan al fragment "**Mis Eventos**".

En cuanto a los elementos empleados para visualizar el activity, hemos usado prácticamente los mismos que en **CrearEvento**, con la diferencia que hemos añadido un ***Button*** para "***Eliminar Evento***".

4.2.9 Activity FilterActiviy



En esta actividad se muestran los distintos campos con los que filtrar los eventos que deseas que te aparezcan. Si no hay ningún evento con esos filtros no saldrá ningún evento disponible al volver al fragment "***EventUp***".

Cuenta con dos botones: uno para filtrar los eventos y otro para volver atrás. Ambos botones vuelven al fragment "***EventUp***".

En cuanto a los elementos empleados para visualizar el activity, hemos usado prácticamente los mismos que en **CrearEvento**.

4.3 Sensores

Ahora procedemos a explicar los sensores que hemos implementado en nuestra aplicación y el uso que les hemos dado.

En primer lugar debemos definir en el archivo ***AndroidManifest.xml*** aquellos sensores que vamos a utilizar, una vez hecho esto, comprobaremos cuando se requiera el uso de alguno si el móvil tiene el sensor incorporado, mostrando un mensaje en caso de no incorporar el sensor.

Hemos implementado para cada sensor los métodos **onResume** y **onPause**, el primero se llama cuando la actividad va a comenzar a interactuar con el usuario y el segundo que la actividad está a punto de ser lanzada a segundo plano.

A continuación presentamos los sensores que hemos incorporado a la aplicación:

4.3.1 Sensor de Proximidad

La finalidad del sensor es poder crear un evento sin necesidad de tener que pulsar dentro del Fragment de ***MisEventos*** en el icono crear nuevo evento.

Cuando al usuario le apetezca crear un evento podrá acercar la mano al sensor y directamente podrá empezar a introducir los datos del nuevo evento solo podrá realizar esta acción si está en los fragments.

El sensor ha sido implementado en la clase ***MenuActivity*** para poder realizar dicha acción en cualquier fragment de la aplicación.

Para poner en funcionamiento el sensor hemos creado una instancia de la clase ***SensorManager,*** la cual hace uso del método **getSystemService()** cuyo argumento **SENSOR_SERVICE** sirve para poder usar cualquier sensor del dispositivo.

Para crear una instancia del sensor ***Proximidad*** llamamos a la función **getDefaultSensor()** y le indicamos el sensor deseado en nuestro caso el de proximidad, por tanto pondremos, **Sensor.TYPE_PROXIMITY**.

Una vez hecho esto, comprobamos que existe el sensor y creamos una instancia de **SensorEventListener** teniendo que implementar los métodos **onSensorChanged** y **onAccuracyChanged**, el segundo lo dejaremos vacío. En el método **onSensorChanged** comprobamos si hemos acercado la mano al sensor, si la acercamos lo suficiente cambiamos la vista.

Para poder cambiar la vista de forma correcta lo que hacemos es comprobar si la variable booleana cambio está a true y si es así creamos una instancia de **Intent** y pasamos la información del usuario a la nueva vista.

4.3.2 Acelerómetro

La finalidad del uso del sensor **Acelerómetro** es salir de la aplicación sin necesidad de tener que pulsar el botón cerrar sesión.

Cuando el usuario desee salir de la aplicación realizará un movimiento rápido para atrás y para adelante con el teléfono móvil volviendo a la pantalla de inicio de sesión.

El sensor al igual que el de **Proximidad** ha sido implementado en la clase ***MenuActivity***.

Para poner en funcionamiento el sensor realizamos el mismo procedimiento que con el sensor de **Proximidad** pero esta vez al llamar a la función **getDefaultSensor()** el sensor que le indicamos es el acelerómetro con **Sensor.TYPE_ACCELEROMETER**.

Una vez realizado esto, comprobamos que existe el sensor y creamos una nueva instancia de **SensorEventListener** y volvemos a implementar los dos métodos como en el sensor de **Proximidad**. En el método **onSensorChanged** comprobamos si hemos realizado un movimiento hacia atrás y posteriormente hacia delante y que el tiempo que hemos tardado en realizar los 2 movimientos es en un corto espacio de tiempo, si es así saldremos de la aplicación.

4.3.3 Magnetómetro

La finalidad del uso del sensor Magnetómetro es conocer la latitud y longitud en la que se encuentra el móvil para así poder guiar al usuario hacia la localización de sus eventos, o hacia localizaciones predefinidas por nosotros en la aplicación.

Cuando el usuario desee ir a la localización de alguno de sus eventos, simplemente tendrá que pulsar el botón **GO**. Justo después, su aplicación Google Maps será abierta con una ruta seleccionada. Esa ruta tendrá como origen su localización actual, y como destino la ubicación del evento.

Este sensor ha sido implementado en dos actividades distintas:

- Abrir localización de eventos: **FragmentActivity** de **MisEventos**.
- Abrir localización predefinida: actividad **Grado**.

Para que el sensor funcione, tenemos que comprobar primeramente que los permisos necesarios están concedidos (P**ackageManager.PERMISSION_GRANTED**). Hemos requerido los dos permisos de localización disponibles para que el punto de partida de la ruta sea lo más exacto posible. Posteriormente, solo tenemos que instanciar el **LOCATION_SERVICE** y usar su **GPS_PROVIDER** para obtener la latitud y la longitud.

4.3.4 Giroscopio

La finalidad del uso del sensor Giroscopio es poder crear un nuevo evento o eliminar un evento de mis eventos a partir de un par de giros concatenados en un intervalo de tiempo. El movimiento para aceptar un evento consiste en inclinar el móvil hacia atrás y posteriormente hacia delante simulando un sí que hacemos con la cabeza, es decir, realizamos un movimiento en el eje x y con esto activamos el botón “Añadir Evento”. Para poder rechazar un evento debemos realizar un movimiento hacia la izquierda seguido de un movimiento a la derecha. Cuando se detecta este movimiento en el eje z, se activa el botón “Eliminar evento”.

Para activar el sensor hacemos uso de **getDefaultSensor(Sensor.*TYPE_GYROSCOPE*)**. El sensor está implementado en dos archivos, ***CrearEvento*** y ***EditarEvento*** mediante las clases **SensorManager** y **Sensor**. Tenemos que crear un **listener** para que cada vez que cambie el sensor se active el botón correspondiente.

Para realizar los movimientos comprobamos que la diferencia en tiempo entre realizar un movimiento hacia atrás y hacia delante o hacia la derecha y hacia la izquierda no supera una cierta cota calculada repitiendo el proceso hasta obtener un tiempo acorde al movimiento. Si esto sucede activamos el botón pertinente con la función **performCLick()** y mostramos un mensaje por pantalla indicando que hemos creado un nuevo evento o eliminado el evento que habíamos seleccionado.