



Título del Proyecto:

SportsMeet

Autor:

Pérez González, Juan Carlos

Director:

Puga Sabio, Gonzalo

TESINA PARA LA OBTENCIÓN DEL TÍTULO DE:

Máster en Desarrollo de Aplicaciones sobre Dispositivos Móviles

Septiembre del 2016



Contenido

| Título del Proyecto: |
|--|
| Autor: |
| Director: |
| Máster en Desarrollo de Aplicaciones sobre Dispositivos Móviles1 |
| Introducción |
| Descripción del problema3 |
| Objetivos |
| Motivación3 |
| Tecnologías utilizadas4 |
| Arquitectura de la aplicación5 |
| Esquema del diseño5 |
| Modelo de datos |
| Servicios Web14 |
| Vistas |
| Alarmas y Notificaciones |
| Conclusiones |
| Anexos |
| Manual de usuario |
| Listado de fuentes entregadas / Código fuente en GitHub45 |
| Glosario de Términos |



Introducción

En las siguientes líneas haremos una breve introducción al presente proyecto mediante la descripción del problema, qué objetivos persigue y de cuál es su motivación.

Descripción del problema

Actualmente correr está de moda, hay muchas personas que salen a corren varios días por semana, muchos de ellos en solitario, por lo que podría ser buena idea crear una aplicación que les ayudase a poder correr en compañía y con ello motivarse unos a otros para alcanzar sus objetivos (mejorar tiempos, terminar una prueba,...).

Por otro lado como corredor aficionado he participado en algunas carreras populares, donde se publicaba el recorrido de la carrera en la web, pero luego no había un modo "fácil" de poder realizar entrenos siguiendo dicho recorrido. Por ello podría ser interesante que disponer de una aplicación que grabase los recorridos o se pudiesen importar a partir de ficheros de rutas GPS y luego se pudiesen seguir en un mapa a través de nuestro dispositivo móvil.

Una vez tenido en cuenta todo lo anterior había que preguntarse por qué deberíamos limitar la aplicación solo a correr cuando la podemos extender a más deportes, por lo que la aplicación debería dar soporte a otras actividades deportivas y ayudar a fomentar con ello la práctica del deporte en general.

Objetivos

El objetivo principal del presente trabajo es el desarrollo de una aplicación para dispositivos móviles Android (a la que llamaremos SportsMeet) que permita a los usuarios organizar "quedadas" deportivas (partidos, carreras, actividades....) y apuntarse a las ya existentes para participar en ellas.

Con dicha aplicación se podrá crear un evento deportivo (marcha de senderismo, entrenamiento de "running", partido de padel,....) indicando el lugar, fecha, hora y una breve descripción del mismo. Así mismo se podrá indicar el recorrido de la actividad (siempre que el deporte del evento sea susceptible de tener recorrido como por ejemplo correr o ciclismo), grabando directamente la ruta a través de la aplicación o importando un fichero con la ruta. Además se avisará a los usuarios de la creación/modificación de los eventos para que puedan apuntarse a los mismos.

Para alcanzar dicho objetivo haremos uso del sistema de localización GPS, almacenaremos los eventos y las rutas de los mismos en BBDD, servicios de Google, enviaremos notificaciones a los usuarios,... Con esto pretendemos alcanzar el segundo objetivo del presente proyecto: poner en práctica los conocimientos adquiridos durante el curso.

Motivación

La motivación que me ha llevado a desarrollar este proyecto es una motivación personal surgida de mi afición por correr y el deporte en general. Aburrido de correr sólo, pensé en una



aplicación que me ayudase a organizar "quedadas" y poder entrenar con otras personas aficionadas a correr.

Además de lo anterior también me decidí por este proyecto porque me permitía poner en práctica varios de los conocimientos adquiridos en las distintas asignaturas (Material Design, notificaciones, tratamiento de XML, acceso a los servicios de Google,....)

Tecnologías utilizadas

En este apartado realizaremos una breve descripción de las tecnologías que se han usado para la realización del proyecto:

- Android: la aplicación se ejecutará en dispositivos Android y ha sido implementada usando Android Studio
- Firebase: para el login, el almacenamiento de datos e imagenes y el envío de notificaciones:
 - Firebase Authentication: nos permite autenticar a los usuarios usando su cuenta de Google, Facebook, Twitter y GitHub. En este caso hemos utilizado sólo la cuenta de Google ya qur al estar dirigida a dispositivos Android los usuarios ya deberían de tener cuenta en Google.
 - o Firebase RealTime Database: base de datos NoSQL alojada en la nube
 - Firebase Storage: nos permite subir y descargar archivos en la nube de forma segura
 - Firebase Cloud Messaging servicio gratuito para enviar notificaciones a los usuarios.
- JSON: para el intercambio de información
- Formatos para importar rutas desde archivos basados en XML:
 - GPX (GPS eXchange Format)
 - TCX (Training Center XML)
 - KML (Keyhole Markup Language)
- XML (procesado con SAX): para la lectura de los ficheros TCX, GPX y KML con los datos de las rutas y para tratar el fichero devuelto por Directions API.
- GPS: utilizado para saber la posición del usuario y la grabación de las rutas.
- Google Maps API v2 para los mapas
- Google Maps Directions API que calcula indicaciones entre ubicaciones y lo usamos para mostrar el camino a seguir desde nuestra ubicación a la salida de la ruta del evento.

Además se ha pretendido seguir las recomendaciones de Material Design a la hora de diseñar cada una de las vistas de la aplicación.



Arquitectura de la aplicación

Esquema del diseño

En este apartado haremos una descripción del software comentando más detalladamente aquellas partes más interesantes. Empezaremos mostrando el esquema general de la aplicación:



Utilizamos Firebase como base de datos (donde almacenamos las actividades deportivas, las rutas,...), para almacenar las imágenes de los eventos deportivos, enviar notificaciones para avisar de modificaciones en los eventos y para administrar la autentificación de usuarios, los cuales utilizan su cuenta de Google para el login.

A continuación mostraremos los diagramas de clases de los Activity principales de la aplicación:

UML Activity MainActivity (Actividad principal desde donde se realiza el login de la aplicación)







UML Activity TablonActivity (Actividad que muestra el listado de eventos deportivos)

UML Activity DetalleActivity (Actividad que muestra los datos del evento deportivo seleccionado por el usuario)



UML Activity DetalleMapsActivity (muestra el detalle del lugar donde se ha quedado para el evento deportivo, la ruta si la hubiese y nos ayuda a realizar un entreno sobre dicha ruta)





UML Activity BusquedaMapaActivity (permite ver todos los eventos en un mapa centrado en nuestra ubicación actual, también permite realizar búsquedas por direcciones)



UML Activity EditarActivity (en esta Actividad se puede crear un evento deportivo para compartirlo con los demás usuarios o modificar uno existente)





UML Activity RutaActivity (nos permite grabar una ruta a partir de la información recogida por el GPS)



En los diagramas anteriores se ha quitado la clase Utils.java para simplificarlos. Esta es usada de forma estática por la mayoría de las clases ya que contiene las constantes de la aplicación (los estados de la grabación, la url para las conexiones con *Firebase*,...) y métodos comunes a la misma (*getDatabase* para obtener una referencia a la base de datos de *Firebase*, *converMSegToMinKm* para pasar de metros por segundo a minutos por kilómetro, *isDeporteConRuta* indica si el deporte es susceptible de tener ruta,...):



Hemos descrito brevemente lo que hace cada Acitivity y mostrado su diagrama de clases. A continuación resumiremos lo que hace el resto de clases mostradas en los diagramas:



- AdaptadorEventos (está clase es la que se encarga de rellenar el RecyclerView con la lista de eventos)
- Aplicacion (se encarga de obtener y mantener los datos que se pasarán a la clase AdaptadorEventos para que los muestre)
- BBDDAlarmas (gestiona la base de datos con las alarmas programadas para el usuario)
- CustomSpinnerAdapter (clase que implementa un combo a medida acorde con el estilo de la aplicación)
- DividierItemDecoration (dibuja la línea que separa los eventos unos de otros en el listado)
- Evento (POJO que representa los eventos deportivos)
- GestorNotificaciones (centraliza la gestión de las notificaciones y alarmas de la aplcicación)
- HeightAnimation (clase utilizada para animar el relleno de un ImageView con la imagen seleccionada por el usuario)
- ManejadorDirectionsAPI (procesa mediante SAX el archive XML devuelto por Google Directions API y obtiene los datos de la ruta)
- ManejadorGPX(procesa mediante SAX un archivo GPX y obtiene los datos de la ruta)
- ManejadorKML(procesa mediante SAX un archivo KML y obtiene los datos de la ruta)
- ManejadorTCX(procesa mediante SAX un archivo TCX y obtiene los datos de la ruta)
- PosicionesGPS(POJO que representa la información referente a los datos obtenidos por el GPS además de información referente a la ruta importante para dibujar y obtener los recorridos del usuario)
- Usuario (POJO que representa al usuario logado)

Modelo de datos

Para guardar los datos de los eventos deportivos utilizamos *Firebase RealTime Database*, que es una base de datos *NoSQL* alojada en la nube. Con *Firebase* los datos se almacenan en formato *JSON* y se sincronizan en tiempo real con cada cliente conectado, lo que nos permitirá fácilmente que cada usuario tenga la información de cada evento actualizada. Firebase permite aplicaciones multiplataforma (iOS, Android y Web), escalabilidad y posibilidad de seguir funcionado sin conexión a la red (hasta que se vuelva a tener conexión se trabaja con una caché y al recuperar la conexión se actualiza la información). Dispone de una consola administrativa que nos permite consultar y administrar los datos y reglas de acceso y uso:

| altim | e Database | | | |
|-------|-----------------|-----------------------------------|---|---|
| os | REGLAS | USO | | |
| | | | | |
| Θ | https://fir-sp | tsmeet.firebaseio.com/ | Φ | Θ |
| fir- | sportsmeet | | | |
| ÷. | apuntados | | | |
| | 0 109260 | 32763812455459 | | |
| ÷. | - eventos | | | |
| | 007b1 | 8-5356-4b6c-9c22-502a3177bd1b | | |
| | 04e031 | a-62d0-4423-97b5-8dbbec0fb37b | | |
| | 0b3824 | 9-7771-4b24-9d05-d83f28ed68f8 | | |
| | 0 304161 | 2-cc80-4d81-929a-dc94248f6997 | | |
| | G- 6fce2e | -b0dd-4b38-8cc5-4e887a437815 | | |
| | 🖸 9adb0t | 9-24a3-4107-9611-3947dcf79f00 | | |
| | 🖬 b176a | 2-b7f7-4f84-ac9e-ed7e7b99df50 | | |
| | 🖬 d7237i | a-d1c8-4ea1-b842-95603352142d | | |
| ė | - rutas | | | |
| | 🗖 ruta_00 | b1db8-5356-4b6c-9c22-502a3177bd1b | | |
| | 🗖 – ruta_04 | 0311a-62d0-4423-97b5-8dbbec0fb37b | | |
| | 🖬 ruta_01 | 82459-7771-4b24-9d05-d83f28ed68f8 | | |
| | - ruta_24 | f964c-2664-4629-932b-ace5141fe1f4 | | |
| | G ruta_5a | fc9ee-6e5d-46c2-bd24-0d1316ffc592 | | |



En nuestra aplicación tenemos dos "tablas" una para los eventos deportivos (Eventos) y otra para las rutas asociadas a dichos eventos ("Rutas"). Como los datos los almacenamos en formato JSON en una base de datos NoSQL el esquema varia un poco respecto a cómo sería en una base de datos relacional tradicional. Así los datos almacenados tienen la siguiente estructura:



Para el acceso a la base de datos y la subida de archivos utilizamos el propio API de Firebase. Si bien para obtener el objeto *FirebaseDatabase* siempre se llama al método estático *getDatabase* de la clase *Utils* para gestionar correctamente el acceso offline a los datos cuando no tenemos conexión a internet (*setPersistenceEnabled*(*true*)) y hacer uso del a caché:

```
private static FirebaseDatabase mDatabase;
public static FirebaseDatabase getDatabase() {
    if (mDatabase == null) {
        mDatabase = FirebaseDatabase.getInstance();
        mDatabase.setPersistenceEnabled(true);
    }
    return mDatabase;
}
```



A continuación mostramos el código para consultar los datos de los eventos deportivos. Una vez construida la *Query* a ejecutar, le asociamos un receptor de eventos (*ChildEventListener*) y sobrescribimos los métodos *onChildAdded*, *onChildChanged*, *onChildRemoved*,... que son llamados automáticamente cada vez que se produce un cambio en la base de datos (cuando se añade un evento, cuando se cambia un dato, cuando se borra un dato....), por lo que en estos métodos iremos actualizando el contenido del RecyclerView donde se muestran dichos eventos. La función *actualizaEventos* se encarga de recoger el *DataSnapshot* que devuelve el API de *Firebase* con los datos del registro, transformarlo a un objeto *Evento* y refrescar el *RecyclerView*:

```
FirebaseDatabase bbddevento = Utils.getDatabase();
//construimos la consulta para obtener los datos según la pestaña seleccionada
if (tipoPestana == Utils. PESTANA TODOS) {
    //recuperamos todos los eventos disponibles
    queryRef = bbddevento.getReference().child("eventos")
                                        .orderByChild("fechaordenacion");
}else {
    if (tipoPestana == Utils. PESTANA MIOS) {
        //recuperamos los eventos creados por mi
        queryRef = bbddevento.getReference().child("eventos")
                                  .orderByChild("idCreador")
                                  .equalTo(Usuario.id);
    } else if (tipoPestana == Utils.PESTANA APUNTADOS) {
        //recuperamos los eventos en los que estoy apuntado
        queryRef = bbddevento.getReference().child("eventos")
                                  .orderByChild("apuntado " + Usuario.id)
                                   .equalTo("1");
    }
}
//asociamos el receptor de eventos
queryRef.addChildEventListener(new ChildEventListener() {
    @Override
    public void onChildAdded(DataSnapshot snapshot, String previousChildKey) {
        Log.d(Utils.TAG, "onChildAdded");
        //se ha añadido un evento y lo añadimos al RecyclerView
        actualizaEventos(snapshot, progreso, false);
    }
    @Override
   public void onChildChanged(DataSnapshot dataSnapshot, String s) {
        Log.d(Utils.TAG, "onchildchanged");
        //se ha modificado un evento y lo actualizamos en el RecyclerView
        actualizaEventos(dataSnapshot, progreso, true);
    }
    @Override
    public void onChildRemoved(DataSnapshot snapshot) {
        String clave= snapshot.child("clave").getValue().toString();
        Log.d(Utils.TAG, "onchildremoved "+ clave);
        if (tablaIndices.containsKey(clave)) {
            //se ha borrado un evento actualizamos el RecyclerView
            //con notifyDataSetChanged
            Integer indice= tablaIndices.get(clave);
            for (int i=indice+1;i<vectorDeportes.size();i++) {</pre>
                tablaIndices.put(vectorDeportes.get(i).getClave(),(i-1));
            }
            vectorDeportes.remove(indice.intValue());
            tablaIndices.remove(clave);
            adaptador.notifyDataSetChanged();
        }
    }
    @Override
```

```
UNIVERSITAT
                                 Alumno: nombre
         POLITÈCNICA
         DE VALÈNCIA
   public void onChildMoved(DataSnapshot dataSnapshot, String s) {
        Log.d(Utils.TAG, "onchildmoved");
    }
    @Override
    public void onCancelled(DatabaseError databaseError) {
        Log.d(Utils.TAG, "onchildcancelled");
        progreso.dismiss();
    }
});
//Recupera los datos obtenidos en la consulta, construye un objeto Evento y
//actualiza el RecyclerView
public void actualizaEventos (DataSnapshot snapshot, ProgressDialog progreso, boolean
isModificacion) {
   Evento e = new Evento();
    if (bInserto && e!=null && snapshot!=null) {
         //obtenemos los datos del DataSnapshot
        if (snapshot.child("lugar") != null &&
        snapshot.child("lugar").getValue() != null) {
            e.setLugar(snapshot.child("lugar").getValue().toString());
        }
        if (snapshot.child("fecha") != null &&
                                                   snapshot.child("fecha").getValue()
!= null) {
            e.setFecha(snapshot.child("fecha").getValue().toString());
        }
        e.setDeporte(snapshot.child("deporte").getValue().toString());
        if (snapshot.child("idCreador") != null &&
        snapshot.child("idCreador").getValue() != null) {
            e.setIdCreador(snapshot.child("idCreador").getValue().toString());
        }
        e.setclave(snapshot.child("clave").getValue().toString());
        e.setDescripcion(snapshot.child("descripcion").getValue().toString());
        e.setHora(snapshot.child("hora").getValue().toString());
        e.setTitulo(snapshot.child("titulo").getValue().toString());
        if (snapshot.child("latitud") != null &&
            snapshot.child("latitud").getValue() != null) {
                 e.setLatitud(Double.parseDouble(snapshot.child("latitud")
                                           .getValue().toString()));
        if(snapshot.child("longitud") != null &&
           snapshot.child("longitud").getValue() != null) {
                 e.setLongitud(Double.parseDouble(snapshot.child("longitud")
                                  .getValue().toString()));
        e.setNumapuntados((int)snapshot.getChildrenCount()-
                          Utils.NUM ELEM EVENTOS);
         if (isModificacion ||
                 tablaIndices.containsKey(snapshot.child("clave")
                 .getValue().toString())) {
                 //es una modificacion
                Integer indice = tablaIndices.get(e.getClave());
                if (vectorDeportes!=null && indice!=null &&
                 vectorDeportes.size()>=indice && vectorDeportes.size()>0) {
                    Log.d(Utils.TAG, "modificación " + indice);
                    vectorDeportes.set(indice.intValue(), e);
                }
        } else {
        //es un alta
        Log.d(Utils.TAG, "alta");
        tablaIndices.put(e.getClave(), vectorDeportes.size());
        vectorDeportes.add(e);
    }
```

Provecto: titulo



```
progreso.dismiss();
adaptador.notifyDataSetChanged();
}
```

Ejemplo código para almacenar los datos de un evento deportivo, donde a partir de una *DatabaseReference* vamos subiendo los datos del objeto Evento a la base de datos mediante el método .child(parametro).setValue(valor):

```
DatabaseReference rootRef = Utils.getDatabase().getReference();
DatabaseReference bbddeventos = rootRef.child("eventos").child(claveUUID);
bbddeventos.child("deporte").setValue(evento.getDeporte());
bbddeventos.child("descripcion").setValue(evento.getDescripcion());
bbddeventos.child("fecha").setValue(evento.getFecha());
```

En cuanto a las búsquedas hay que indicar que no ha sido posible implementar consultas "*like*" (obtener todos los datos que contienen determinada cadena) como las que se realizan en bases de datos relacionales. En su lugar se obtienen todos los datos que comienzan por el filtro introducido haciendo uso de las funciones *startAt* y *endAt*:

Además en la aplicación podemos asociar una fotografía a cada evento, esta imagen la almacenamos utilizando Firebase Storage. Para subir las imágenes y poder descargarlas posteriormente las almacenamos con el siguiente nombre: "evento_" + clave uuid del evento al que pertenece como se ve en la siguiente imagen de la consola de Firebase Storage:

| ARCHIVO | S REG | LAS | | | | |
|---------|-------------|--|-----------|------------|---------------------|---|
| | | | | | | |
| | gs://fireba | se-sportsmeet.appspot.com > images | | 1 | SUBIR ARCHIVO | Ð |
| | | Nombre | Tamaño | Тіро | Última modificación | |
| | | | | | | |
| | | evento_007b1db8-5356-4b6c-9c22-502a3177 | 289,55 KB | image/jpeg | 8 sept. 2016 | |
| | | evento_04e0311a-62d0-4423-97b5-8dbbec0f | 284,67 KB | image/jpeg | 8 sept. 2016 | |
| | | evento_0b382459-7771-4b24-9d05-d83f28ed | 339,81 KB | image/jpeg | 5 sept. 2016 | |
| | | evento_1911d265-faf4-4e62-9f1b-c2a46dbbf | 105,78 KB | image/jpeg | 24 ago. 2016 | |

Código utilizado para subir una imagen:

```
// Obtenemos la referencia donde se guardan los archivos de la app
StorageReference storageRef = FirebaseStorage.getInstance().getReference();
// creamos la refencia que tendrá la imagen
StorageReference imgRef = storageRef.child("images/"+nombre+".jpg");
// subimos los datos
UploadTask uploadTask = imgRef.putBytes(data);
uploadTask
```



```
.addOnFailureListener(new OnFailureListener() {
    @Override
    public void onFailure(@NonNull Exception exception) {
        //se lanza cuando ha habido un error en la subida
    }
}).addOnSuccessListener(new
OnSuccessListener<UploadTask.TaskSnapshot>() {
    @Override
    public void onSuccess(UploadTask.TaskSnapshot taskSnapshot) {
        //se ha subido correctamente
        Uri downloadUrl = taskSnapshot.getDownloadUrl();
    }
});
```

Código utilizado para descargar una imagen y mostrarla en el ImageView:

```
// Obtenemos la referencia donde se guardan los archivos de la app
StorageReference storageRef = FirebaseStorage.getInstance().getReference();
// obtenemos la refencia donde debería estar almacenada la imagen del evento
StorageReference imgRef = storageRef.child("images/evento_"+evento.getClave()+".jpg");
```

```
final long ONE MEGABYTE = 1024 * 1024;
```

```
imgRef.getBytes(ONE MEGABYTE).addOnSuccessListener(new
                                                          OnSuccessListener<byte[]>()
{
            Override
            public void onSuccess(byte[] bytes) {
               //Se ha descargado con éxito
               Bitmap bitmap = BitmapFactory.decodeByteArray
                                 (bytes, 0, bytes.length);
               //cargamos la imagen en el ImageView
               iv.setImageBitmap(bitmap);
               progreso.dismiss();
   }
}).addOnFailureListener(new OnFailureListener() {
   QOverride
   public void onFailure( Exception exception) {
        //ha habido un error en la descarga
        progreso.dismiss();
   }
});
```

Por otro lado también tenemos otra base de datos local donde almacenamos los identificadores de las alarmas programadas (se lanzan 24 horas antes del inicio de una actividad deportiva) haciendo uso de la base de datos SQLITE que incorpora Android. Almacenamos estos identificadores para poder cancelar la alarma si el usuario así lo decide o se desapunta de una actividad deportiva. La clase que la gestiona es *BBDDAlarmas* que hereda de *SQLiteOpenHelper*. En esta base de datos únicamente tendremos una tabla ("Alarmas") con la siguiente estructura:



Servicios Web

En cuanto a servicios web usamos Google Maps Directions API, que es un servicio que calcula indicaciones entre ubicaciones usando una solicitud HTTP. Accedemos a http://maps.google.com/maps/api/directions/xml con la latitud y longitud de la posición actual del



usuario y la latitud y posición de la salida de la ruta del evento. En el siguiente código se muestra como se construye la url a la que vamos a acceder (clase *DetalleMapsActivity*):

```
urlString.append("http://maps.google.com/maps/api/directions/xml?origin=");
urlString.append(mejorLocaliz.getLatitude());
urlString.append(",");
urlString.append(mejorLocaliz.getLongitude());
urlString.append("&destination=");
urlString.append(posSal.getLatitud());
urlString.append(",");
urlString.append(posSal.getLongitud());
urlString.append("&sensor=true&mode=walking&output=dragdir");
```

El acceso a este servicio lo hacemos en la clase interna *DescargarRuta* (de la clase *DetalleMapsActivity*) la cual hereda de *AsynTask*:

```
private class DescargarRuta extends AsyncTask<String, Void, String> {
    @Override
    protected String doInBackground(String... param) {
        String data = "";
        trv{
             //obtenemos el XML de salida del servicio
            data = downloadUrl(param[0]);
        } catch (Exception e) {
            Log.d("Background Task", e.toString());
        }
        return data;
    }
@Override
protected void onPostExecute(String result) {
    super.onPostExecute(result);
    try{
        //leemos y procesamos los datos de la ruta del xml obtenido
        SAXParserFactory fabrica = SAXParserFactory.newInstance();
        SAXParser parser = fabrica.newSAXParser();
        XMLReader lector = parser.getXMLReader();
        ManejadorDirectionsAPI manejadorXML = new ManejadorDirectionsAPI();
        lector.setContentHandler(manejadorXML);
        lector.parse(new InputSource(new StringReader(result)));
        ArrayList<ArrayList<LatLng>> points =
                 manejadorXML.getArrPosicionesGPS();
        //dibujamos la ruta en el mapa
        PolylineOptions lineOptions = null;
        MarkerOptions markerOptions = new MarkerOptions();
        lineOptions = new PolylineOptions();
        for(int i=0;i<points.size();i++) {</pre>
            lineOptions.addAll(points.get(i));
            lineOptions.width(8);
            lineOptions.color(Color.CYAN);
        if (rutaHastaSalida !=null) rutaHastaSalida.remove();
        rutaHastaSalida= mMap.addPolyline(lineOptions);
    } catch (Exception e) {
        Log.d("Background Task",e.toString());
}
//Esta función se encarga de conectarse a la url del servicio y obtener la //respuesta
```



```
private String downloadUrl(String strUrl) throws IOException {
    String data = "";
    InputStream iStream = null;
    HttpURLConnection urlConnection = null;
    try{
        URL url = new URL(strUrl);
        urlConnection = (HttpURLConnection) url.openConnection();
        urlConnection.connect();
        iStream = urlConnection.getInputStream();
        BufferedReader br =new BufferedReader(new InputStreamReader(iStream));
        StringBuffer sb = new StringBuffer();
        String line = "";
        while( ( line = br.readLine()) != null) {
            sb.append(line);
        }
        data = sb.toString();
        br.close();
    } catch (Exception e) {
        Log.d(Utils.TAG, e.getMessage());
    }finally{
        iStream.close();
        urlConnection.disconnect();
    1
    return data;
}
}
```

Así un ejemplo de la url creada sería

http://maps.google.com/maps/api/directions/xml?origin=40.16565204,-3.89660868&destination=40.16565270,-3.89660870&sensor=true&mode=walking&output=dragdir y la respuesta obtenida es un archivo XML como el que se muestra a continuación:

```
<?xml version="1.0" encoding="UTF-8" ?>
<DirectionsResponse>
       <status>OK</status>
       <route>
             <summary>Calle Borox</summary>
             <leq>
                    <step>
                          <travel mode>WALKING</travel mode>
                          <start_location>
                                  <lat>40.1658566</lat>
                                  <lng>-3.8964117</lng>
                           </start location>
                          <end location>
                                  <lat>40.1658568</lat>
                                  <lng>-3.8964121</lng>
                           </end location>
                           <polyline>
                                  <points>s{ctFp xV</points>
                           </polyline>
                           <duration>
                                  <value>0</value>
                                  <text>1 min</text>
                           </duration>
                           <html instructions>Dirígete al <b>noroeste</b> por
                                Calle Borox</html instructions>
                           <distance>
                                  <value>0</value>
                                  <text>1 m</text>
                            </distance>
                     </step>
                     <duration>
```



```
<value>0</value>
                          <text>1 min</text>
                    </duration>
                    <distance>
                          <value>0</value>
                          <text>1 m</text>
                    </distance>
                    <start location>
                          <lat>40.1658566</lat>
                          <lng>-3.8964117</lng>
                    </start_location>
                    <end location>
                          <lat>40.1658568</lat>
                          <lng>-3.8964121</lng>
                    </end_location>
                    <start address>Calle Borox, 27, 45216 Carranque, Toledo,
                        España</start address>
                    <end address>Calle Borox, 27, 45216 Carranque, Toledo,
                        España</end address>
             </lea>
             <copyrights>Datos de mapas ©2016 Google, Inst. Geogr.
                 Nacional</copyrights>
             <overview polyline>
                   <points>s{ctFp_xV</points>
             </overview polyline>
             <warning>Las rutas a pie están en versión beta. Ten cuidado. - En esta
           ruta puede que no haya aceras o pasos para
                                                        peatones.</warning>
            <bounds>
                   <southwest>
                          <lat>40.1658566</lat>
                          <lng>-3.8964121</lng>
                   </southwest>
                    <northeast>
                          <lat>40.1658568</lat>
                          <lng>-3.8964117</lng>
                    </northeast>
             </bounds>
      </route>
      <geocoded waypoint>
             <geocoder status>OK</geocoder status>
             <type>street address</type>
             <place id>ChIJN0tdTAPyQQ0R8KUVe_hw6dQ</place id>
      </geocoded_waypoint>
      <geocoded waypoint>
             <geocoder status>OK</geocoder status>
             <type>street address</type>
             <place id>ChIJN0tdTAPyQQ0R8KUVe_hw6dQ</place id>
      </geocoded waypoint>
</DirectionsResponse>
```

Este XML lo procesamos por SAX en la clase *ManejadorDirectionsAPI*. Del XML anterior nos quedamos con el contenido de *DirectionsResponse/route/leg/step/polyline/points* para obtener la ruta a seguir desde la posición actual a la salida del evento y mostrarla en el mapa:

```
@Override
public void startDocument() throws SAXException {
    cadena = new StringBuilder();
    arrLatLng = new ArrayList <ArrayList<LatLng>>();
}
@Override
public void startElement(String uri, String nombreLocal, String nombreCualif,
Attributes atr) throws SAXException {
    cadena.setLength(0);
    if (nombreLocal.equals("step")) {
        numRow++;
    }
}
```



La función *decodePoly* es una función que decodifica el contenido del elemento "points" a un array de *LatLng*, está función se ha obtenido a partir del siguiente artículo donde se explica el funcionamiento del Google Maps Directions API (<u>http://wptrafficanalyzer.in/blog/route-between-two-locations-with-waypoints-in-google-map-android-api-v2/</u>):

```
private ArrayList<LatLng> decodePoly(String encoded) {
    int index = 0, len = encoded.length();
    int lat = 0, lng = 0;
    ArrayList<LatLng> ruta= new ArrayList<LatLng>();
    while (index < len) {</pre>
        int b, shift = 0, result = 0;
        do {
            b = encoded.charAt(index++) - 63;
            result |= (b & 0x1f) << shift;
            shift += 5;
        } while (b >= 0x20);
        int dlat = ((result & 1) != 0 ? ~(result >> 1) : (result >> 1));
        lat += dlat;
        shift = 0;
        result = 0;
        do {
            b = encoded.charAt(index++) - 63;
            result |= (b & 0x1f) << shift;
            shift += 5;
        } while (b >= 0x20);
        int dlng = ((result & 1) != 0 ? ~(result >> 1) : (result >> 1));
        lng += dlng;
        LatLng p = new LatLng((((double) lat / 1E5)),
                (((double) lng / 1E5)));
        ruta.add(p);
    }
    return ruta;
}
```

Una vez que tenemos el array de *LatLng* podemos dibujar fácilmente la ruta en el mapa en el *onPostExecute* de la clase *DetalleMapsActivity* :

```
ArrayList<ArrayList<LatLng>> points = manejadorXML.getArrPosicionesGPS();
PolylineOptions lineOptions = new PolylineOptions();
MarkerOptions markerOptions = new MarkerOptions();
for(int i=0;i<points.size();i++) {
    lineOptions.addAll(points.get(i));
    lineOptions.width(8);
    lineOptions.color(Color.CYAN);
}</pre>
```



//borramos la ruta anterior si existe
if (rutaHastaSalida !=null) rutaHastaSalida.remove();
//añadimos el polyline con la ruta al mapa
rutaHastaSalida= mMap.addPolyline(lineOptions);

Además del anterior servicio también accedemos al servicio de notificaciones Firebase Cloud Messaging (acceso mediante http a <u>https://fcm.googleapis.com/fcm/send</u>) el cual veremos en mayor detalle en el apartado Alarmas y Notificaciones.

Vistas

En este apartado mostraremos las vistas de la aplicación y el esquema de navegación entre las mismas.



Además de la navegación directa entre las vistas también disponemos de una barra de navegación lateral (*Navigation Drawer*) la cual puede desplegarse pulsando en el icono el la barra de acciones o desplazándola desde el lado izquierdo al derecho de la pantalla.



Vista de preferencias de usuario (PreferenciasActivity)

Vista de edición de eventos (EditarActivity)

A continuación describiremos brevemente lo que se puede hacer en cada vista principal (para mayor detalle consultar el anexo con el manual de usuario (anexo 1):





| <u>Vista listado de los</u> <u>eventos</u> (ActivityTablon <u>)</u> | | En esta vista aparece un listado con los eventos disponibles. Podemos realizar búsquedas de eventos por nombre, fecha o deporte según lo hayamos configurado y existen tres pestañas para ver solo los eventos en los que estamos apuntados, los creados por mí y todos los disponibles. Desde ella se puede acceder al detalle, editar, borrar, compartir y apuntarse al evento seleccionado. |
|--|---|--|
| <u>Vista de edición de</u> <u>eventos</u> (EditarActivity) | 2° 4° 3° 14.03 Editar Evento : Actividad Correr Nombre del Evento Lugar Fecha SEL. FECHA Hora IMPORTAR RUTA GRABAR RUTA Descripción | Desde esta vista se puede crear un evento nuevo o modificar uno existente. Se puede introducir una foto, el deporte, nombre, descripción, fecha, hora y lugar del evento. Además se permite para determinados deportes (correr, ciclismo,) importar o grabar el recorrido que tendrá la actividad deportiva. |
| <u>Vista Grabar Ruta</u> (RutaActivity) | Image: construction of the construc | Nos permite grabar una ruta que se asociara al evento. Durante la grabación se mostrará la distancia recorrida, el ritmo, la altitud y el tiempo empleado. Además se informará cada kilómetro por mensaje de voz la distancia recorrida y el ritmo actual. |







| <u>Vista preferencias de</u> <u>usuario</u> (PreferenciasActivity <u>)</u> | A the contract of the | El usuario puede configurar determinadas características de la aplicación, si quiere que le lleguen avisos o no, por qué parámetro van a realizarse las búsquedas (nombre, fecha o deporte del evento). |
|--|--|---|
| | | |

En la aplicación también tenemos otras ventanas auxiliares que nos ayudan a la introducción de datos y a mostrar información al usuario:







Alarmas y Notificaciones

El usuario puede configurar la aplicación para que le lleguen alarmas y notificaciones, en concreto puede activar:

 que le lleguen notificaciones cada vez que se crea/modifica/borra un evento de uno de sus deportes favoritos



• que se le avise 24 horas antes del comienzo de un evento al que está apuntado.



Para las notificaciones se hace uso de Firebase Cloud Messaging (FCM). FCM es una solución multiplataforma que te permite enviar, de forma gratuita y segura, mensajes y notificaciones.



El usuario al activar la opción de que le lleguen notificaciones y seleccionar un deporte favorito la aplicación le suscribe automáticamente al "tema" de ese deporte. Con ello cada vez que se mande una notificación de ese "tema" también le llegará al usuario:

```
FirebaseMessaging.getInstance().subscribeToTopic(deporte);
```

Y si un deporte deja de ser favorito eliminamos la anterior suscripción:

FirebaseMessaging.getInstance().unsubscribeFromTopic(deporte);

Para el envío de las notificaciones se ha creado la clase *GestorNotificaciones* que gestiona dichos envíos y las alarmas. Dentro de esta clase tenemos el método *sendMessage* que realiza la conexión con el servidor y construye el objeto JSON que recibirá FCM y con el que preparará la notificación final que se envía a los usuarios. Esta función recibe como parámetros el título de la notificación, el cuerpo del mensaje y el grupo/tema al que va dirigido.

```
public void sendMessage( final String title, final String body, final String grupo ) {
    new AsyncTask<String, String, String>() {
        @Override
        protected String doInBackground(String... params) {
            try {
                JSONObject root = new JSONObject();
                JSONObject notification = new JSONObject();
                notification.put("body", body);
                notification.put("title", title);
                notification.put("icon", "icono2.png");
                JSONObject data = new JSONObject();
                data.put("message", "");
                root.put("notification", notification);
                root.put("data", data);
                root.put("to", "/topics/"+grupo);
                String result = postToFCM(root.toString());
                return result;
            } catch (Exception ex) {
                ex.printStackTrace();
            }
            return null;
        }
        @Override
        protected void onPostExecute(String result) {
        }
    }.execute();
```

```
}
```

Un ejemplo de solicitud POST enviada a FCM desde la aplicación tendrías la siguiente estructura:

| Cabecera HTTP | <pre>https://fcm.googleapis.com/fcm/send Content-Type:application/json (indicamos que lo que enviamos es un objeto JSON) Authorization:key= QU16YVlJQjgza2F (la clave del servidor)</pre> |
|---------------|---|
| Cuerpo HTTP | <pre>{ notification :{ body: "Carrera Solidaria feria de Carranque" title:"Evento Modificado" icon:"icono2.png" },</pre> |



```
data: {
    message=""
},
to: "/topics/correr"
}
```

Al modificar/crear/borrar un evento deportivo se envía una notificación haciendo uso de la anterior función:

```
GestorNotificaciones gestorNot = new GestorNotificaciones();
```

```
gestorNot.sendMessage(
    "Evento Modificado",
    evento.getTitulo()+" -- "+ evento.getFecha()+" "+evento.getHora(),
    evento.getDeporte().replaceAll(" ","_")
);
```

Hasta aquí hemos explicado brevemente el envío de notificaciones ahora indicaremos como se reciben dichas notificaciones. Para ello en el *AndroidManifest.xml* declaramos un servicio que extiende de *FirebaseMessagingService* (*MyFirebaseMessagingService*):

Este servicio es el que se encarga de recibir la notificación desde FCM y mostrarla al usuario.

```
public class MyFirebaseMessagingService extends FirebaseMessagingService
     @Override
    public void onMessageReceived(RemoteMessage remoteMessage) {
        //obtenemos los datos del mensaje enviado por FCM
        sendNotification(remoteMessage.getNotification().getTitle(),
                         remoteMessage.getNotification().getBody());
    }
   private void sendNotification(String messageTitle,String messageBody) {
        //construimos la notificación que verá el usuario
        NotificationCompat.Builder notific = (NotificationCompat.Builder)
                 new NotificationCompat.Builder(this)
                          .setContentTitle(messageTitle)
                          .setSmallIcon(R.drawable.icono2)
                          .setDefaults (Notification. DEFAULT ALL)
                          .setContentText(messageBody);
        //al pulsar sobre la notificación se abrirá la aplicacion
        NotificationManager notificationManager = (NotificationManager)
                         getSystemService (Context.NOTIFICATION_SERVICE);
        Intent i = new Intent(this, MainActivity.class);
        PendingIntent intencionPendiente = PendingIntent.getActivity(this, 0,
                         i, PendingIntent.FLAG UPDATE CURRENT);
        notific.setContentIntent(intencionPendiente);
        notificationManager.notify((int) System.currentTimeMillis(),
                         notific.build());
    }
}
```



Así cuando recibimos una notificación de FCM indicando un cambio en uno de los eventos deportivos se nos mostrará una notificación como la siguiente:



Como ya hemos dicho la clase *GestorNotificaciones* se encarga de gestionar también las alarmas (avisamos 24 horas antes del comienzo de una actividad deportiva al que estemos apuntado). Para este propósito tenemos el método programarAlarma donde hacemos uso de una BBDD sqlite para almacenar el id del *PendingIntent* de la alarma y poder cancelarla a posteriori, por ejemplo si el usuario deja de estar apuntado a un evento:

```
public void programarAlarma(Evento e, Context contexto) {
    //almacenamos el id de la alarma en una BBDD local para poder eliminar la
    //alarma si el usuario deja de estar apuntado
    BBDDAlarmas bbdd = new BBDDAlarmas(contexto);
    bbdd.guardarAlarma(e.clave);
    int idAlarma=bbdd.getIdAlarma(e.clave);
    //Programamos la alarma
    AlarmManager alarmManager = (AlarmManager)
                          contexto.getSystemService(Context.ALARM_SERVICE);
    Calendar cal = Calendar.getInstance();
    //recuperamos la hora del evento
    String hora = "0";
    String minuto = "0";
    String strHora = e.getHora();
    if (strHora.indexOf(":") > -1) {
        hora = strHora.substring(0, strHora.indexOf(":"));
        minuto = strHora.substring(strHora.indexOf(":")+1);
    }
    //recuperamos el dia mes y año de la fecha del evento
    String strfecha = e.getFecha();
    StringTokenizer token = new StringTokenizer(strfecha, "-");
    String ano = "";
    String mes = "";
    String dia = "";
    if (token.hasMoreTokens()) {
        dia = token.nextToken();
        mes = token.nextToken();
        ano = token.nextToken();
    }
    cal.set(Calendar.YEAR, Integer.parseInt(ano));
    cal.set(Calendar.MONTH, Integer.parseInt(mes)-1);
    cal.set(Calendar.DAY_OF_MONTH, Integer.parseInt(dia));
    cal.set(Calendar.HOUR_OF_DAY, Integer.parseInt(hora));
cal.set(Calendar.MINUTE, Integer.parseInt(minuto));
    cal.set(Calendar.SECOND, 0);
    cal.set(Calendar.MILLISECOND, 0);
    //le restamos 24 horas a la fecha del evento que es cuando se lanzará el
    //aviso
    cal.setTimeInMillis(cal.getTimeInMillis()-1000*60*60*24);
    if (System.currentTimeMillis()<(cal.getTimeInMillis())) {</pre>
        //si la alarma es mayor al dia actual se progama para que salte la
         //notificación a la hora y dia calculados
        SimpleDateFormat dateFormat = new SimpleDateFormat("dd MMM yyyy
                 hh:mm:ss zzz");
        Intent intentForService = new
```

```
UNIVERSITAT
                                Alumno: nombre
         POLITÈCNICA
         DE VALÈNCIA
                         Intent(contexto.getApplicationContext(),
                         ServicioAvisoEmpiezaEvento.class);
        intentForService.putExtra("Titulo", "Evento proximo a
                comenzar");
       intentForService.putExtra("Desc", e.getTitulo() +
                " comenzará el " + e.getFecha() + " " + e.getHora());
       intentForService.putExtra("clave", e.getClave());
        //establecemos el id del PendingIntent con la clave almacenada
              //en bbdd para poder borrar dicha alarma
        PendingIntent service = PendingIntent.getService(contexto,
          idAlarma, intentForService, PendingIntent.FLAG UPDATE CURRENT);
        alarmManager.set(AlarmManager.RTC WAKEUP,
                cal.getTimeInMillis(), service);
  }
}
```

Provecto: titulo

Para cancelar una alarma tenemos el método cancelarAlarma:

```
public void cancelarAlarma(Evento e, Context contexto) {
    //recuperamos el id de la alarma asocida al evento
    BBDDAlarmas bbdd = new BBDDAlarmas(contexto);
    int idAlarma=bbdd.getIdAlarma(e.clave);
   bbdd.borrarAlarma(e.clave);
    if(idAlarma>-1) {
        //si existe la alarma la cancelamos
        final AlarmManager alarmManager = (AlarmManager)
        contexto.getSystemService(Context.ALARM SERVICE);
        Intent intentForService = new Intent(contexto.getApplicationContext(),
                                  ServicioAvisoEmpiezaEvento.class);
        PendingIntent service = PendingIntent.getService(contexto, idAlarma,
                         intentForService, PendingIntent.FLAG UPDATE CURRENT);
        service.cancel();
        alarmManager.cancel(service);
    }
}
```

Finalmente declaramos en el AndroidManifest el servicio llamado por la alarma, ServicioAvisoEmpiezaEvento:

```
<service android:name=".ServicioAvisoEmpiezaEvento" />
```

Que en el método onStartCommand lanza la notificación:

```
@Override
public int onStartCommand(Intent intenc, int flags, int idArranque) {
     if(intenc!=null) {
        String titulo = intenc.getStringExtra("Titulo");
        String desc = intenc.getStringExtra("Desc");
        String clave = intenc.getStringExtra("clave");
        if(titulo!=null && desc!=null) {
            SharedPreferences pref =
                  PreferenceManager.getDefaultSharedPreferences(this);
             //comprobamos que tiene activado el aviso de 24 horas antes
            if (pref.getBoolean("pref_notificaciones", false) &&
                pref.getBoolean("pref aviso24", false)) {
                 //lanzamos la notificación
                GestorNotificaciones gn = new GestorNotificaciones();
                gn.mostrarNotificacion(titulo, desc, this);
                 //borramos la alarma de la bbdd
                BBDDAlarmas bbdd = new BBDDAlarmas(this);
                bbdd.borrarAlarma(clave);
            }
        }
```



```
}
return START_STICKY;
```

}

Así al apuntarse un usuario a un evento programamos el aviso simplemente con este código:

```
GestorNotificaciones gn = new GestorNotificaciones();
gn.programarAlarma(evento,getApplicationContext());
```

Y cuando el usuario se desapunta del evento cancelamos la alarma con:

```
GestorNotificaciones gn = new GestorNotificaciones();
gn.cancelarAlarma(evento,getApplicationContext());
```

Conclusiones

Hay que indicar que se ha logrado realizar todos los objetivos iniciales propuestos, si bien se han detectado una serie de posibles líneas de mejora y desarrollo que se podrían incluir en versiones futuras del proyecto:

- Paginar las consultas de los listados de eventos
- Poder consultar que usuarios están apuntados a un evento determinado
- Implementar funcionalidades para la comunicación entre los usuarios apuntados a un evento (chat, foro,...)
- Entrenamiento guiado
- Almacenar los entrenamientos realizados para poder consultarlos y compararlos.
- Compartir y exportar rutas
- Posibilidad de realizar el pago de las inscripciones de eventos deportivos de pago a través de la app.
-

Personalmente el desarrollo de este proyecto me ha servido para afianzar los conocimientos adquiridos y ver que soy capaz de realizar un aplicación móvil desde cero. También destacaría que es la primera vez que utilizo Firebase y he de decir que facilita bastante la implementación de aplicaciones que precisen almacenar datos, ficheros, envío de notificaciones,....y sobre todo la actualización online de los datos. Finalmente indicar que me encuentro particularmente satisfecho del diseño de las vistas en donde he intentado seguir las consideraciones de Material Desing.

Anexos

Manual de usuario

En este punto se va a presentar un breve manual de usuario en el que detallaremos el funcionamiento de la aplicación y las opciones de la misma (dar de alta eventos deportivos,



apuntarse a los mismos, importar, grabar y entrenar rutas,...). En primer lugar indicar que la aplicación está disponible en inglés y español.

A.1 Vista de Login

Al iniciar la aplicación nos aparece la vista de inicio desde la cual el usuario puede iniciar la sesión mediante su cuenta de Google pulsando en el botón "Iniciar sesión":



Una vez iniciada la sesión se mostrará la vista con los eventos deportivos disponibles que se describirá en el siguiente apartado.

A.2 Vista Listado Eventos Disponibles (Tablón)

En esta vista se muestra un listado de los eventos deportivos disponibles (los eventos con fecha de inicio anterior a la de hoy no se muestran).





A continuación detallaremos los componentes de esta ventana.

En el menú superior tenemos los botones 📃, 🔍 y 💷:

• El botón desplegará el menú lateral común a toda la aplicación desde donde podremos acceder a las distintas opciones de la aplicación. Este menú se detallará en el apartado (apartado A.6).



- Pulsando en se podrán filtrar los eventos, dependiendo de la opción de configuración que tengamos seleccionada, por las siguientes condiciones:
 - El nombre del evento (aparecerán todos los eventos cuyo nombre empiece por el filtro introducido):

Buscar Por Titulo...

 La fecha del evento (se mostrarán todos los eventos que se inicien en la fecha introducida según el formato dd/mm/yyyy):

Buscar Por Fecha...

- El deporte del evento (aparecerán todos los del deporte indicado):
 ← Buscar Por Deporte...
- Pulsando en accedemos a la opción de lanzar la ventana de información acerca de (
 Acerca de...
) que mostrará (común a todas las pantallas):





Debajo del menú disponemos de tres pestañas que pulsando sobre ellas nos permiten obtener el listado de los eventos (con fecha igual o posterior a la actual):

- APUNTADO
 los que estamos apuntados
- los creados por el usuario
- todos los eventos

A continuación aparece la lista de los eventos deportivos:





La información que se muestra por cada evento es:



Si pulsamos sobre un evento la aplicación nos dirigirá a la ventana con el detalle del evento (*apartado A.4*) y si pulsamos sobre nos aparecerán una serie de acciones que se pueden realizar sobre el evento:



- Editar (solo disponible si el usuario ha creado este evento)→para modificar algún dato del evento (*apartado A.3*)
- Borrar (solo disponible si el usuario ha creado este evento) →para borrar el evento, antes del borrado se mostrará un mensaje de confirmación



 Compartir →para compartir el evento con otros usuarios a través del mail, redes sociales,...:





Y finalmente tenemos un botón flotante ve que nos permite acceder directamente a la pantalla de edición de eventos (apartado A.3)

A.3 Vista Edición Eventos

Desde esta vista podemos crear eventos nuevos o modificar eventos ya existentes:



Para cada evento podemos introducir:

 Una foto, para lo cual se debe pulsar sobre seleccionar el 'cartel' del evento. La imagen seleccionada se mostrará en la página de edición:



Una vez cargada la imagen si pulsamos sobre ella se nos mostrará la imagen original:





• El deporte del evento:

| atividad |
|---------------|
| Correr |
| Escalada |
| Futbol |
| Natacion |
| Padel |
| Patinaje |
| Senderismo |
| Tenis |
| Tenis de Mesa |
| Frail |
| /olevball |

- El nombre del evento
- El lugar donde se quedará para iniciar el evento. Para ello se pulsará sobre el botón "SEL. LUGAR" y se mostrará una ventana para seleccionar la ubicación a partir de la señal del GPS o a partir de una dirección:



• La fecha en la que tendrá lugar el evento pulsando sobre el botón "SEL. FECHA":





La hora del evento pulsando sobre "SEL.HORA":
 ▶ ↔ ☺ ೫ ♥ ○ ♥ 4℃ ↓ 3℃ ↓



- La descripción del evento.
- Si el deporte seleccionado es susceptible de tener una ruta asociada (correr, senderismo, ciclismo,...) se muestran los siguientes botones para añadir una ruta al evento:
 - "IMPORTAR RUTA" nos permite importar una ruta desde un fichero con extensión:
 - .tcx (Training Center XML)
 - .kml (Keyhole Markup Language)
 - .gpx (GPS eXchange Format)



 "GRABAR RUTA" se nos mostrará una nueva ventana para la grabación de la ruta del evento (apartado A.3.1)

Al pulsar en el botón aceptar se valida que se hayan rellenado todos los datos obligatorios para poder crear un evento (nombre del evento, actividad, lugar, fecha, hora y descripción). Si se han introducido todos, se graba el evento y se envía una notificación, a todos los usuarios que tienen activada la opción de recibir notificaciones de eventos de ese deporte, indicando dicha creación/modificación. Si no se han introducido todos los datos se muestra un mensaje en rojo debajo del dato erróneo:

| Nombre del Evento | |
|---------------------------------|--|
| 1 | |
| Introduces al nombro del suento | |

A.3.1 Grabar Ruta

Desde esta vista se puede grabar la ruta del evento:



Al iniciar esta actividad se comprueba si el usuario ha dado el permiso de "Ubicación" a la aplicación, en caso contrario se muestra un aviso para que se conceda:





Así mismo se muestra el siguiente mensaje hasta que se obtiene una ubicación valida:



Una vez tenemos señal GPS podemos iniciar la ruta pulsando en el botón "Iniciar Ruta", una vez iniciada se nos irán mostrando los siguientes botones para pausar/reanudar, detener, descartar (se descarta la ruta y no se asociará al evento) y finalizar la ruta (al pulsar en finalizar se asociara la ruta al evento):



En el mapa se representa con el icono P la salida y con V la meta. La ruta se va dibujando en el mapa con color verde claro:





Además se nos muestra la siguiente información debajo del mapa:



Durante la grabación la aplicación nos irá informando mediante mensajes de voz de los cambios del estado de GPS ("buscando GPS", "GPS encontrado", "GPS deshabilitado",...), de los kilómetros recorridos y del ritmo que llevamos.

A.4 Vista Detalle Evento

Si pulsamos sobre un evento en la lista de eventos accedemos a esta vista en la cual se muestra el detalle del evento:



En esta pantalla se muestra:

• La foto del evento, si pulsamos sobre ella se verá la imagen original:





- El nombre, la actividad y la fecha del evento superpuestos a la foto del evento.
- La descripción.
- La distancia en Km (siempre que tenga una ruta asociada)
- La dirección de donde se ha quedado para realizar la actividad deportiva
- Un mapa con la ruta y el lugar donde se ha quedado
- Finalmente una serie de botones para:
 - Borrar el evento (si eres el creador del mismo)
 - Editar el evento (si eres el creador del mismo)
 - Apuntarte al evento y el número de apuntados
 - Compartir el evento a través del mail, redes sociales,...

Si pulsamos sobre el mapa accedemos a otra pantalla donde se nos muestra en mayor detalle la ruta y el lugar donde se ha quedado *(apartado A.4.1)*

A.4.1 Detalle del Mapa del Evento

Al pulsar sobre el mini mapa del detalle del evento se mostrara esta vista:





Aquí tenemos un mayor detalle del mapa, donde se muestra:

- el lugar donde se ha quedado para el evento con un icono que cambia dependiendo del
 - deporte de la actividad, por ejemplo para correr 🦻, para natación 🏓 ,.... Si se pulsa sobre dicho icono se mostrará información del evento:



- la ruta del evento en color verde claro
- la salida y la meta de la ruta.
- una gráfica con la altimetría de la ruta



3° Además tenemos un botón flotante para realizar un "entrenamiento" del evento. Al pulsar sobre él:

se mostrará el camino a seguir en el mapa (en un color verde azulado) desde la ubicación del usuario 💹 a la salida de la ruta:



los kilómetros recorridos, el ritmo, la altimetría actual y el tiempo que llevamos haciendo la ruta (similar a la información que se muestra al grabar una ruta).



se irán indicando mediante mensajes de voz los kilómetros recorridos, si estamos en la salida del evento o si hemos llegado a la meta.

Si se quiere volver a ver la altimetría bastaría con pulsar el botón 🔛

A.5 Búsqueda Eventos en Mapa

A esta vista accedemos a través de la opción "Buscar Evento Mapa" del menú lateral, en ella se muestran los eventos disponibles (los eventos con fecha de inicio anterior a la de hoy no se muestran) en un mapa centrado en la ubicación del usuario:



Si pulsamos sobre los iconos de los eventos se nos mostrara un panel informativo del evento, y si se pulsa sobre el panel se nos mostrará la pantalla del detalle del evento seleccionado *(apartado A.4):*



Si pulsamos sobre el botón flotante se centrará el mapa en la ubicación del usuario y pulsando sobre la opción del menú superior, el mapa se centrará sobre la dirección indicada para mostrar los eventos próximos a ella:





A.6 Menú Lateral

Al pulsar sobre el botón 🗮 del menú superior se desplegará el menú lateral. Esté menú es común a toda la aplicación:



En este menú se muestra el nombre y mail del usuario logado. A través de él se puede acceder a las distintas pantallas de la aplicación y cerrar la sesión:

- Tablón \rightarrow accedemos al listado de los eventos (apartado A.2).
- Buscar Evento Mapa \rightarrow accedemos a la pantalla del mapa de eventos (apartado A.6).
- Crear Evento
 →accedemos la pantalla de creación de eventos
 (apartado A.5).
- Configuracion →accedemos a la ventana de preferencias del usuario



(apartado A.6.1).

×

Cerrar →se cierra la sesión del usuario y se finaliza la aplicación

A.6.1 Preferencias de usuario

Desde esta vista el usuario podrá configurar distintas opciones del funcionamiento de la aplicación:



El usuario puede activar las notificaciones para que le lleguen avisos. Al activar esta opción se pueden activar las siguientes notificaciones:

• <u>24 horas antes</u> (se envía una notificación de recordatorio 24 horas antes del comienzo de todos los eventos a los que se haya apuntado):



 <u>evento deportes favoritos</u> (se envía una notificación cuando se crea, borra o modifica un evento de los deportes marcados como favoritos).



Si se activa esta opción se pueden seleccionar los deportes favoritos del usuario:



| Deportes favoritos | | | | |
|--------------------|------------------|--|--|--|
| | Badminton | | | |
| | Baloncesto | | | |
| | Ciclismo | | | |
| | Correr | | | |
| | Escalada | | | |
| | Futbol | | | |
| | Natacion | | | |
| | Padel | | | |
| | Patinaje | | | |
| | CANCELAR ACEPTAR | | | |

También se puede indicar sobre que se quiere que se filtren los eventos al pulsar en 🥄 en el listado de eventos (por nombre, por fecha o por deporte):

| Buscar por | | | | |
|------------|-------------|----------|--|--|
| ۲ | Por Nombre | | | |
| 0 | Por Fecha | | | |
| 0 | Por Deporte | | | |
| | | CANCELAR | | |

Código fuente en GitHub

El código fuente de este proyecto está subido a GitHub en la siguiente url:

https://github.com/juancarperez/SportsMeet

Pruebas

Las pruebas del presente proyecto se han realizado directamente en dos terminales reales sin utilizar el emulador:

- Moto G4 Plus versión 5.0 de Android
- Lenovo K3 Note versión 2.3.6 de Android

Glosario de Términos

GPX GPS eXchange Format, es un esquema XML pensado para transferir datos GPS entre aplicaciones. Se puede usar para describir puntos (waypoints), recorridos (tracks), y rutas (routes).

GPS Global Position System, es el primer sistema de navegación por satélite, desarrollado por el departamento de defensa del gobierno estadounidense.



JSON: JavaScript Object Notation, formato ligero de intercambio de datos básicamente. JSON describe los datos con una sintaxis dedicada que se usa para identificar y gestionar los datos.

KML: Keyhole Markup Language, formato basado en XML que se utiliza para mostrar datos geográficos en un navegador terrestre(Google Earth, Google Maps).

NoSQL: Not Only SQL, sistemas de gestión de bases de datos que difieren del modelo clásico del sistema de gestión de bases de datos relacionales siendo el más destacado el hecho de que no usan SQL como el principal lenguaje de consultas.

TCX: Training Center XML, es un formato de intercambio de datos similar a GPX introducida en 2007 como parte del producto Garmin Training Center.

XML: eXtensible Markup Language, es un metalenguaje extensible de etiquetas desarrollado por el W3C. Es una simplificación y adaptación del SGML y permite definir la gramática de lenguajes específicos.