Universidad de Murcia

Grado en Ingeniería Informática
Curso 2020/2021

Computación Móvil

Profesores:
Pedro Javier Ferández Ruiz
Jesús García Rodríguez

Práctica II : Entregable Android

Ignacio Pagán Molera - 49275872M - ignacio.paganm@um.es
Jaime Ortiz Aragón - 49196689B - jaime.ortiza@um.es

Índice

Descripción técnica de la aplicación

La práctica consiste en la implementación de una aplicación Android, y la realización de un estudio derivado de su utilización. La funcionalidad básica de la aplicación es mostrar por pantalla un mapa de cobertura del nivel de señal máximo percibido por el teléfono, en función de la ubicación del usuario, y según la tecnología seleccionada. Es decir, se creará un mapa que se irá actualizando conforme el usuario vaya variando su posición. La aplicación es capaz de mostrar estos mapas para tres tecnologías distintas: GSM, WCDMA y LTE (2G, 3G y 4G respectivamente).

En cuanto al diseño, la aplicación está formada únicamente por dos actividades. A pesar de que las comentaremos en detalle en los próximos apartados, vamos a adelantar su funcionalidad principal.

La primera actividad mostrará por pantalla un menú en el cual se podrá seleccionar la tecnología utilizada para la creación del mapa. Es por tanto una pantalla de inicio que carece de mayor funcionalidad.

La segunda actividad, mediante el uso de GoogleMaps, realizará el despliegue del mapa en la pantalla. En esta actividad se mostrará la ruta realizada por el usuario, en la cual podremos ver el nivel de señal máximo medido en las distintas ubicaciones en las que hemos estado de forma visual.

Antes de comentar con más detalle el diseño y el desarrollo de las actividades que acabamos de presentar, vamos a comentar algunos aspectos técnicos de la aplicación.

Ficheros de configuración

En primer lugar, vamos a comentar el proceso que hemos seguido para seleccionar los colores de la aplicación. El primer paso fue definir en el fichero **colors.xml** los colores que pensamos que íbamos a necesitar:

```
<color name="colorPrimary">#FFC107</color>
<color name="colorPrimaryDark">#FFC107</color>
<color name="colorAccent">#FF9800</color>
<color name="BackGroundColor">#FFE57F</color>
```

A continuación, definimos el estilo AppTheme en el fichero **styles.xml**, y seleccionamos para este estilo los colores previamente definidos.

```
<!-- Base application theme. -->
<style name="AppTheme" parent="Theme.AppCompat.Light.DarkActionBar">
    <!-- Customize your theme here. -->
    <item name="colorPrimary">@color/colorPrimary</item>
    <item name="colorPrimaryDark">@color/colorPrimaryDark</item>
    <item name="colorAccent">@color/colorAccent</item>
    <item name="android:textSize">22dp</item>
    <item name="android:windowBackground">@color/BackGroundColor</item>
</style>
```

Para que este estilo se adoptara como el estilo por defecto de la app, añadimos en el archivo **AndroidManifest.xml** la siguiente directiva: **android:theme="@style/AppTheme".**

En este fichero también hemos añadido la siguiente directiva: **android:allowBackup="true"** que permite a la aplicación volver a otra actividad anterior. Aunque si se retrocede a la actividad anterior la información obtenida en la actividad se perderá. Es decir, la actividad se iniciará de nuevo sin guardar información de sesión.

Otro aspecto importante es que hemos añadido la función *setRequestedOrientation(ActivityInfo.**SCREEN_ORIENTATION_LOCKED**);* en el método *OnCreate()* de ambas actividades para que la aplicación no permita al dispositivo que se cambie la vista al girar la pantalla. Hemos optado añadir esta funcionalidad puesto que en los mapas si girabas el dispositivo toda la información almacenada se perdía, reiniciándose la actividad.

Actividad Main

Esta actividad será la primera que se muestre cuando se lance la aplicación. En cuanto a su ciclo de vida, sólo debemos tener en cuenta la función *onCreate()* que se encarga de cargar el layout.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_LOCKED);
}
```

Este layout, como se puede apreciar en las posteriores capturas de pantalla puede aparecer tanto en castellano como en inglés, en función de los ajustes del lenguaje de entrada del dispositivo. Por defecto viene en inglés, pero con la ventana de *Open editor* y seleccionando el icono de la bola del mundo los strings se traducen al idioma que seleccionemos, en nuestro caso, castellano.

La única funcionalidad de esta actividad viene dada por los 3 botones que se muestran, referentes a las 3 tecnologías de geolocalización : **GSM, WCDMA** y **LTE**. En cuanto al tratamiento de las cadenas utilizadas para los botones o demás aspectos visuales de las pantallas, hemos decidido definir variables en el fichero ***strings.xml*** con el fin de generar un código de los layouts más reutilizables y fáciles de modificar.

En la siguiente captura tenemos un ejemplo del valor asociado a cada una de las cadenas correspondientes a la tecnología. En este caso podemos ver que la directiva *"translatable"* se encuentra a *"false"* puesto que el nombre de las tecnologías es indistinto tanto en castellano como en inglés.

```
<string name="g2" translatable="false">GSM</string>
<string name="g3" translatable="false">WCDMA</string>
<string name="g4" translatable="false">LTE</string>
```

Estas cadenas son las que se usarán para asignar los nombres de los botones. De esta forma, podemos definir sobre qué objeto se realiza la acción derivada del click sobre el botón. Dado que son 3 botones, deberemos realizar 3 acciones. Realmente es la misma acción, pero cada botón ejecuta una función *onClick* que llamará a la actividad *Maps* con un identificador de la tecnología que representa, para actuar en consecuencia.

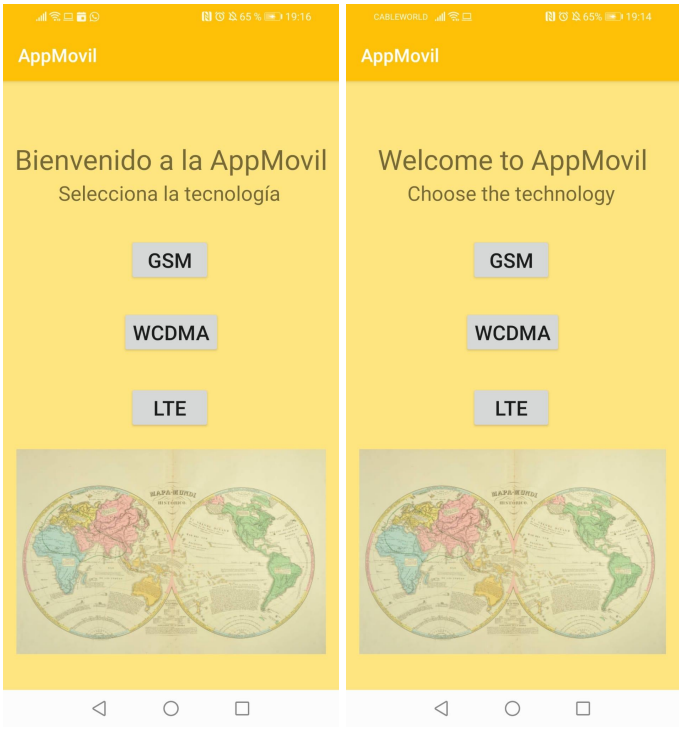
Este identificador se incluye mediante un objeto *intent* al que se le añade un extra con la cadena asociada a la tecnología. Como podemos apreciar en la siguiente captura:

```
public void onClick2g(View v){
    Intent intent = new Intent( packageContext: this,ActividadMaps.class);
    intent.putExtra( name: "tecnologia", value: 2);
    startActivity(intent);
}

public void onClick3g(View v){
    Intent intent = new Intent( packageContext: this,ActividadMaps.class);;
    intent.putExtra( name: "tecnologia", value: 3);
    startActivity(intent);
}

public void onClick4g(View v){
    Intent intent = new Intent( packageContext: this,ActividadMaps.class);
    intent.putExtra( name: "tecnologia", value: 4);
    startActivity(intent);
}
```

En las siguientes imágenes, podemos ver la apariencia de esta actividad principal, observando lo comentado anteriormente acerca del lenguaje de las cadenas: al haber incluido en el fichero ***strings.xml*** la traducción para el inglés de las cadenas definidas, en caso de que el lenguaje del dispositivo cambie al inglés, el idioma de las palabras también cambiaría en tiempo de ejecución.



Actividad Maps

En esta actividad mostraremos un mapa mediante las librerías de Google Maps, en el que se mostrará nuestra ubicación actual, y se creará un mapa mediante el dibujo de puntos de colores en función del nivel de señal obtenido en la ubicación actual del usuario.

En el método *OnCreate()* podemos observar como se carga el layout, no se permite el cambio de orientación de la pantalla como ya hemos comentado anteriormente. A continuación, mediante un objeto *Bundle* se recupera la información extra que la actividad Main almacenó en un *Intent.* Esta información es la tecnología seleccionada por el usuario, y se guardará en una variable global booleana que será utilizada posteriormente en otra función.

Posteriormente se crea una instancia del cliente de proveedor de ubicación combinada. Y mediante un objeto *LocationCallback*, con la función *onLocationResult()* se accede al código en el que se obtiene la ubicación cada vez que se cumple la condición del callback. Las condiciones son establecidas en la función *showLastLocation()*, que comentaremos posteriormente. Cuando se cumplen las condiciones se obtiene la ubicación, se mueve la cámara al punto donde esté el usuario, se obtiene el nivel de señal máximo para esa ubicación mediante la función *getAllCells()*, y se imprime un punto, de un color en función del nivel de señal obtenido, en el mapa.

Por último, se creará el mapa en la actividad, y un *telephonyManager*, que será el objeto que nos permita recuperar la información de las torres de telefonía.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_actividad_maps);
    setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_LOCKED);
    Bundle extras = getIntent().getExtras();
    if (!extras.isEmpty()) {
        Object extra = extras.get("tecnologia");
        if (extra instanceof Integer) {
            if (extra == (Integer) 2) {
                gsm = true;
            }

            if (extra == (Integer) 3) {
                wcdm = true;
            }
            if (extra == (Integer) 4) {
                lte = true;
            }
        }
    }
    fusedLocationClient = LocationServices.getFusedLocationProviderClient( activity: this);
    callback = (LocationCallback) onLocationResult(locationResult) -> {
            Location location = locationResult.getLastLocation();
            double latitud = location.getLatitude();
            double longitud = location.getLongitude();
            LatLng ubicacion = new LatLng(latitud,longitud);
            mMap.moveCamera(CameraUpdateFactory.newLatLngZoom(ubicacion,  v: 15));
            int level= getAllCells();
            imprimirPuntos(latitud, longitud, level);
    };
    SupportMapFragment mapFragment = (SupportMapFragment) getSupportFragmentManager().findFragmentById(R.id.map);
    mapFragment.getMapAsync( onMapReadyCallback: this);
    telephonyManager = (TelephonyManager) getSystemService(Context.TELEPHONY_SERVICE);

}
```

Esta función es la que genera el mapa (objeto de la librería de GoogleMaps), y posteriormente llama a 3 funciones para poner en marcha toda la funcionalidad de la aplicación.

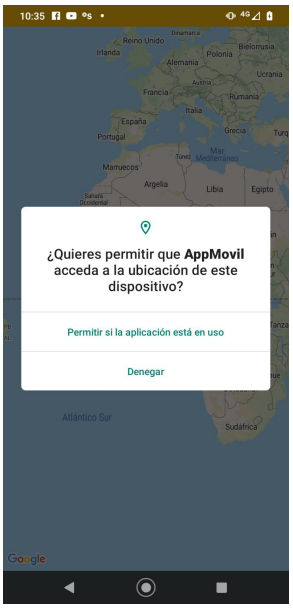
```
@Override
public void onMapReady(GoogleMap googleMap) {
    mMap = googleMap;
    showPosition();
    showLastLocation();
    comprobarUbi();
}
```

En esta función, se pide al usuario todos los permisos para que la aplicación pueda acceder a la aplicación. Si el usuario permite el acceso a la ubicación, el mapa pasará a poder mostrar la ubicación del usuario.

```
private void showPosition() {
    if (checkSelfPermission(Manifest.permission.ACCESS_FINE_LOCATION) != PackageManager.PERMISSION_GRANTED &&
            checkSelfPermission(Manifest.permission.ACCESS_COARSE_LOCATION) != PackageManager.PERMISSION_GRANTED) {
        ActivityCompat.requestPermissions( activity: this, new String[]{
                Manifest.permission.ACCESS_FINE_LOCATION
        }, requestCode: 0);
        return;
    }
    mMap.setMyLocationEnabled(true);

}
```

Este es el mensaje que aparece en la aplicación la primera vez que esta se ejecuta:



En caso contrario, si el usuario no permitió el acceso de la aplicación a la ubicación del dispositivo, se accede a esta función, que imprimirá por pantalla el siguiente mensaje: “Se necesita poder acceder a la ubicación para que la aplicación funcione”.

Como podemos ver, esta función actúa como rutina de error en caso de que se produzca algún fallo en lo que consideramos que debería ser el correcto funcionamiento de la aplicación. En este caso solo hemos determinado que fuera necesario imprimir por pantalla este mensaje en caso de que el usuario no proporcione los permisos para acceder a la aplicación. Pero se podría añadir otros códigos de error, para realizar demás tratamientos de error.

```
@Override
public void onRequestPermissionsResult(int requestCode, String[] permissions, int[] grantResults) {
    switch (requestCode) {
        case 0: {
            if (grantResults.length > 0 && grantResults[0] == PackageManager.PERMISSION_GRANTED) {
                onMapReady(mMap);
            } else {
                Toast.makeText( context: this,  text: "Se necesita poder accerder a la ubicación para que la aplicación funcione", Toast.LENGTH_LONG).show();
            }
        }

    }

}
```

La función *showLastLocation()* crea un objeto LocationRequest, que será al que añadamos todas las condiciones necesarias para que se pueda acceder al callback del método *onCreate()* una vez estas se cumplan. Estas condiciones son, que la ubicación se actualice cada 5 segundos, pudiéndose actualizar antes si fuera posible, pero dejando como mínimo un delay de 3 segundos entre cada actualización.

El otro requisito es que la ubicación se haya modificado 50m desde la última vez que se produjo una actualización de ubicación. Hemos decidido esta distancia puesto que hemos considerado que era lo suficientemente amplia para que se puedan experimentar cambios en los niveles de señal.

```
    private void showLastLocation() {
        if (checkSelfPermission(Manifest.permission.ACCESS_FINE_LOCATION) != PackageManager.PERMISSION_GRANTED &&
                checkSelfPermission(Manifest.permission.ACCESS_COARSE_LOCATION) != PackageManager.PERMISSION_GRANTED) {
        }
        LocationRequest locationRequest = LocationRequest.create();
        locationRequest.setInterval(5000);
        locationRequest.setFastestInterval(3000);
        locationRequest.setPriority(locationRequest.PRIORITY_HIGH_ACCURACY);
        locationRequest.setSmallestDisplacement(50);
        fusedLocationClient.requestLocationUpdates(locationRequest, callback,  looper: null);
    }
```

Hemos añadido una función *comprobarUbi()*, puesto que la aplicación la primera vez que se ejecuta pide los permisos de usuario, pero no comprueba que la ubicación del dispositivo se encuentre activa. Por ello, cada vez que se ejecuta la aplicación, mediante la función *isLocationAvailable()* comprobamos si la ubicación se encuentra activa en el dispositivo.

En caso de que no se encuentre activa, se imprime por pantalla el siguiente mensaje: “Se debería activar la ubicación”. Si se encuentra ya activa, no se realiza nada, ya que es el funcionamiento esperado.

```
private void comprobarUbi()
{
    if (checkSelfPermission(Manifest.permission.ACCESS_FINE_LOCATION) != PackageManager.PERMISSION_GRANTED &&
            checkSelfPermission(Manifest.permission.ACCESS_COARSE_LOCATION) != PackageManager.PERMISSION_GRANTED) {
        return;
    }
    fusedLocationClient.getLocationAvailability().addOnSuccessListener(new OnSuccessListener<LocationAvailability>() {
        @Override
        public void onSuccess(LocationAvailability locationAvailability)
        {
            if (locationAvailability.isLocationAvailable())
            {

            }
            else Toast.makeText(getApplicationContext(),  text: "Se debería activar la ubicación", Toast.LENGTH_LONG).show();
        }
    });
}
```

Por último, nos quedan las funciones relativas a la obtención de señal, y a la impresión de puntos en el mapa. La función *getAllCells()* obtiene una lista de objetos CellInfo. Debemos conocer lo que representa este tipo de objetos. Los CellInfo son celdas de radio que pertenecen a una red de celdas, cada una tiene su propio transmisor, y son conocidas como estación base. Las celdas se usan con la finalidad de cubrir diferentes áreas geográficas, para poder proporcionar cobertura de radio sobre una zona más grande que lo que podría proporcionar una única celda.

Se realiza un for each para cada una de estas celdas obtenidas, comprobando en cada iteración que la celda sea de la tecnología que se ha seleccionado. Después se obtiene la señal que proporciona esa celda mediante la función *getCellSignalStrength().getLevel()* Esta función puede devolver 5 valores. En la siguiente tabla mostramos los nombres de las constantes que devuelve esta función, asociado a sus valores numéricos, y a los colores que nosotros les hemos asignado para realizar los mapas:

Señal	**Valor**	**Color**
SIGNAL_STRENGTH_NONE_OR_UNKNOWN	0	**Negro**
SIGNAL_STRENGTH_POOR	1	Rojo
SIGNAL_STRENGTH_MODERATE	2	**Naranja**
SIGNAL_STRENGTH_GOOD	3	Amarillo
SIGNAL_STRENGTH_GREAT	4	Verde

Se comprueba si el nivel de señal obtenido es mayor que el máximo nivel obtenido por alguna de las demás celdas. En caso de que sea mayor, se actualiza el valor del máximo.
Una vez se recorren todas las celdas, se devuelve el valor de la señal máxima obtenida.
A continuación mostramos el código descrito:

```
public int getAllCells() {
    StringBuilder text = new StringBuilder();
    if (checkSelfPermission(Manifest.permission.ACCESS_COARSE_LOCATION) != PackageManager.PERMISSION_GRANTED) {
        return Integer.parseInt( s: null);
    }
    List<CellInfo> cellInfoList = telephonyManager.getAllCellInfo();
    text.append("Found ").append(cellInfoList.size()).append(" cells\n");
    int contador=0;
```

```
for (CellInfo info : cellInfoList) {
    if ((info instanceof CellInfoGsm) && (gsm = true)) {
        CellInfoGsm cellInfoGsm = (CellInfoGsm) info;
        CellIdentityGsm id = cellInfoGsm.getCellIdentity();
        text.append("GSM ID:{cid: } ").append((id.getCid()));
        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.P) {
            text.append(" mcc: ").append(id.getMccString());
        } else text.append(" mcc: ").append(id.getMcc());
        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.P) {
            text.append(" mnc: ").append(id.getMncString());
        } else text.append(" mnc: ").append(id.getMnc());
        text.append((" lac: ")).append(id.getLac());
        text.append("} Level ").append(cellInfoGsm.getCellSignalStrength().getLevel()).append("\n");
        if (cellInfoGsm.getCellSignalStrength().getLevel() > contador)
        {
            contador = cellInfoGsm.getCellSignalStrength().getLevel();
        }
```

El código del for each, se repite para las CellInfo de LTE y WCDMA. Lo único que cambia es en la primera comprobación, que en vez de ***gsm = true*** se comprobará si en vez de gsm es lte o wcdma para 4g y 3g respectivamente. Por último, como ya hemos comentado la función devuelve la variable *contador*, que contiene el valor máximo de cobertura que ha obtenido de entre todas las celdas a las que se ha conectado.

Una vez tenemos el nivel del señal máximo para la ubicación actual, debemos mostrarlo en el mapa mediante un círculo de color. Únicamente realizamos la función addCircle de GoogleMap, con un círculo de diámetro 30, con distintos colores según la señal obtenida (los colores correspondientes a los valores de señal ya los hemos indicado en la tabla anterior):

```
private void imprimirPuntos(double latitud, double longitud, int level) {
    switch (level)
    {
        case 0:
            mMap.addCircle(new CircleOptions().center(new LatLng(latitud,  longitud)).radius(30).fillColor(Color.BLACK).strokeWidth(3));
            break;

            case 1:
            mMap.addCircle(new CircleOptions().center(new LatLng(latitud,  longitud)).radius(30).fillColor(Color.RED).strokeWidth(3));
            break;

        case 2:
            mMap.addCircle(new CircleOptions().center(new LatLng(latitud,  longitud)).radius(30).fillColor(Color.rgb( red: 255,  green: 128,  blue: 0)).strokeWidth(3));
            break;

        case 3:
            mMap.addCircle(new CircleOptions().center(new LatLng(latitud,  longitud)).radius(30).fillColor(Color.YELLOW).strokeWidth(3));
            break;

        case 4:
            mMap.addCircle(new CircleOptions().center(new LatLng(latitud,  longitud)).radius(30).fillColor(Color.GREEN).strokeWidth(3));
            break;
    }
}
```

Análisis teórico de los mapas

En este apartado vamos a realizar un estudio sobre los mapas de cobertura que hemos obtenido mediante la ruta que hemos llevado a cabo. Nuestro trayecto comprende parte de la Avenida Juan de Borbón (una calle muy amplia y grande), parte del barrio de la Flota (un barrio residencial en el cual nos hemos adentrado en calles estrechas con numerosos edificios) y parte de zona rural, como es Santiago del Zaraiche (llegando prácticamente al palmeral)

Hemos escogido dicha ruta puesto que queríamos combinar distintas zonas, ya que considerábamos interesante observar si se producían cambios significativos en la cobertura en función del terreno. En los siguientes apartados vamos a comentar los 3 mapas obtenidos:

GSM

Global System Communications se trata de la segunda generación de redes móviles, que sustituyó el acceso fijo a la red telefónica conmutada que existía hasta el momento, conocido como *PSTN* (Public Switched Telephone Network). Se podría decir que es la primera generación de redes móviles que realmente tuvo un éxito real, debido al gran número de países en los que se implantó y el número de usuarios que acogió.

En cuanto al despliegue de la arquitectura, es necesario saber que cualquier sistema GSM consta a su vez de 3 subsistemas. Por un lado el *RSS* (Radio Subsystem), el *NSS* (Network Switching Subsystem) y *OSS* (Operation Subsystem). En nuestro caso, como la información se intercambia entre el terminal móvil y las estaciones, nos vamos a centrar en el RSS.

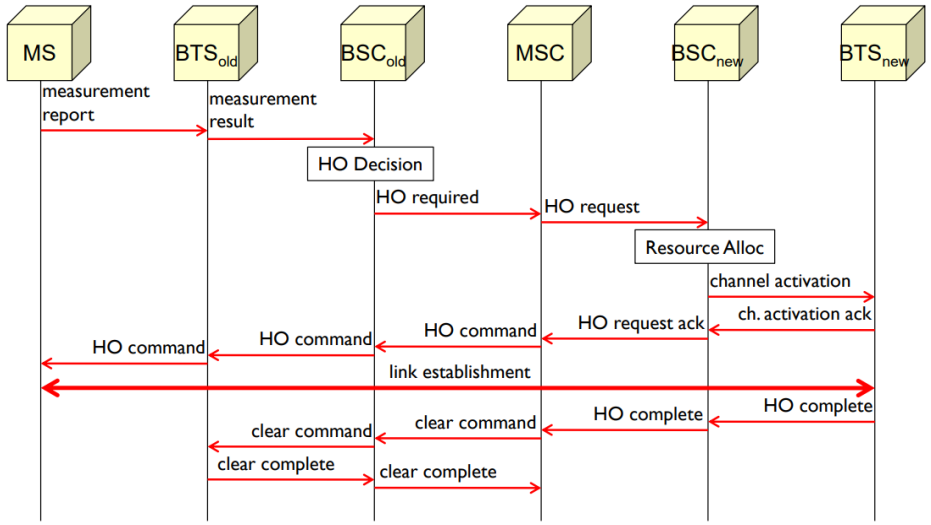
El dispositivo móvil está formado por el propio terminal, es decir tanto el smartphone físico como el software, nuestra app, por ejemplo. Este se identifica por su *IMEI*, que es el identificador del dispositivo. Además del *ME*, que conjunta lo mencionado anteriormente, cuenta con la propia *SIM* (Subscriber Identity Module), la cual almacena los datos relativos a GSM y se identifica por el *ICCID*.

El RSS, además de por el ME, está formado por el *BSS* (Base Station Subsystem), que está compuesto por el *BTS* (Base Transceiver Station) y *BSC* (Base Station Controller). Hilando fino, lo que a nosotros nos interesa para estudiar el mapa es el BTS, ya que alberga las antenas y celdas a las que se hacen referencia en la actividad *maps* cuando se trata de obtener el nivel de señal. Para tener completamente localizado el MS, se debería usar un *LA* (Location Area) por cada celda, sin embargo, esto traería consigo un esquema poco escalable, ya que existiría una gran sobrecarga en la señalización. Es por esto por lo que realmente, una LA representa un grupo de celdas, en vez de sólo una.

Como se puede esperar, al realizar un recorrido como el que hemos realizado nosotros, se pasará por un gran número de celdas, incluso nos conectaremos a varias antenas distintas. Este movimiento entre las celdas se llama *handover* y se produce cuando durante una sesión activa, el MS se mueve entre celdas. Cuando el dispositivo se aleja del BTS al que está

conectado, se empiezan a producir errores debido al descenso de la señal recibida, es por esto por lo que necesita cambiar de estación.

Debido a la escala de nuestro caso, el handover más típico es el que se muestra en la siguiente captura, el handover inter-celda intra-BSC. Es decir, el MS cambia de celda pero sigue bajo el control del mismo BSC, por lo que este le asignará una nueva celda. El MS realiza mediciones periódicas del canal downlink, mientras que el BTS realiza mediciones periódicas del canal uplink para medir la calidad del enlace.

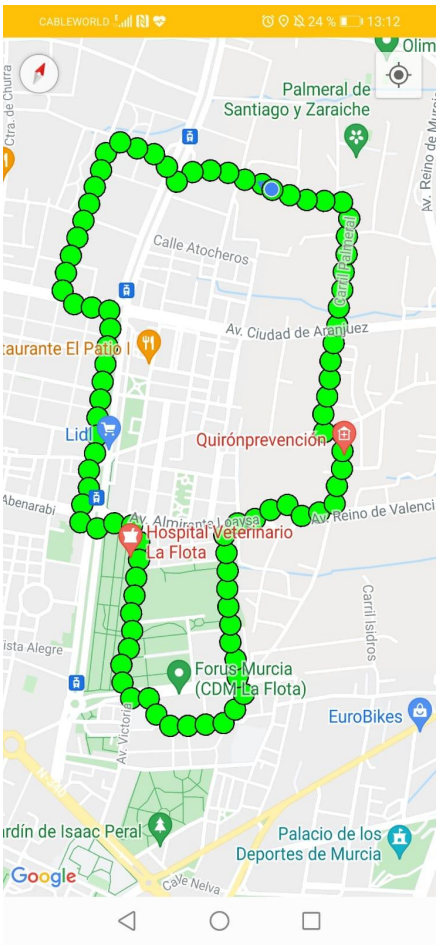


El parámetro que marca la calidad de la señal se llama *RSSI* (Received Signal Strength Indicator) y se representa con dos valores: *dBm* y *asu*. En las redes móviles 2G, el RSSI expresado en *dBm*, siempre posee un valor negativo. La máxima potencia de señal se corresponde al valor -51 y la mínima con la que se puede establecer una conexión corresponde a -113. En cuanto al valor *asu,* su valor es proporcional a la fuerza de la señal. En redes 2G oscila desde 0 (la señal más débil) hasta 31 (la más fuerte).

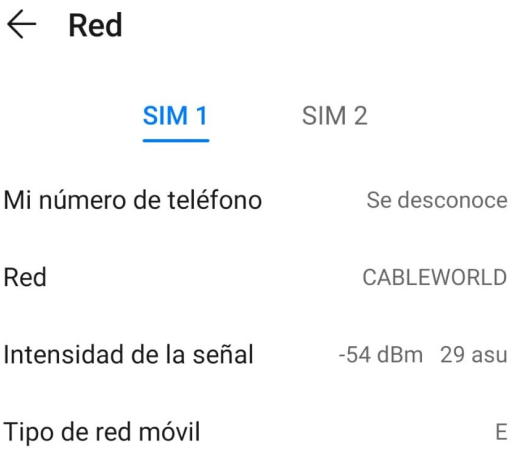
En la siguiente imagen podemos ver los baremos de la cobertura medida en *dBm* para la tecnología 2G:

5 barras: -51 hasta -79 — Señal excelente
4 barras: -80 hasta -88 — Señal buena
3 barras: -89 hasta -95 — Señal regular
2 barras: -96 hasta -104 — Señal mala
1 barra: -105 hasta -112 — Señal débil

La siguiente captura refleja el mapa obtenido del siguiente trayecto:



Según la leyenda que hemos realizado podemos observar como se ha obtenido el máximo nivel de cobertura en todos los puntos del mapa. Para comprobar que la aplicación funciona de forma correcta hemos ido comprobando el estado de red del dispositivo, en el que se muestra el nivel de señal en dBm, con el que hemos realizado la medición. En la siguiente captura de pantalla podemos ver como dicha señal es mayor a -79, que es el valor límite de la señal excelente.



WCDMA

Wideband CDMA es la solución que se adoptó para cumplir los requisitos de la evolución de las redes 2G con *UMTS* (Universal Mobile Telecommunication System). Este sistema sufrió distintas evoluciones desde el conocido 3G hasta el 3.5G con UMTS R99, UMTS R4, UMTS R5, UMTS R6 y UMTS R7. La primera estructura inicial es la R99, en la cual nos vamos a centrar. Esta aprovecha el núcleo de red de GSM e involucra una nueva red de acceso radio: *UTRAN* (UMTS Terrestrial Access Network). Uno de los objetivos de esta evolución era que se pudiera seguir utilizando 2G, por lo que el núcleo de red conserva la estructura de la red GSM, separando los dominios de circuitos y paquetes.

En la mencionada nueva interfaz radio se sustituyen los BTS que conocíamos de 2G por nodos B, y los BSC por los RNC. Se utiliza ATM como protocolo de transporte. En resumen, este sistema tenía como objetivo seguir cubriendo 2G, pero aportar mejoras tales como el aumento del ancho de banda.

En este caso, la arquitectura de red es jerárquica. El mapa de cobertura se divide en celdas de gran tamaño con el fin de proporcionar una amplia cobertura, y estas a su vez envuelven otras celdas de menor tamaño en algunas zonas es las que sea necesaria la ampliación del sistema.
De esta forma, cuando un dispositivo se enciende y comienza a operar en la red, lo primero que hace es escuchar su canal físico *SCH* para obtener información de sincronización. Este tiene un identificador único para toda la celda. Además, también espera recibir el código que usa *BCH* en toda la celda. Una vez captada esta información, el dispositivo decodifica el canal físico BCH para obtener toda la información que necesite del canal lógico *BCCH.*

Esta localización de dónde se encuentra el terminal móvil debe soportar la movilidad como principio de UMTS. UMTS está destinado a usuarios móviles, que pueden desplazarse mientras están transmitiendo y recibiendo información. Estos desplazamientos conllevan cambios en una misma celda o entre celdas. Esto ya ocurría con GMS, y al igual que entonces, se denomina *handover*.

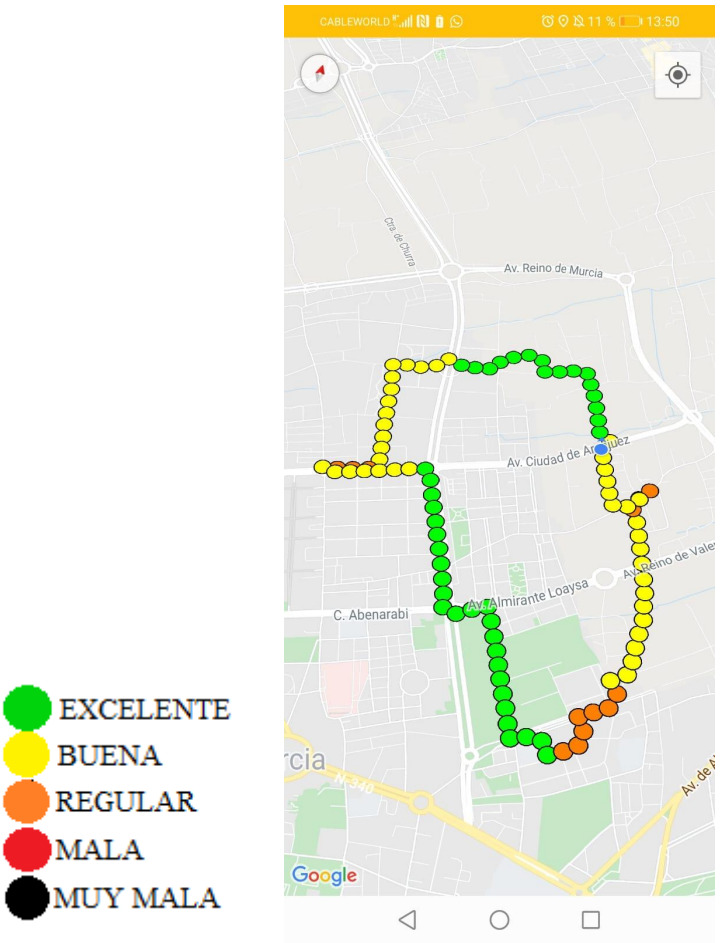
En UTRA existen tres tipos de handovers: hard handover, soft handover y softer handover. Al tratarse de datos, cabe destacar que estaremos hablando de conmutación de paquetes, ya que la QoS es menor que con las llamadas.

En el primero de ellos, la nueva conexión en la celda destino no se produce hasta que no se ha liberado la celda en la que se encontraba anteriormente. Estas celdas tienen frecuencias distintas, lo que se conoce como inter-frecuencia. Esto sucede porque los dispositivos móviles no pueden recibir datos de dos frecuencias distintas simultáneamente. Sin embargo, este no es el tipo de handover más común, porque pensamos que las frecuencias de las celdas a las que se conecta el dispositivo estarán a la misma frecuencia, al ser bastante uniforme. Sí que podríamos decir que se produce un hard HO en la Avenida Ciudad de Aranjuez en ambos lados del mapa, ya que pasamos de zonas más rurales y silvestres hacia zonas urbanas. En algunos puntos concretos, en concreto donde la calidad de servicio es peor (los puntos

naranjas), se tratan de zonas bastante urbanizadas, por lo que puede ser que el mecanismo sea forzado por la propia red, actuando como balanceador de carga, con el fin de que no haya tanto ruido en dicha celda.

El que diríamos que es el cambio de celda más común es el soft HO, ya que en muchos tramos encontramos que la calidad de la red recibida es bastante estable en colores. En este caso, la nueva conexión a la celda destino se produce antes de liberar los recursos de la celda en la que se encuentre anteriormente el dispositivo. Ambas celdas tienen la misma frecuencia, por lo que este HO se denomina como intra-frecuencia. Este tipo de cambio de celda es novedoso en 3G, ya que en UMTS el dispositivo puede utilizar dos códigos distintos para dos antenas distintas.

Finalmente, el softer HO no es como tal un cambio de celda, sino un cambio de sector dentro del mismo nodo-B.

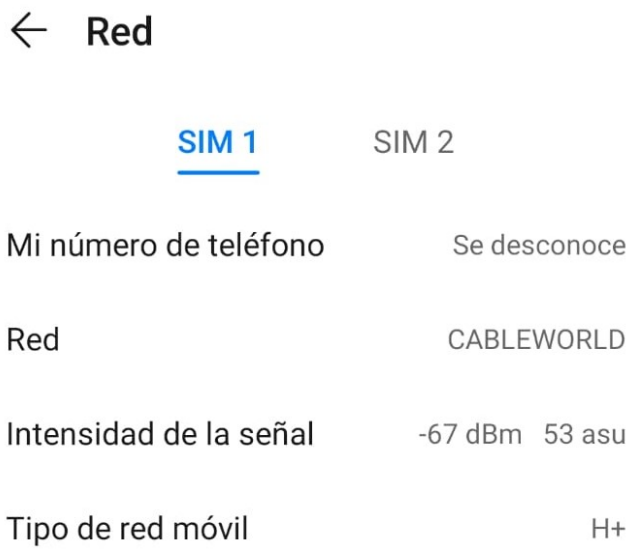


En cuanto a cómo se mide la calidad de señal en 3G, además del valor RSSI como se veía en 2G, también encontramos otras dos medidas. *RSCP* (Received Signal Code Power) es la fortaleza de la señal que es procesada por el móvil, expresada también en dBm. La tercera es *EcNo* (Energy Per Chip). Este es un parámetro que indica la relación de RSCP entre RSSI. Su valor es negativo expresado en Db.

En el dispositivo móvil, la medida que se nos proporciona es el de RSSI, como se muestra a continuación.

Potencia de la señal	RSCP
Excelente	-50 a -75 dBm
Buena	-76 a -90 dBm
Regular	-91 a -100 dBm
Mala	-101 a -120 dBm

Adjuntamos una captura del valor del RSSI obtenido en el mismo tramo del mapa que en el apartado anterior. En 2G veíamos que el valor era -54, y en este caso es de -67 dBm.



LTE

La tecnología LTE (Long Term Evolution) 4G supone el inicio de la cuarta generación de telefonía móvil. Es un estándar de comunicaciones móviles desarrollado por la 3GPP (asociación que desarrolló y se encarga del mantenimiento de GSM y UMTS) Sus objetivos fueron el acceso radio de banda ancha, y la convergencia de servicios móviles basados en Internet , proporcionándose todos los servicios sobre redes IP basadas en conmutación de paquetes.

A diferencia de 2G y 3G, que tienen una arquitectura de red jerárquica, 4G favorece una arquitectura de red plana. En E-UTRAN (red de acceso especificada para LTE) no existe un controlador central, y la funcionalidad BSC/RNC se halla distribuída entre los eNB, reduciéndose latencia, costes, y eliminando posibles puntos único de fallo.

LTE se define sobre una gran cantidad de bandas de frecuencia 40 perfiles distintos: banda uplink/downlink y modo dúplex (FDD/TDD).

También debemos comentar que tiene altas tasas de transferencia de datos debido a las siguientes técnicas: Tecnologías multi-carrier, antenas MIMO, y diversas técnicas de adaptación de enlace (planificación en tiempo, frecuencia y modulación/codificación)

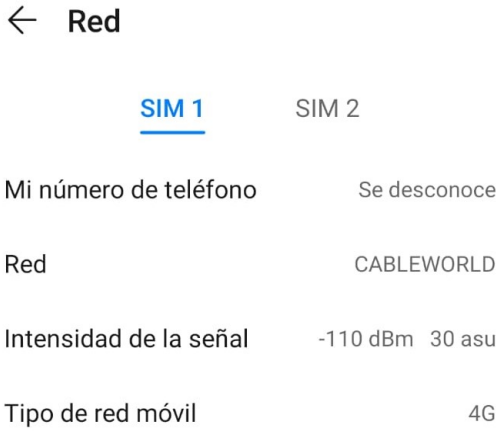
En cuanto al mapa desarrollado, podemos observar que se producen grandes diferencias: nos encontramos con distintas zonas en la que señal es mala (color rojo), algo que no había ocurrido en ninguna zona ni de 3G ni de 4G. Esto se debe a que los dBm necesitados para que la señal sea del mejor tipo es mucho mayor a diferencia de las otras dos tecnologías. Posteriormente mostraremos la tabla de valores dBm necesarios para alcanzar cada nivel de señal, y podremos comprobar que existen estas grandes diferencias.

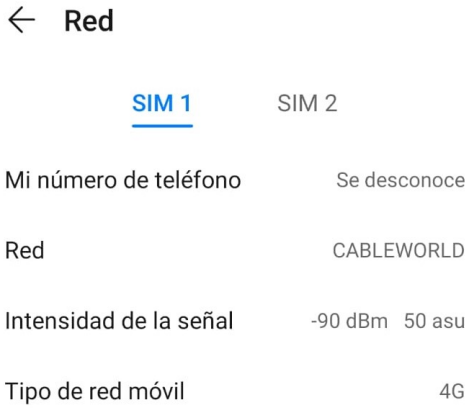
Cabe destacar que la gran mayoría de zonas naranjas y rojas han sido obtenidas en tramos de huerta, o en barrios de calles estrechas que se encuentran entre numerosos edificios altos. En las dos rectas amplias por las que hemos estado podemos ver que la señal era verde y amarilla (zona de Juan de Borbón y Avenida Príncipe de Asturias) La recta de la derecha vemos que también es muy larga, pero se corresponde a zona de huerta.

Mostramos aquí la tabla que relaciona la potencia de la señal y el nivel de señal obtenido. El nivel de señal es la intensidad con la que la señal llega al móvil desde la celda a la que el dispositivo se encuentra conectado. Como hemos comentado anteriormente, se puede apreciar que para 4g los baremos son bastante mayores. En el caso de 2G una señal excelente era aquella en la que su señal estaba comprendido entre -51 y -79, y en este caso tenemos que la señal excelente es aquella menor a -88. Este es el motivo por el cual en el mapa de 2G el mapa se encontraba pintado entero de verde.

Potencia de la señal	RSCP
Excelente	-75 a -88 dBm
Buena	-89 a -96 dBm
Regular	-97 a -105 dBm
Mala	-106 a -112 dBm
Pésima	-113 a -125 dBm

A continuación mostramos unas capturas del estado de red en algunos de los puntos del mapa, pudiendo comprobar que según los valores de la tabla anterior, la señal de -110dBm correspondería con una zona pintada de rojo (mala cobertura), y la segunda captura corresponde a una zona amarilla (buena cobertura).





Conclusiones

El desarrollo de estas prácticas nos ha parecido de gran interés, puesto que hemos aprendido a desarrollar una aplicación sencilla mediante un entorno de desarrollo como es Android Studio que ninguno de los dos habíamos utilizado antes. No nos ha costado gran trabajo adaptarnos a este entorno gracias a los vídeo-tutoriales y a la resolución de todas las dudas que se nos ha proporcionado.

En cuanto al apartado más técnico, describir el código de la actividad principal nos ha servido para conocer mejor el proceso que realizan las aplicaciones de geolocalización. La realización de los mapas también nos ha ayudado a dominar algunos aspectos teóricos de los que teníamos algunas dudas.

Por tanto, podemos decir que ha sido una de las prácticas que más hemos disfrutado durante la carrera.