Universidad Nacional de La Matanza

Cristian Marcelo Rios – 40.015.557

Evaluación de Aprendizaje II

Sistemas Operativos Avanzados – Comisión Turno Miércoles Noche

Sistema Operativo Android

Contenidos

Introducción de la aplicación:

La utilidad de la presente aplicación entregada se puede definir en:

- Registrar un usuario
- Loguearse con su respectivo usuario registrado y válido
- Validar estado de conexión de internet
- Leer valor de distintos sensores del dispositivo, así como registrar ciertos eventos en un servidor proveído por la cátedra
- Reproducir música tras un delay definido

La finalidad de la aplicación es poder demostrar los conocimientos adquiridos por la cátedra en forma tal que implemente la comunicación respectiva con el servidor para registrar el usuario, utilizando Retrofit para la comunicación con el API y Gson para la conversión de datos y objetos al protocolo de notación JSON. Asimismo, también se lo usa para el logueo respectivo de eventos relacionados con los sensores del dispositivo y sus valores leídos. En todos los casos se envía la información en las request según el formato que requiere el servidor (dispuesto por la cátedra), ya que de lo contrario se obtiene un resultado de petición erróneo, por ejemplo por no cumplir con el formato de la petición, y se muestra el determinado mensaje al usuario.

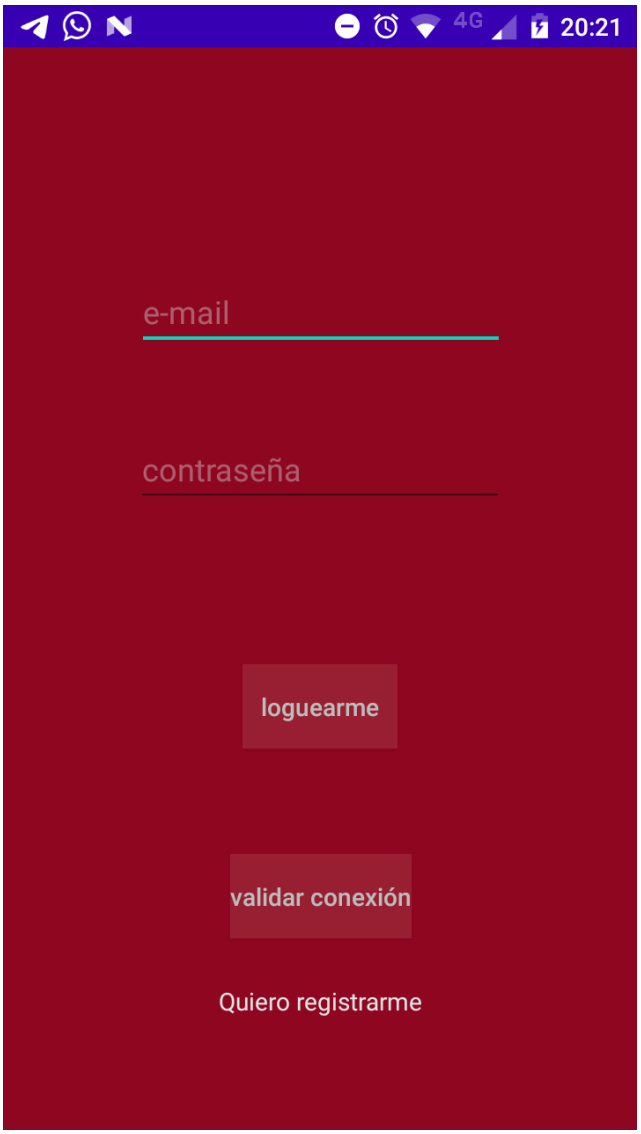
Una vez ingresado, el usuario puede ver y leer en pantalla los valores en tiempo real de los sensores de:

- Aceleración
- Giróscopo
- Sensor gravitacional
- Proximidad
- Luminosidad

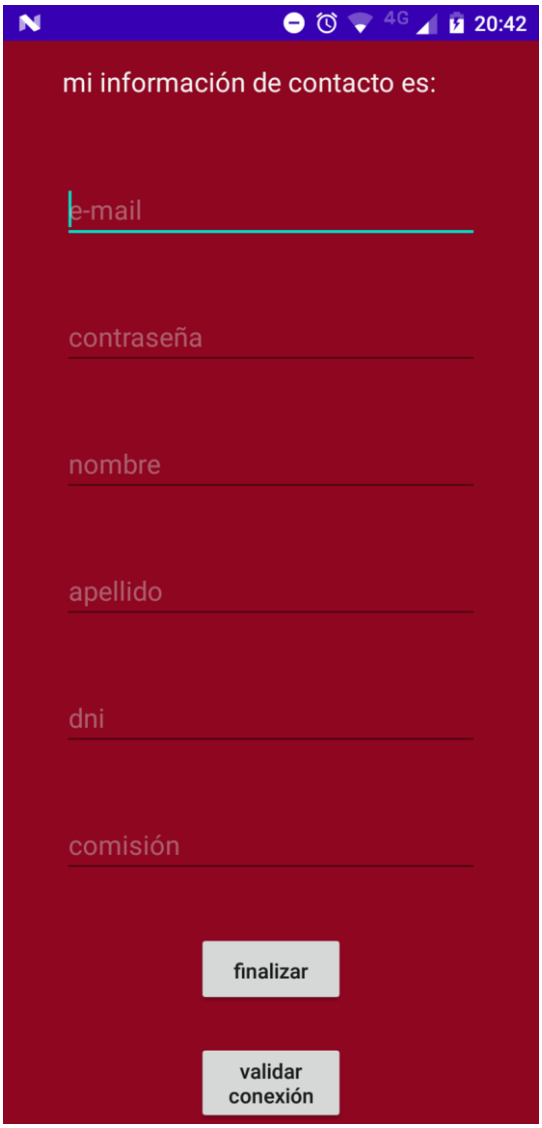
La aplicación cuenta con un botón adicional para escuchar música, de forma de demostrar el funcionamiento e implementación de hilos secundarios (no bloqueantes para el hilo principal) y servicios (en este caso, un servicio que se encarga de reproducir música pasado un cierto tiempo).

Breve manual del usuario:

La primera vista en visualizarse por el usuario es la **pantalla de logueo**. La misma se conforma de los campos respectivos para ingresar su Usuario y Contraseña **ya registrados**. Una vez completados estos cambios, debe presionar el botón “loguearme” para que se envíe la petición al servidor con el usuario y mail ingresado y se obtenga la validación, es decir, si efectivamente su usuario se encuentra registrado. Si las credenciales son correctas, se redirige al usuario a la vista de visualización de sensores. De lo contrario, se le muestra un mensaje de error en el segmento inferior de la pantalla – como mensaje “Toast”.



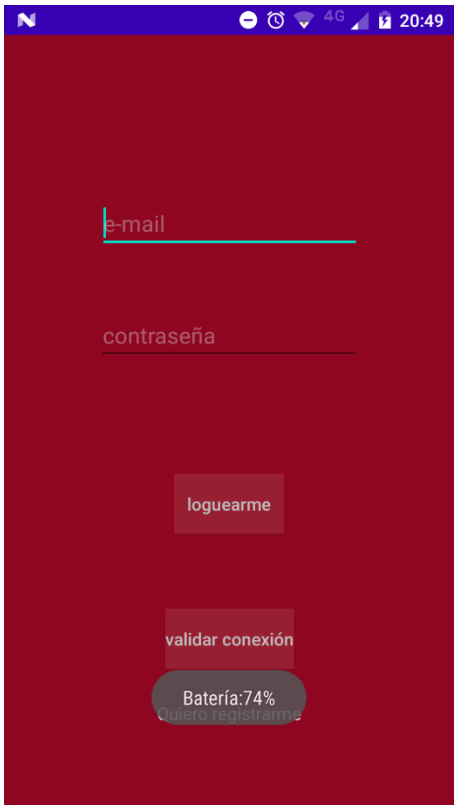
En caso de no poseer usuario, puede pulsar la etiqueta “Quiero registrarme” para pasar a la **vista de alta de usuario**, en donde debe completar todos los campos allí requeridos de forma válida.



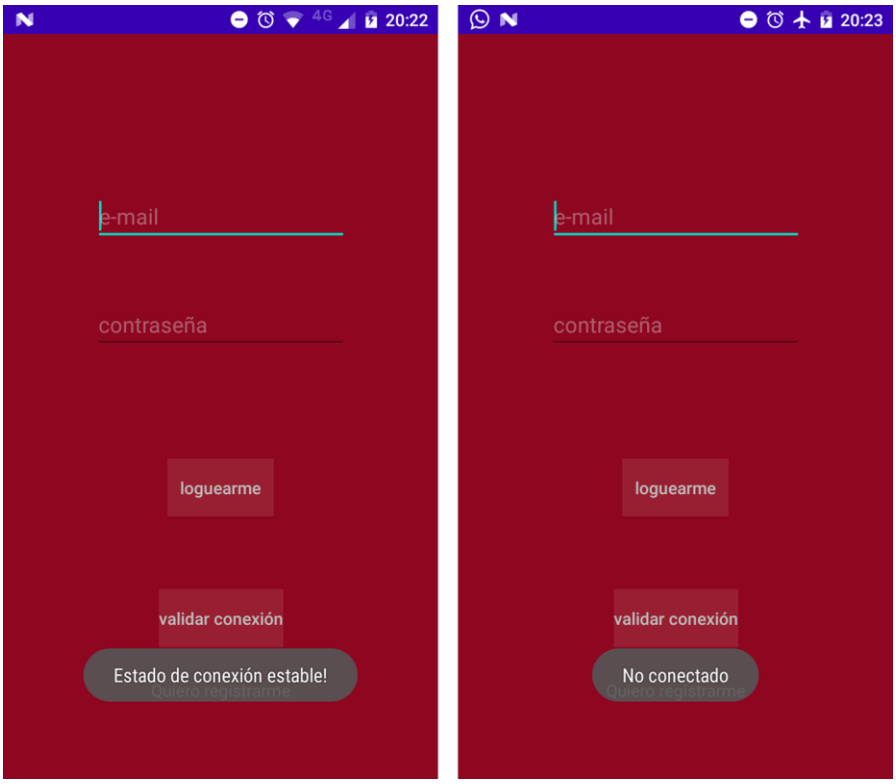
Una vez completados todos los datos, el usuario puede apretar el botón **finalizar** en donde, si los datos son válidos y el registro en el servidor es exitoso, se le mostrará un mensaje de bienvenida y se le redirigirá a la vista de visualización de sensores. De lo contrario, se le mostrará un mensaje de error.

Finalmente, tras el registro, la pantalla de visualización mostrará los distintos valores de los sensores que se detallaron en el apartado anterior. Adicionalmente se posee el botón de escuchar música, con un timer de 20 segundos (al apretarse el botón, la música se reproducirá 20 segundos después, y recién ahí se podrá detener la reproducción de la misma).

Cabe aclarar asimismo, que apenas el usuario abre la aplicación se le muestra como información en forma de mensaje Toast el nivel de batería del dispositivo.



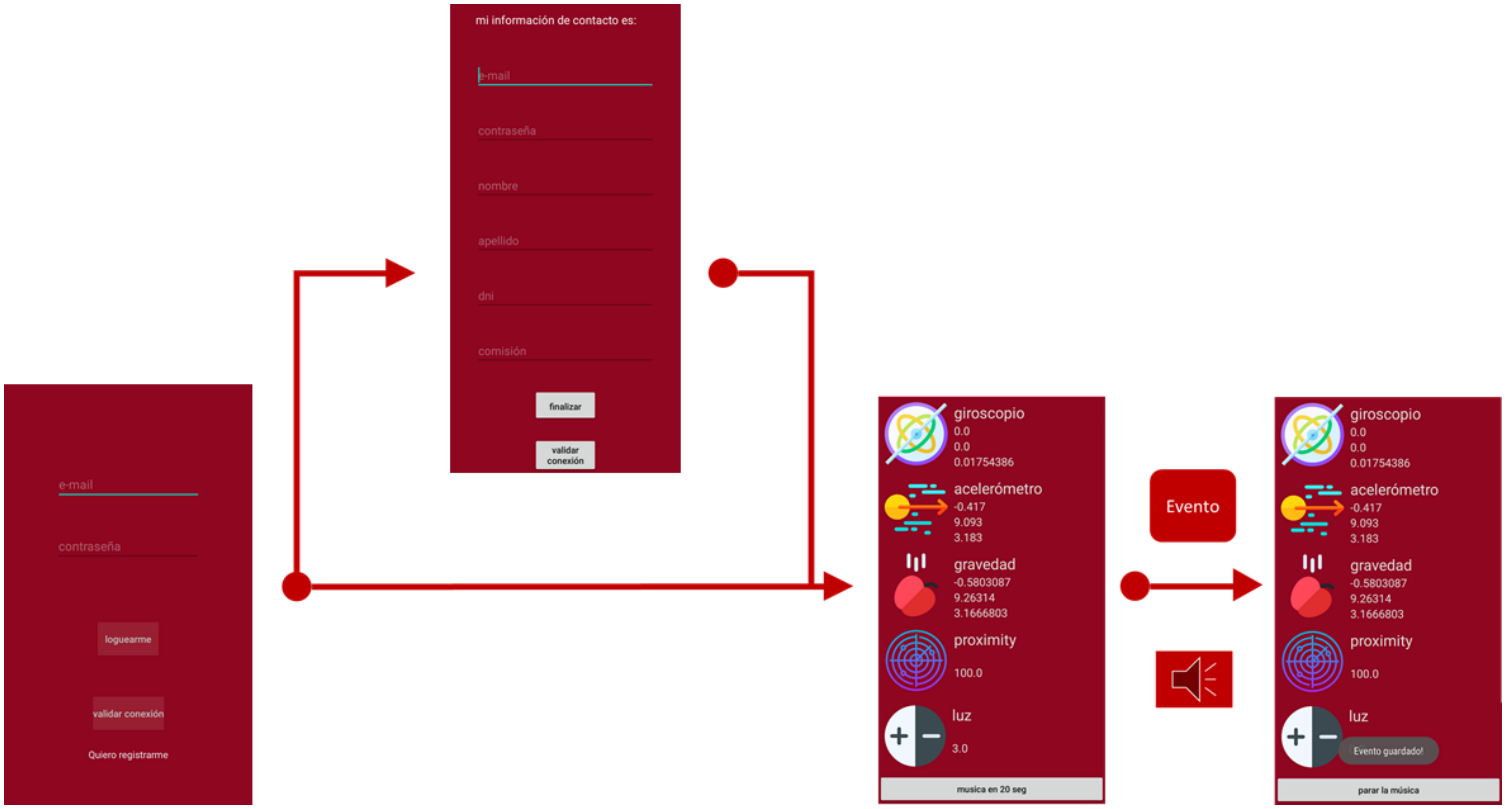
La aplicación también cuenta con el botón **validar conexión** para verificar mediante un mensaje en pantalla si se encuentra o no estable la conexión a Internet. Este botón se encuentra en ambas pantallas de **Logueo y Registro**.



Repositorio de GitHub:

https://github.com/cristian-rios/sensor-visualizer

Diagrama funcional de navegación de Activities:



En orden:

1) Logueo inicial del usuario con las credenciales de e-mail y contraseña - **LoginActivity**
2) En caso de querer registrarse, vista de registro de usuario con campos de entrada de datos e-mail, contraseña, nombre, apellido, DNI y comisión – **RegisterActivity**
3) En caso de logueo/registro exitoso, vista de visualización de sensores – **SensorActivity**. Desde esta vista se visualizarán los eventos ocurridos relacionados a los umbrales de los sensores (predefinido en la aplicación)
4) En caso de ocurrir un evento, se muestra en la misma activity el mensaje de confirmación de evento ocurrido (en este caso, que se haya pulsado para reproducir música) – misma **SensorActivity** con mensaje de evento

Sincronización

Para realizar el Timer de los 20 segundos que empiezan a correr una vez se apreta el botón, se utilizó el método de sincronización de AsyncTasks. En la vista de SensorActivity, una vez pulsado el botón, se crea el Thread asincrónico asignándole un Id. En el primer método que se ejecuta de este thread, **onPreExecute**, se muestra al usuario un mensaje Toast con la leyenda "sonando en 20 segundos" como indicación de que comenzó la ejecución del thread y se deshabilita el botón de música. En este momento, este nuevo hilo ya se desvinculó del anterior.

En el método **doInBackground**, el cual es en sí mismo la ejecución del hilo, se seteó la espera del mismo con la instrucción Thread.sleep(..). Dado que no necesitamos en este caso ir devolviendo datos al hilo principal, no estamos haciendo uso del método **onProgressUpdate**, sino que esperamos a que termine dicho hilo (timer) para recién ahí pasar a la ejecución del método **onPostExecute**, en donde se da la orden de inicio al servicio de música, se muestra el mensaje en pantalla al usuario y se habilita el botón respectivo con la leyenda actualizada "parar la música".

Por otra parte, también se utilizaron los sensores de la clase SensorManager, los cuales registran "listeners" en hilos separados por cada sensor. Esto es, dado que todos ejecutan el mismo método **onSensorChanged**, todos los hilos de los sensores deben ser sincronizados para evitar errores de concurrencia, lo cual se realizó con la declaración synchronized (this). De esta forma, dentro de la función de la sincronización, podemos saber cuál fue el sensor que generó el evento de cambios, obtener sus valores y trabajar con ellos, notificar el evento, mostrarlo por pantalla, etc.

Comunicación entre Activities

Un punto muy importante de la aplicación son las request, y dado que durante el tiempo de uso casi todas se corresponden a POST de eventos, es necesario trasladar el token de autenticación que se recibe en la vista de login o register a la de sensores, encargada de estas últimas peticiones.

Para esto, se hace uso de la clase Intent. Al trasladarse entre activities, se genera un nuevo objeto Intent con la información (parámetros del constructor) del activity actual en el que me encuentro, y la clase del activity al que deseo ir. En esencia, si estoy en la vista de Login, el login fue exitoso y necesito moverme a la vista de Sensores, debería hacer lo siguiente (código copiado del proyecto):

```
token = response.body().getToken();
Intent intent = new Intent(LoginActivity.this, SensorActivity.class);
intent.putExtra("token", token);
startActivity(intent);
```

Esto es, obtenemos cuál fue el token del cuerpo de la request, creamos un nuevo objeto Intent con la información previamente mencionada, le cargamos el token como parte de la información que recibirá SensorActivity y finalmente la levantamos.

Dentro de SensorActivity, recuperamos esta información como:

```
Bundle bundle = getIntent().getExtras();
token = bundle.getString("token");
```

Esto es, obtenemos el Intent desde el Bundle, y desde la información que se cargó anteriormente se obtiene el token enviado. Una vez hecho esto ya se puede usar en esta activity.

Registro de eventos

Según se explicó anteriormente, la aplicación registra determinados eventos en un servidor externo a una API abierta proveída por la cátedra. Si bien ya se ha mencionado, los eventos registrados se listan a continuación:

type_events	description	Disparo
EVENTO_LOGIN	Se ha logueado correctamente	En inicio exitoso
EVENTO_ACELEROMETRO	Se ha leido valor de aceleracion sobre el umbral	Cuando la magnitud leída por el acelerómetro en cualquiera de los ejes supera el valor de 10 como umbral
EVENTO_GRAVITY	Se ha leido un valor de gravedad sobre el umbral para los tres ejes	Cuando la magnitud leída por el sensor de gravedad supera el valor de 5 en los tres ejes al mismo tiempo
EVENTO_REGISTRO	Se ha creado el usuario	En registro de usuario exitoso
EVENTO_MUSICA	Se ha comenzado la música	Cuando se ha superado el tiempo de 20 segundos del timer, una vez apretado el botón para escuchar música y la misma empieza a reproducirse

Comunicación con el servidor

Para la comunicación con el Servidor se utilizó la librería Retrofit, mediante la cual se pueden realizar requests a la API expuesta por la cátedra en http://so-unlam.net.ar/, y la librería GSON proveía por Google preparar y parsear los datos y llevarlos como string a la notación de objetos JSON.

Según la documentación oficial de Retrofit:

“Retrofit convierte tu API HTTP en una interfaz Java. […]

Usa anotaciones para describir la petición HTTP:

- Soporte para reemplazar parámetro URL y parámetro de consulta
- Conversión de objeto a cuerpo de petición (por ejemplo, JSON, protocol buffers).
- Subida de archivos y Cuerpo de petición multiparte" [1]

Para definir las request, el proyecto incluye clases de Request y Response para cada endpoint de la API – es decir, para cada request distinta en este caso, aunque sean del mismo tipo POST. Dentro de la misma se define la forma en que viajará el cuerpo de la request. Por ejemplo, siendo lo requerido por la API como cuerpo de la **request**:

{
"env": << "TEST" | "PROD">>,
"name": <<String>>,
"lastname": <<String>>,
"dni": <<Numerico>>,
"email": <<String>>,
"password": <<String>>,
"commission": <<Numerico>>,
}
Debemos definir una clase Request:

```
public class RegisterRequest {
    @SerializedName("env")
    private String env;
    @SerializedName("email")
    private String email;
    @SerializedName("password")
    private String password;
    @SerializedName("name")
    private String name;
    @SerializedName("lastname")
    private String lastname;
    @SerializedName("dni")
    private long dni;
    @SerializedName("commission")
    private long commission;
```

Mediante las anotaciones SerializedName, se definen los nombres de los atributos una vez serializados en formato JSON por la librería antes mencionada Gson.

Ahora, sabiendo que la respuesta esperada es del tipo:

{
"success": Boolean,
"env" : String,
"token": String,
"token_refresh": String,
}
Debemos definir también la clase cuyo formato se tomará para la deserealización de los datos:

```
public class RegisterResponse {
    @SerializedName("success")
    private Boolean success;
```

```
    @SerializedName("env")
    private String env;
    @SerializedName("token")
    private String token;
    @SerializedName("token_refresh")
    private String tokenRefresh;
    @SerializedName("msg")
    private String msg;
```

Una vez tenemos las dos clases, definí un servicio con la interfaz de los métodos que realizarían la request:

```
public interface RequestService {
    @POST("api/api/register")
    @Headers({
            "content-type: application/json",
    })
    Call<RegisterResponse> register(@Body RegisterRequest registerRequest);
}
```

Finalmente, para realizar la comunicación, se debe instanciar la request, configurar el builder de Retrofit y encolar la llamada al endpoint que necesitamos.

```
requestService = new Retrofit.Builder()
        .addConverterFactory(GsonConverterFactory.create())
        .baseUrl(getString(R.string.uri))
        .build()
        .create(RequestService.class);
…
Call<RegisterResponse> call = requestService.register(miRequest);
call.enqueue(new Callback<RegisterResponse>() {
    @Override
    public void onResponse(Call<RegisterResponse> call,
Response<RegisterResponse> response) {
…
```

Problemas durante el desarrollo del proyecto

Hubieron varios problemas que implicaron una leve demora respecto al desarrollo, uno de ellos fue el ubicar por qué, al cambiar de la vista de Login a Sensores, la aplicación dejaba de funcionar con un mensaje que indicaba esto mismo. Revisando los logs de la app, obtenía un IllegalArgumentException (java.lang.IllegalArgumentException: Receiver not registered), supuestamente dado que no se podía detener la activity, pues había un Receiver que no había sido registrado. Esto se debía a que había una llamada a:

```
unregisterReceiver(batteryReceiver);
```

dentro del método onStop(), y el receiver no había sido registrado correctamente.

Otro error que me tomó tiempo encontrar fue que el servicio MusicService era instanciado y llamado, pero nunca se ejecutaba el método onStartCommand. No obstante, todas las llamadas eran correctas (igualmente el código del servicio es simple en sí mismo). Encontré luego en StackOverflow que los servicios habían que declararlos en el AndroidManifest.xml de la siguiente forma:

```
<service android:name="services.MusicService"/>
```

Luego de declararlo allí, las llamadas funcionaron perfectamente.

También me tomó un tiempo averiguar la forma de acomodar los componentes en xml en el ScrollView, o mas bien, en el LinearLayout interno, ya que automáticamente se asignaba un valor a los campos de distancias del campo con respecto a sus bordes. Cambiando estos campos respectivos a match_parent y wrap_content donde corresponda, se pudo centrar las imágenes con los TextView correspondientes. [2]

Finalmente, un error que también tuve por el comienzo fue una excepción construyendo el proyecto del tipo:

```
Invoke-customs are only supported starting with android 0 --min-api 26
```

Lo que se arregló cambiando las siguientes configuraciones en el build.gradle:

```
android { [3]
  compileOptions {
    sourceCompatibility JavaVersion.VERSION_1_8
    targetCompatibility JavaVersion.VERSION_1_8
  }
} [4]
```

Bibliography

[1] Retrofit ~ 2013 Square, Inc., [En línea]. Available: https://square.github.io/retrofit/.

[2] StackOverflow - How do you make a LinearLayout scrollable?, [En línea]. Available: https://stackoverflow.com/questions/4055537/how-do-you-make-a-linearlayout-scrollable.

[3] Create a background service, [En línea]. Available: https://developer.android.com/training/run-background-service/create-service.

[4] StackOverflow - Invoke-customs are only supported starting with android 0 --min-api, [En línea]. Available: https://stackoverflow.com/questions/49891730/invoke-customs-are-only-supported-starting-with-android-0-min-api-26,.

[5] M. t. B. L. a. C. State. [En línea]. Available: https://developer.android.com/training/monitoring-device-state/battery-monitoring.

[6] C. d. y. s. e. e. d. l. conectividad. [En línea]. Available: https://developer.android.com/training/monitoring-device-state/connectivity-monitoring?hl=es.