

# Aplicaciones Móviles

---



---

*Autores:*  
María Sanz Gómez  
Álvaro Asensio Calvo  
Gonzalo Guerrero Torija

7 de julio de 2021



## Índice

<b>1. Funcionalidad de la aplicación</b>	<b>2</b>
1.1. Registro de actividades físicas . . . . .	2
1.2. Registro de macro nutrientes . . . . .	3
1.3. Historial con todas las actividades físicas registradas . . . . .	4
1.4. Registro de usuario . . . . .	5
<b>2. Listado de componentes</b>	<b>6</b>
2.1. Actividades . . . . .	6
2.1.1. LoadingActivity.java . . . . .	6
2.1.2. MainActivity.java . . . . .	6
2.1.3. SportActivity.java . . . . .	6
2.1.4. MacrosActivity.java . . . . .	6
2.1.5. ScanningActivity.java . . . . .	6
2.1.6. ProfileActivity.java . . . . .	7
2.1.7. SavedActivity.java . . . . .	7
2.1.8. SavedActActivity.java . . . . .	7
2.2. Content providers . . . . .	7
<b>3. Listado de fuentes y librerías externas</b>	<b>8</b>
3.1. Actividades . . . . .	8
3.1.1. LoadingActivity.java . . . . .	8
3.1.2. SportActivity.java . . . . .	8
3.1.3. MacrosActivity.java . . . . .	8
3.1.4. ScanningActivity.java . . . . .	8
3.1.5. ProfileActivity.java . . . . .	9
3.1.6. SavedActivity.java, SavedActActivity.java y ActivityDbAdapter.java . . . . .	9
<b>4. Consideraciones</b>	<b>9</b>
<b>5. Contribuciones del equipo</b>	<b>10</b>
5.1. María . . . . .	10
5.2. Álvaro . . . . .	10
5.3. Gonzalo . . . . .	10

## 1. Funcionalidad de la aplicación

VivaVida es una aplicación desarrollada cuyo objetivo es mejorar los hábitos saludables de los usuarios. Para cumplir este objetivo se le proporciona al usuario una serie de herramientas útiles como son el registro de macro nutrientes personalizado y registro de actividades físicas para llevar a cabo una vida saludable y activa.

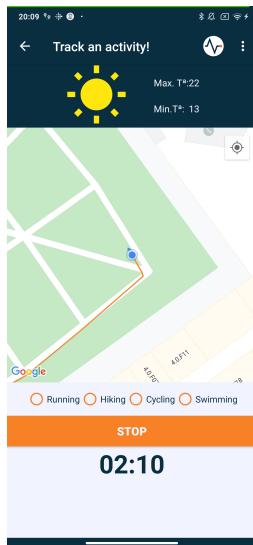


Pantalla de carga

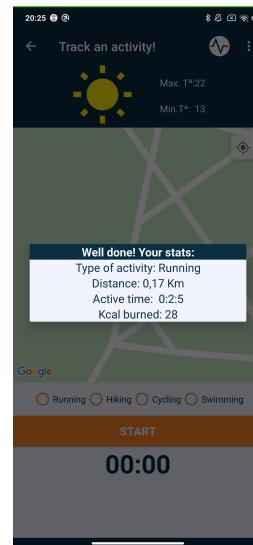


Menú principal con todas las funcionalidades disponibles

### 1.1. Registro de actividades físicas

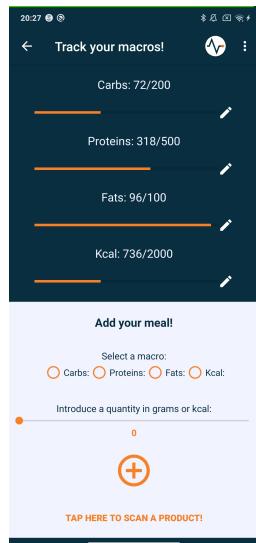


Registro de la actividad con trazado de la ruta en directo en un mapa, condiciones meteorológicas de la ubicación actual y cronómetro.

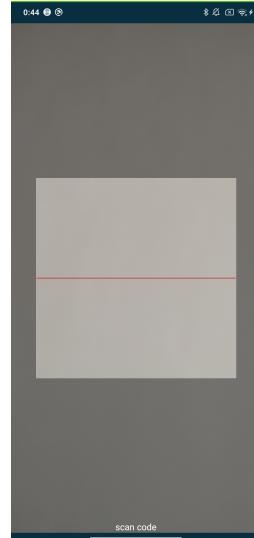


Estadísticas de la actividad una vez finalizada. Incluye tipo de actividad, tiempo en activo, distancia y aproximación de calorías quemadas.

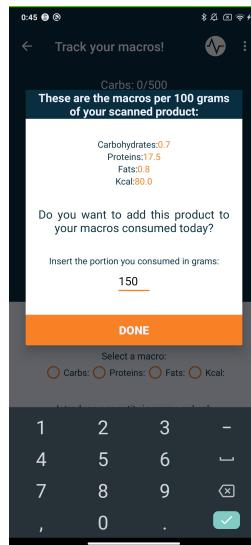
## 1.2. Registro de macro nutrientes



Registro manual de macros y calorías, posibilidad de editar los objetivos, seleccionar macro y ajustar cantidad con slider. Las macros y calorías registradas se resetean cada 24h.



Posibilidad de escanear cualquier producto alimenticio con un escáner en la cámara. Se escanea el código del producto y se comprueba si está en la base de datos de OpenFoodFacts.

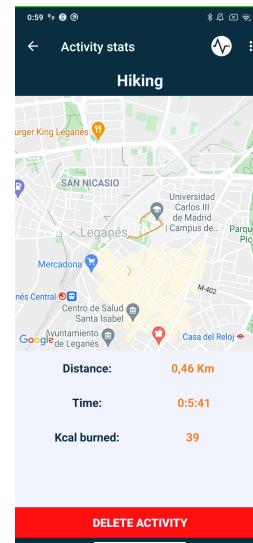


Devolución de las macros del producto escaneado (si está registrado en OpenFoodFacts), y registro de las macros según la cantidad ingerida en gramos por el usuario.

### 1.3. Historial con todas las actividades físicas registradas



Historial de todas las actividades físicas registradas, incluyendo el tipo de actividad y fecha como título de la entrada. Posibilidad de eliminar todas las actividades del historial. Las actividades se registran en una base de datos *SQLite*. Se pueden obtener más datos de una actividad pulsando sobre ella.



Estadísticas de la actividad almacenada, accediendo desde el anterior listado. Incluye tipo de actividad, tiempo en activo, distancia y aproximación de calorías quemadas, así como el trazado sobre un mapa del recorrido realizado. Posibilidad de eliminar individualmente la actividad seleccionada.

## 1.4. Registro de usuario



Registro de los datos del usuario. Posibilidad de tomar una foto o escoger una de la galería para seleccionarla como foto de perfil, pulsando sobre el ícono por defecto o la foto seleccionada. Entre los datos se incluyen nombre, género, altura y peso. Este último se usa para el cálculo aproximado de las calorías quemadas en una actividad.

## 2. Listado de componentes

### 2.1. Actividades

VivaVida se compone de un total de 8 actividades distintas que permiten la integración de las funcionalidades descritas anteriormente.

#### 2.1.1. LoadingActivity.java

Actividad correspondiente con la pantalla de carga. Cuando arranca la aplicación, se muestra durante 1 segundo y llama a `MainActivity.java`.

#### 2.1.2. MainActivity.java

Actividad con el menú principal. Integra 4 botones con los que se puede acceder a cada una de las funcionalidades.

#### 2.1.3. SportActivity.java

Actividad que permite el registro de una actividad física (a elegir entre *running*, *hiking*, *cycling* y *swimming*), con sus estadísticas y el trazado del recorrido en el mapa. También muestra las condiciones climatológicas de la ubicación actual. No se puede acceder a esta actividad si no se otorgan los permisos de ubicación necesarios. Las estadísticas se guardan en una base de datos adaptada a la actividad física. Esta base de datos consta de 2 tablas, una para registrar el tipo de actividad, distancia, tiempo y calorías, y la otra tabla para almacenar los puntos del recorrido.

El recorrido que se realiza mientras se registra la actividad (tras pulsar botón **START**) se muestra en tiempo real en la pantalla, al igual que el tiempo medido por el widget de cronómetro. El usuario no puede retroceder al menú principal hasta que no finalice la actividad en curso. Para calcular las calorías quemadas, es necesario que el usuario introduzca su peso en `ProfileActivity.java`. Cuando el usuario pulsa el botón de **STOP**, se muestra un diálogo con las estadísticas obtenidas, y se guardan en la base de datos.

#### 2.1.4. MacrosActivity.java

Actividad que permite el registro de macro nutrientes de forma manual a través de una selección y un *slider*, o bien escaneando el producto que quiere consumir el usuario. A medida que se van introduciendo las macros, bien de forma manual o escaneada, una serie de barras de progreso correspondientes a carbohidratos, proteínas, grasas y kilocalorías se van completando según los objetivos especificados por el usuario en esta misma actividad. Estas barras de progreso, por tanto el seguimiento diario, se restauran cada día.

Para el registro manual el usuario debe elegir la macro a registrar, seleccionar la cantidad con el *slider* y pulsar el botón con el icono de "+". Para el registro por escaneo, el usuario debe de pulsar el texto que indica esta funcionalidad ("**TAP HERE TO SCAN A PRODUCT!**"), se abre `ScanningActivity.java` y permite el escaneo de un producto, que a través de una conexión con el servidor de *OpenFoodFacts*, permite obtener toda la información nutricional del producto. Estas estadísticas son mostradas al usuario en un diálogo, y este debe introducir la cantidad en gramos que quiere registrar. Automáticamente, en función de la información y de la cantidad, se actualizan las barras de progreso.

#### 2.1.5. ScanningActivity.java

Actividad que abre la cámara con una interfaz para escanear un producto. Devuelve el código del producto escaneado, si existe. No se puede acceder a esta actividad si no se otorgan los permisos de cámara necesarios.

### 2.1.6. ProfileActivity.java

Actividad con los datos del usuario, que son almacenados en preferencias. Para la foto de perfil, el usuario debe pulsar sobre el icono por defecto para cambiarla. Un diálogo le muestra dos opciones, tomar una foto, que abre la cámara, permite al usuario tomarse una foto, y establecerla como foto de perfil, o bien escogerla desde la galería. Para ello se guarda la URI en el caso de tomar una foto (se toma la foto y se crea un fichero, del cual se obtiene la URI), o el directorio en el caso de elegirla desde la galería. Una vez elegida una opción, la foto se establece, y podrá cambiarla pulsando sobre la misma. El resto de información; nombre, sexo, peso y altura también se guarda en preferencias y se muestran cada vez que el usuario entra en esta actividad. No se puede acceder a esta actividad si no se otorgan los permisos de cámara necesarios.

### 2.1.7. SavedActivity.java

Actividad que muestra todas las actividades registradas en la base de datos en forma de lista, incluyendo únicamente el tipo de actividad y la fecha cuando se realizó. Si se pulsa sobre un elemento de la lista, se abre SavedActActivity.java, y muestra todas las estadísticas. Consta de un botón ("**DELETE ALL ACTIVITIES**") que permite eliminar todas las entradas de la base de datos. Para confirmar esta acción, se muestra un diálogo al usuario, preguntando si desea eliminar todas las actividades registradas.

### 2.1.8. SavedActActivity.java

Actividad que muestra todas las estadísticas de una actividad registrada (tipo de actividad, distancia, tiempo y calorías quemadas) así como el recorrido trazado en dicha actividad, que se muestra sobre un mapa. Consta de un botón ("**DELETE ACTIVITY**") que permite eliminar esa actividad (estadísticas y puntos del recorrido) de la base de datos.

## 2.2. Content providers

El principal content provider de la aplicación es una base de datos de tipo *SQLite*, que consta de dos tablas, una para almacenar las estadísticas básicas de la actividad a registrar, y otra para almacenar los pares de puntos (latitud y longitud) que conforman el trazado de la ruta.

La primera tabla está formada por 7 columnas:

- KEY\_ROWID1: Id de cada fila, en este caso, de cada actividad.
- KEY\_TYPE: Tipo de actividad.
- KEY\_DATE: Fecha en la que se realizó la actividad.
- KEY\_DISTANCE: Distancia recorrida en la actividad.
- KEY\_TIME: Tiempo de duración de la actividad.
- KEY\_ROUTEID: Id de la ruta realizada. Se comparte con la segunda tabla.

La segunda tabla está formada por cuatro columnas:

- KEY\_ROWID2: Id de cada fila, en este caso, de cada par de puntos.
- KEY\_LAT: Latitud del punto.
- KEY\_LNG: Longitud del punto.
- KEY\_ROUTEID: Id de la ruta realizada. Varios pares de puntos comparten este ID, que va a asociado a una ruta de una actividad.

Para poder usar la base de datos se ha creado una clase adaptadora con el objetivo de facilitar el manejo con *SQLite*. Esta clase es ActivityDbAdapter.java.

### 3. Listado de fuentes y librerías externas

A continuación se especifican las fuentes y librerías externas que se han usado en cada actividad o clase, y la función que realizan dentro de ella. Cada una de las siguientes contiene un hipervínculo a la url que incluye documentación de la librería o resuelve el problema expuesto.

#### 3.1. Actividades

##### 3.1.1. LoadingActivity.java

- Cómo hacer una *Splash Screen*.

##### 3.1.2. SportActivity.java

- SDK de Maps para Android: Se ha empleado tanto para mostrar el mapa y configurarlo, así como para emplear la herramienta de trazado *Polyline*.
- AEMET *OpenData*: Se ha utilizado su API para poder obtener el tiempo a través de una conexión con un servidor HTTPS y lectura de un fichero .xml.
- Trazado de rutas con *Polyline*.
- Añadir punto a punto - *Polyline*: Para poder ir añadiendo puntos a un trazado *Polyline*.
- Método *distanceBetween*: Para calcular la distancia entre dos puntos.
- *Google Maps API Demos*: Proyecto que incluye múltiples ejemplos de distintos usos del SDK de Maps. De estos ejemplos se ha obtenido la clase *PermissionUtils.java*.
- *AsyncTask*: Para establecer la conexión al servidor de AEMET.
- *XmlPullParser*: Para entender y realizar el parseado xml devuelto por AEMET.
- Proyectos de ejemplo proporcionados por los profesores: *HelloAppMov*, *NetworkUsage* y *HttpsAppMov*.

##### 3.1.3. MacrosActivity.java

- Cómo obtener JSON dado un código de barras: Para que una vez devuelto el código de barras por escaneo, abrir conexión con *OpenFoodFacts*, y obtener la información básica a través de un JSON devuelto.
- Cómo utilizar una barra de progreso.

##### 3.1.4. ScanningActivity.java

- Código abierto de *ZXing*: Código que permite el uso de un lector de código de barras dentro de la aplicación de manera sencilla.
- Cómo integrar el escáner de código de barras *ZXing* en forma vertical.

### 3.1.5. ProfileActivity.java

- Cómo hacer una foto y guardarla en un fichero: Para poder utilizar la opción de tomar una foto para la foto de perfil, y gestionar su URI en preferencias.
- Cómo capturar una imagen de la cámara y mostrarla en una actividad: Para ver los permisos necesarios y comparar con una forma alternativa de sacar una foto.
- Dialogo para escoger una foto o tomarla desde la cámara: Para mostrar un dialogo que de a elegir al usuario entre dos opciones, y como gestionar cada una de ellas, incluyendo los permisos necesarios en cada caso.
- Cómo manejar *ImeOptions done button click*: Para que cuando al usuario le salga el teclado para introducir sus datos, cuando le de al botón de aceptar (tick en el teclado), se guarden esos datos en preferencias.

### 3.1.6. SavedActivity.java, SavedActActivity.java y ActivityDbAdapter.java

- Proyecto de ejemplo NotepadDBAppMov: Se ha modificado la clase adaptadora para la creación de una base de datos con dos tablas y de las keys concretadas en la [sección 2.2](#). Se han adaptado los métodos y creado nuevos para la correcta gestión de los datos. También se han usado las clases de las actividades para entender el funcionamiento de la base de datos y mostrar sus entradas en un *ListView*.

## 4. Consideraciones

Al usar el código abierto de *ZXing*, es necesario añadir a las dependencias del `build.gradle(Module)` las implementaciones indicadas en el *GitHub* donde se hospeda el código. Sin embargo al compartirlo entre los componentes del grupo, en `ScanningActivity.java`, no se reconocía la librería `com.journeyapps.barcodescanner.CaptureActivity`. Si se da dicho error, se soluciona añadiendo a las dependencias de `build.gradle(Module)` las siguientes implementaciones:

```
implementation 'com.journeyapps:zxing-android-embedded:3.0.2@aar'  
implementation 'com.google.zxing:core:3.2.0'
```

Por otro lado, en `MacrosActivity.java`, el tiempo establecido para que se reinic peace los datos es de 30 segundos, debido a fines de testeo. Habría que sustituirlo si se quiere que se restablezcan cada 24 horas. Este valor aparece indicado en la parte del código donde tendría que cambiarse.

Puntualizar que en `ProfileActivity.java`, cuando el usuario introduce su nombre y datos, es necesario que le pulse al tick del teclado para que el manejador de *ImeOptions* pueda guardar los datos en preferencias. Si no, a pesar de que los datos aparezcan en el cuadro de texto, no se guardan.

Para obtener el tiempo en la ubicación en la que se encuentra el usuario, se obtiene un punto geográfico, y a partir de él, un código postal que se le pasa a la petición HTTPS a AEMET. Sin embargo, el código de localidad de AEMET no se corresponde con el código postal. Únicamente coinciden en los dos primeros dígitos. Por ello, para que sea totalmente funcional se ha de tener una base de datos que relacione código postal con código de localidad. Con el fin de probar la aplicación, solo se obtienen los dos primeros dígitos, y los tres restantes se han dejado a 079, de tal manera que en todas las pruebas se obtiene el código 28079, que corresponde con la localidad de Madrid.

También se ha de tener en cuenta que la localización tiene que estar activada para que funcionen correctamente los servicios de localización.

Finalmente indicar algunos errores/*warnings* que aparecen en la pestaña de *Run*, que no resultan críticos para la ejecución de la aplicación pero consideramos que habría que solucionarlos:

- W/xample.vivavid: Accessing hidden method Lsun/misc/Unsafe; ->getLong(Ljava/lang/Object;J)J (greylist,core-platform-api, linking, allowed)  
W/xample.vivavid: Accessing hidden method Lsun/misc/Unsafe; ->putLong(Ljava/lang/Object;JJ)V (greylist, linking, allowed)
- E/libprocessgroup: set\_timerslack\_ns write failed: Operation not permitted
- W/System: A resource failed to call close.

El primer *warning* se desconoce la causa y es el más recurrente. Los otros dos se han investigado, el segundo se debe al uso conjunto de mapas y diálogos, y el ultimo se debe a una llamada *close()* de la base de datos que no debería realizarse, puesto que ya está cerrada.

Los emuladores que se han usado han sido los siguientes:

- Pixel XL: 1440 x 2560 (560 dpi) y API de Android versión 30
- Nexus 5: 1080 x 1920 (xxhdpi) y API de Android versión 30
- Nexus 6: 1440 x 2560 (560 dpi) y API de Android versión 30

Los dispositivos físicos que se han usado han sido los siguientes:

- Xiaomi Mi 9T Pro: 1080 x 2340 y API de Android versión 29
- Samsung A52: 1080 x 2400 y API de Android versión 30
- Samsung Galaxy A5 2017: 1920 x 1080 y API de Android versión 26

## 5. Contribuciones del equipo

### 5.1. María

Creación de *MainActivity.java*. Creación del registro manual de macros en *MacrosActivity.java*, incluyendo sus respectivos diálogos, manejo de botones, barras de progreso, valores obtenidos así como guardado y obtención de valores a través de preferencias. Creación en *SportActivity.java* de los botones y su manejo, y gestión de la petición HTTPS al servidor de AEMET, incluyendo el parseado xml y presentación en el *layout* del tiempo atendiendo a los datos obtenidos. Depurado general.

### 5.2. Álvaro

Creación del registro por escaneo de macros en *MacrosActivity.java*. Guardado de las estadísticas de la actividad en la base de datos en *SportActivity.java*. Gestión y creación de la primera tabla de base de datos, que almacena las estadísticas básicas de una actividad registrada. Visualizado de las entradas de la base de datos en *SavedActivity.java* y visualizado de las estadísticas guardadas en *SavedActActivity.java*.

### 5.3. Gonzalo

Diseño estético de la aplicación (logos, colores). Creación de la pantalla de carga *LoadingActivity.java*. Integración del cronómetro y mapa con capacidad de trazar el recorrido en tiempo real en *SportActivity.java*, y guardarlo en la base de datos. Creación de *ProfileActivity.java*. Gestión y creación de la segunda tabla de base de datos, que almacena los puntos del recorrido realizado. Integración del mapa y trazado guardado en *SavedActActivity.java*. Gestión de los botones para borrar entradas en la base de datos. Depurado general y retoque de todos los *layouts*.