**Juan Camilo Restrepo Velez 000373886**

Tabla de contenido

[1) Ajuste MariaDB 2](#_Toc40317796)

[2) Ajuste API REST 3](#_Toc40317797)

[2.1) updateData.php 3](#_Toc40317798)

[2.2) postData.php 6](#_Toc40317799)

[3) ConexionApiRest 8](#_Toc40317800)

[3.1) Constructor 8](#_Toc40317801)

[3.2) getData 8](#_Toc40317802)

[3.3) setData 9](#_Toc40317803)

[3.4) updateData 9](#_Toc40317804)

[3.5) downloadData 10](#_Toc40317805)

[3.6) setHttps 10](#_Toc40317806)

[3.7) tryConnect 11](#_Toc40317807)

[4) EU Estudiante 12](#_Toc40317808)

[4.1) MainActivity 12](#_Toc40317809)

[4.2) ActividadActivity 16](#_Toc40317810)

[5) EU Profesores Prototipo 17](#_Toc40317811)

[5.1) MainActivity 17](#_Toc40317812)

[5.2) SesionActivity 18](#_Toc40317813)

[5.3) NewSesionActivity 21](#_Toc40317814)

[5.4) ActividadActivity 22](#_Toc40317815)

[Referencias 25](#_Toc40317816)

# 1) Ajuste MariaDB

Se debe adicionar en la base de datos EduApps de MariaDB las siguientes tablas:

USE EduApps;

CREATE TABLE Estados

    (ID INT NOT NULL AUTO\_INCREMENT PRIMARY KEY,

     Descripcion VARCHAR(255) NOT NULL);

CREATE TABLE Estudiantes\_Actividades

    (Estudiante\_ID INT NOT NULL,

     Actividad\_ID INT NOT NULL,

     Estado\_ID INT NOT NULL,

     Observaciones VARCHAR(255) NOT NULL,

     PRIMARY KEY (Actividad\_ID, Estudiante\_ID),

     CONSTRAINT FKEstudiante\_Actividad

     FOREIGN KEY (Actividad\_ID) REFERENCES Actividades(ID)

      ON DELETE CASCADE

      ON UPDATE RESTRICT,

     CONSTRAINT FKActividad\_Estudiante

     FOREIGN KEY (Estudiante\_ID) REFERENCES Estudiante(ID)

      ON DELETE CASCADE

      ON UPDATE RESTRICT,

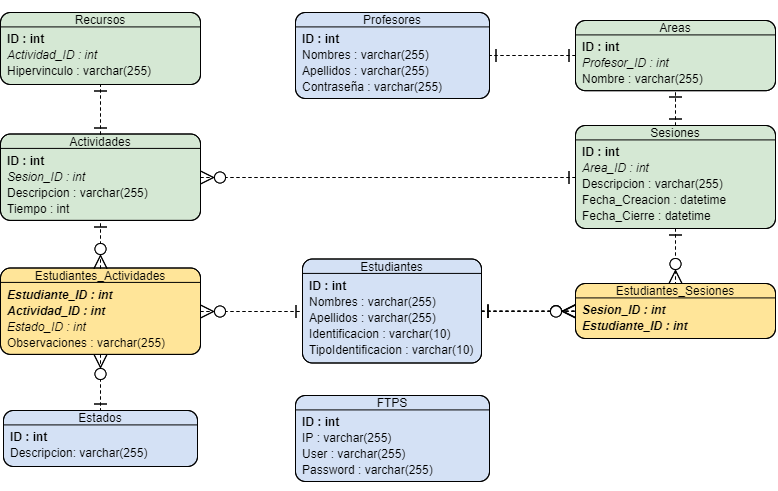
     CONSTRAINT FKActividad\_Estado

     FOREIGN KEY (Estado\_ID) REFERENCES Estados(ID)

      ON DELETE CASCADE

      ON UPDATE RESTRICT);

Dejando el siguiente modelo relacional en el nodo:



Por otro lado, se deben realizar la siguiente inserción:

INSERT INTO Estados VALUES (0,‘Iniciada’),(0,‘Finalizada’),(0,‘Abandonada’);

# 2) Ajuste API REST

Como el API REST que se decidió implementar no está completo (faltan las funciones de update y delete) se nos presenta el impedimento para poder actualizar el estado de la actividad por lo cual debemos implementar al menos la funcionalidad de atualizar los datos, además de mejorar la función de insertar datos ya que no nos devuelve el registro que se acaba de insertar (se requiere para notificar con cuál ID quedó dicho registro).

## 2.1) updateData.php

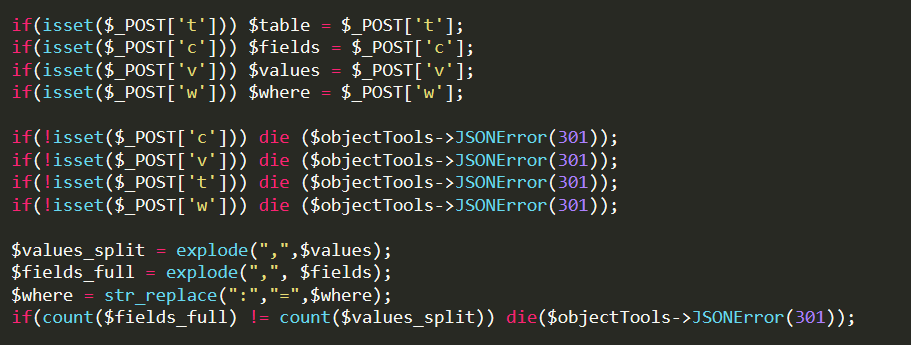
Ahora bien, para poder actualizar datos por medio del API REST se escribe el código necesario en el archivo updateData.php para implementar dicha función por medio de la URL que se deben enviar los siguientes parámetros mediante el método POST:

* Tabla
* Columna
* Valor
* Condición

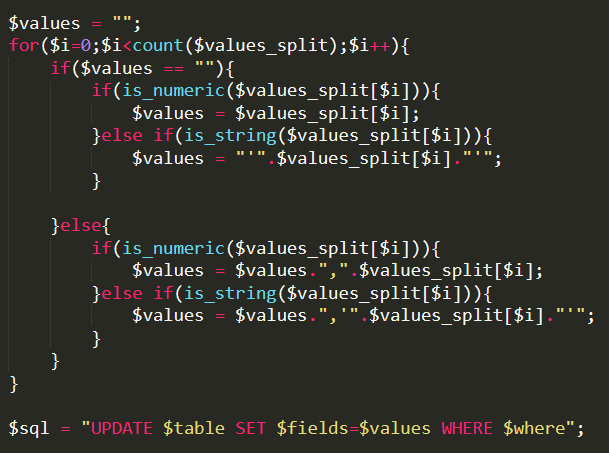
Entonces la URL queda (recordar que da tanto con HTTP como HTTPS por la instalación del certificado del sprint 3):

*https://192.168.0.101/ApiRest/updateData.php?t=table&c=column&v=newValue&w=condicion*

Para el ***código en php*** se reutilizan todas las validaciones que se hacen los demás archivos php por lo que solo se mostrará lo esencial como la obtención de los parámetros:



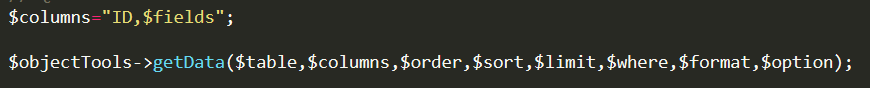
Luego se realizan las validaciones posteriormente mencionadas, para luego realizar la construcción de la instrucción update



Luego se utiliza una clase Tools que se encarga de la conexión con el servidor MariaDB y aplica la sentencia update:

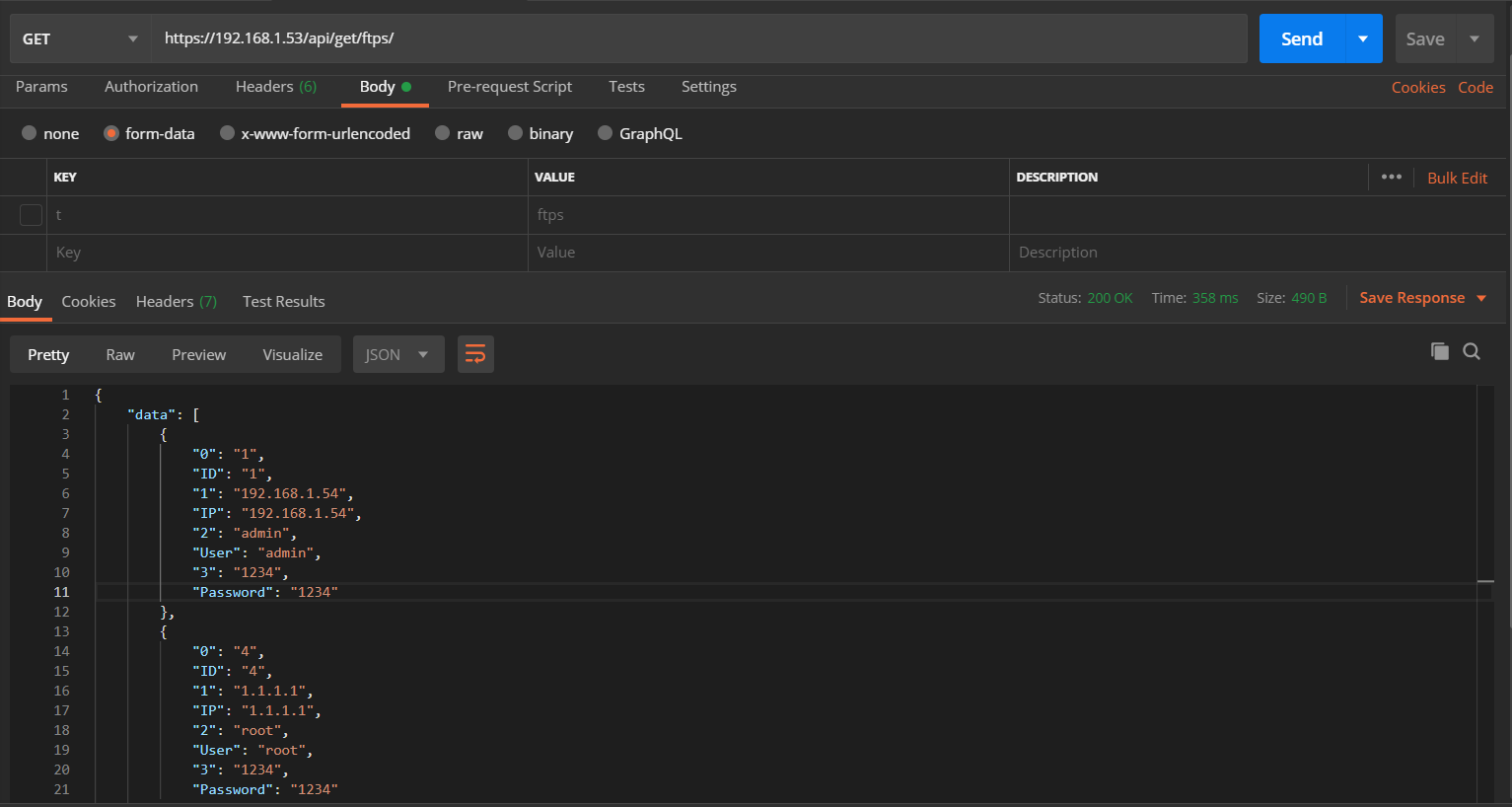


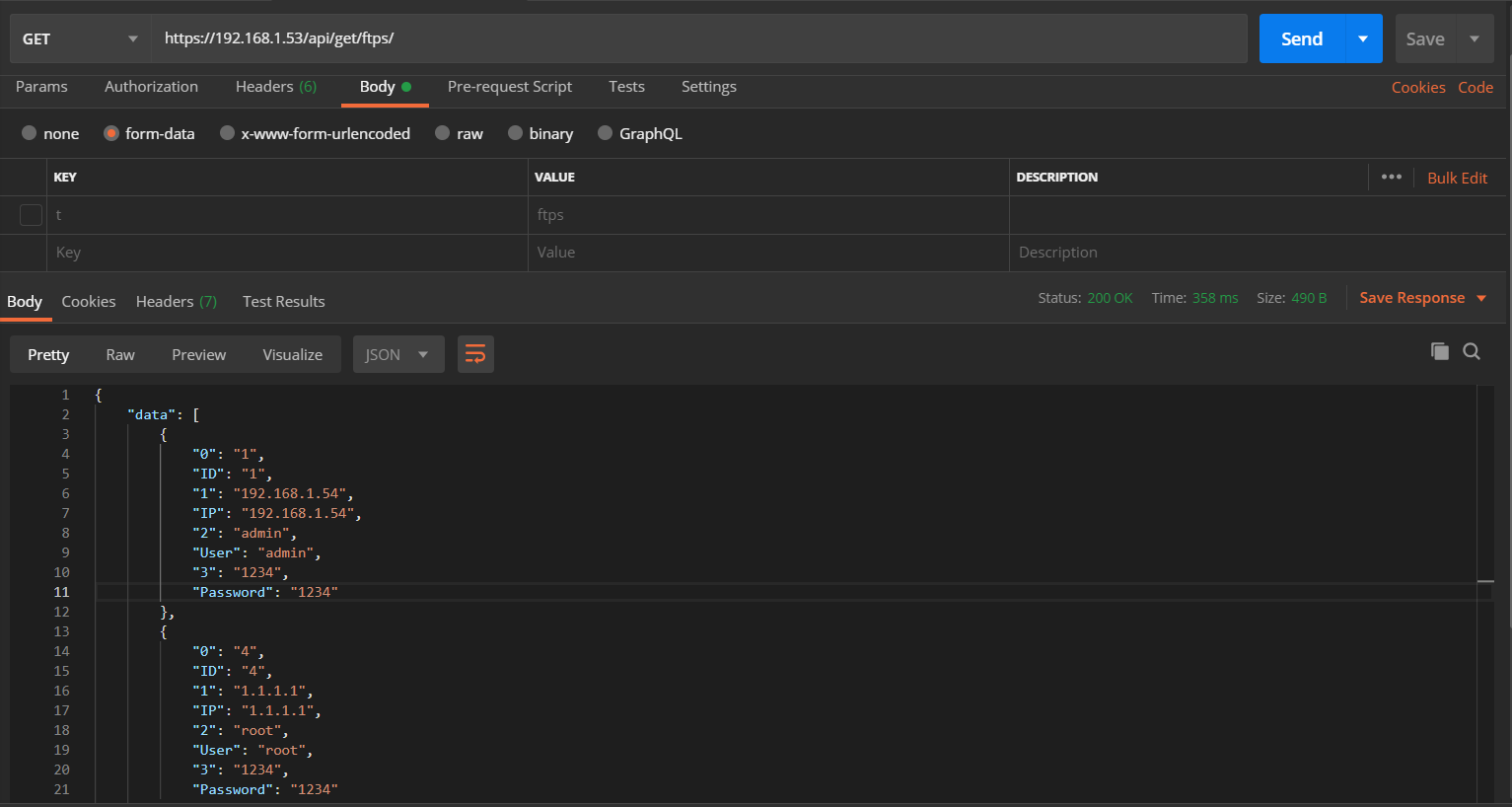
Luego se procede a llamar nuevamente la clase, pero esta vez devolver un JSON con el ID y la columna modificada de la tabla:



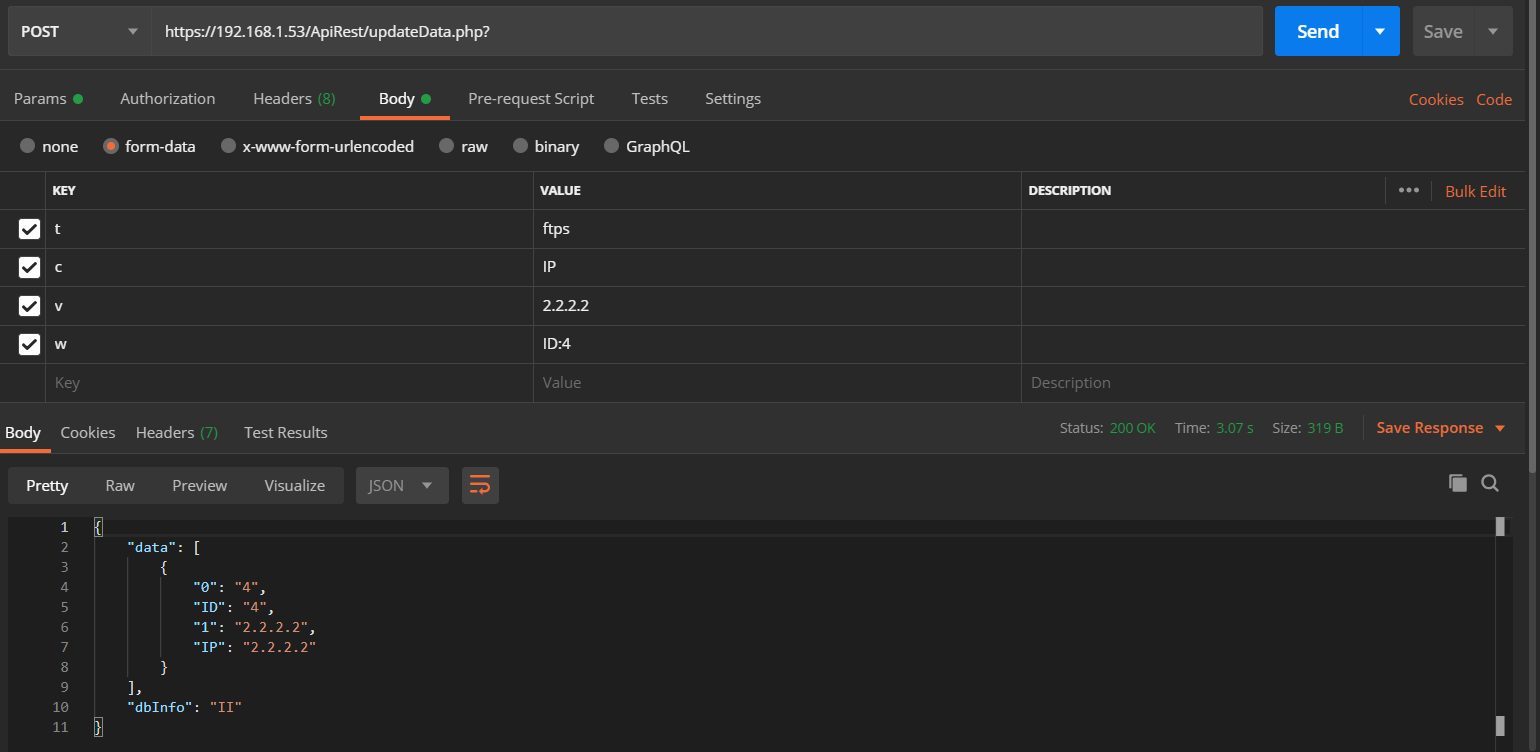
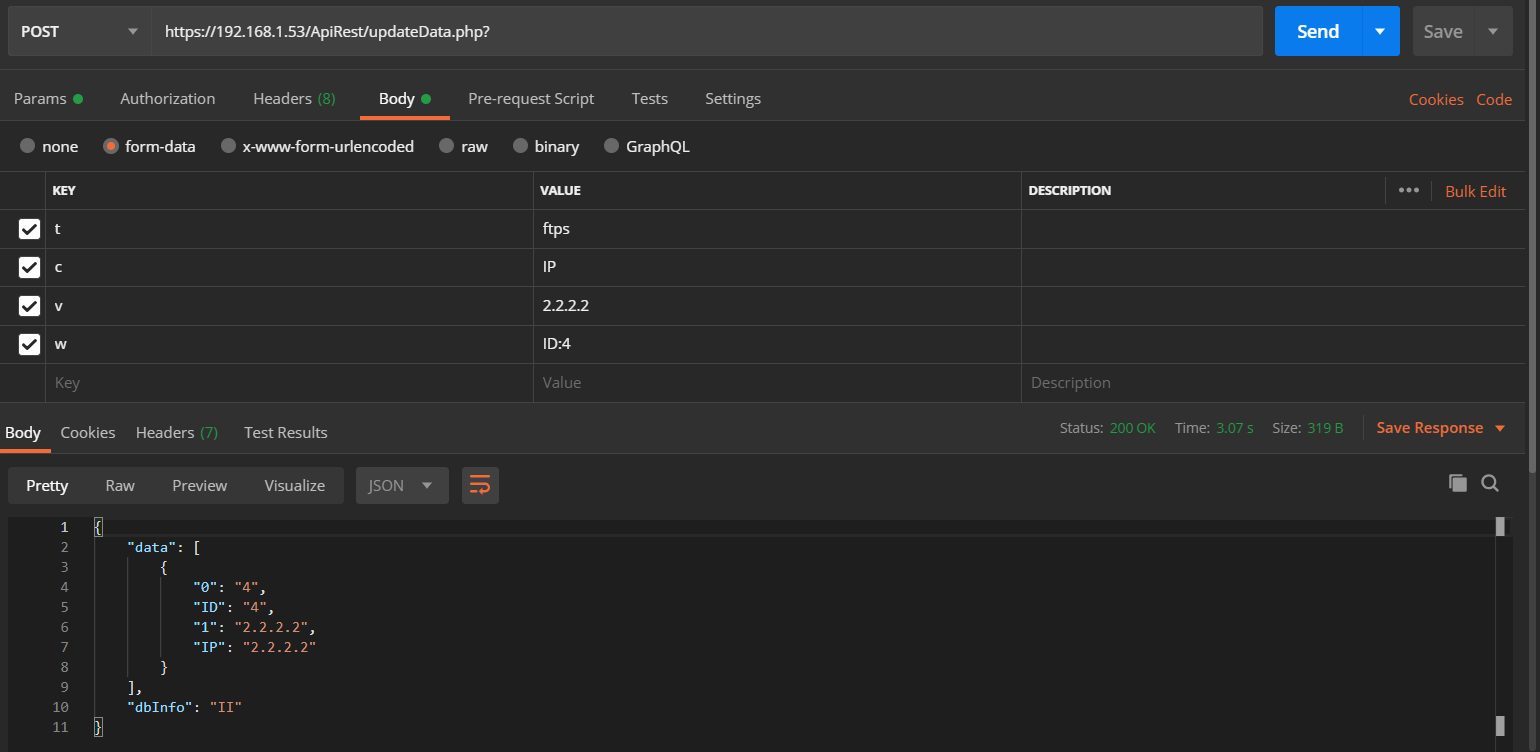
***Prueba desde Postman*** que es un programa que nos permite probar las funcionalidades de nuestra API REST para enviar parámetros por la URL por medio de los diferentes modos (GET, POST, PUT, entre otros).

Primero consultamos los registros que tiene la tabla ftps (Estos son los del servidor de prueba):

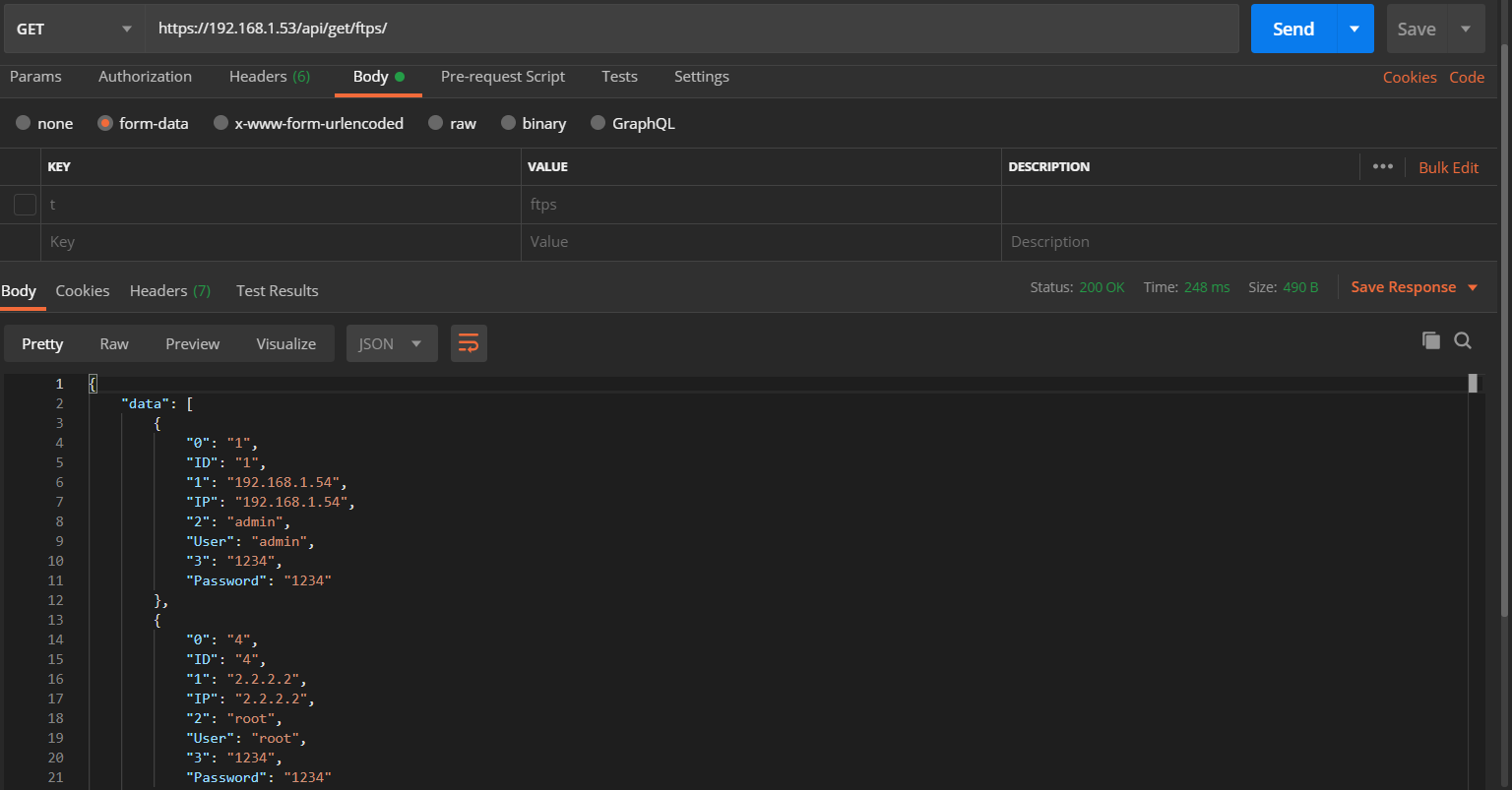
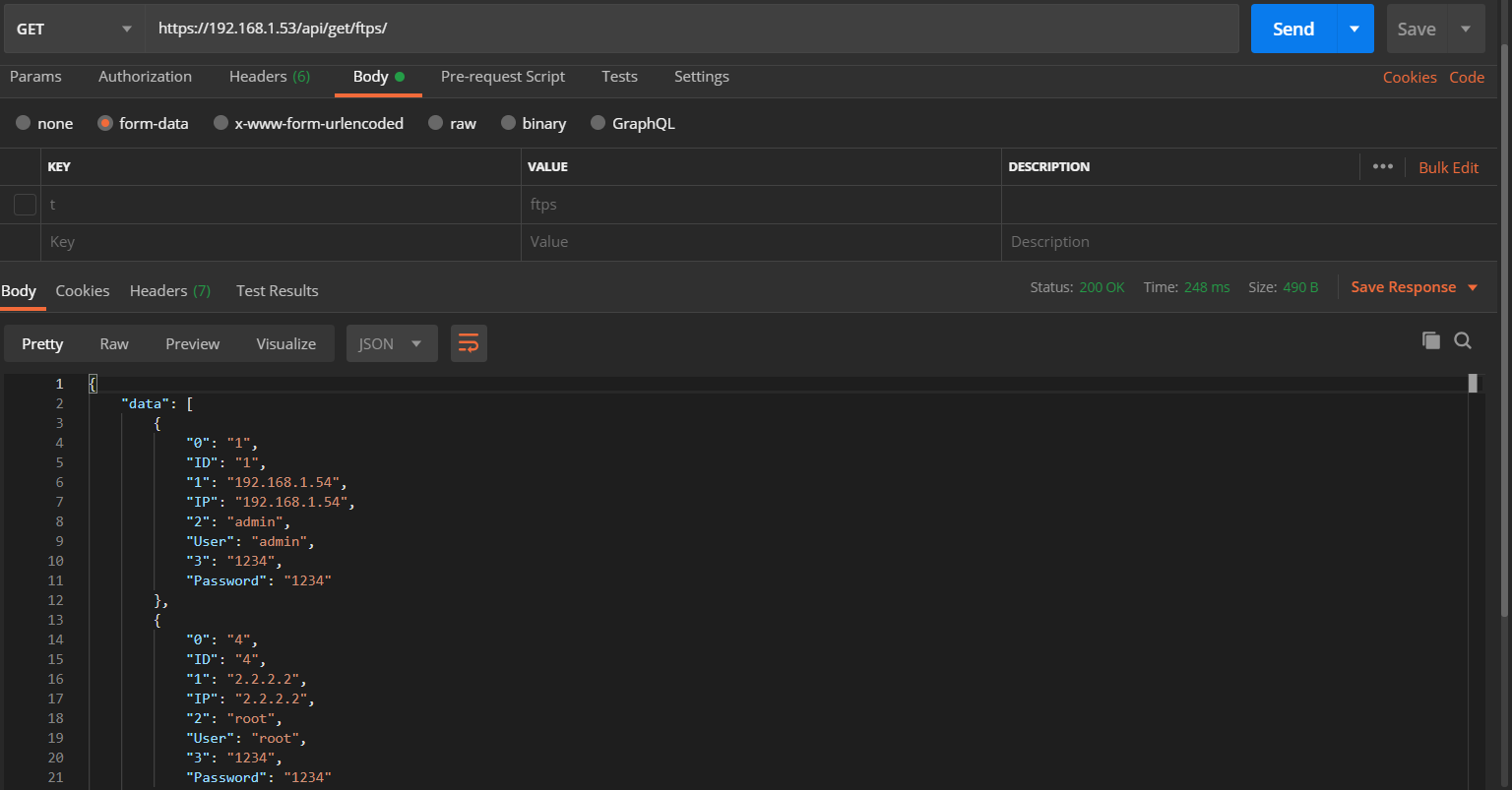




Luego se procede a cambiar la IP del registro con ID igual a 4:

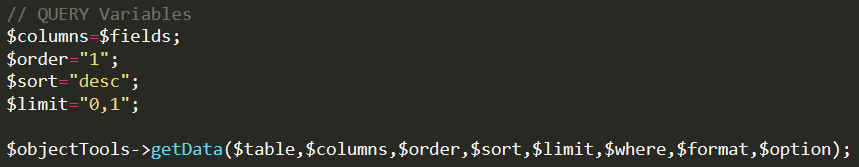


Y ahora volvemos a consultar todos los registros:



## 2.2) postData.php

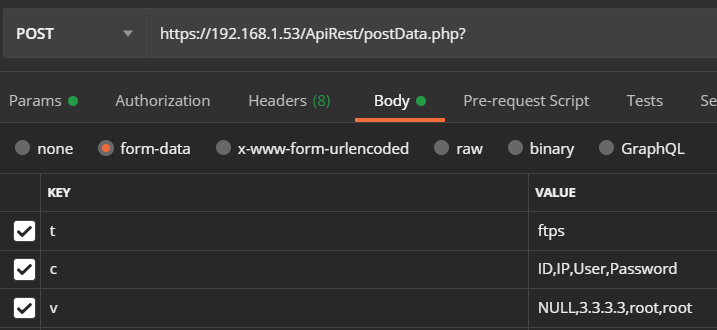
Por otro lado, la opción de insertar datos ya viene implementada, pero esta retorna el ID incorrecto del registro que se acaba de insertar por lo que se modifica la parte después de construir la consulta insert, que se hace lo mismo que con update, se llama un método de la clase Tools para consultar el último registro de la tabla, por ello se especifican todas las columnas, se ordena descendentemente por el ID y se limita solo a un resultado:

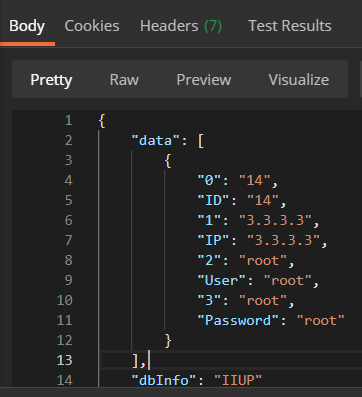


Al igual que update, se envían los parámetros por la URL así con el método POST:

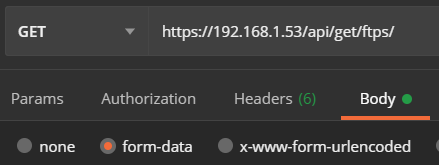
*https://192.168.0.101/ApiRest/postData.php?t=table&c=column1,column2&v=value1,value2*

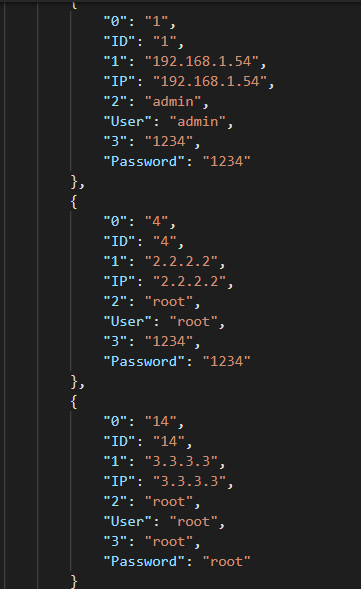
***La prueba en el programa Postman*** para insertar un nuevo registro:





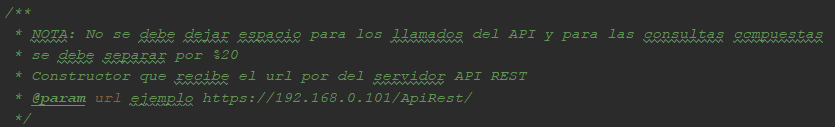
Y la consulta sobre la tabla donde se ve el nuevo registro:





# 3) ConexionApiRest

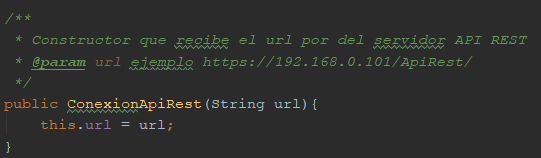
Con un mejor entendimiento del API REST se realiza una mejora de la clase ***ConexionApiRest*** haciendo que sea de más alto nivel para facilitar su uso y comprensión, además todos los métodos llevan el comentario con su descripción de lo que realiza, recibe y retornar, ejemplo:



Y como se ve en el comentario, en todos los llamados de los métodos del API REST no se puede dejar espacio ya que estos se hacen por la url, entonces para las consultas compuestas se debe separar escribiendo *%20*

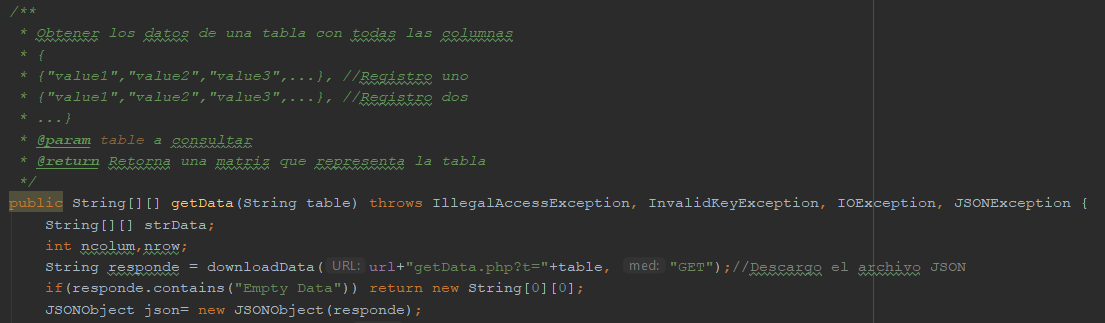
## 3.1) Constructor

Recibe la URL del servidor: *https://192.168.0.101/ApiRest/*

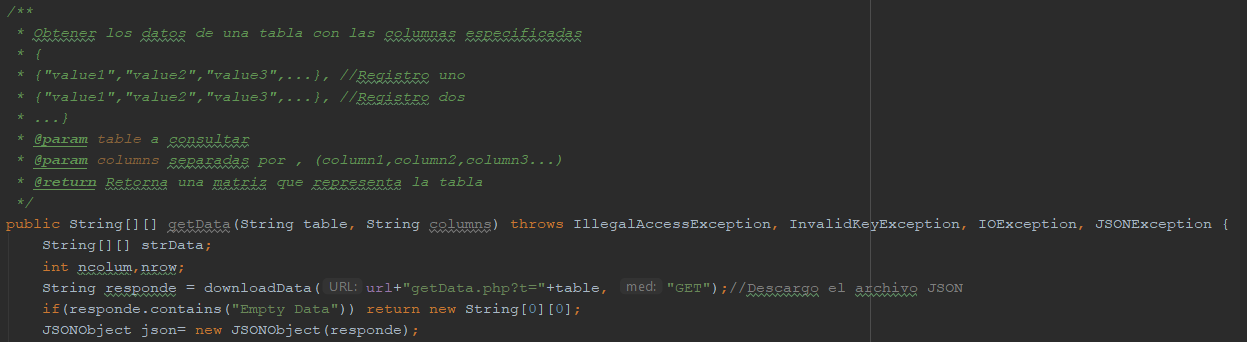


## 3.2) getData

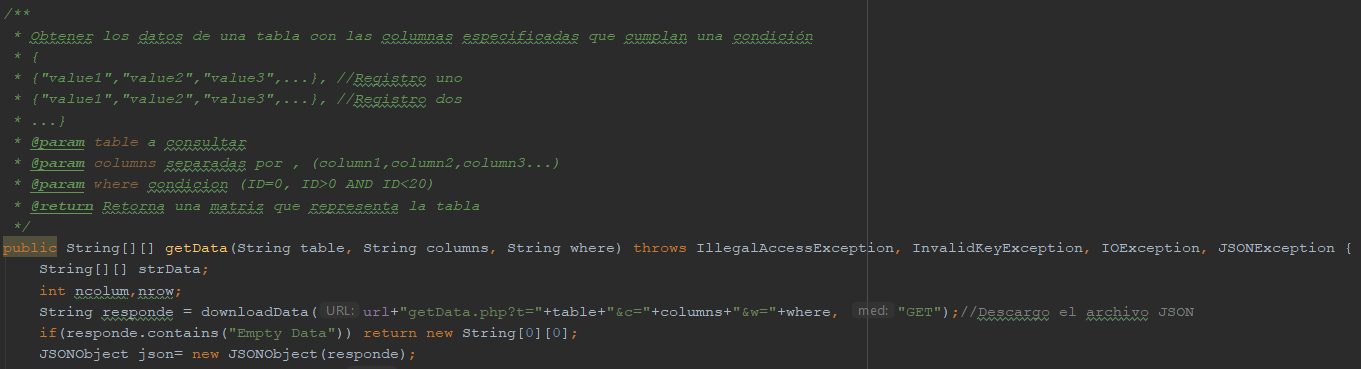
En la decodificación del JSON se mantiene, cambia la forma en que como se llama y en los parámetros que recibe. En este sentido puede obtener el contenido de toda la tabla:



Obtener solo algunas columnas de la tabla:

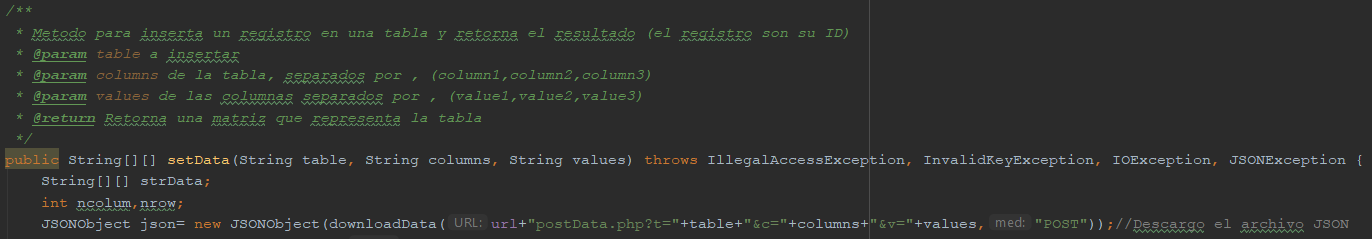


Obtener algunas columnas filtradas por un where:



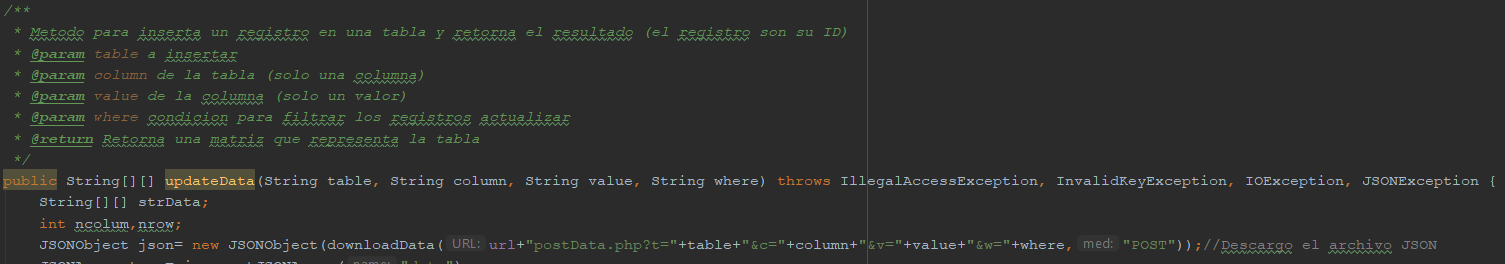
## 3.3) setData

El método enviar por la URL los parámetros para realizar la inserción por el método POST de HTTP y como también devuelve un JSON, tiene la misma decodificación que el método getData, por lo que cambia el URL a enviar a download y se especifica el método POST:



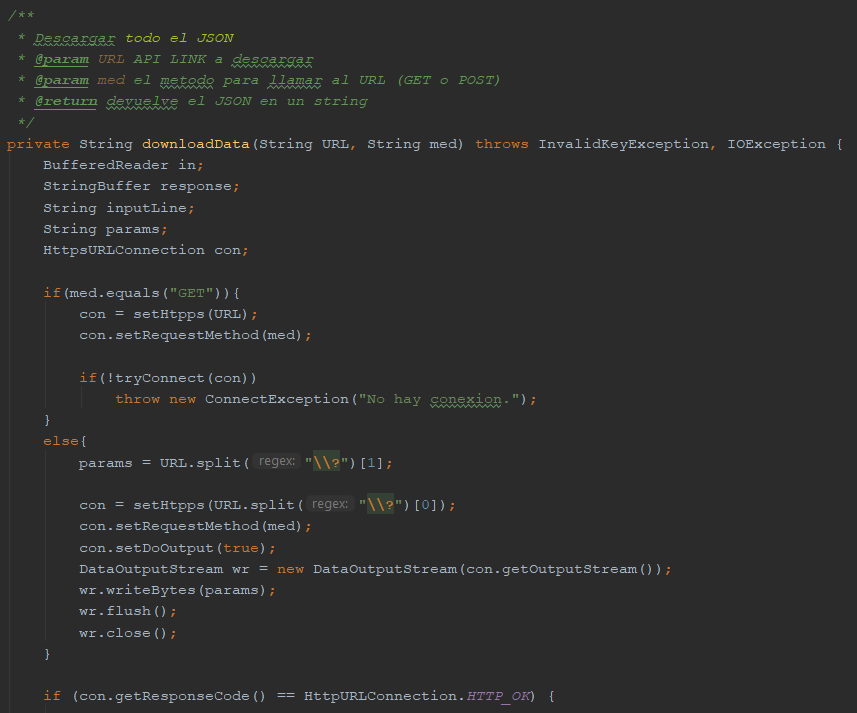
## 3.4) updateData

Al igual que setData, solo cambia el URL a enviar a download, ya que también se devuelve un JSON con el ID y la columna cambiada:



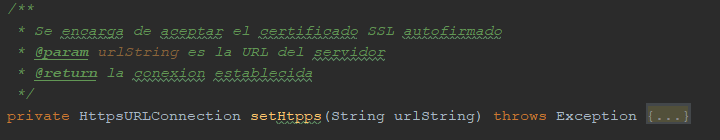
## 3.5) downloadData

Método privado que se encarga de descargar los JSON que retornan los API LINK, además de llamar un método para comprobar la conexión con la página cuando se van a consultar datos:



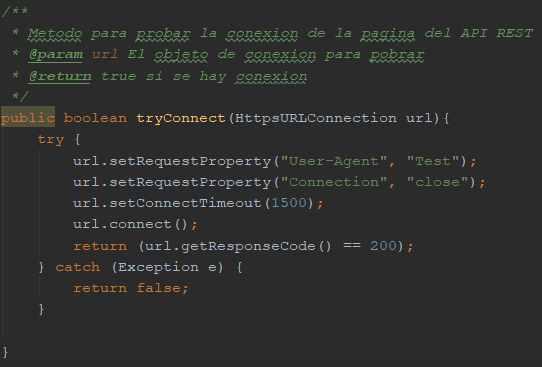
## 3.6) setHttps

Metodo privado que se encarga de gestionar el permiso para aceptar la conexión con servidor ya que este tiene un certificado SSL autofirmado:



## 3.7) tryConnect

Método para probar la conexión a la página de forma rápida y evitando que se cuelgue la app cuando no se pueda acceder al nodo:



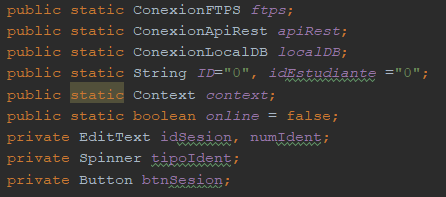
# 4) EU Estudiante

Se hará una breve recopilación de los llamados a las clases auxiliares para manejar la conexión con el servidor MariaDB y con el servidor FTP, así como las clases necesarias para el manejo del front y las nuevas implementaciones que abarca el manejo de la base de datos SQLite interna y todo el código para iniciar, actualizar y sincronizar los estados de las actividades.

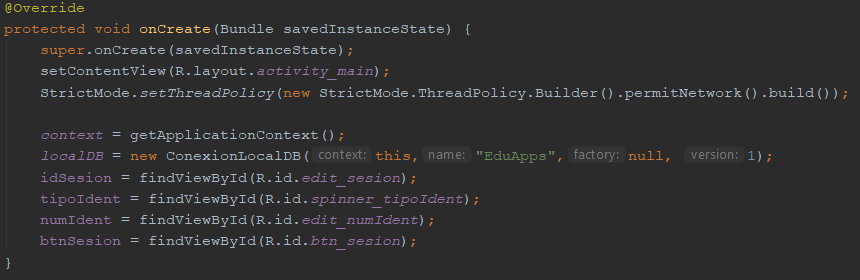
## 4.1) MainActivity

Esta actividad se encarga de iniciar el estado de la actividad, si es primera vez que el estudiante inicia sesión para dichas actividades, insertando los datos del nodo a la base de datos local o actualizando las tablas locales con la base de datos que está en el nodo (sin modificar la tabla que tiene el estado de las actividades), esta última opción si ya se ha iniciado sesión en el dispositivo con anterioridad. Además, si el dispositivo ya tiene información de la sesión que se desea iniciar y no hay conexión con el nodo, se permite entrar en estado *offline* para que el estudiante pueda visualizar las actividades y poder cambiar el estado de cada una de ser necesario.

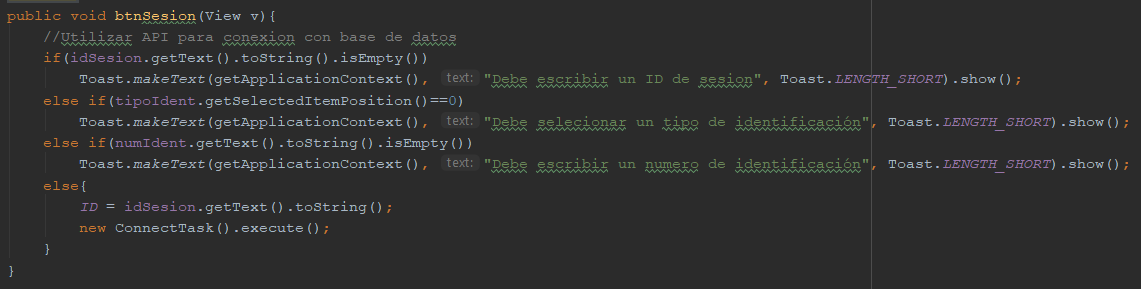
Primero, se establecen las variables para los objetos de las vistas y las instancias de conexión global como estáticas para no estar creando múltiples objetos con el mismo fin:



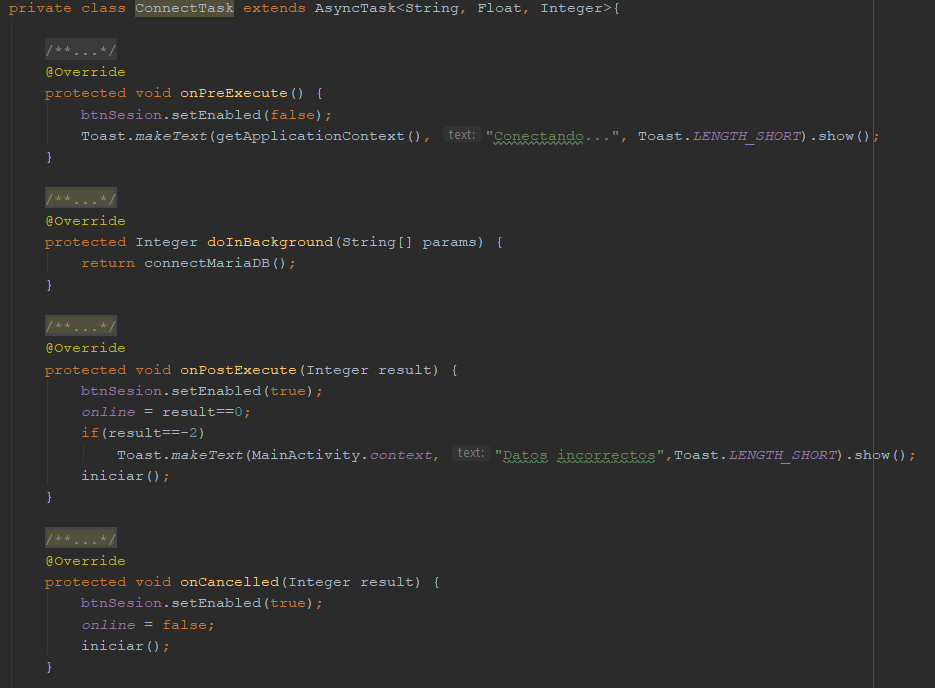
Luego, en el onCreate se hacen las asignaciones necesarias y se habilita la regla de descargar la información desde el hilo principal con la línea de *StrictMode*:



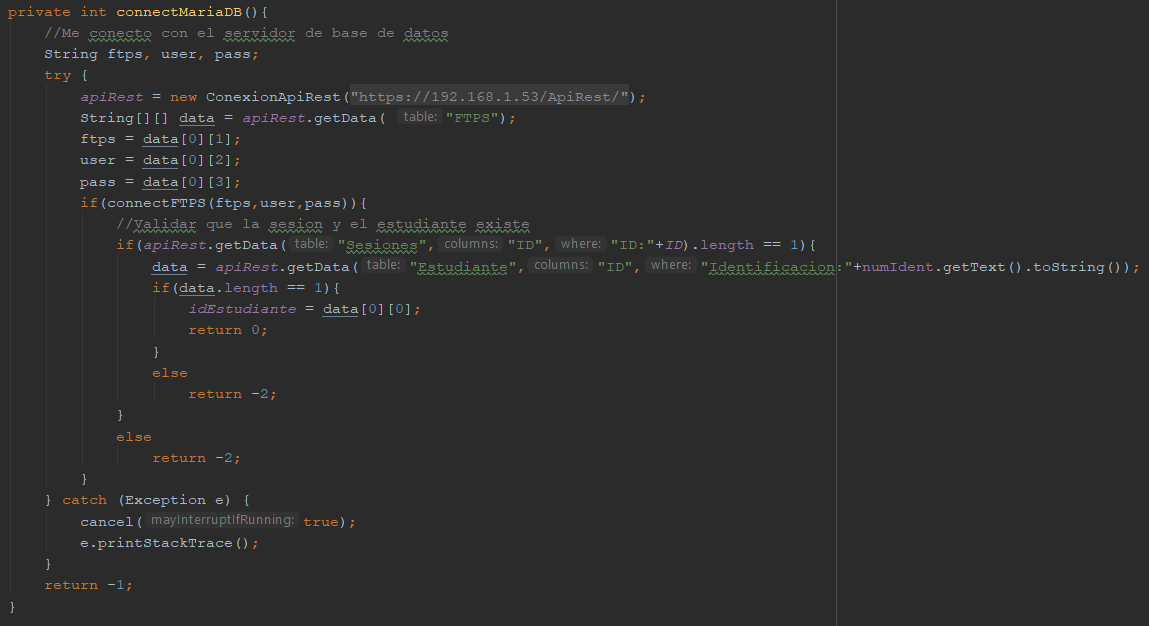
Cuando se dé clic en el botón *Iniciar sesión* se obtiene la información de los objetos haciendo una primera validación de que estos no estén nulos:



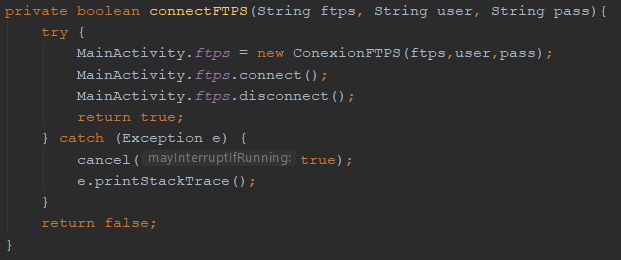
Luego se ejecuta un hilo para probar la conexión con el API REST, que puede retornar 3 valores; 0 (conexión exitosa), -1 (conexión fallida), -2 (Datos erróneos). Además, antes de que se ejecute el hilo se deshabilita el botón de iniciar para evitar que el usuario haga múltiples peticiones:



En el método *connectMariaDB* se realiza la prueba con el API REST, que si es exitosa se establece la variable *online* en *true* y se obtienen los datos del servidor FTP, la sesión y el estudiante para validar que los datos ingresados sean correctos:

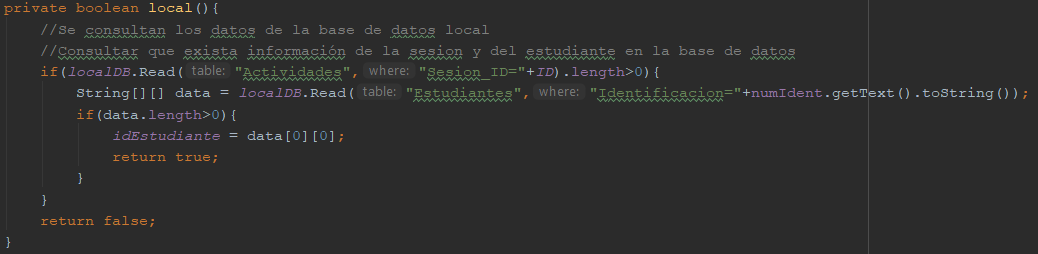


Luego, en el método *connectFTPS*, se valida la conexión con el servidor FTPS según los datos del nodo:



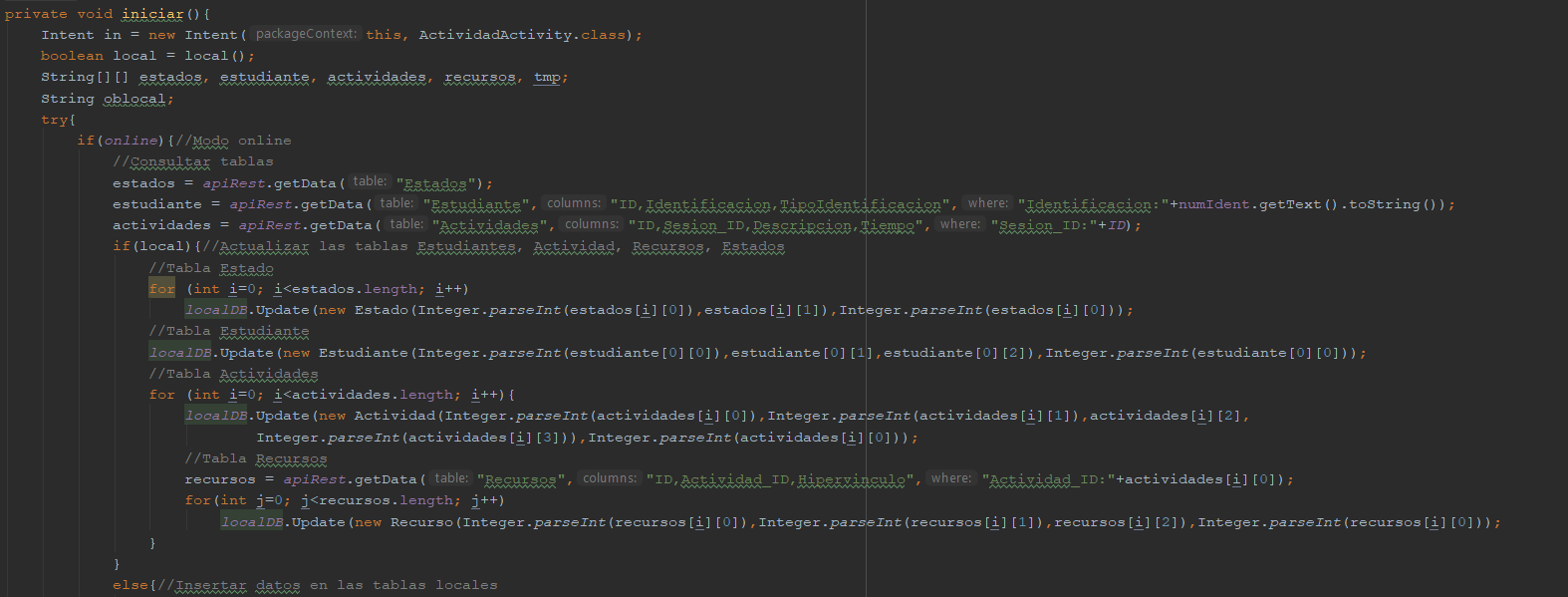
***Iniciar el estado de la actividad***

Posterior a la ejecución del hilo, se ejecuta el método *iniciar* que llama al método *local* para saber que hay información de la sesión en la base de datos local:

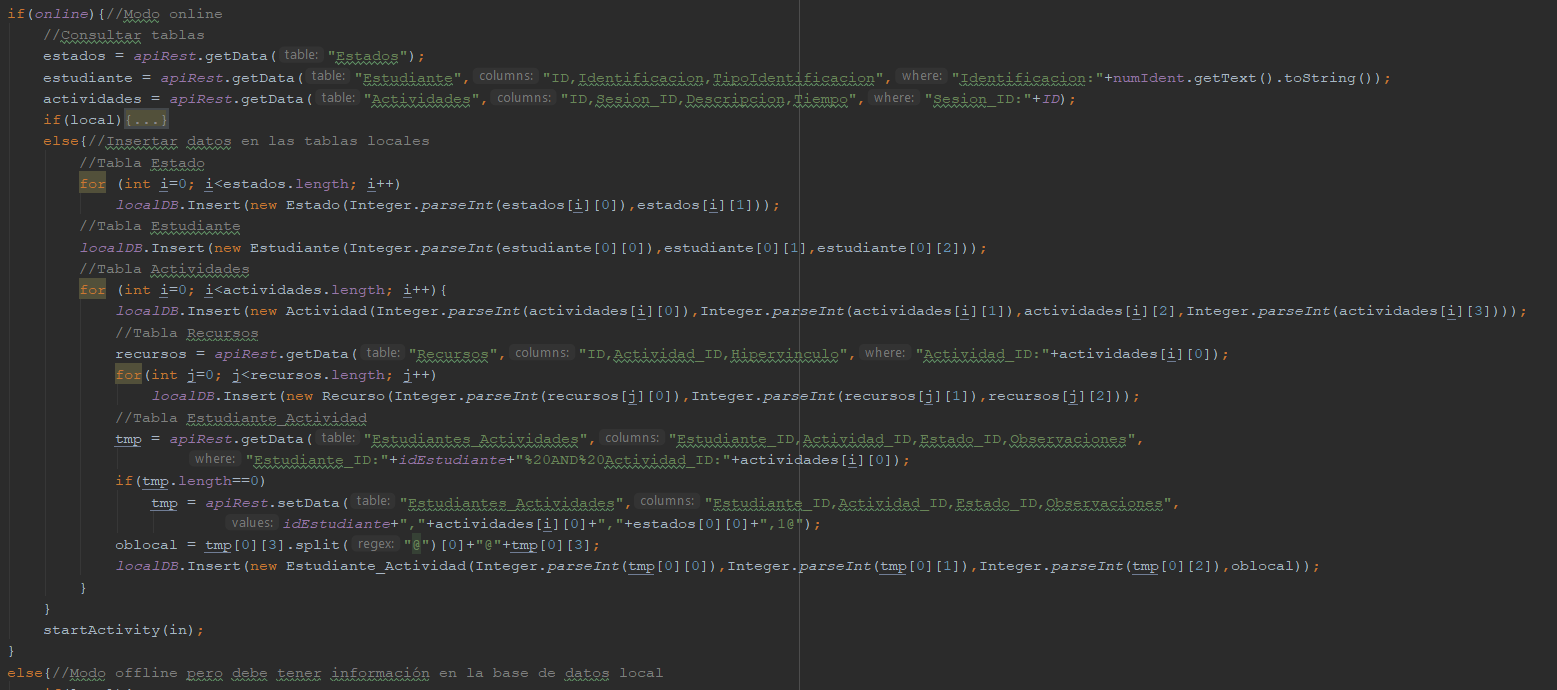


Luego de esto puede pasar lo siguiente:

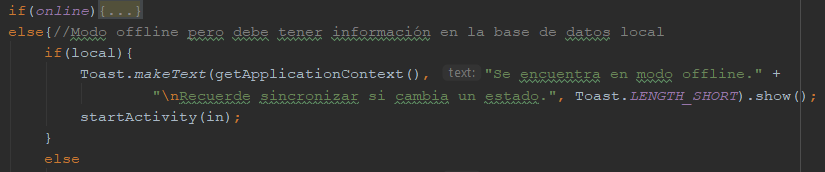
* Si se está en modo *online* y hay información en la base de datos local se actualizan las tablas locales excepto la tabla que contiene los estados de las actividades ya que posteriormente (en la *ActividadActivity* se deben sincronizar), y luego se inicia la actividad siguiente:



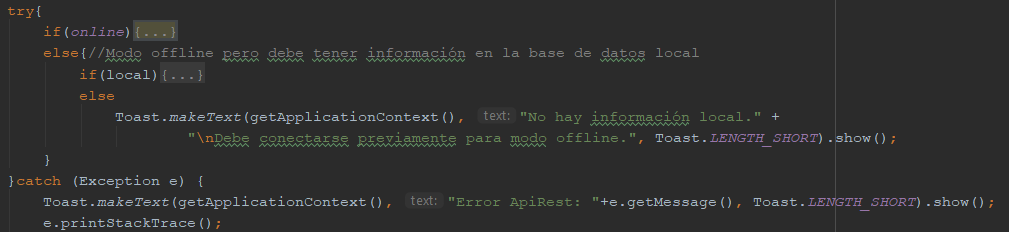
* Si se está en modo *online* pero sin datos en la base de datos local, se insertan en todas las tablas de la base de datos SQLite local y se consulta si es la primera vez que se va a ingresar a la sesión consultando los datos de los estados, **si no existe información se insertan los datos y se ponen las actividades en estado *Iniciada***, y luego se inicia la *ActividadActivity*:



* Si se está en modo *offline* y con datos locales se inicia la *ActividadActivity* que se encarga de cargar la información para visualizar las actividades:



* Si se está en modo *offline* pero sin datos en la base de datos local, no se puede iniciar debido a que debe haber un ingreso previó *online* para poder tener información en la base de datos local:



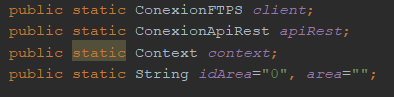
## 4.2) ActividadActivity

# 5) EU Profesores Prototipo

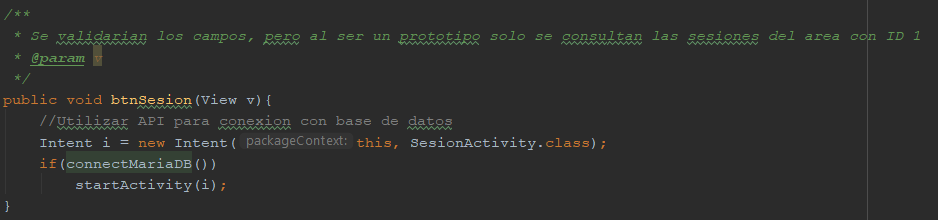
Esta aplicación es un prototipo de la app Profesores en la cual se centrará en crear sesiones sobre una misma área y siempre se crearán tres actividades, cada una con tres recursos (una imagen, un documento, un apk) en código estático (“quemado”).

## 5.1) MainActivity

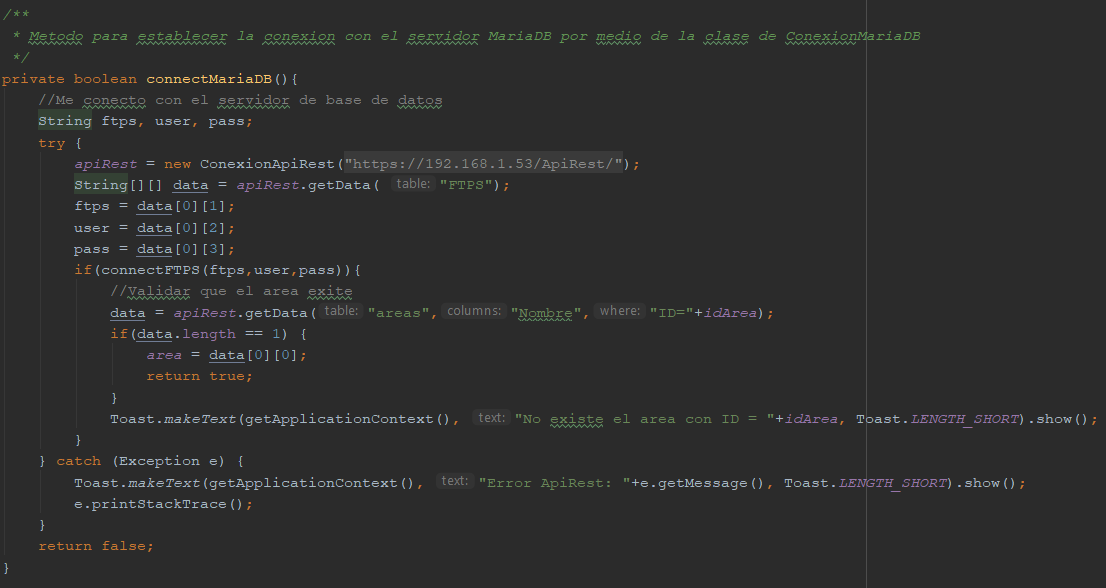
Es la ventana principal donde se debería validar el LogIn del profesor, pero al ser un prototipo simplemente se entrará con el filtro de que el ID del Area es igual a 1, por lo que en esta clase se definen los atributos que se requerirán en las demás clases:



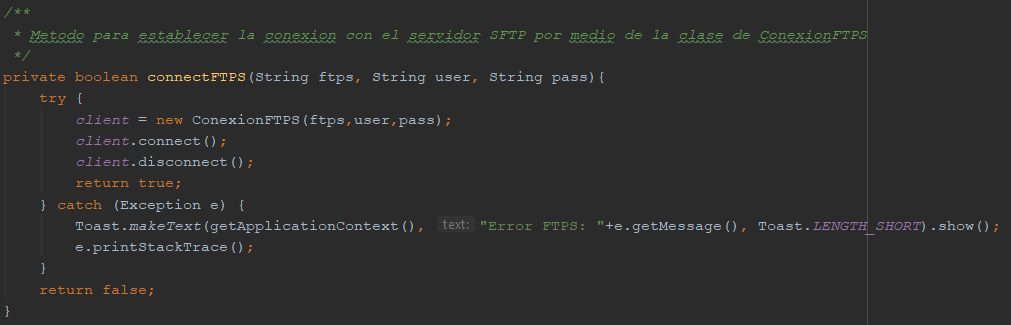
En el onCreate se inicializar el IdArea en 1 y el context. Luego está el botón para iniciar sesión:



Probará la conexión con MariaDB por medio del API REST:

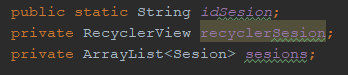


Y con los datos del servidor FTP obtenidos probará la conexión con el servidor FTP:

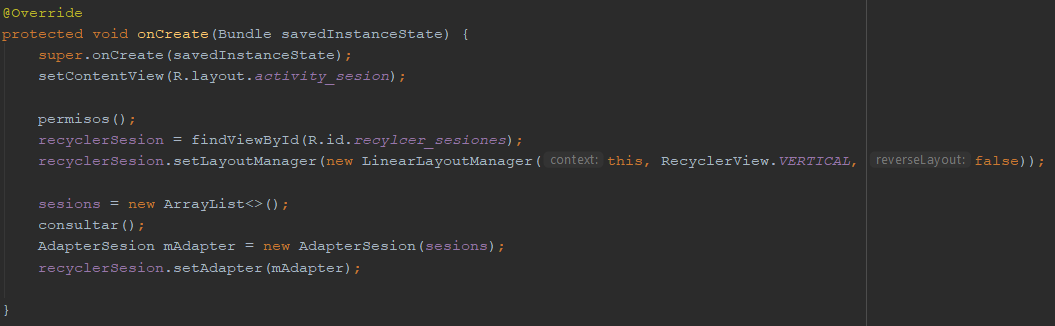


## 5.2) SesionActivity

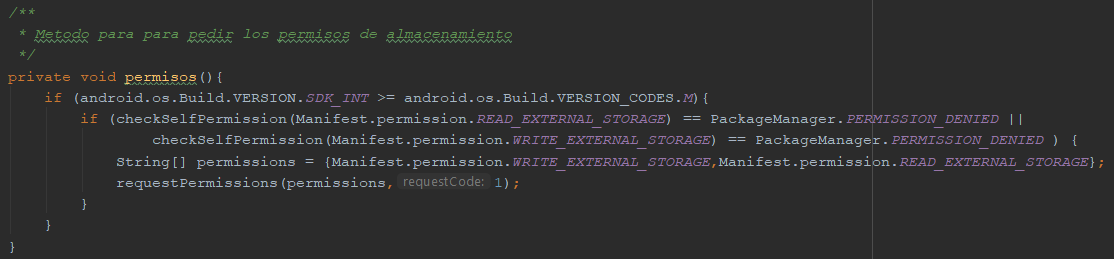
Esta clase estará encargada del manejo lógico para visualizar las sesiones que están creadas en el área cuyo ID es igual a 1, además de dar la opción de crear una sesión y de ver las actividades de una sesión especifica. Para ello primero se crean los atributos necesarios como es el RecyclerView para mostrar la lista de sesiones:



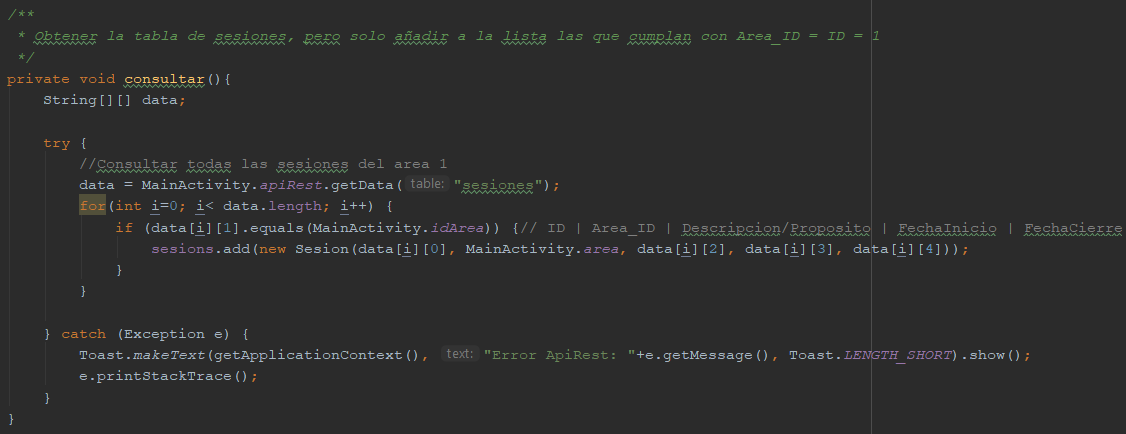
Luego en el onCreate se inicializan las variables y se piden los permisos de almacenamiento, además de consultar todas las sesiones que pertenecen al área con ID 1:



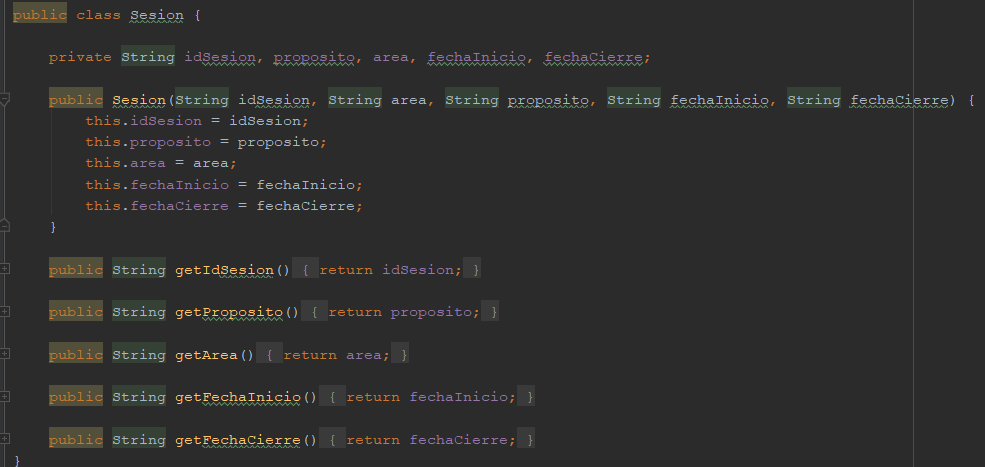
El código de los permisos es exactamente el mismo:



Y para consultar, se debe obtener toda la tabla de sesiones, pero solo añadir a la lista las que tengan la FK (la llave foránea) Area\_ID igual al ID del MainActivity, es decir, que sean igual a 1:



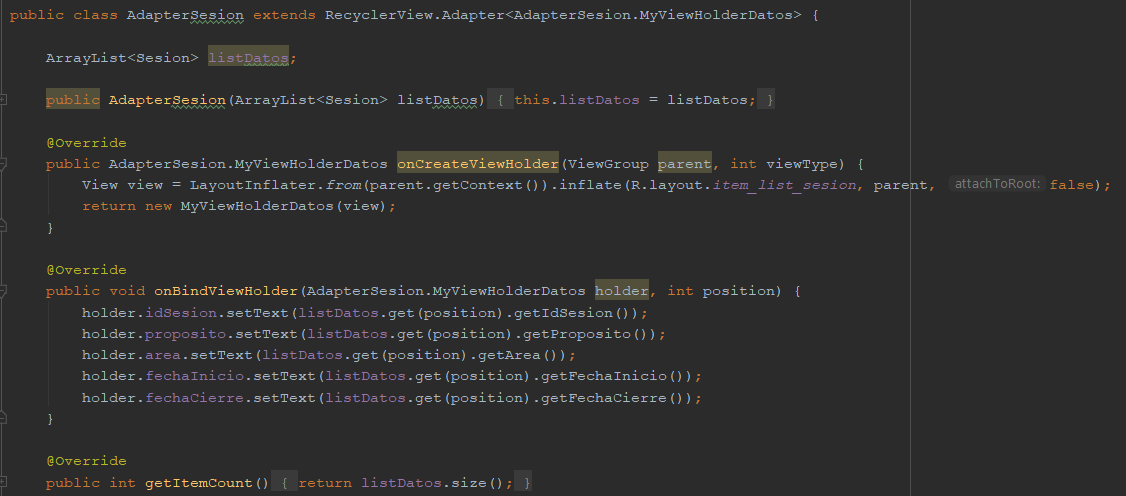
Cabe resaltar que, para esto, la lista donde se agregan los resultados es de tipo Sesion que tiene un atributo para cada columna:



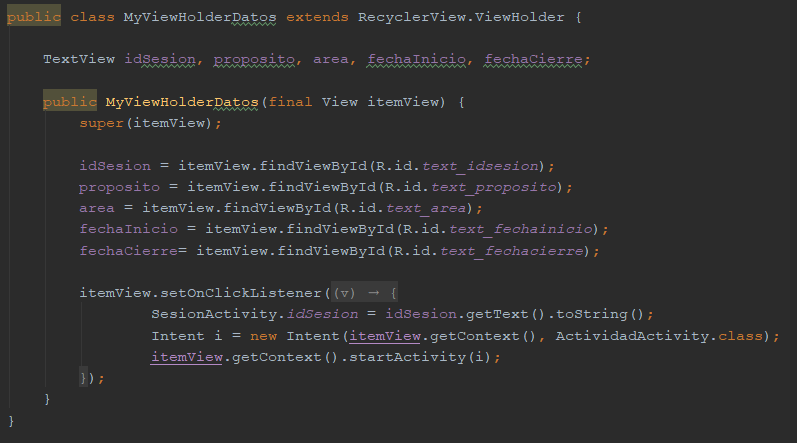
Que son los mismo que se mostrarán en el CardView ya que no se implementará ningún tipo de botón en la tarjeta:



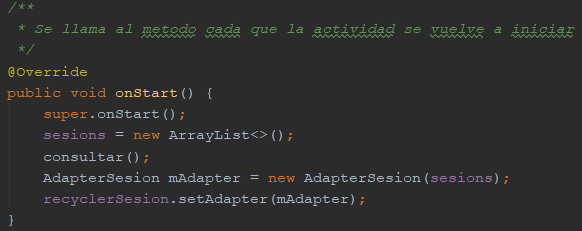
Que para poder mostrar tantas veces las tarjetas como sesiones existentes se crea la clase *AdapterSesion*que hará esto posible:



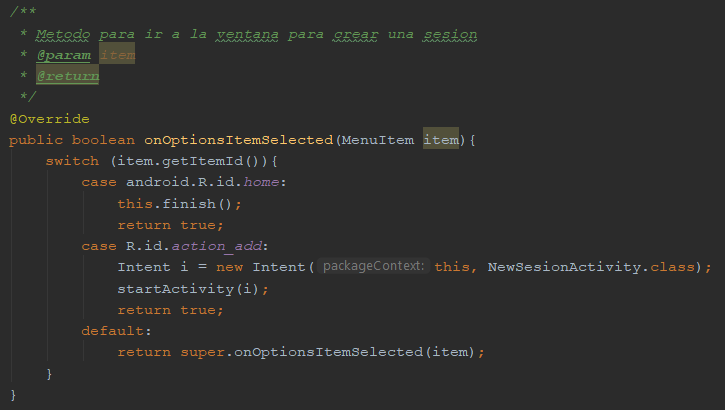
Y es en la clase *MyViewHolderDatos* donde se debe programar el código para cuando se dé clic sobre una tarjeta ir a la ventana con todas las actividades que tiene la sesión seleccionada:



Además, está el método onStart() que se llama cuando la actividad vuelve a iniciar por lo que vuelve a consultar las sesiones por si hubo algún cambio.

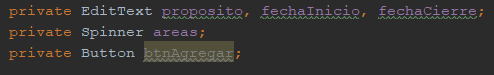


Por último, cuando se dé clic en el boton “+” de la venta de sesión se abrirá una ventana para poner los datos necesarios para crear una sesión y sus actividades con sus recursos por debajo:

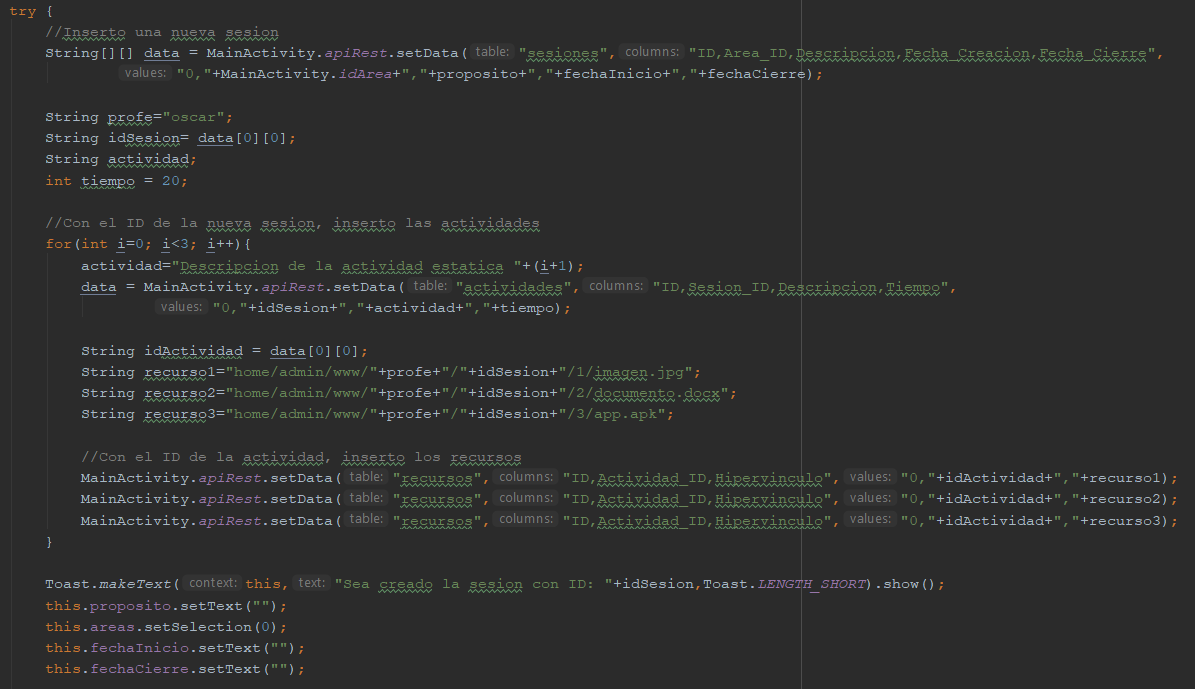


## 5.3) NewSesionActivity

En esta clase se corresponde a crear una nueva sesión en el Area\_ID 1 y que esta contenga 3 actividades que su vez, cada una contiene 3 recursos, entonces primero se identifican los campos de la sesión:

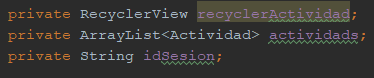


Y cuando se hace clic en el botón agregar se validan los campos y crea la sesión, se obtiene el ID con la cual quedo agregado para crear las tres actividades, que se guarda el ID de la actividad para crearle los tres recursos:

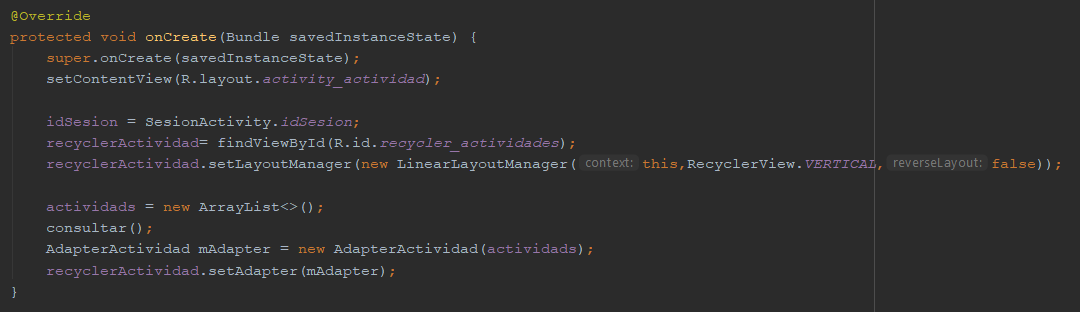


## 5.4) ActividadActivity

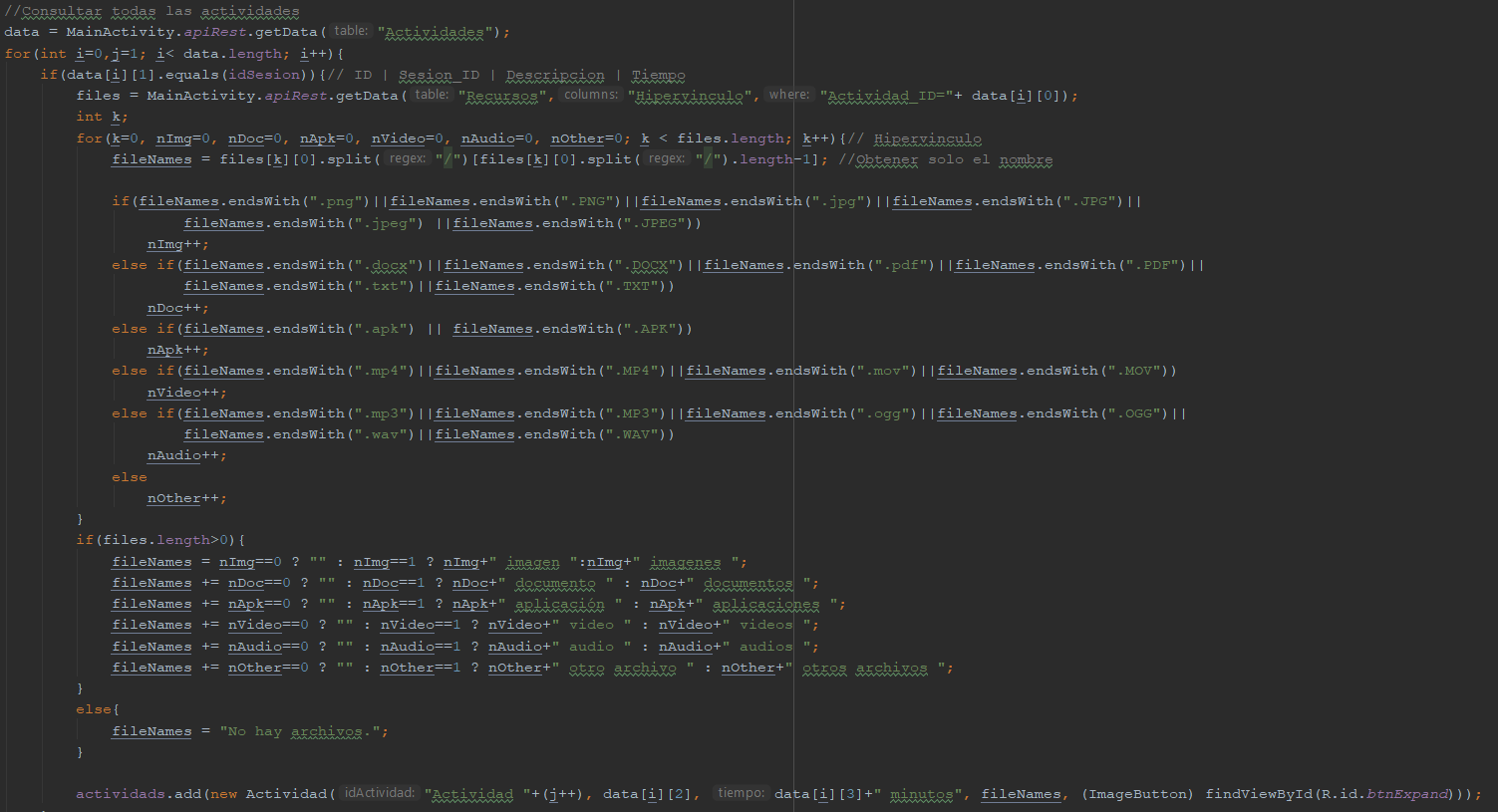
Se encarga de mostrar la lista de actividades que tiene una sesión en específico, solo se muestra la información sin botones, entonces en primer lugar se crean las variables necesarias como las listas de actividades y el RecyclerView y el idSesion que se utilizará:



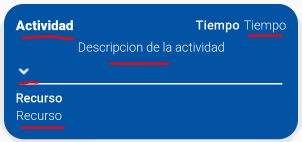
Luego en el onCreate() se hace las asignaciones e inicializaciones:



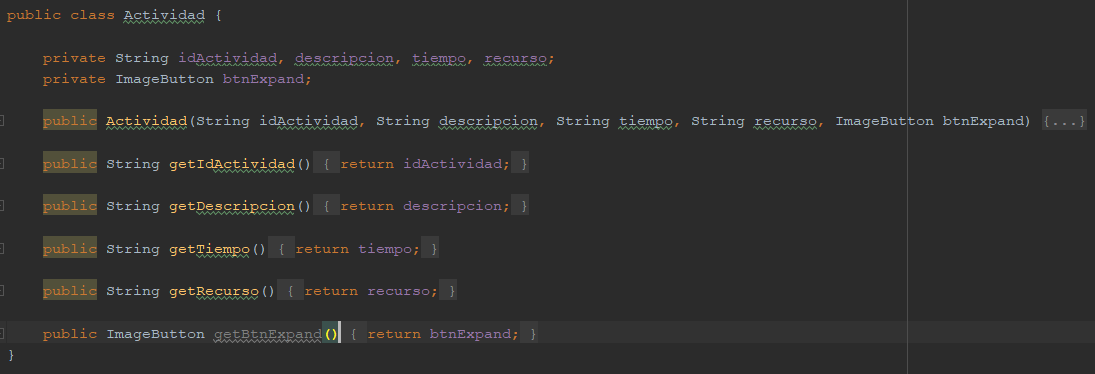
Como ya se ha hecho anteriormente, el método consultar se encargará de llenar la lista con los registros de la base de datos que cumplan con la condición de que el ID de sesión sea igual al idSesion:



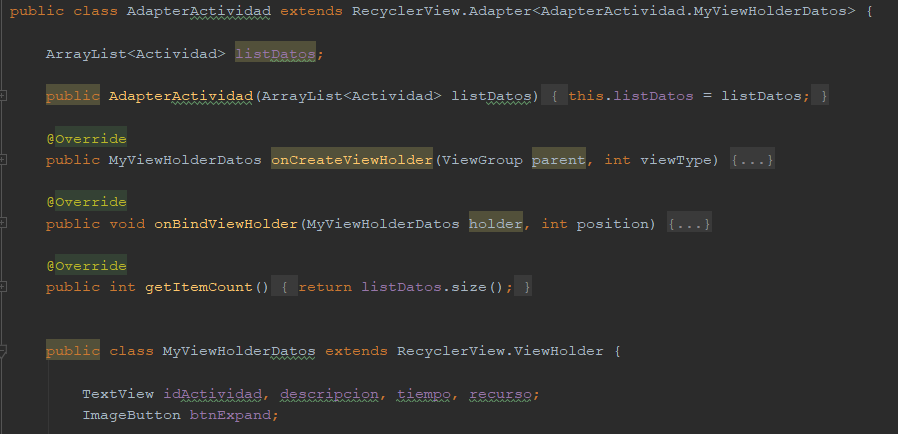
Solo se tiene un objeto demás en la lista que es el icono para expandir o colapsar la tarjeta para ver u ocultar, respectivamente, los recursos de la actividad, por lo que cumple que están todos los objetos de la CardView:



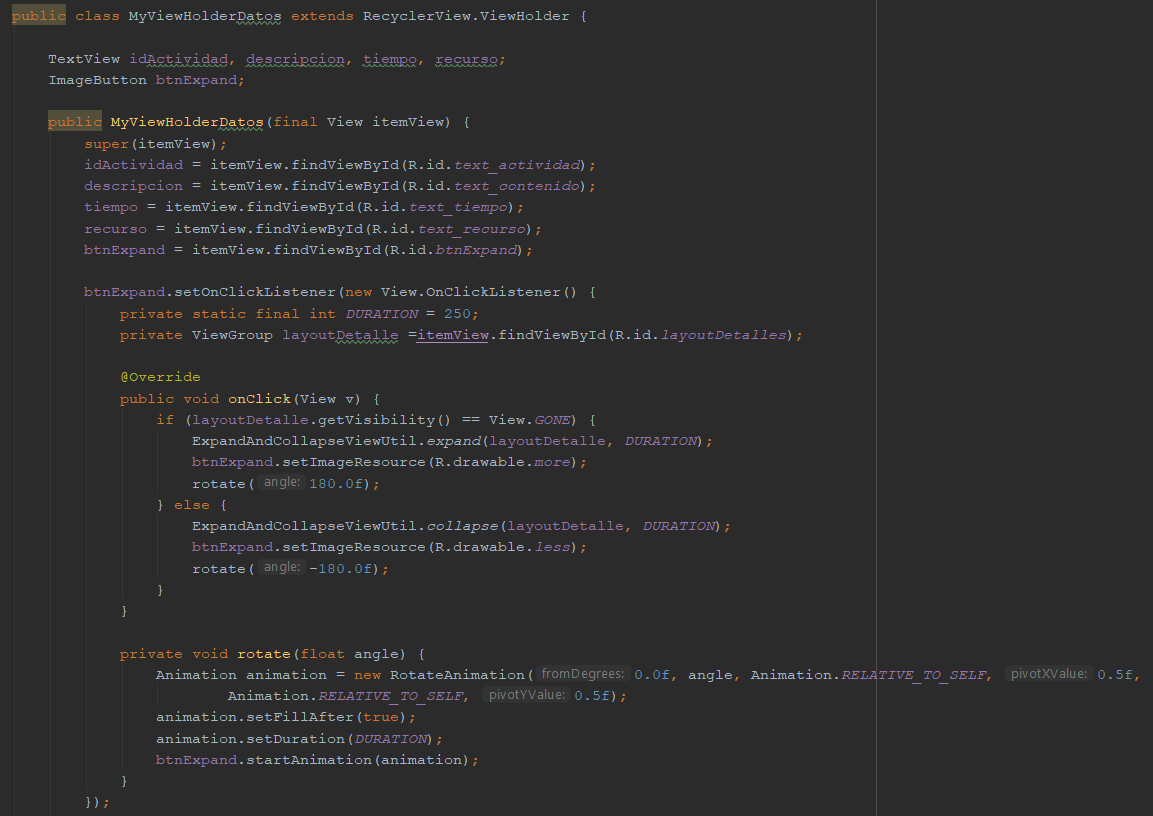
Y la clase que nos ayuda para crear una lista con todos estos objetos:



Por último, el *AdapterActividad* que nos replica la tarjeta en el RecyclerView:



Recordar que en la clase interna *MyViewHolderDatos* es donde se programa el código cuando se hace clic en la tarjeta o en algún objeto de esta:



# Referencias

Rodriguez. (2020). Automatic API REST Droidcon. Slideshare.net. Retrieved 5 May 2020, from <https://www.slideshare.net/AlejandroEsquiva/automatic-apirestdroidcon>

Diseño Android: Tarjetas con CardView. danielme.com. (2020). Retrieved 5 May 2020, from <https://danielme.com/2015/08/12/diseno-android-tarjetas-con-cardview/>

Optimizando la interfaz Android - Compound Drawables. Optimizando la interfaz Android - Compound Drawables. (2020). Retrieved 5 May 2020, from <https://elbauldelprogramador.com/optimizando-la-interfaz-android-compound-drawables/>

Diseño Android: Spinner. danielme.com. (2020). Retrieved 5 May 2020, from <https://danielme.com/2013/04/25/diseno-android-spinner/>

Cómo interpretar el ciclo de vida de una actividad. developer. (2020). Retrieved 6 May 2020, from <https://developer.android.com/guide/components/activities/activity-lifecycle?hl=es>