	UNIVERSIDAD DON BOSCO FACULTAD DE INGENIERÍA ESCUELA DE COMPUTACIÓN		CICLO 01-2022
	MATERIA	DESARROLLO DE SOFTWARE PARA MÓVILES	GUIA DE LABORATORIO N° 6
	PRÁCTICA	CONSUMO DE API REST CON RETROFIT	

I. OBJETIVOS

1. Conocer las generalidades de la librería Retrofit.
2. Conocer el concepto de API REST.
3. Consumir un API REST usando Retrofit en una aplicación Android.
4. Construir una API REST y consumirla con la librería Retrofit utilizando los métodos HTTP adecuados (GET, POST, PUT y DELETE).

II. INTRODUCCIÓN TEÓRICA – PRIMERA PARTE

En la actualidad consumir APIs es una tarea casi obligatoria a la hora de desarrollar una aplicación Android, sobre todo si se busca que dicha app sea útil y de mucho valor al usuario, y es que una aplicación sin conectividad con el mundo exterior es muy pobre en funcionalidades y alcance.

Afortunadamente el ecosistema de desarrollo Android es bastante rico en librerías y utilidades que hacen muy simple y sencillo implementar muchas de las características comunes en una app actual, una de esas es Retrofit.

Retrofit es una librería utilizada en Android para consumir servicios REST, facilita la gestión de errores, el manejo del request y response así como el procesamiento de JSON y su conversión en objetos Java. Pero, antes de entrar de lleno a la utilización de Retrofit como tal es necesario comprender el concepto de API REST.

¿Qué es un API REST?


En términos simples API REST es un servicio que provee de las funciones necesarias para obtener información de un servicio externo, como por ejemplo, una base de datos alojada en cualquier parte del mundo desde dentro de nuestra propia aplicación.

Instagram por ejemplo, una aplicación con millones de usuarios, es inviable tener la información de cada usuario dentro de la aplicación así que para solventar el problema se utilizan servicios API REST.

Siguiendo con el ejemplo, al entrar en la app el usuario se autentica (ingresando sus credenciales), esto sería el primero de los servicios, ya que se envían al servidor el usuario y contraseña y este devolverá si tenemos o no permitido el acceso a la aplicación.

Disponemos de cuatro tipos distintos de peticiones como norma general.

- **GET:** Son las peticiones más sencillas, solo nos devuelven información. Si necesitamos pasarle un parámetro a la petición será a través de la url. Es decir si por ejemplo tenemos que hacer una petición que depende de una id (ej la identificación del usuario) la url se formaría así <https://ejemplo.com/informacion/1>, siendo 1 el parámetro que le pasamos.
- **POST:** Similar a Get pero los parámetros no se pasan por url sino por el request body, generalmente se usa para crear registros.
- **PUT:** Se suele usar para actualizar información, es decir, si pensamos en un servicio como el acceso a una base de datos, este invocará el UPDATE del usuario.
- **DELETE:** Sería el último de los cuatro que nos permitiría borrar los registros de la base de datos.

	UNIVERSIDAD DON BOSCO FACULTAD DE INGENIERÍA ESCUELA DE COMPUTACIÓN		CICLO 01-2022
	MATERIA	DESARROLLO DE SOFTWARE PARA MÓVILES	GUIA DE LABORATORIO N° 6
	PRÁCTICA	CONSUMO DE API REST CON RETROFIT	


La información suele venir en dos formatos distintos, XML o JSON.

Formato JSON

Json es un formato de texto simple, es el acrónimo de JavaScript Object Notation. Se trata de uno del estándar para el traspaso de información entre plataformas, tiene una forma muy legible que permite entender su contenido sin problema. Un ejemplo sencillo sería este.

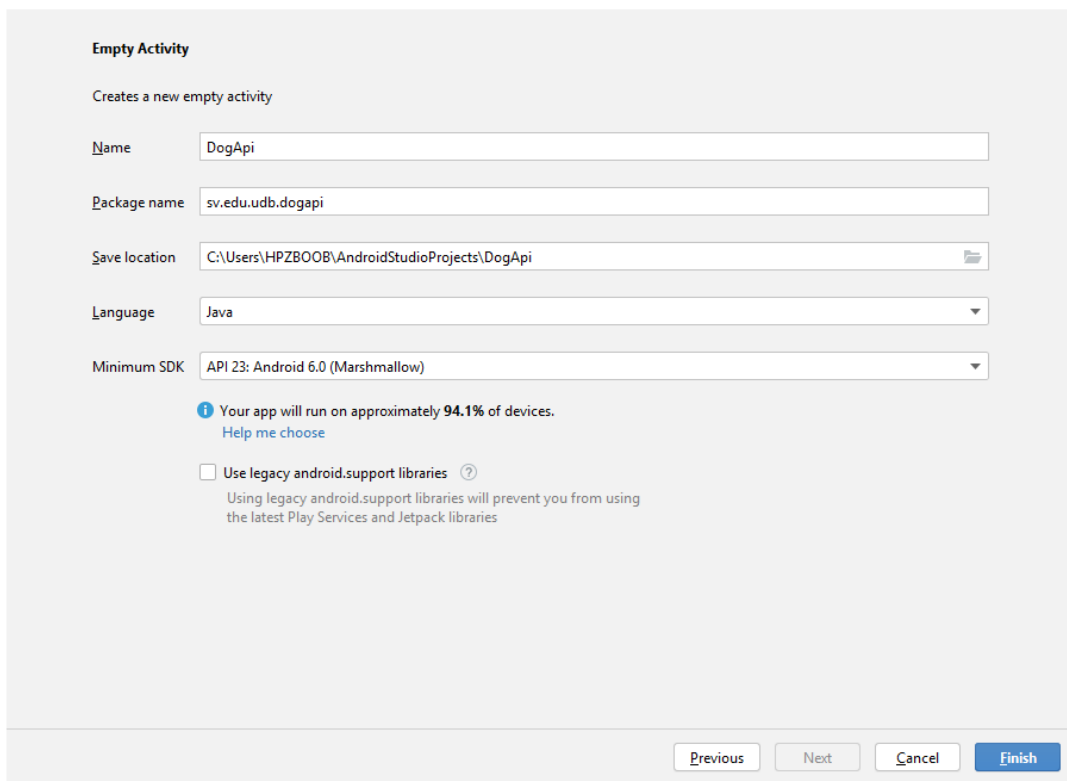
```
{
  "data": {
    "employees": [
      {
        "id": "1",
        "firstName": "Tom",
        "lastName": "Cruise",
        "photo": "https://jsonformatter.org/img/tom-cruise.jpg"
      },
      {
        "id": "2",
        "firstName": "Maria",
        "lastName": "Sharapova",
        "photo": "https://jsonformatter.org/img/Maria-Sharapova.jpg"
      },
      {
        "id": "3",
        "firstName": "Robert",
        "lastName": "Downey Jr.",
        "photo": "https://jsonformatter.org/img/Robert-Downey-Jr.jpg"
      }
    ]
  }
}
```

Todo formato Json empieza y termina con llaves y tiene una clave-valor. La clave **data** contiene a su vez una lista de **employees** (se utiliza los corchetes para definir un arreglo de valores), que este almacena id, firstName, lastName y photo. Así podemos pasarnos gran cantidad de información de una plataforma a otra con un estándar que nos ayudan a simplificar el proceso.

	UNIVERSIDAD DON BOSCO FACULTAD DE INGENIERÍA ESCUELA DE COMPUTACIÓN		CICLO 01-2022
	MATERIA	DESARROLLO DE SOFTWARE PARA MÓVILES	GUIA DE LABORATORIO N° 6
	PRÁCTICA	CONSUMO DE API REST CON RETROFIT	

III. DESARROLLO DE PRÁCTICA – PRIMERA PARTE

1. Ejecuta Android Studio y crea un nuevo proyecto. En **Project Template** selecciona **Empty Activity** y presiona **Next**.
2. Como **Name** coloca **DogApi**, en **Package name** escribe **sv.edu.udb.dogapi** en **Save location** escoge la carpeta de tu preferencia, en Language elige **Java** y el **Minimum SDK** selecciona **API 23: Android 6.0 (Marshmallow)**
3. Presiona **Finish**



Empty Activity

Creates a new empty activity

Name: DogApi

Package name: sv.edu.udb.dogapi

Save location: C:\Users\HPZBOOB\AndroidStudioProjects\DogApi

Language: Java

Minimum SDK: API 23: Android 6.0 (Marshmallow)

Your app will run on approximately 94.1% of devices.
[Help me choose](#)

☐ Use legacy android.support libraries [?](#)
 Using legacy android.support libraries will prevent you from using the latest Play Services and Jetpack libraries


Previous Next Cancel Finish

4. Edita el archivo AndroidManifest.xml y agrega el permiso de conectividad a Internet.
`<uses-permission android:name="android.permission.INTERNET"/>`
5. Edita el archivo build.gradle y agrega las librerías que utilizaras para construir la aplicación.
 - a. **Picasso**: Esta librería nos permitirá transformar esas urls en imágenes.
 - b. **Retrofit 2**: Librería encargada del consumo de las API.
 - c. **Retrofit 2 Converter Gson**: Esta herramienta será un complemento a la anterior y nos simplificará el proceso de pasar un JSON a una clase Java.

implementation "com.squareup.picasso:picasso:2.71828"

implementation "com.squareup.retrofit2:retrofit:2.9.0"

implementation "com.squareup.retrofit2:converter-gson:2.9.0"

	UNIVERSIDAD DON BOSCO FACULTAD DE INGENIERÍA ESCUELA DE COMPUTACIÓN		CICLO 01-2022
	MATERIA	DESARROLLO DE SOFTWARE PARA MÓVILES	GUIA DE LABORATORIO N° 6
	PRÁCTICA	CONSUMO DE API REST CON RETROFIT	

6. Edita nuevamente el archivo build.gradle para utilizar el View Binding en tu aplicación, para eso en la sección **android** agrega la siguiente líneas de código:

```

android {
    compileSdkVersion 30
    buildToolsVersion "30.0.3"

    buildFeatures{
        viewBinding = true
    }

    defaultConfig {
        ....

```

7. Modifica tu MainActivity

```

package sv.edu.udb.dogapi;

import androidx.appcompat.app.AppCompatActivity;
import android.os.Bundle;
import sv.edu.udb.dogapi.databinding.ActivityMainBinding;

public class MainActivity extends AppCompatActivity {

    ActivityMainBinding binding;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        binding = ActivityMainBinding.inflate(getLayoutInflater());
        setContentView(binding.getRoot());
    }
}


```

8. Modifica el layout activity_main.xml

```

<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/viewRoot"
    android:layout_width="match_parent"

```

	UNIVERSIDAD DON BOSCO FACULTAD DE INGENIERÍA ESCUELA DE COMPUTACIÓN		CICLO 01-2022
	MATERIA	DESARROLLO DE SOFTWARE PARA MÓVILES	GUIA DE LABORATORIO N° 6
	PRÁCTICA	CONSUMO DE API REST CON RETROFIT	

```

android:layout_height="match_parent"
tools:context=".MainActivity">

<androidx.appcompat.widget.SearchView
    android:id="@+id/searchDogs"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    app:layout_constraintVertical_bias="0"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toTopOf="parent" />

<androidx.recyclerview.widget.RecyclerView
    android:id="@+id/listDogs"
    android:layout_width="0dp"
    android:layout_height="0dp"
    app:layout_constraintTop_toBottomOf="@+id/searchDogs"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintBottom_toBottomOf="parent"/>

</androidx.constraintlayout.widget.ConstraintLayout>

```

9. Crea la clase DogsResponse, que servirá para procesar como objeto la respuesta JSON que proviene del API.

```

package sv.edu.udb.dogapi;

import com.google.gson.annotations.SerializedName;
import java.util.List;


public class DogsResponse {

    @SerializedName("status")
    private String status;

    @SerializedName("message")
    private List<String> images;

    public String getStatus() {
        return status;
    }
}

```

	UNIVERSIDAD DON BOSCO FACULTAD DE INGENIERÍA ESCUELA DE COMPUTACIÓN		CICLO 01-2022
	MATERIA	DESARROLLO DE SOFTWARE PARA MÓVILES	GUIA DE LABORATORIO N° 6
	PRÁCTICA	CONSUMO DE API REST CON RETROFIT	

```

public void setStatus(String status) {
    this.status = status;
}

public List<String> getImages() {
    return images;
}

public void setImages(List<String> images) {
    this.images = images;
}
}

```

10. Crea el contrato que defina la llamada de Retrofit, es decir, vamos a crear una interfaz que lo que hará será definir el tipo de consumo de API y lo que nos va a devolver. Crea la interfaz **ApiService**.

```

package sv.edu.udb.dogapi;

import retrofit2.Call;
import retrofit2.http.GET;
import retrofit2.http.Path;

public interface ApiService {

    @GET("{raza}/images")
    Call<DogsResponse> getDogsByBreed(@Path("raza") String raza);

}

```

11. Crea la instancia de Retrofit, agregando el siguiente método en tu MainActivity.

```


private ApiService getApiService() {

    Retrofit retrofit = new Retrofit.Builder()
        .baseUrl("https://dog.ceo/api/breed/")
        .addConverterFactory(GsonConverterFactory.create())
        .build();

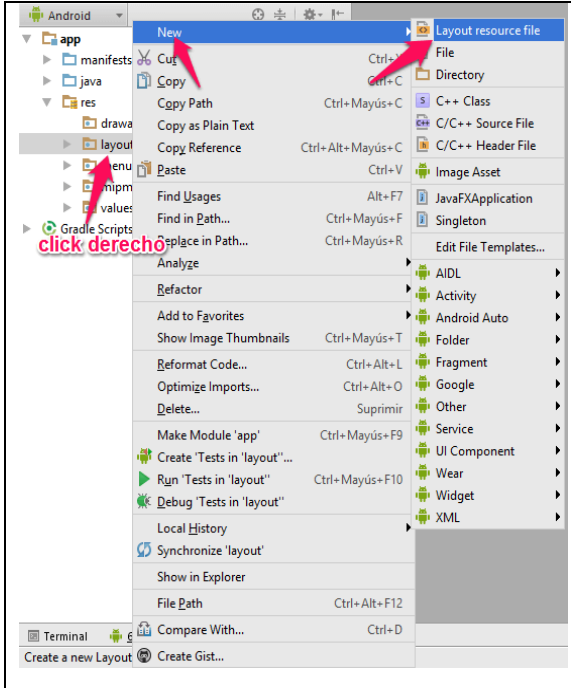
    ApiService service = retrofit.create(ApiService.class);

    return service;
}

```

	UNIVERSIDAD DON BOSCO FACULTAD DE INGENIERÍA ESCUELA DE COMPUTACIÓN		CICLO 01-2022
	MATERIA	DESARROLLO DE SOFTWARE PARA MÓVILES	GUIA DE LABORATORIO N° 6
	PRÁCTICA	CONSUMO DE API REST CON RETROFIT	

12. Crea el layout **item_dog.xml** que se usará para el RecyclerView



```

<?xml version="1.0" encoding="utf-8"?>
<androidx.cardview.widget.CardView
xmlns:android="http://schemas.android.com/apk/res/android"
android:layout_width="match_parent"
xmlns:app="http://schemas.android.com/apk/res-auto"
app:cardCornerRadius="16dp"
android:background="@color/black"
android:layout_margin="16dp"
android:layout_height="320dp">

<ImageView
    android:id="@+id/ivDog"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:scaleType="centerCrop"/>

</androidx.cardview.widget.CardView>

```

13. Crea una nueva clase llamada **DogViewHolder**, que también se utilizará para el RecyclerView.

```

package sv.edu.udb.dogapi;

import android.view.View;
import androidx.recyclerview.widget.RecyclerView;
import com.squareup.picasso.Picasso;
import sv.edu.udb.dogapi.databinding.ItemDogBinding;

public class DogViewHolder extends RecyclerView.ViewHolder {


    private ItemDogBinding itemDogBinding;

    public DogViewHolder(View view) {
        super(view);
        itemDogBinding = ItemDogBinding.bind(view);
    }

    public void bind(String imageUrl) {
        Picasso.get().load(imageUrl).into(itemDogBinding.ivDog);
    }

}

```

	UNIVERSIDAD DON BOSCO FACULTAD DE INGENIERÍA ESCUELA DE COMPUTACIÓN		CICLO 01-2022
	MATERIA	DESARROLLO DE SOFTWARE PARA MÓVILES	GUIA DE LABORATORIO N° 6
	PRÁCTICA	CONSUMO DE API REST CON RETROFIT	

14. Crea una nueva clase llamada **DogAdapter**, que también se utilizará para el RecyclerView.

```
package sv.edu.udb.dogapi;

import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import androidx.annotation.NonNull;
import androidx.recyclerview.widget.RecyclerView;
import java.util.List;

public class DogAdapter extends RecyclerView.Adapter<DogViewHolder> {


    private List<String> images;

    public DogAdapter(List<String> images){
        this.images = images;
    }

    @NonNull
    @Override
    public DogViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int viewType) {
        View view = LayoutInflater.from(parent.getContext())
            .inflate(R.layout.item_dog, parent, false);
        return new DogViewHolder(view);
    }

    @Override
    public void onBindViewHolder(@NonNull DogViewHolder holder, int position) {
        holder.bind(images.get(position));
    }

    @Override
    public int getItemCount() {
        return images != null ? images.size() : 0;
    }
}
```


	UNIVERSIDAD DON BOSCO FACULTAD DE INGENIERÍA ESCUELA DE COMPUTACIÓN		CICLO 01-2022
	MATERIA	DESARROLLO DE SOFTWARE PARA MÓVILES	GUIA DE LABORATORIO N° 6
	PRÁCTICA	CONSUMO DE API REST CON RETROFIT	

15. Ahora que ya están todos los componentes listos, solo queda ensamblar todo en el MainActivity, para eso modifica tu actividad para que quede de la siguiente manera.

```

package sv.edu.udb.dogapi;

import androidx.annotation.Nullable;
import androidx.appcompat.app.AppCompatActivity;
import androidx.appcompat.widget.SearchView;
import androidx.recyclerview.widget.LinearLayoutManager;
import androidx.os.Bundle;
import android.widget.Toast;
import java.util.ArrayList;
import java.util.List;
import retrofit2.Call;
import retrofit2.Callback;
import retrofit2.Response;
import retrofit2.Retrofit;
import retrofit2.converter.gson.GsonConverterFactory;
import sv.edu.udb.dogapi.databinding.ActivityMainBinding;

public class MainActivity extends AppCompatActivity implements SearchView.OnQueryTextListener {

    ActivityMainBinding binding;
    DogAdapter dogAdapter;
    List<String> images = new ArrayList<>();

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        binding = ActivityMainBinding.inflate(getLayoutInflater());
        setContentView(binding.getRoot());
        initRecyclerView();
        binding.searchDogs.setOnQueryTextListener((SearchView.OnQueryTextListener) this);
    }

    private void initRecyclerView() {
        dogAdapter = new DogAdapter(images);
        binding.listDogs.setLayoutManager(new LinearLayoutManager(this));
        binding.listDogs.setAdapter(dogAdapter);
    }


    private ApiService getApiService() {

        Retrofit retrofit = new Retrofit.Builder()
            .baseUrl("https://dog.ceo/api/breed/")
            .addConverterFactory(GsonConverterFactory.create())
            .build();

        ApiService service = retrofit.create(ApiService.class);

        return service;
    }
}

```

	UNIVERSIDAD DON BOSCO FACULTAD DE INGENIERÍA ESCUELA DE COMPUTACIÓN		CICLO 01-2022
	MATERIA	DESARROLLO DE SOFTWARE PARA MÓVILES	GUIA DE LABORATORIO N° 6
	PRÁCTICA	CONSUMO DE API REST CON RETROFIT	

```

private void searchByName(String raza) {

    final Call<DogsResponse> batch = getApiService().getDogsByBreed(raza);

    batch.enqueue(new Callback<DogsResponse>() {
        @Override
        public void onResponse(@Nullable Call<DogsResponse> call, @Nullable Response<DogsResponse>
response) {
            if (response != null && response.body() != null)
            {

                List<String> responselImages = response.body().getImages();
                images.clear();
                images.addAll(responselImages);
                dogAdapter.notifyDataSetChanged();
            }
        }

        @Override
        public void onFailure(@Nullable Call<DogsResponse> call, @Nullable Throwable t) {
            if(t!=null)
            {
                showError();
            }
        }
    });
}


private void showError() {
    Toast.makeText(this, "Ha ocurrido un error", Toast.LENGTH_SHORT).show();
}

@Override
public boolean onQueryTextSubmit(String query) {
    if(!query.isEmpty()){
        searchByName(query.toLowerCase());
    }
    return true;
}

@Override
public boolean onQueryTextChange(String newText) {
    return true;
}
}

```

16. Ejecuta la aplicación.

	UNIVERSIDAD DON BOSCO FACULTAD DE INGENIERÍA ESCUELA DE COMPUTACIÓN		CICLO 01-2022
	MATERIA	DESARROLLO DE SOFTWARE PARA MÓVILES	GUIA DE LABORATORIO N° 6
	PRÁCTICA	CONSUMO DE API REST CON RETROFIT	

Es importante aclarar que por tratarse de un ejemplo didáctico se ha concentrado mucho código en la actividad principal, *esto de ninguna manera es una buena práctica* y se le invita al estudiante que haga uso de arquitecturas de diseño como **MVP** o **MVVC** para construir su aplicación

IV. INTRODUCCIÓN TEÓRICA – SEGUNDA PARTE

En el ejemplo anterior estudiamos la forma de consumir una API usando Retrofit para consultar información utilizando el método GET como petición HTTP. Sin embargo para este ejemplo desarrollaremos una API REST local que nos permite tener el funcionamiento de los métodos GET, POST, PUT y DELETE, los cuales serán consumidos desde una aplicación móvil.

Los códigos de respuesta HTTP estandarizan una forma de informar al cliente sobre el resultado de su solicitud. Estos son algunos códigos de respuesta HTTP, que a menudo se utilizan en las API REST:

- **200 OK** : Este código de respuesta indica que la solicitud se ha realizado correctamente.
- **201 Created** : Esto indica que la solicitud tuvo éxito y se creó un recurso. Se utiliza para confirmar el éxito de una solicitud PUT o POST.
- **400 Bad Request** : La solicitud fue malformada. Esto sucede especialmente con las solicitudes POST y PUT, cuando los datos no pasan la validación o están en el formato incorrecto.
- **404 Not Found** : Esta respuesta indica que no se pudo encontrar el recurso necesario. Esto generalmente se devuelve a todas las solicitudes que apuntan a una URL sin recurso correspondiente.
- **401 Unauthorized** : Este error indica que debe realizar la autenticación antes de acceder al recurso.
- **405 Method Not Allowed** : El método HTTP utilizado no es compatible con el de este recurso.
- **409 Conflict** : Esto indica un conflicto. Por ejemplo, está utilizando una solicitud PUT para crear el mismo recurso dos veces.
- **500 Internal Server Error** : Cuando todo lo demás falla, en general, se utiliza una respuesta 500 cuando el procesamiento falla debido a circunstancias imprevistas en el lado del servidor, lo que provoca el error del servidor.

A continuación se explica el código que se utilizara en la aplicación:

Utilizaremos la siguiente dirección IP como ejemplo, esta deberá ser cambiada por la IP de su equipo.


URL base: `"http://192.168.1.2/APIUDB/producto/"`

1- URL para consultar todos los registros de una tabla de productos:

`http://192.168.1.2/APIUDB/producto/`

Resultado del ejemplo:

```
[
  {
    "codigo": "1009",
    "descripcion": "Laptop Dell X",
    "precio": 500
  },
]
```

	UNIVERSIDAD DON BOSCO FACULTAD DE INGENIERÍA ESCUELA DE COMPUTACIÓN		CICLO 01-2022
	MATERIA	DESARROLLO DE SOFTWARE PARA MÓVILES	GUIA DE LABORATORIO N° 6
	PRÁCTICA	CONSUMO DE API REST CON RETROFIT	

```
{
  "codigo": "11",
  "descripcion": "Teclado gamer rojo",
  "precio": 21
},
{
  "codigo": "25",
  "descripcion": "Bocina Sony",
  "precio": 51.35
},
.
.
]
```

Observa que el resultado es un arreglo de objetos que tienen 3 atributos cada uno: código, descripción y precio.

Para poder leer este resultado será necesario crear una clase modelo que tenga definido estos 3 atributos con sus respectivos métodos accesorios (getters).

Estructura de clase modelo:


```
public class Producto {
    private String codigo;
    private String descripcion;
    private float precio;

    public Producto(String codigo, String descripcion, float precio) {
        this.codigo = codigo;
        this.descripcion = descripcion;
        this.precio = precio;
    }

    public String getCodigo() {
        return codigo;
    }

    public String getDescripcion() {
        return descripcion;
    }

    public float getPrecio() {
        return precio;
    }
}
```

	UNIVERSIDAD DON BOSCO FACULTAD DE INGENIERÍA ESCUELA DE COMPUTACIÓN		CICLO 01-2022
	MATERIA	DESARROLLO DE SOFTWARE PARA MÓVILES	GUIA DE LABORATORIO N° 6
	PRÁCTICA	CONSUMO DE API REST CON RETROFIT	

2- URL para consultar un solo registro de producto a través de su código:

<http://192.168.1.2/APIUDB/producto/?codigo=32>

Observa que al final de la url está incluido el código del producto y el resultado será similar al siguiente:

```
{
  "ok": true,
  "resultado": {
    "codigo": "32",
    "descripcion": "Teclado",
    "precio": 15
  }
}
```

Observa que obtenemos 2 atributos:

ok: que es booleano

resultado: es un objeto que tiene los 3 atributos antes mencionados.

Será necesario contar con una clase que represente a dicha estructura:

Estructura de clase modelo

```
public class RespProducto {
    private boolean ok;
    private Producto resultado;

    public boolean getOk() {
        return ok;
    }

    public Producto getResultado() {
        return resultado;
    }
}
```

Observa que resultado es de tipo Producto (la clase declarada anteriormente).

3- URL para eliminar un registro usando la API:


<http://192.168.1.2/APIUDB/producto/eliminar/?codigo=32>

Donde el valor de 1000 que esta al final representa el código de ejemplo de un producto. Si dicho producto existe y es borrado satisfactoriamente obtendremos un resultado como el siguiente:

```
{
  "ok": true,
  "resultado": 1
}
```

ok: representa que hicimos un requerimiento correcto al intentar borrar dicho registro.

Resultado 1: que el registro ha sido borrado satisfactoriamente. Cuando el resultado es 0 (cero) es por que intentamos borrar un producto cuyo código no existía.

	UNIVERSIDAD DON BOSCO FACULTAD DE INGENIERÍA ESCUELA DE COMPUTACIÓN		CICLO 01-2022
	MATERIA	DESARROLLO DE SOFTWARE PARA MÓVILES	GUIA DE LABORATORIO N° 6
	PRÁCTICA	CONSUMO DE API REST CON RETROFIT	

Clase modelo que representa el resultado:

```
public class Respuesta {
    private boolean ok;
    private int resultado;

    public boolean getOk() {
        return ok;
    }

    public int getResultado() {
        return resultado;
    }
}
```

4- URL para permitir agregar un nuevo registro usando la API:

<http://192.168.1.2/APIUDB/producto/agregar/>

En esta ocasión vamos a tener que enviar un objeto además de la ruta de la URL para poder enviar el producto que deseamos agregar. Este objeto será de tipo Producto (La clase definida anteriormente). La API en cuestión fue diseñada para esperar a este objeto en un parámetro llamado **Body**. Por lo tanto al momento de enviar dicho objeto lo haremos canalizado en ese parámetro. (**Body**)

Resultado cuando el producto fue agregado satisfactoriamente:

```
{
  "ok": true,
  "resultado": 1
}
```

Ok: representa que hicimos un requerimiento correcto al intentar agregar dicho registro.


Resultado: 1. El registro se agrego satisfactoriamente. Cero (0) cuando el registro no se pudo agregar satisfactoriamente (podría ser por que el código ya existía anteriormente)
La clase modelo para poder leer este resultado es el mismo que el caso anterior: la clase **Respuesta**.

5- URL para permitir actualizar un registro usando la API:

<http://192.168.1.2/APIUDB/actualizar/?codigo=32>

Esta URL es similar a la de borrado (DELETE) pero en esta ocasión vamos a necesitar enviar un objeto de tipo Producto en el parámetro BODY y la respuesta obtenida será similar a la anterior:

```
{
  "ok": true,
  "resultado": 1
}
```

	UNIVERSIDAD DON BOSCO FACULTAD DE INGENIERÍA ESCUELA DE COMPUTACIÓN		CICLO 01-2022
	MATERIA	DESARROLLO DE SOFTWARE PARA MÓVILES	GUIA DE LABORATORIO N° 6
	PRÁCTICA	CONSUMO DE API REST CON RETROFIT	

Ok: true. Pudimos comunicarnos correctamente con el servidor al intentar actualizar el registro.

Resultado: 1. Si recibimos un 1 es porque el registro pudo actualizarse correctamente. De lo contrario recibiremos un cero (0), esto podría ser por que el código del producto que intentamos actualizar no existía en la base de datos al momento de actualizar.

La clase que representa esta estructura es la misma que la anterior: **Respuesta**.

V. DESARROLLO DE PRÁCTICA – SEGUNDA PARTE

Antes de comenzar con el desarrollo de nuestra aplicación, realicemos la siguiente actividad:

- 1- Verificar que tengamos instalado el gestor de Base de Datos MySQL y el servidor Apache para la utilización de PHP, en caso contrario debería de descargar el software e instalarlo. Puede utilizar la siguiente dirección para descargar el paquete completo de servidor apache + PHP <https://www.apachefriends.org/es/index.html>.
- 2- Verifique la dirección IP de su equipo, esta nos servirá para la conectividad de nuestra aplicación.
- 3- En los recursos proporcionados encontrara un archivo DDL.sql, el cual contiene la creación de la base de datos que utilizaremos.
- 4- Copie los recursos en el servidor apache, la estructura de su directorio seria APIADB/producto.

Una vez realizadas las anteriores verificaciones, procedamos a clonar el siguiente proyecto:

<https://github.com/Alexjimenez7/EjemploRetrofit/tree/master>

Procedamos a realizar las siguientes modificaciones al proyecto clonado:

1. Localice nuestro archivo AndroidManifest.xml y agreguemos el siguiente permiso:
`<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />`
2. Localicemos el archivo interfaces/APIService y actualicemos el siguiente código:

```
public interface APIService {

    @GET(".")
    Call<List<Producto>> getProducts();


    @GET("index.php")
    Call<RespProducto> getProductByld(@Query("codigo") String codigo);

    @POST("agregar/index.php")
    Call<Respuesta> insertProduct(@Body Producto producto);

    @DELETE("eliminar/index.php")
    Call<Respuesta> deleteProduct(@Query("codigo") String codigo);

    @PUT("actualizar/index.php")
    Call<Respuesta> updateProduct(@Query("codigo") String codigo, @Body Producto producto);

}
```

	UNIVERSIDAD DON BOSCO FACULTAD DE INGENIERÍA ESCUELA DE COMPUTACIÓN		CICLO 01-2022
	MATERIA	DESARROLLO DE SOFTWARE PARA MÓVILES	GUIA DE LABORATORIO N° 6
	PRÁCTICA	CONSUMO DE API REST CON RETROFIT	

Observa como cada uno de los encabezados de métodos definidos en la interface representa a cada tipo de solicitud que realizaremos en la API.

- Consulta de todos los registros
- Consulta de un registro específico
- Agregar un producto
- Borrar un producto
- Actualizar un producto

En los casos de agregar y actualizar un producto observa como usamos el parámetro @Body para enviar el objeto de tipo Producto.

Procedimiento que consulta todos los productos: (MainActivity)


```

Call<List<Producto>> call = Servicio.service.getProducts();
call.enqueue(new Callback<List<Producto>>() {
    @Override
    public void onResponse(Call<List<Producto>> call, Response<List<Producto>> response) {
        if (response.code() == 200) {
            productos.clear();
            List<Producto> prods = response.body();
            productos.addAll(prods);
            productAdapter.notifyDataSetChanged();

        } else {
            Toast.makeText(getBaseContext(), "Error:" + response.code(),
                Toast.LENGTH_SHORT).show();
        }
    }
});

@Override
public void onFailure(Call<List<Producto>> call, Throwable t) {
    Toast.makeText(getBaseContext(), "Error:" + t.getMessage(),
        Toast.LENGTH_SHORT).show();
}
});

```


	UNIVERSIDAD DON BOSCO FACULTAD DE INGENIERÍA ESCUELA DE COMPUTACIÓN		CICLO 01-2022
	MATERIA	DESARROLLO DE SOFTWARE PARA MÓVILES	GUIA DE LABORATORIO N° 6
	PRÁCTICA	CONSUMO DE API REST CON RETROFIT	

Procedimiento que consulta un producto a la vez (MainActivity)

```

Call<RespProducto> call = Servicio.service.getProductById(codigo);
call.enqueue(new Callback<RespProducto>() {
    @Override
    public void onResponse(Call<RespProducto> call, Response<RespProducto> response) {
        if (response.code()==200) {
            RespProducto producto = response.body();
            if (producto.getResultado()==null) {
                Toast.makeText(getApplicationContext(),"Código NO existe",
                    Toast.LENGTH_LONG).show();
                return;
            } else {
                productos.clear();
                Producto prod = response.body().getResultado();
                productos.add(prod);
                productAdapter.notifyDataSetChanged();
            }
        }
    }
});

@Override
public void onFailure(Call<RespProducto> call, Throwable t) {

}
});


```

Procedimiento que borra un producto (en ProductoAdapter)

```

Call<Respuesta> call = Servicio.service.deleteProduct(codigo);
call.enqueue(new Callback<Respuesta>() {
    @Override
    public void onResponse(Call<Respuesta> call, Response<Respuesta> response) {
        Respuesta r = response.body();
        if (r.getResultado()==1) {
            Toast.makeText(context, "Registro borrado",
                Toast.LENGTH_LONG).show();
        } else {Toast.makeText(context, "Registro NO SE borro",
            Toast.LENGTH_LONG).show();
        }
    }
});

```

	UNIVERSIDAD DON BOSCO FACULTAD DE INGENIERÍA ESCUELA DE COMPUTACIÓN		CICLO 01-2022
	MATERIA	DESARROLLO DE SOFTWARE PARA MÓVILES	GUIA DE LABORATORIO N° 6
	PRÁCTICA	CONSUMO DE API REST CON RETROFIT	

```

@Override
public void onFailure(Call<Respuesta> call, Throwable t) {
    Toast.makeText(context, "Error:" + t.getMessage(),
        Toast.LENGTH_LONG).show();
}
});

```

Procedimiento para agregar un producto (AgregarActivity)

```

String codigo = binding.edtCodigo.getText().toString().trim();
String descripcion = binding.edtDescripcion.getText().toString().trim();
float precio = Float.valueOf(binding.edtPrecio.getText().toString().trim());
Producto producto = new Producto(codigo,descripcion,precio);

Call<Respuesta> call = Servicio.service.insertProduct(producto);
call.enqueue(new Callback<Respuesta>() {
    @Override
    public void onResponse(Call<Respuesta> call, Response<Respuesta> response) {
        if (response.code()==200) {
            Toast.makeText(getBaseContext(),"Registro Agregado satisfactoriamente",
                Toast.LENGTH_LONG ).show();
            finish();
        } else
        {
            Toast.makeText(getBaseContext(),"Error : " + response.code(),
                Toast.LENGTH_LONG ).show();
        }
    }
}

@Override
public void onFailure(Call<Respuesta> call, Throwable t) {
    Toast.makeText(getBaseContext(),"Error : " + t.getMessage(),
        Toast.LENGTH_LONG ).show();
}
});

```


Procedimiento para actualizar un producto (También en AgregarActivity)

```

String codigo = binding.edtCodigo.getText().toString().trim();
String descripcion = binding.edtDescripcion.getText().toString().trim();
float precio = Float.valueOf(binding.edtPrecio.getText().toString().trim());

Producto producto = new Producto(codigo,descripcion,precio);

```

	UNIVERSIDAD DON BOSCO FACULTAD DE INGENIERÍA ESCUELA DE COMPUTACIÓN		CICLO 01-2022
	MATERIA	DESARROLLO DE SOFTWARE PARA MÓVILES	GUIA DE LABORATORIO N° 6
	PRÁCTICA	CONSUMO DE API REST CON RETROFIT	

```

Call<Respuesta> call = Servicio.service.updateProduct(codigo,producto);
call.enqueue(new Callback<Respuesta>() {
    @Override
    public void onResponse(Call<Respuesta> call, Response<Respuesta> response) {
        Respuesta respuesta = response.body();
        if (respuesta.getResultado()==1) {
            Toast.makeText(getBaseContext(),"Registro actualizado",
                Toast.LENGTH_LONG).show();
            finish();
        } else {
            Toast.makeText(getBaseContext(),"Error:" +response.code(),
                Toast.LENGTH_LONG).show();
        }
    }
});

@Override
public void onFailure(Call<Respuesta> call, Throwable t) {
    Toast.makeText(getBaseContext(),"Error:" + t.getMessage(),
        Toast.LENGTH_LONG).show();
}
});

```

3. Localicemos el siguiente archivo interfaces/Servicio y actualicemos su código:

```

public class Servicio {

    private static String IP=""; //Agregar su dirección IP
    public static Retrofit retrofit = new Retrofit.Builder()
        .baseUrl("http://"+IP+"/APIUDB/producto/")
        .addConverterFactory(GsonConverterFactory.create())
        .build();

    public static ApiService service =
        retrofit.create(ApiService.class);
}

```


4. Localicemos el siguiente archivo models/Producto y actualicemos su código:

```

public class Producto {

    @SerializedName("codigo")
    private String codigo;
    @SerializedName("descripcion")

```

	UNIVERSIDAD DON BOSCO FACULTAD DE INGENIERÍA ESCUELA DE COMPUTACIÓN		CICLO 01-2022
	MATERIA	DESARROLLO DE SOFTWARE PARA MÓVILES	GUIA DE LABORATORIO N° 6
	PRÁCTICA	CONSUMO DE API REST CON RETROFIT	

```

private String descripcion;
@SerializedName("precio")
private float precio;

public Producto(String codigo, String descripcion, float precio) {
    this.codigo = codigo;
    this.descripcion = descripcion;
    this.precio = precio;
}

public String getCodigo() {
    return codigo;
}

public void setCodigo(String codigo) {
    this.codigo = codigo;
}

public String getDescripcion() {
    return descripcion;
}

public void setDescripcion(String descripcion) {
    this.descripcion = descripcion;
}

public float getPrecio() {
    return precio;
}

public void setPrecio(float precio) {
    this.precio = precio;
}
}

```


5. Actualicemos el código del siguiente archivo models/RespProducto

```

public class RespProducto {
    private boolean ok;
    private Producto resultado;
    private String mensaje;

    public boolean getOk() {
        return ok;
    }
}

```

	UNIVERSIDAD DON BOSCO FACULTAD DE INGENIERÍA ESCUELA DE COMPUTACIÓN		CICLO 01-2022
	MATERIA	DESARROLLO DE SOFTWARE PARA MÓVILES	GUIA DE LABORATORIO N° 6
	PRÁCTICA	CONSUMO DE API REST CON RETROFIT	

```

public Producto getResultado() {
    return resultado;
}

public String getMensaje() {
    return mensaje;
}
}

```

6. Actualicemos el código del siguiente archivo models/Respuesta

```

public class Respuesta {
    private boolean ok;
    private int resultado;
    private String mensaje;

    public boolean getOk() {
        return ok;
    }

    public int getResultado() {
        return resultado;
    }

    public String getMensaje() {
        return mensaje;
    }
}

```


7. Actualicemos el código de la clase AgregarActivity, busque los métodos y reemplácelos:

```

public void agregar() {
    String codigo = binding.edtCodigo.getText().toString().trim();
    String descripcion = binding.edtDescripcion.getText().toString().trim();
    float precio = Float.valueOf(binding.edtPrecio.getText().toString().trim());
    Producto producto = new Producto(codigo, descripcion, precio);

    Call<Respuesta> call = Servicio.service.insertProduct(producto);
    call.enqueue(new Callback<Respuesta>() {
        @Override
        public void onResponse(Call<Respuesta> call, Response<Respuesta> response) {
            Respuesta respuesta = response.body();
            String mensaje = respuesta.getMensaje().toString();
            Log.i("INFORMACION", mensaje);
            if (response.code() == 200) {
                Toast.makeText(getApplicationContext(), mensaje,

```

	UNIVERSIDAD DON BOSCO FACULTAD DE INGENIERÍA ESCUELA DE COMPUTACIÓN		CICLO 01-2022
	MATERIA	DESARROLLO DE SOFTWARE PARA MÓVILES	GUIA DE LABORATORIO N° 6
	PRÁCTICA	CONSUMO DE API REST CON RETROFIT	

```

        Toast.LENGTH_LONG).show();
    finish();
} else {
    Toast.makeText(getBaseContext(), "Error : " + mensaje + "Codigo: " + response.code(),
        Toast.LENGTH_LONG).show();
    }
}


@Override
public void onFailure(Call<Respuesta> call, Throwable t) {
    Toast.makeText(getBaseContext(), "Error : " + t.getMessage(),
        Toast.LENGTH_LONG).show();
    }
}

public void actualizar() {
    String codigo = binding.edtCodigo.getText().toString().trim();
    String descripcion = binding.edtDescripcion.getText().toString().trim();
    float precio = Float.valueOf(binding.edtPrecio.getText().toString().trim());

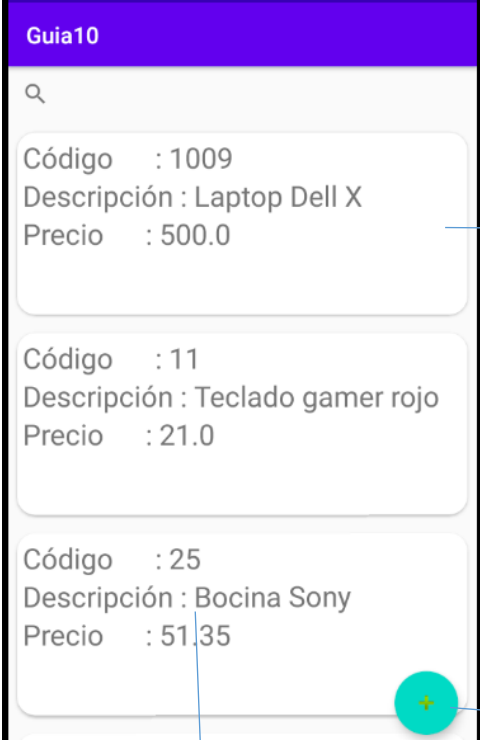
    Producto producto = new Producto(codigo, descripcion, precio);
    Call<Respuesta> call = Servicio.service.updateProduct(codigo, producto);
    call.enqueue(new Callback<Respuesta>() {
        @Override
        public void onResponse(Call<Respuesta> call, Response<Respuesta> response) {
            Respuesta respuesta = response.body();
            String mensaje = respuesta.getMensaje().toString();
            Log.i("INFORMACION", mensaje);
            if (respuesta.getResultado() == 1) {
                Toast.makeText(getBaseContext(), mensaje,
                    Toast.LENGTH_LONG).show();
                finish();
            } else {
                Toast.makeText(getBaseContext(), "Error:" + response.code(),
                    Toast.LENGTH_LONG).show();
            }
        }
    })

    @Override
    public void onFailure(Call<Respuesta> call, Throwable t) {
        Toast.makeText(getBaseContext(), "Error:" + t.getMessage(),
            Toast.LENGTH_LONG).show();
    }
}
}


```

	UNIVERSIDAD DON BOSCO FACULTAD DE INGENIERÍA ESCUELA DE COMPUTACIÓN		CICLO 01-2022
	MATERIA	DESARROLLO DE SOFTWARE PARA MÓVILES	GUIA DE LABORATORIO N° 6
	PRÁCTICA	CONSUMO DE API REST CON RETROFIT	

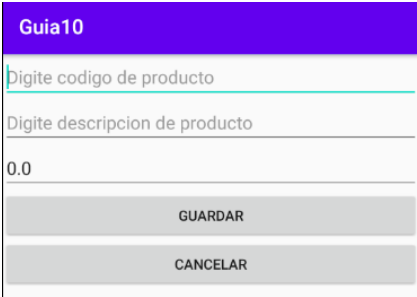
8. Ejecute su aplicación y verifique su funcionamiento.
9. Cuando ejecute la app podrás utilizarla de la siguiente manera forma:

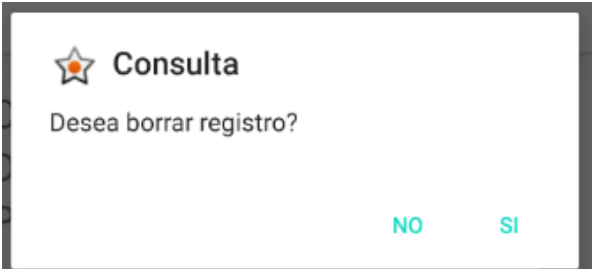


Cuando hagas clic en un registro podrás editarlo




Cuando hagas clic en este botón podrás agregar un nuevo





Cuando mantengas presionado un registro podrás eliminarlo si respondes afirmativamente la pregunta

	UNIVERSIDAD DON BOSCO FACULTAD DE INGENIERÍA ESCUELA DE COMPUTACIÓN		CICLO 01-2022
	MATERIA	DESARROLLO DE SOFTWARE PARA MÓVILES	GUIA DE LABORATORIO N° 6
	PRÁCTICA	CONSUMO DE API REST CON RETROFIT	

VI. DISCUSIÓN DE RESULTADOS

1. Crea una aplicación Android que se conecte al API REST de Github que permita buscar por nombre de usuario (username) y muestre un listado de sus repositorios públicos.
2. Para la aplicación de la segunda parte deberas de mejorar los siguientes aspectos:
 - a. Si se borra un producto, no se actualiza automáticamente la lista de productos hasta que forzamos a que se actualice buscando otro producto.
 - b. Si agregamos y/o editamos un producto sucede algo similar, puedes hacer que la aplicación refresque automáticamente la lista de productos.
 - c. Modifica la app para que realice estas operaciones automáticamente.

VII. BIBLOGRAFÍA

- Documentación Oficial Android
<https://developer.android.com/guide/topics/ui/layout/recyclerview#java>
- Documentación Oficial Retrofit
<https://square.github.io/retrofit/>