	UNIVERSIDAD DON BOSCO **FACULTAD DE INGENIERÍA** **Escuela de Computación**	
CICLO: I/2021	**GUIA DE LABORATORIO #09**	
	Nombre de la Práctica:	Consumir API REST con Retrofit
	MATERIA:	Desarrollo de Software para Móviles

I. OBJETIVOS

1. Conocer la librería Retrofit.
2. Conocer el concepto de API REST.
3. Consumir un API REST usando Retrofit en una aplicación Android.

II. INTRODUCCION TEORICA

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.

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 de los 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** (fijaros que en vez de llaves tiene corchetes), que este almacena id, firstName, lastName y photo. Así podemos pasarnos gran cantidad de información de una plataforma a otra con unos estándar que nos ayudan a simplificar el proceso.

III. PROCEDIMIENTO

La aplicación a desarrollar en esta ocasión tratará de un buscador de razas de perros. Es decir, en el buscador vamos a poner una raza de perro (en inglés) y recuperaremos imágenes de dicha raza que mostraremos en un RecyclerView, accederemos a internet para mostrar imágenes que nuestra app no tiene.

El API a consumir será Dog API, es totalmente gratuita y tiene una documentación sencilla y práctica. Si vamos a la documentación podemos ver todos los tipos de llamada que hay, a nosotros nos interesará uno solo By breed, es decir por raza.

1. Ejecuta Android Studio y crea un nuevo proyecto. En **Project Template** selecciona **Empty Activity** y presiona **Next**.

2. Como **Name** coloca **Guia09App,** en **Package name** escribe **sv.edu.udb.guia09app** en **Save location** escoge la carpeta de tu preferencia, en Language elige **Java** y el **Minimum SDK** selecciona **API 16: Android 4.1 (Jelly Bean)**

Nota: En esta práctica se hará uso del lenguaje de programación Java pero, se invita al estudiante a realizarla también utilizando Kotlin.

3. Presiona **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"
```

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.guia09app;

import androidx.appcompat.app.AppCompatActivity;
import android.os.Bundle;
import sv.edu.udb.guia09app.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"
  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.guia09app;

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;
    }

    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**.

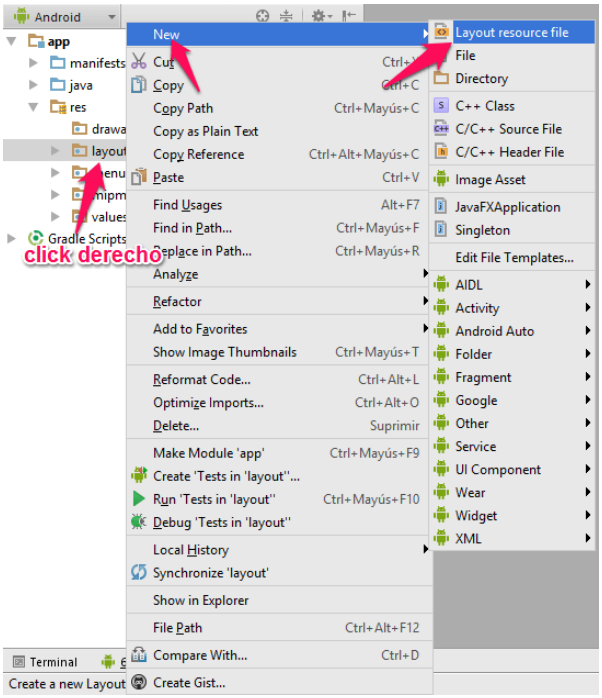
```
package sv.edu.udb.guia09app;

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;
}
```

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




```
<?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.guia09app;

import android.view.View;
import androidx.recyclerview.widget.RecyclerView;
import com.squareup.picasso.Picasso;
import sv.edu.udb.guia09app.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);
  }

}
```

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

```
package sv.edu.udb.guia09app;

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;
    }

}
```

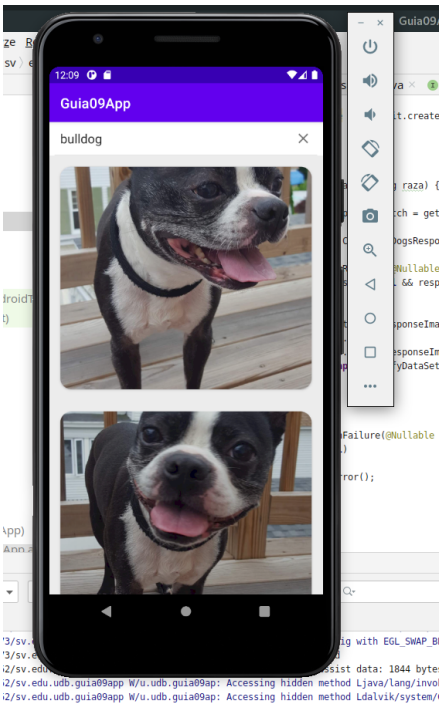
15. Ahora que ya están todos los componentes listos, solo queda ensamblar todo en el MainActivity, para eso modifica tu actividad para que quede de la siguiente manera.

```
package sv.edu.udb.guia09app;

import androidx.annotation.Nullable;
import androidx.appcompat.app.AppCompatActivity;
import androidx.appcompat.widget.SearchView;
import androidx.recyclerview.widget.LinearLayoutManager;
import android.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.guia09app.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;
    }

    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> responseImages = response.body().getImages();
                    images.clear();
                    images.addAll(responseImages);
                    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. Presiona **run app**.



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. 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.

V. BIBLIOGRAFÍA

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