Instituto Superior
Tecnológico del Azuay

# GUÍA PRÁCTICA UNIDAD IV
# CONSUMO SERVICIOS WEB


**ESTUDIANTE**

Fausto Campoverde


**CURSO**

M4A

**MATERIA**

Desarrollo de Aplicaciones Móviles


**DOCENTE**

Ing. Patricio Pacheco


**FECHA**

09/03/2022

# CONSUMO DE API REST CON RETROFIT

## 1. Introducción

Los servicios web permiten la comunicación interactiva entre el cliente y el servidor, solicita los datos a la base de datos y los recoge, para que estos puedan ser presentados en pantalla. Por lo que hoy en día, la gran cantidad de tecnologías utilizan esta arquitectura, las aplicaciones móviles también lo incorporan mediante el uso de librerías o recursos de software, que tienen como función acceder a una API y consumir los servicios que esta lo establece.

## 2. Objetivos

Consumir una API (servicios web) utilizando el IDE de Desarrollo de Android Studio con la librería Retrofit y procesar la respuesta JSON obtenida.

## 3. Procedimiento

### 3.1. Añadir librería retrofit a nuestro proyecto

```
dependencies {

    implementation 'androidx.appcompat:appcompat:1.4.1'
    implementation 'com.google.android.material:material:1.5.0'
    implementation 'androidx.constraintlayout:constraintlayout:2.1.3'

    /*Dependencias para retrofit  */
    implementation 'com.squareup.retrofit2:retrofit:2.3.0'
    implementation 'com.squareup.retrofit2:converter-gson:2.3.0'
    implementation 'com.squareup.okhttp3:logging-interceptor:3.9.1'

    /* ---------------- */
    testImplementation 'junit:junit:4.+'
    androidTestImplementation 'androidx.test.ext:junit:1.1.3'
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.4.0'
}
```

### 3.2. Solicitar permisos

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.faustoc.consumoapi_retrofit">

    <uses-permission android:name="android.permission.INTERNET"/>
```

## 3.3. Crear una clase y una interfaz

### 3.3.1. Interfaz con los métodos de acuerdo a su petición http

```java
public interface MyApiService {

    @GET("api")
    Call<List<Testimonio>> getTestimonials();

}
```

### 3.3.2. Clase para crear un método que recibe acceso a la baseUrl de la API

```java
public class MyApiAdapter {

    private MyApiService apiService;

    public MyApiService getApiService(String baseUrl){

        HttpLoggingInterceptor logging = new HttpLoggingInterceptor();
        logging.setLevel(HttpLoggingInterceptor.Level.BODY);

        OkHttpClient.Builder httpClient = new OkHttpClient.Builder();
        httpClient.addInterceptor(logging);

        Retrofit retrofit = new Retrofit.Builder()
                .baseUrl(baseUrl)
                .addConverterFactory(GsonConverterFactory.create())
                .client(httpClient.build())
                .build();

        apiService = retrofit.create(MyApiService.class);
        return apiService;

    }
}
```

### 3.3.3. Creación de la Clase Entidad que recibirá los atributos del JSON

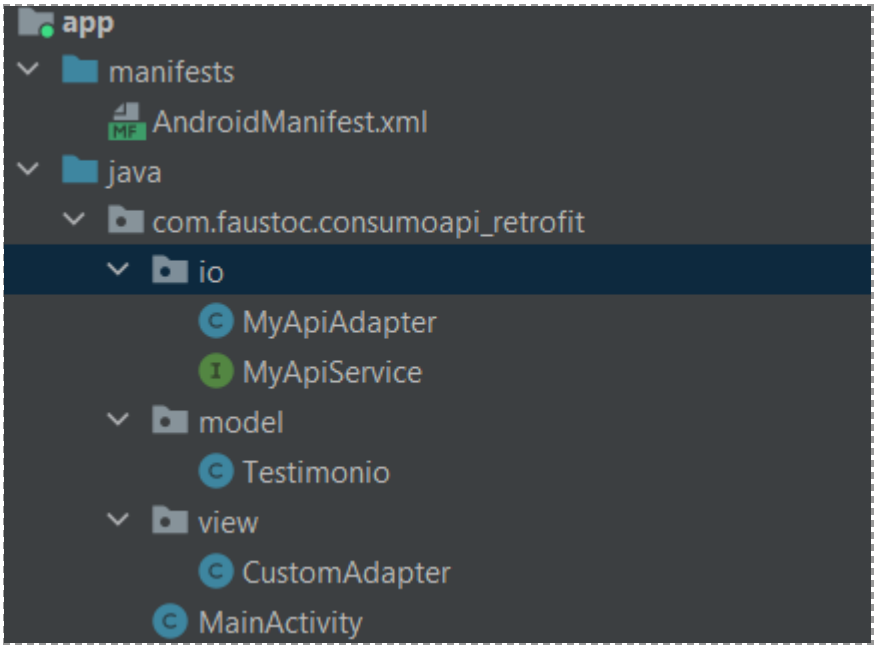
```java
public class Testimonio {

    private int id;
    private String name;
    private String location;
    private String designation;
    private String avatar;
    private String message;
    private String lorem;
    private double raiting;
    private String audio;


    public int getId() { return id; }

    public void setId(int id) { this.id = id; }

    public String getName() { return name; }

    public void setName(String name) { this.name = name; }

    public String getLocation() { return location; }

    public void setLocation(String location) { this.location = location; }

    public String getDesignation() { return designation; }

    public void setDesignation(String designation) { this.designation = designation
```

3.3.4. Creación de atributos y métodos dentro de la Clase Main para el uso de la lógica creada

```java
public class MainActivity extends AppCompatActivity {


    private MyApiAdapter myApiAdapter;
    private String baseUrl;

    private ListView listView;
    private CustomAdapter adapter;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        listView = findViewById(R.id.listView);
        baseUrl = "https://testimonialapi.toolcarton.com/";
        myApiAdapter = new MyApiAdapter();
        searchData();

    }
```

```java
protected void searchData() {

    MyApiService myApiService = myApiAdapter.getApiService(baseUrl);

    Call<List<Testimonio>> call = myApiService.getTestimonials();
    call.enqueue(new Callback<List<Testimonio>>() {
        @Override
        public void onResponse(Call<List<Testimonio>> call, Response<List<Testimonio>> response) {
            if (response.isSuccessful()) {
                List<Testimonio> testimonioResponse = response.body();
                adapter = new CustomAdapter(getApplicationContext(),testimonioResponse);
                listView.setAdapter(adapter);
                  for (Testimonio t:
                     testimonioResponse) {
                    System.out.println(t.getMessage());
                }*/


            } else {
                System.out.println(response.errorBody());
            }

        }

        @Override
        public void onFailure(Call<List<Testimonio>> call, Throwable t) {
            System.out.println(t.getMessage());
        }
    });
```

### 3.3.5. Organización del código




## 4. Conclusiones

La implementación de la librería retrofit al igual que la librería volley, permite consumir servicios web desde nuestra aplicación móvil, sin embargo retrofit según la guía presentada, permitió el consumo de estos servicios de una manera más legible, la modularización que se implementó en la guía favoreció una mejor comprensión de los métodos que ofrece la librería, la obtención de los datos del JSON para ser incorporados dentro de una clase Java, para que estos al final puedan ser presentados en consola o en un componente de interfaz de Android Studio.