	UNIVERSIDAD DON BOSCO **FACULTAD DE INGENIERÍA** **ESCUELA DE COMPUTACIÓN**	
CICLO: I/2021	**GUIA DE LABORATORIO #10**	
	Nombre de la Práctica:	Consumir API REST con Retrofit (Segunda Parte)
	MATERIA:	Desarrollo de Software para Móviles

I. OBJETIVOS

1. Conocer la librería Retrofit para enviar datos y actualizar, agregar y eliminar además de consultar información.

2. Consumir un API REST usando Retrofit en una aplicación Android.

II. INTRODUCCION TEORICA

En la guía anterior estudiamos la forma de consumir una API usando Retrofit para consultar información usando el verbo GET.

En la presente guía veremos una aplicación que hace uso de los 4 verbos para permitir hacer un CRUD con una tabla que se encuentra en un servidor SQL Server que esta en internet. También la API esta en un servidor de Amazon Web Services (AWS) y fue desarrollada en noje.js.

Esta API estará disponible hasta junio del presente año (2021).

Analicemos cada una de las URL que nos permitirán acceder a dicha API.

URL base : **http://18.206.142.59:5000/**

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

   http://18.206.142.59:5000/sql

Resultado Ejemplo:


```
[
    {
        "codigo": "1009",
        "descripcion": "Laptop Dell X",
        "precio": 500
    },
    {
        "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 accesores (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;
    }
}
```

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

   http://18.206.142.59:5000/sql/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.
   Sera 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)

URL para eliminar registros usando la API:

http://18.206.142.59:5000/sql/1000

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.

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;
    }
}
```

3. URL para permitir agregar un nuevo registro con la API:

http://18.206.142.59:5000/sql/agrega

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**.

4. URL para permitir actualizar un producto a través de la API.

   http://18.206.142.59:5000/sql/1000

   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
}
```

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

   Resultado: 1. Si recibimos un 1 es por que el registro pudo actualizares 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**.

III. PROCEDIMIENTO

Como hemos dicho, la aplicación permitirá hacer un CRUD con una tabla de productos usando la API que acabamos de describir.

Puedes obtener el proyecto ejemplo Android Studio de la siguiente ruta en GitHub: https://github.com/Alexjimenez7/EjemploRetrofit/tree/master

El proyecto cuenta con la interface respectiva de Retrofit para consumir la API con los 5 casos descritos anteriormente:

```
public interface APIService {

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

    @GET()
    Call<RespProducto> getProductById(@Url String url);

    @POST("agrega")
    Call<Respuesta> insertProduct(@Body Producto producto);

    @DELETE()
    Call<Respuesta> deleteProduct(@Url String url);

    @PUT()
    Call<Respuesta> updateProduct(@Url String url, @Body Producto producto);

}
```

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

1- Consulta de todos los registros
2- Consulta de un registro especifico
3- Agregar un producto
4- Borrar un producto
5- 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();
    }
});
```

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(getBaseContext(),"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();

        }
    }

    @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)

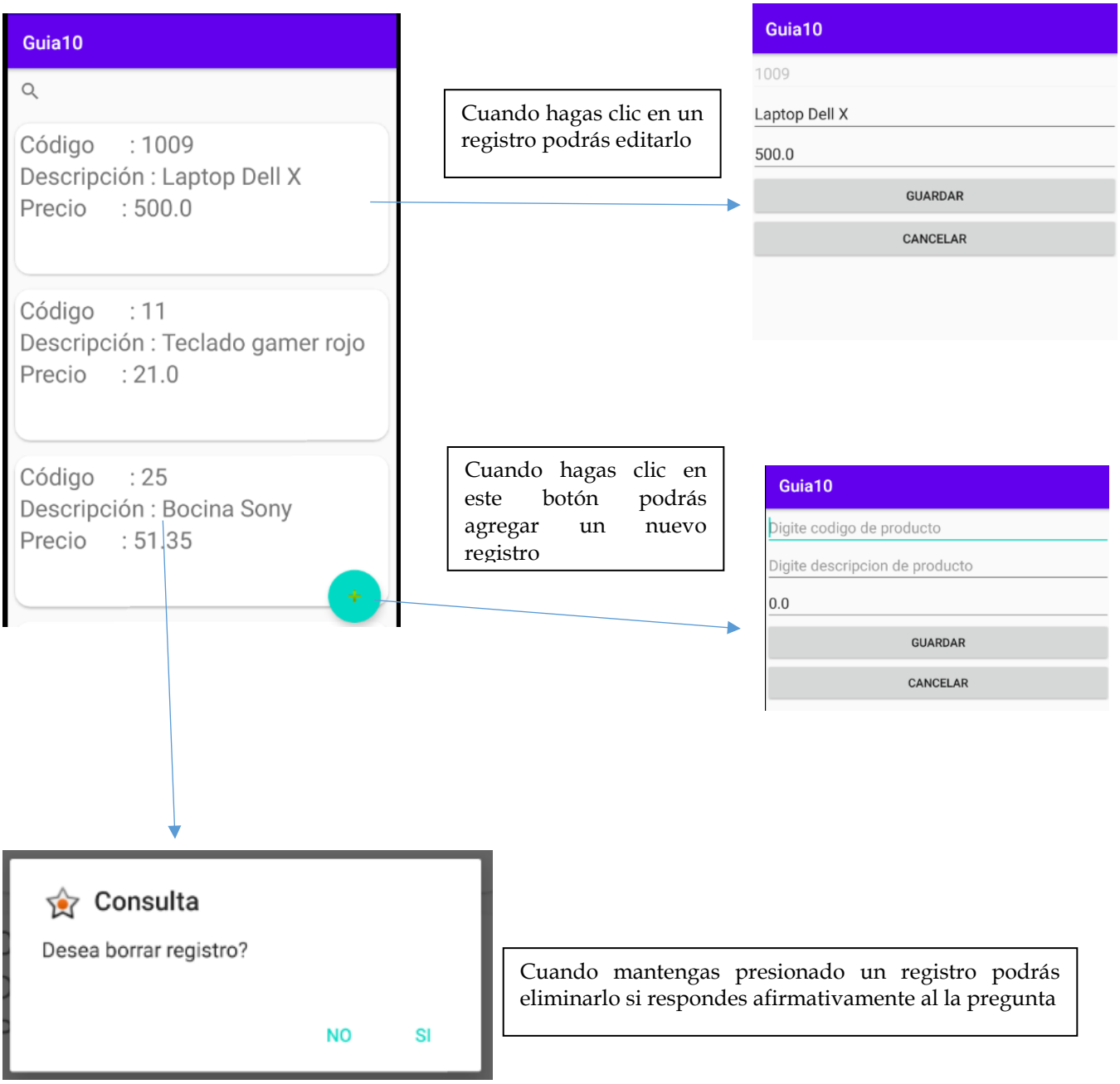
```
String codigo =  binding.edtCodigo.getText().toString().trim();
String descripcion = binding.edtDescripcion.getText().toString().trim();
float precio =
Float.valueOf(binding.edtPrecio.getText().toString().trim());

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

Cuando ejecutes la app podrás utilizarla de la siguiente forma:



IV. DISCUSIÓN DE RESULTADOS

1. Puedes mejorar algunos aspectos de la aplicación:

    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.

V. BIBLIOGRAFÍA

- Documentación Oficial Android
  https://developer.android.com/guide/topics/ui/layout/recyclerview#java

- Documentación Oficial Retrofit
  https://square.github.io/retrofit/