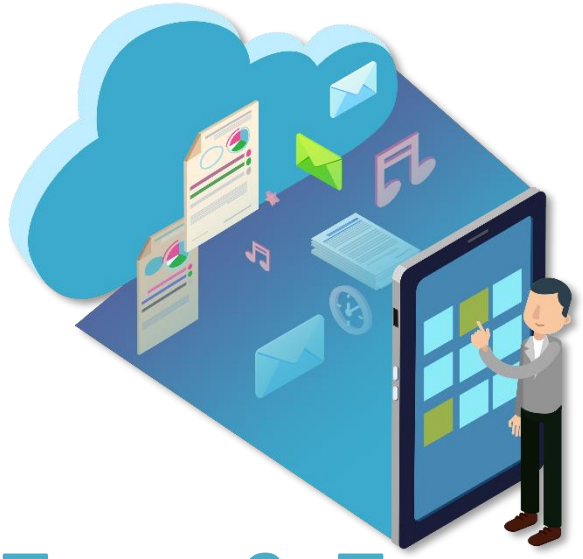
2024



Tema 9: Temario

ALMACENAMIENTO REMOTO

STUDIUM

9.1 Introducción

En un tema anterior se han estudiado varias maneras de guardar información localmente, en el propio dispositivo. Esto solamente es interesante si se trata de poca información o temporalmente hasta tener conexión, ya sea **GSM** o por **Wifi**. Lo habitual es que nuestras aplicaciones trabajen con gran cantidad de datos y que estos solamente estén disponibles en el dispositivo el tiempo necesario para ser mostrados en pantalla para consultarlos, actualizarlos, ... y poco más. Así conseguimos **aplicaciones ligeras** que no ocupan mucho espacio en el dispositivo, ni la aplicación en sí misma ni los datos que maneja.

Para que nuestras aplicaciones trabajen de esta forma, necesitamos que éstas se comuniquen con un **repositorio de datos remotos**, ya sea una base de datos directamente, o, lo que es más habitual, un servicio de conexión que hace de intermediario entre los datos y la propia aplicación.

A estos servicios se les suele denominar **API** de Application Programming Interface o Interfaz de Programación de Aplicaciones. Existen varios tipos (SOAP, Rest), aunque hoy en día los más utilizados son los que usan la arquitectura REST para las comunicaciones.

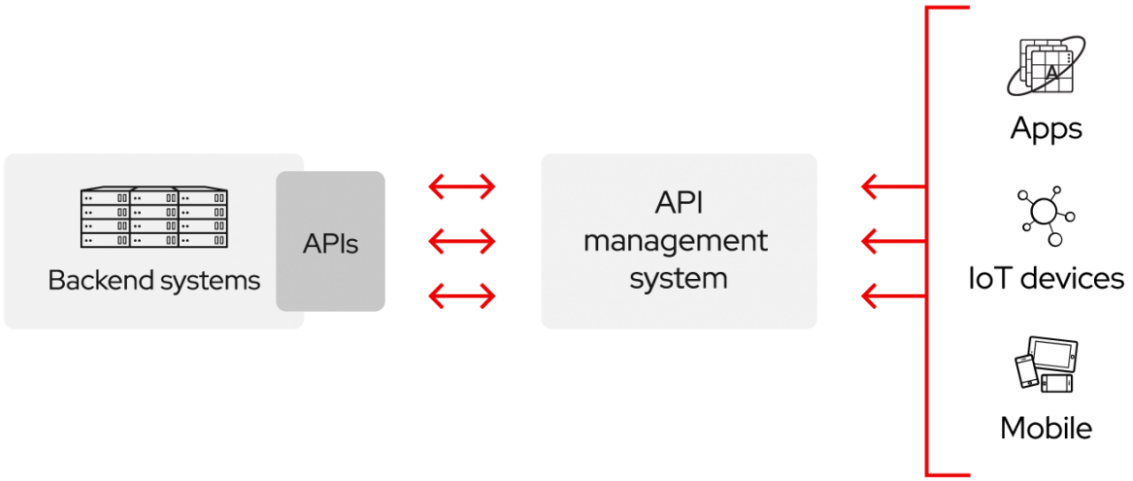
Mediante esta tecnología, podemos conectar nuestras aplicaciones Android con servidores remotos donde tendremos disponible la funcionalidad necesaria. Esa funcionalidad puede estar "montada" en JAVA, en PHP, en JavaScript o en .NET.

E igual que hablamos de Android, estas APIs pueden dar servicio también a iOS, con lo cual, todo el conjunto es tremendamente funcional y eficiente.

9.2 Api Rest

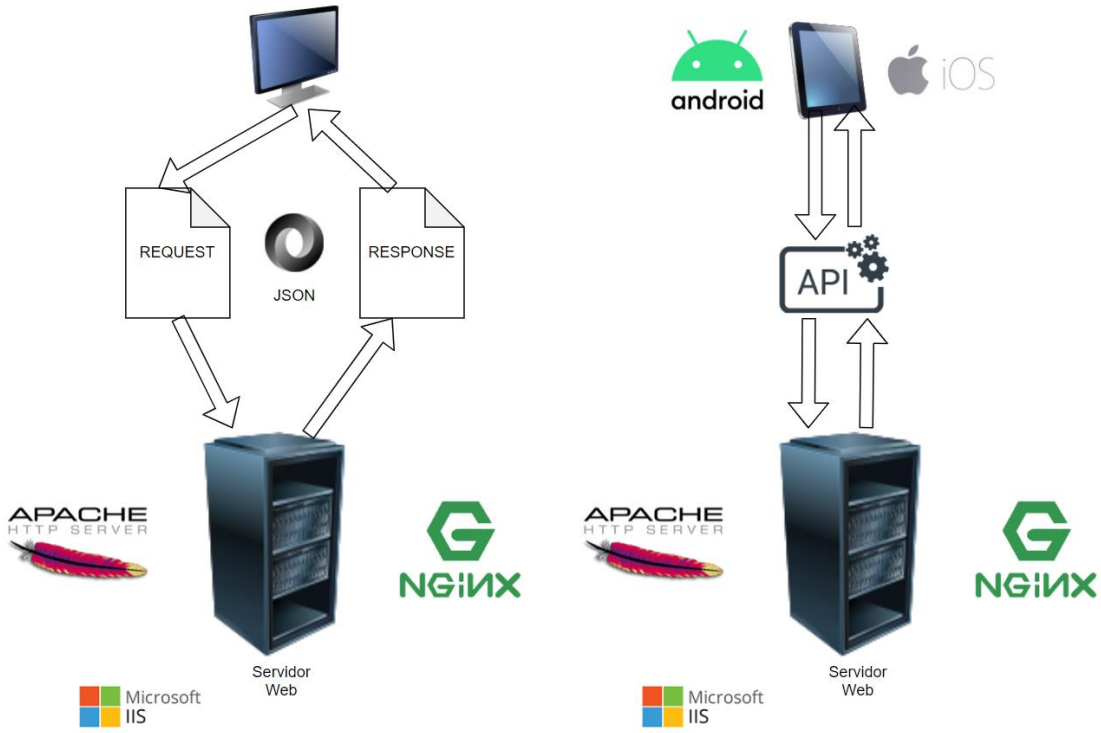
Una **API de transferencia de estado representacional** (REST), o API de RESTful, es una interfaz de programación de aplicaciones que se ajusta a los límites de la arquitectura REST.

Una **API** o **interfaz de programación de aplicaciones** es un conjunto de definiciones y protocolos que se usa para diseñar e integrar el software de aplicaciones. Suele considerarse como el **contrato** entre un proveedor de información y un usuario, donde se establece el contenido que se requiere del consumidor (la llamada) y el que necesita el productor (la respuesta). Por ejemplo, una **API de servicio meteorológico** podría solicitar que el usuario escribiera un código postal y que el productor diera una respuesta en dos partes: la primera sería la temperatura máxima y la segunda, la mínima.



En otras palabras, si desea interactuar con una computadora o un sistema para obtener datos o ejecutar una función, las API le permiten comunicar lo que desea al sistema, para que este comprenda la solicitud y la cumpla.

Puede considerarse como el **mediador** entre los usuarios o clientes y los recursos que quieren obtener. También es una forma en que las empresas comparten recursos e información mientras conservan la seguridad y el control, y pueden definir los accesos de cada usuario.



Otra ventaja de las API es que el desarrollador de la aplicación móvil no necesita saber cómo se recibe el recurso o de dónde proviene, solamente debe conocer las funciones necesarias, cómo enviar las consultas y cómo recibir los resultados de dichas consultas.

Por ejemplo, imagínese una empresa distribuidora de libros. Podría ofrecer a los clientes una **aplicación** que les permita a los empleados de la librería verificar la disponibilidad de los libros con el distribuidor. El desarrollo de esta aplicación podría ser costoso, estar limitado por la plataforma y requerir mucho tiempo de desarrollo y mantenimiento continuo.

Otra opción es que la distribuidora de libros proporciona una API para verificar la disponibilidad en inventario. Existen varios beneficios de este enfoque:

- Permite que los clientes accedan a los datos con una API que les ayude a añadir información sobre su inventario en un solo lugar.
- La distribuidora de libros podría realizar cambios en sus sistemas internos sin afectar a los clientes, siempre y cuando el comportamiento de la API fuera el mismo.
- Con una API disponible de forma pública, los desarrolladores que trabajan para la distribuidora de libros, los vendedores o los terceros podrían desarrollar una aplicación para ayudar a los clientes a encontrar los libros que necesiten. Esto podría dar como resultado mayores ventas u otras oportunidades comerciales.

En resumen, las **API** le permiten habilitar el acceso a sus recursos y, al mismo tiempo, mantener la seguridad y el control. Cómo habilitar el acceso y a quiénes dependen del desarrollador de la API.

REST no es un protocolo ni un estándar, sino que se trata de un conjunto de principios de arquitectura. Los desarrolladores de las API pueden implementarlo de distintas maneras.

Cuando se envía una solicitud a través de una API de RESTful, ésta transfiere una representación del estado del recurso requerido a quien lo haya solicitado. La información se entrega por medio de **HTTP** en uno de estos formatos: JSON (Javascript Object Notation), HTML, XLT o texto sin formato. JSON es el más popular, ya que tanto las máquinas como las personas lo pueden comprender y no depende de ningún lenguaje.

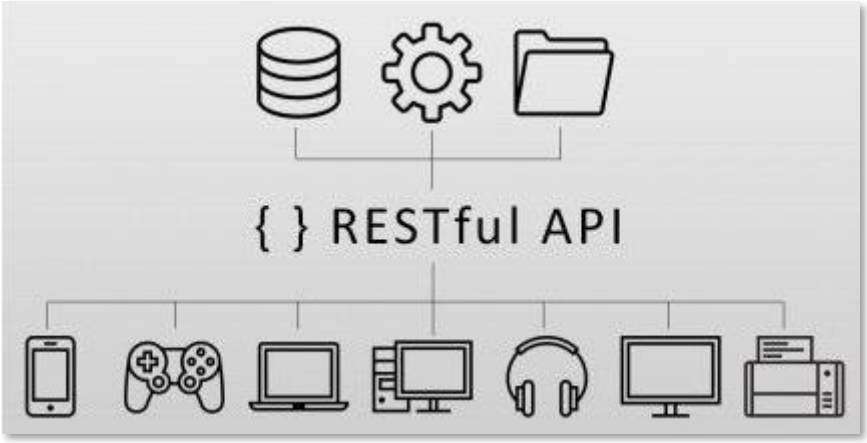

¿Qué es {JSON}?
JSON (JavaScript Object Notation) es un formato ligero de intercambio de datos, que usa la sintaxis de objeto de JS.
JSON, not Jason!
{} myfile.json
{
"name" : "Lupita Code",
"id" : 123,
"isProgrammer" : true,
"email" : null,
"languages" : ["Python", "JS"],
"address" : {
"street" : "5 main st",
"city" : "Boston"
}
}
JSON RULES:
El JSON Object comienza y termina con llaves {}
Pares: key:value separados por comas.
Los valores pueden ser: string, number, boolean, null, object y array.
Las claves deben ser únicas y deben estar entre comillas dobles.
USOS
API
En API REST o Graphql
Archivos de configuración
Interacciones AJAX o WebSocket

Para que una API se considere de RESTful, debe cumplir los siguientes **criterios**:

- Arquitectura cliente-servidor compuesta de clientes, servidores y recursos, con la gestión de solicitudes a través de HTTP.
- Comunicación entre el cliente y el servidor sin estado, así que no se almacena la información del cliente entre las solicitudes; cada una es independiente y está desconectada del resto.
- Datos que pueden almacenarse en caché y optimizan las interacciones entre el cliente y el servidor.
- Una interfaz uniforme entre los elementos, para que la información se transfiera de forma estandarizada. Para ello deben cumplirse las siguientes condiciones:
    - Los recursos solicitados deben ser identificables e independientes de las representaciones enviadas al cliente.
    - El cliente debe poder manipular los recursos a través de la representación que recibe, ya que esta contiene suficiente información para permitirlo.
    - Los mensajes autodescriptivos que se envíen al cliente deben contener la información necesaria para describir cómo debe procesarla.
    - Debe contener hipermedios, lo cual significa que cuando el cliente acceda a cierto recurso, debe poder utilizar hipervínculos para buscar las demás acciones disponibles en ese momento.
- Un sistema en capas que organiza en jerarquías invisibles para el cliente cada uno de los servidores (los encargados de la seguridad, del equilibrio de carga, etc.) que participan en la recuperación de la información solicitada.
- Código disponible según se solicite (opcional), es decir, la capacidad de enviar código ejecutable del servidor al cliente cuando se solicite, lo cual amplía las funciones del cliente.

Si bien la API de REST debe cumplir todos estos parámetros, resulta más fácil de usar que un protocolo definido previamente, como SOAP (*Simple Object Access Protocol o Protocolo Simple de Acceso a Objetos*), el cual tiene requisitos específicos, como la mensajería XML y la seguridad y el cumplimiento de las operaciones integrados, que lo hacen más lento y pesado.

Por el contrario, REST es un conjunto de pautas que pueden implementarse según sea necesario. Por esa razón, las API de REST son más rápidas y ligeras, y resultan ideales para el Internet de las Cosas (IoT) y el desarrollo de aplicaciones para dispositivos móviles.



Primeros pasos

Multitud de **empresas** y **organismos** ofrecen sus datos públicos, en la mayoría de las veces de forma totalmente gratuita para que sean usados por quien quiera. Ayuntamientos, oficinas de estadística, y un largo etcétera ponen a disposición del público en general estos datos de manea que puedan ser "consumidos" por las aplicaciones que sean, sobre todo para el tratamiento de dichos datos: medias, modas, distribuciones, estudios, ...

Como en casi la totalidad de las veces se ofrecen en formatos estándares como JSON o XML, desde un simple navegador podemos acceder a dicha información, como veremos a continuación.

En **Swapi**, un repositorio de "La Guerra de las Galaxias" on-line, ofrece, entre otros servicios, la posibilidad de consultar mediante un servicio API la información de sus bases de datos para su consulta. Por ejemplo, si en una pestaña de un navegador escribimos:

```
https://swapi.dev/api/people
```

Obtendremos la información de todos los personajes de la base de datos:

```
HTTP 200 OK
Content-Type: application/json
Vary: Accept
Allow: GET, HEAD, OPTIONS

{
    "count": 82,
    "next": "https://swapi.dev/api/people/?page=2",
    "previous": null,
    "results": [
        {
            "name": "Luke Skywalker",
            "height": "172",
            "mass": "77",
            "hair_color": "blond",
            "skin_color": "fair",
            "eye_color": "blue",
            "birth_year": "19BBY",
            "gender": "male",
            "homeworld": "https://swapi.dev/api/planets/1/",
            "films": [
                "https://swapi.dev/api/films/1/",
                "https://swapi.dev/api/films/2/",
                "https://swapi.dev/api/films/3/",
                "https://swapi.dev/api/films/6/"
            ],
```

Podemos delimitar la búsqueda con parámetros:

https://swapi.dev/api/people/3

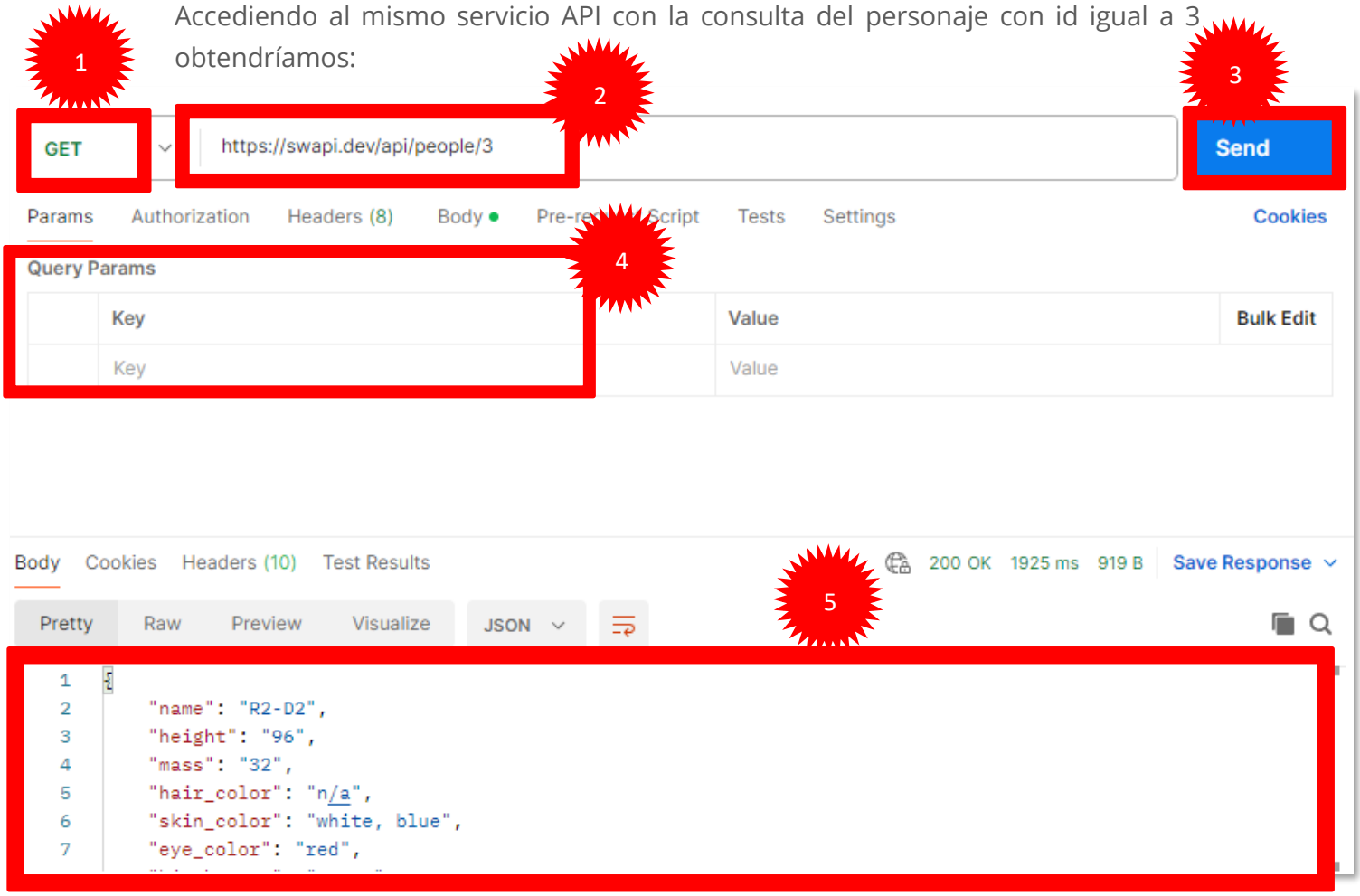

```
HTTP 200 OK
Content-Type: application/json
Vary: Accept
Allow: GET, HEAD, OPTIONS

{
    "name": "R2-D2",
    "height": "96",
    "mass": "32",
    "hair_color": "n/a",
    "skin_color": "white, blue",
    "eye_color": "red",
    "birth_year": "33BBY",
    "gender": "n/a",
    "homeworld": "https://swapi.dev/api/planets/8/",
```


Como podemos observar, el parámetro de la consulta, el valor del WHERE si fuera un SELECT de SQL sería igualmente un para clave=valor, en este caso id=3.

Y no solamente podemos hacer Consultas (GET). Si el API ha desarrollado el CRUD completo, podremos hacer Altas (POST), Bajas (DELETE) y Modificaciones (PUT) también.

Existen aplicaciones que nos facilitan la labora de interacción con las APIs web. Es el caso de **Postman**, que, mediante un interfaz gráfico sencillo, pero muy completo, nos permite lanzar peticiones de todo tipo (Altas, Bajas, Modificaciones y Consultas) y recoger las respuestas en los formatos más habituales, así como presentar dicha información de manera más amigable.

Accediendo al mismo servicio API con la consulta del personaje con id igual a 3 obtendríamos:



En 1 indicamos el tipo de operación:

- **GET** para Consultar, sus parámetros se colocan en la pestaña “Params”
- **POST** para Insertar, sus parámetros se colocan en la pestaña “Body”
- **PUT** para Modificar, sus parámetros se indican en la pestaña “Params”, y
- **DELETE** para Borrar, sus parámetros se colocan en la pestaña “Params”

En 2 se indica la dirección del API. Se pueden poner los parámetros oportunos, aunque también se pueden poner en 4.

Una vez "montada" la petición, se ejecuta en 3 y en 5 se nos muestran los resultados.

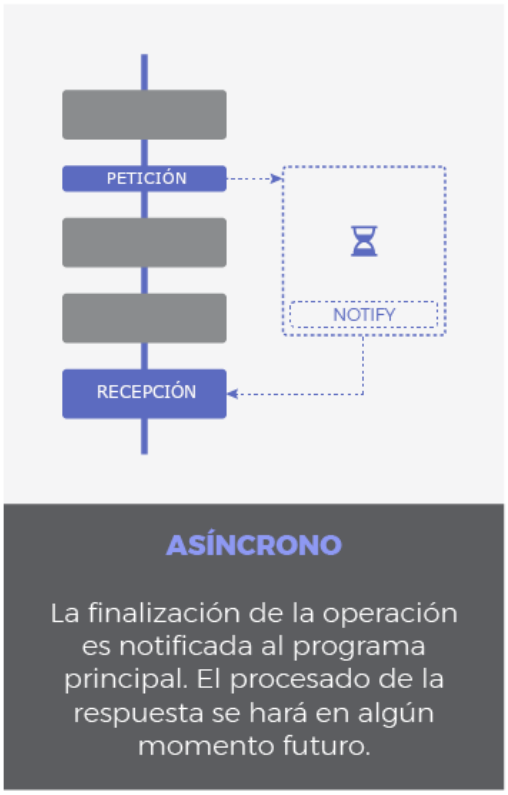
Ahora que ya sabemos cómo interactuar con una API mediante una **petición**, solamente debemos conocer el formato de **respuesta**, interpretar esta información y mostrar en nuestras Apps lo que deseemos: un listado, un "Alta Correcta", una "Modificación Correcta" o un "Error en la Baja", según proceda.

9.3 Comunicaciones

Tenemos dos formas de conectar nuestras aplicaciones a los servicios de datos por sus correspondientes APIs: de forma **síncrona** y de forma **asíncrona**.

En el primer modo, nuestra App realiza la conexión para realizar las operaciones oportunas y no se libera el recurso, ni en la App ni en el servidor remoto, hasta que desde la App se indica el fin de la transacción. Es una **forma sencilla** de comunicación, pero ante peticiones que consuman muchos recursos podemos bloquear la propia aplicación en incluso el dispositivo. Se supone que el servidor API debe estar preparado para asumir este consumo de datos, pero también puede llegar a saturarse.

Por otro lado, cuando las Apps se comunican con los servicios de datos a través de APIs en modo asíncrono, lo que ocurre es que desde nuestro dispositivo se establece la conexión igual que antes, pero solo el tiempo necesario para realizar la petición y se desconecta sin esperar la respuesta. Cuando esta esté preparada, el servidor será quien establezca la comunicación con nuestra App y le enviará los datos, cerrándose inmediatamente el canal de transmisión para hacer un uso más eficiente de los recursos. Por el contrario, su puesta en marcha es un **poco más elaborado**.



Comenzaremos viendo un ejemplo sencillo, con comunicación síncrona para establecer los primeros contactos con esta forma de trabajar para pasar más adelante a otros ejemplos más elaborados ya usando comunicación asíncrona.

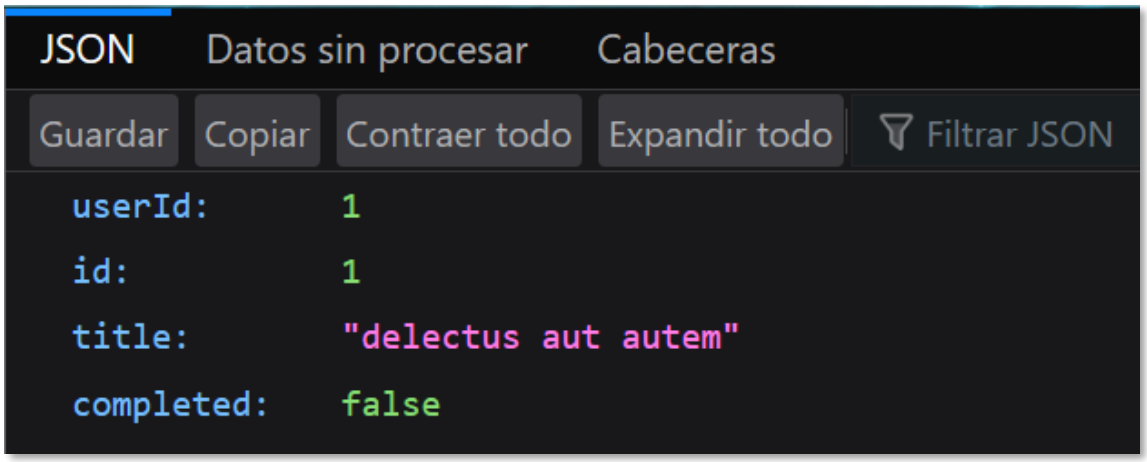
Es importante recalcar que Android establece unos modos con unas clases para estas comunicaciones y en versiones posteriores las deja de usar sacando otros modos más eficientes. El usar una forma u otra debe depender tanto de las propias necesidades de la aplicación y el entorno donde se ejecute, como la versión de API requerida para el desarrollo de las aplicaciones usando las clases apropiadas.

9.4 Ejemplos

Veamos ahora una serie de ejemplos que nos permitan conocer las instrucciones para la conexión remota a servicios API y la posterior recepción de datos para su tratamiento.

Ejemplo con servidor API público

Comencemos con un ejemplo sencillo. Consistirá en una aplicación que se conecte a la url https://jsonplaceholder.typicode.com/todos/1 para obtener información que luego mostraremos en pantalla. La información que envía este servidor API con la anterior petición de prueba es:



Comenzamos un proyecto nuevo llamado "ApiSimple" tipo "Empty Views Activity".

Lo primero que haremos será incorporar la librería **OkHttp** en la sección de dependencias del build.gradle de módulo:

```
dependencies
{
    implementation libs.appcompat
    implementation libs.material
    implementation libs.activity
    implementation libs.constraintlayout
    testImplementation libs.junit
    androidTestImplementation libs.ext.junit
    androidTestImplementation libs.espresso.core
    implementation 'com.squareup.okhttp3:okhttp:3.0.1'
}
```

No olvidemos de "Sincronizar"...

Lo siguiente será dar permisos de acceso a Internet para poder conectarnos con esta API. Esto lo haremos en el **Manifest** del proyecto:

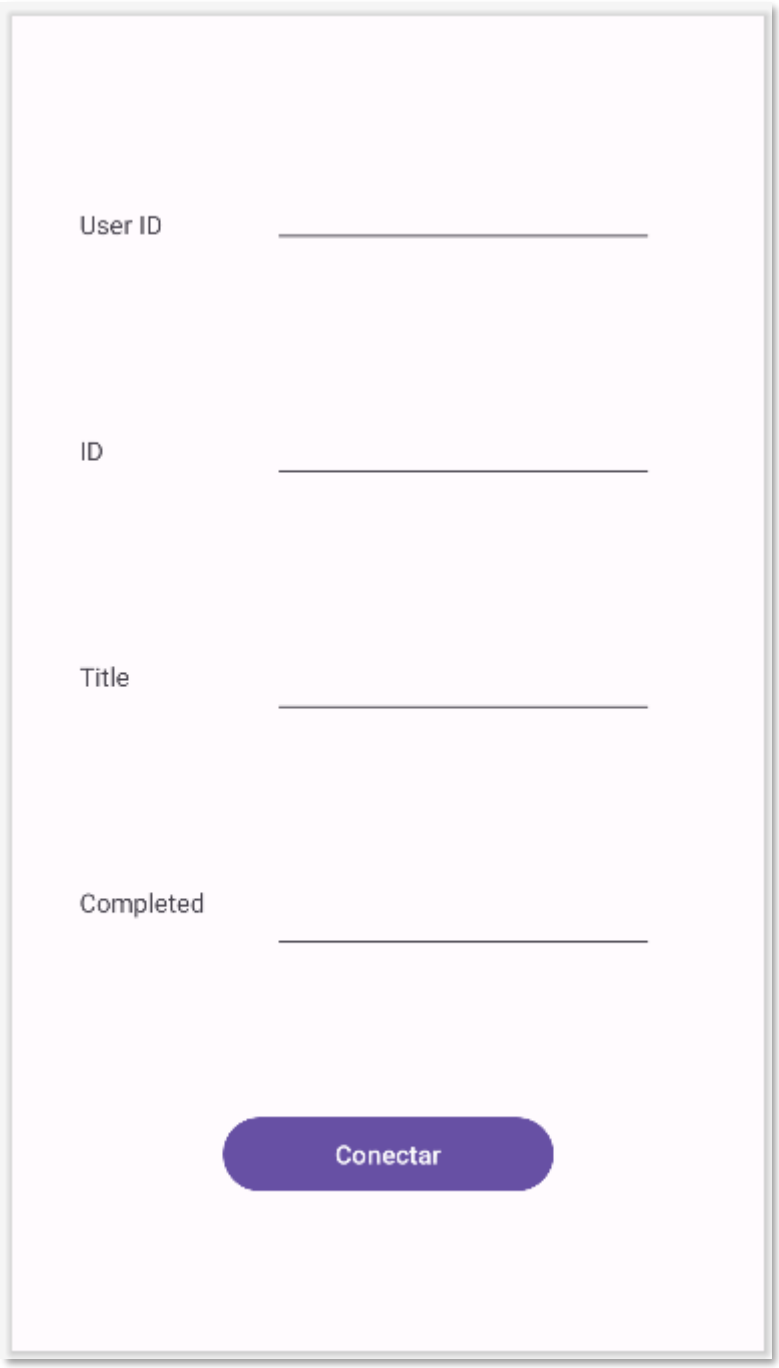
```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools">

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

    <application
        android:allowBackup="true"
        android:dataExtractionRules="@xml/data_extraction_rules"
        android:fullBackupContent="@xml/backup_rules"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.ApiSimple"
        tools:targetApi="31"
        android:usesCleartextTraffic="true">
        <activity
            android:name=".MainActivity"
            android:exported="true">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

La instrucción de la línea 19 nos permite el intercambio de información entre el cliente (Nuestra App) y el servidor API **sin encriptar**. No se recomienda que tenga valor “true”, pero para estas pruebas, no importa demasiado. De todos modos, la conexión a esta API lleva cierta protección en forma de encriptación de la información que se intercambia (https).

Creamos unas etiquetas en strings.xml:

```
<resources>
   <string name="app_name">ApiSimple</string>
   <string name="etiquetaUserId">User ID</string>
   <string name="etiquetaId">ID</string>
   <string name="etiquetaTitle">Title</string>
   <string name="etiquetaCompleted">Completed</string>
   <string name="etiquetaBoton">Conectar</string>
</resources>
```

Nuestro activity_main.xml puede tener un diseño tal que así:



Cuatro etiquetas, cuatro cuadros de texto y un botón. La idea es que cuando se pulse el botón, la aplicación haga una petición al API y muestre los datos obtenidos como respuesta.

En código sería algo similar a esto:

```
<?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:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <Button
        android:id="@+id/button"
        android:layout_width="181dp"
        android:layout_height="48dp"
        android:text="@string/etiquetaBoton"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/editTextCompleted" />

    <EditText
        android:id="@+id/editTextTitle"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:ems="10"
        android:inputType="textPersonName"
        app:layout_constraintBottom_toTopOf="@+id/editTextCompleted"
        app:layout_constraintStart_toEndOf="@+id/textView2"
        app:layout_constraintStart_toStartOf="@+id/editTextId"
        app:layout_constraintTop_toBottomOf="@+id/editTextId" />

    <EditText
        android:id="@+id/editTextUserId"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:ems="10"
        android:inputType="textPersonName"
        app:layout_constraintBottom_toTopOf="@+id/editTextId"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.504"
        app:layout_constraintStart_toEndOf="@+id/textView4"
        app:layout_constraintTop_toTopOf="parent" />

    <TextView
        android:id="@+id/textView4"
        android:layout_width="wrap_content"
```

```
    android:layout_height="wrap_content"
    android:text="@string/etiquetaUserId"
    app:layout_constraintBottom_toTopOf="@+id/textView"
    app:layout_constraintStart_toStartOf="@+id/textView"
    app:layout_constraintTop_toTopOf="parent" />

<TextView
    android:id="@+id/textView3"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/etiquetaCompleted"
    app:layout_constraintBottom_toTopOf="@+id/button"
    app:layout_constraintEnd_toStartOf="@+id/editTextCompleted"
    app:layout_constraintHorizontal_bias="0.5"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/textView2" />

<EditText
    android:id="@+id/editTextCompleted"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:ems="10"
    android:inputType="textPersonName"
    app:layout_constraintBottom_toTopOf="@+id/button"
    app:layout_constraintStart_toEndOf="@+id/textView3"
    app:layout_constraintStart_toStartOf="@+id/editTextTitle"
    app:layout_constraintTop_toBottomOf="@+id/editTextTitle" />

<TextView
    android:id="@+id/textView2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/etiquetaTitle"
    app:layout_constraintBottom_toTopOf="@+id/textView3"
    app:layout_constraintStart_toStartOf="@+id/textView3"
    app:layout_constraintTop_toBottomOf="@+id/textView" />

<EditText
    android:id="@+id/editTextId"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:ems="10"
    android:inputType="textPersonName"
    app:layout_constraintBottom_toTopOf="@+id/editTextTitle"
    app:layout_constraintStart_toEndOf="@+id/textView"
    app:layout_constraintStart_toStartOf="@+id/editTextUserId"
    app:layout_constraintTop_toBottomOf="@+id/editTextUserId" />

<TextView
    android:id="@+id/textView"
    android:layout_width="wrap_content"
```

```
    android:layout_height="wrap_content"
    android:text="@string/etiquetaId"
    app:layout_constraintBottom_toTopOf="@+id/textView2"
    app:layout_constraintStart_toStartOf="@+id/textView2"
    app:layout_constraintTop_toBottomOf="@+id/textView4" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

Todo el código del acceso remoto lo implementaremos en una clase aparte de la principal, una llamada "ConsultaRemota" con el siguiente código:

```
package es.studium.apisimple;

import android.util.Log;
import android.widget.TextView;
import org.json.JSONException;
import org.json.JSONObject;
import java.io.IOException;
import okhttp3.OkHttpClient;
import okhttp3.Request;
import okhttp3.Response;

public class ConsultaRemota
{
    JSONObject result;
    String userId = "";
    String id = "";
    String title = "";
    String completed = "";
    // Crear una instancia de OkHttpClient
    OkHttpClient client = new OkHttpClient();
    Request request = new Request.Builder()
            .url("https://jsonplaceholder.typicode.com/todos/1")
            .build();

    ConsultaRemota(TextView txtUserId,TextView txtId,TextView
txtTitle,TextView txtCompleted)
    {
        try
        {
            // Realizar la solicitud
            Response response = client.newCall(request).execute();
            // Procesar la respuesta
            if (response.isSuccessful())
            {
                result = new JSONObject(response.body().string());
                userId = result.getString("userId");
                id = result.getString("id");
                title = result.getString("title");
                completed = result.getString("completed");
                txtUserId.setText(userId);
                txtId.setText(id);
                txtTitle.setText(title);
                txtCompleted.setText(completed);
            }
```

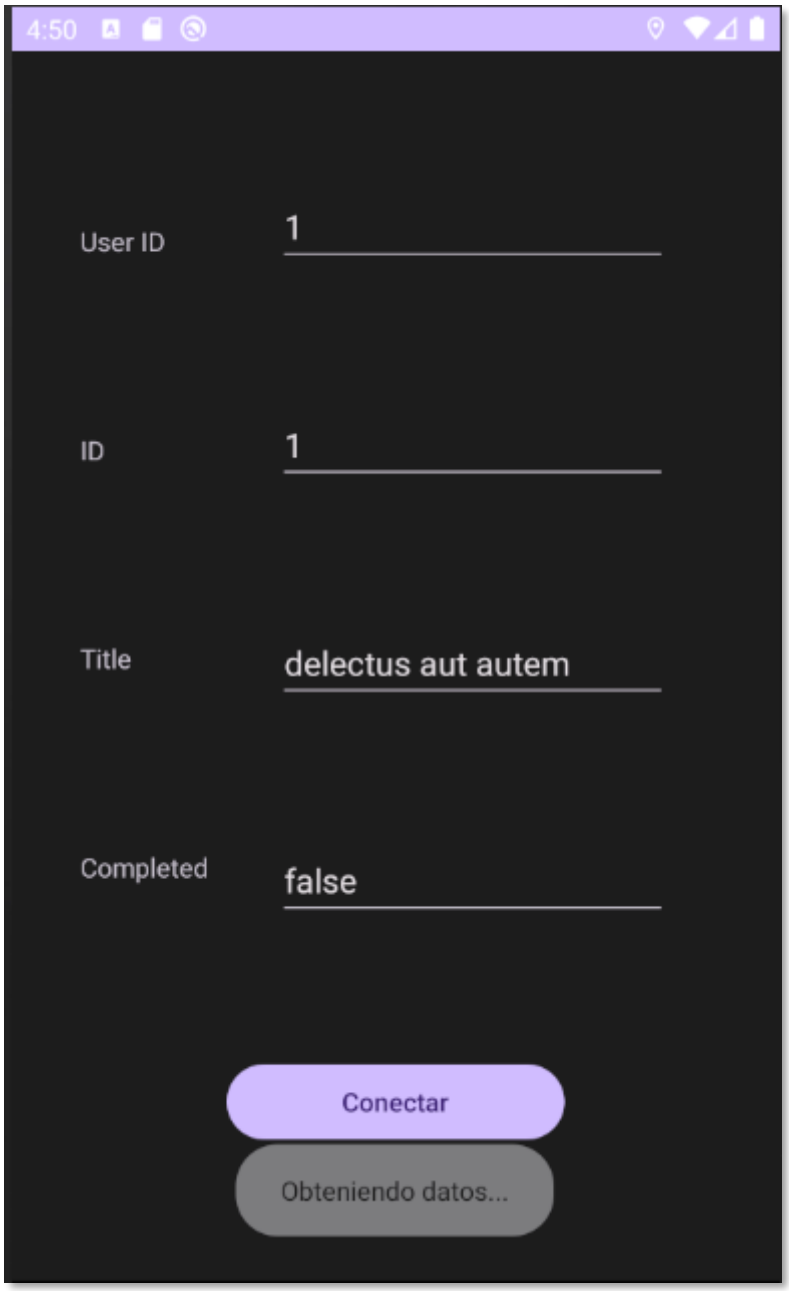
```
            else
            {
                Log.e("MainActivity", response.message());
            }
        }
        catch (IOException e)
        {
            Log.e("MainActivity", e.getMessage());
        }
        catch (JSONException e)
        {
            throw new RuntimeException(e);
        }
    }
}
```

Por último, el MainActivity.java quedaría de la siguiente forma:

```
package es.studium.apisimple;

import android.os.Bundle;
import android.os.StrictMode;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;
import androidx.appcompat.app.AppCompatActivity;

public class MainActivity extends AppCompatActivity implements
View.OnClickListener
{
    Button btnConectar;
    TextView txtUserId;
    TextView txtId;
    TextView txtTitle;
    TextView txtCompleted;

    @Override
    protected void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        btnConectar = findViewById(R.id.button);
        btnConectar.setOnClickListener(this);

        txtUserId = findViewById(R.id.editTextUserId);
        txtId = findViewById(R.id.editTextId);
        txtTitle = findViewById(R.id.editTextTitle);
        txtCompleted = findViewById(R.id.editTextCompleted);

        // Definir comportamiento de Hilos optimizados
        if (android.os.Build.VERSION.SDK_INT > 9)
        {
            StrictMode.ThreadPolicy policy = new
StrictMode.ThreadPolicy.Builder().permitAll().build();
```

```
                StrictMode.setThreadPolicy(policy);
            }
        }

        @Override
        public void onClick(View v)
        {
            new ConsultaRemota(txtUserId, txtId, txtTitle, txtCompleted);
Toast.makeText(this, "Obteniendo datos...", Toast.LENGTH_SHORT).show();
        }
}
```

Ya podemos ejecutar y pulsar el botón de "Conectar" para obtener los datos del API...



Ejemplo con servidor API propio

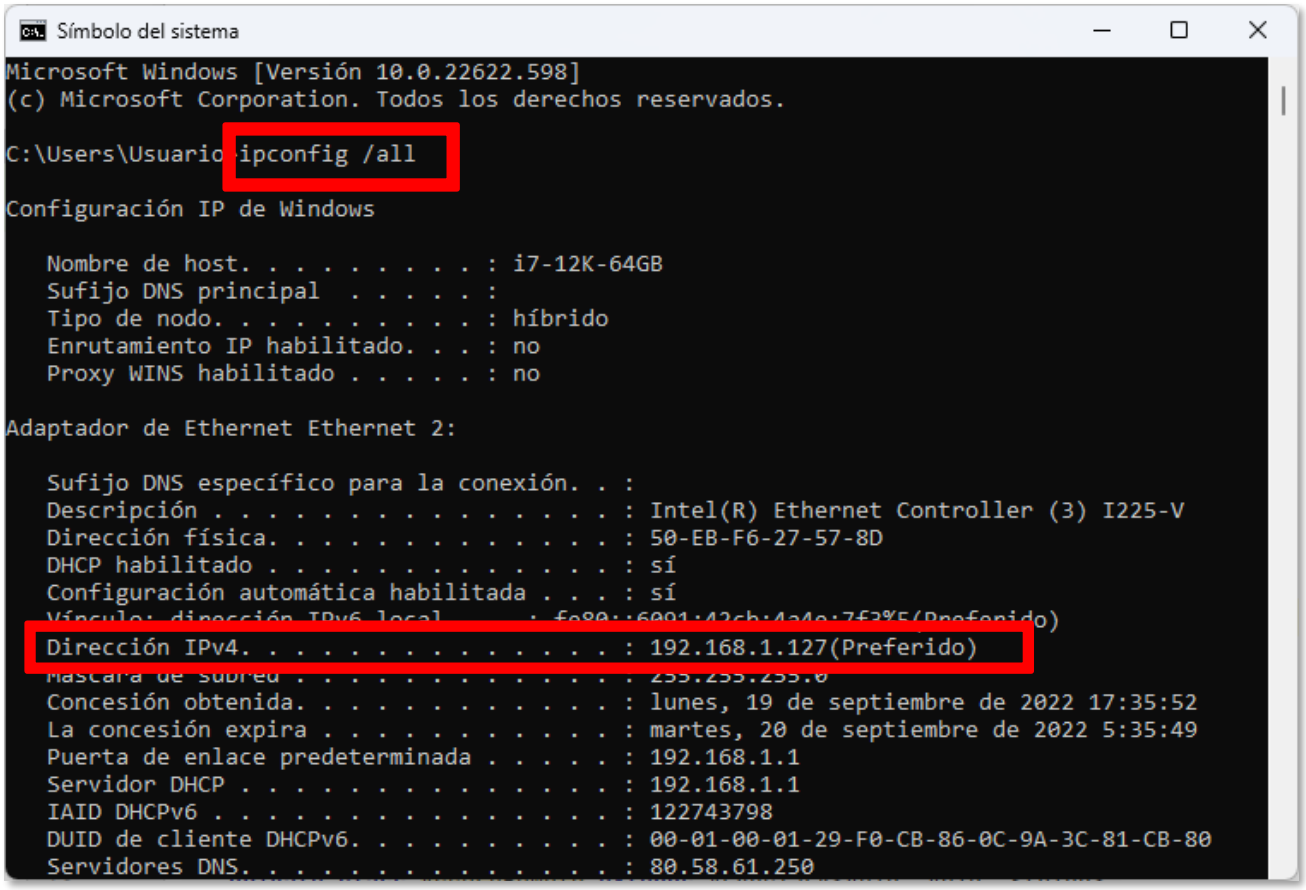
A menos que se especifique un API concreto, el que usaremos para estos ejemplos lo vamos a montar nosotros mismos. Consistirán en un equipo donde tendremos montado un **MySQL Server** que contendrá la base de datos con la información con la que trabajarán nuestras Apps de ejemplo. Por otro lado, un **servidor web Apache** soportará la API desarrollada en PHP.

Tooling

Instalando la aplicación **XAMPP**, o similar, podemos tener de una sola vez los elementos requeridos: Servidor MySQL y Servidor Web con soporte para PHP.

Otro elemento importante para tener en cuenta será la IP del equipo que soporte este sistema, pues será necesaria suministrarla a la hora de conectar nuestra App con la API.

Para obtenerla, simplemente abrimos una consola de Windows con el comando cmd y ejecutamos la instrucción ipconfig /all tal como se aprecia en la siguiente imagen:



Anotamos la IP para indicarla en la aplicación Android y ahora nos disponemos a crear la **base de datos** que va a contener la información que necesita nuestra

aplicación y con la que interactuará para hacer Altas, Bajas, Modificaciones y Consultas.

Base de Datos

Se trata de la base de datos "agenda" con una tabla llamada "contactos" con un par de registros, que podemos crear con las siguientes instrucciones:

```
CREATE DATABASE agenda CHARSET utf8mb4 COLLATE utf8mb4_spanish2_ci;
USE agenda;
CREATE TABLE contactos (idContacto INT AUTO_INCREMENT, nombreContacto VARCHAR(45), apellidosContacto VARCHAR(45), telefonoContacto VARCHAR(9), PRIMARY KEY(idContacto));
INSERT INTO contactos VALUES(null, 'Elon', 'Musk', '654654654'),(null, 'Steve', 'Jobs', '555666777');
```

Así dispondremos al principio de datos para mostrar al hacer una consulta simple.

También debemos disponer de un usuario de MySQL con los suficientes privilegios para poder interactuar con la base de datos. Siempre tendremos al usuario "root" pero no es recomendable trabajar con él, así que crearemos uno llamado "rest":

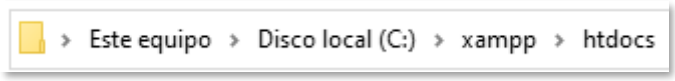
```
CREATE USER 'rest'@'localhost' IDENTIFIED BY 'Studium2020;';
GRANT ALL ON agenda.* TO 'rest'@'localhost';
FLUSH PRIVILEGES;
```

NOTA: Si nuestro sistema MySQL Server usara el tipo de validación de claves Legacy, sería mejor crear el usuario así:

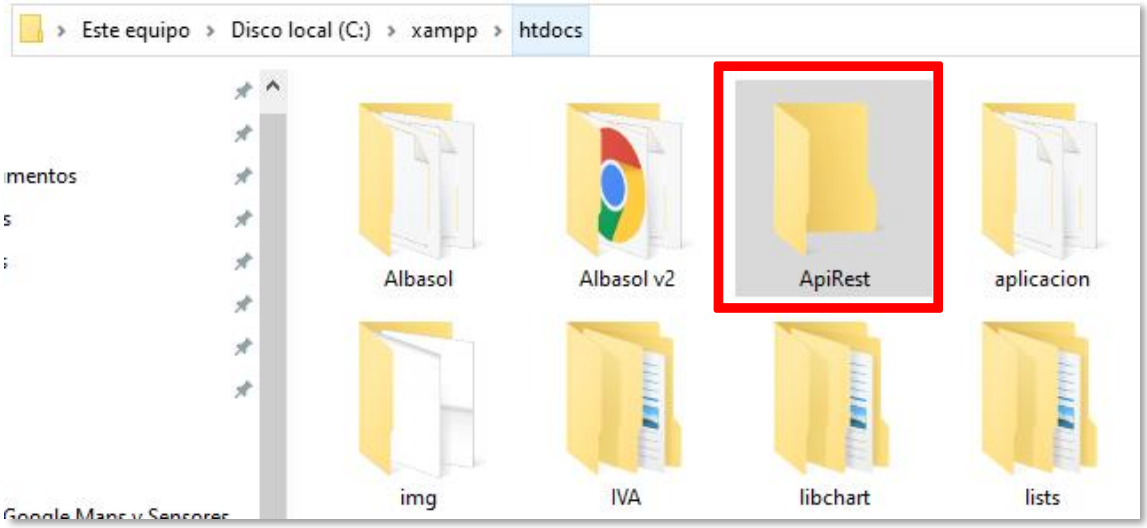
```
CREATE USER 'rest' IDENTIFIED WITH mysql_native_password BY 'Studium2020;';
```

Construyendo el API

Una vez instalado el XAMPP, se nos crea en el disco de destino, una carpeta llamada */xampp* dentro de la cual encontraremos la carpeta de proyectos denominada */htdocs*.



En dicha carpeta */htdocs* crearemos otra carpeta llamada ApiRest donde incluiremos todos los ficheros necesarios del Api:



En dicha carpeta debemos incluir tres ficheros PHP:

- config.php, que contendrá los datos básicos de conexión a la base de datos, porque recordemos que nuestra App no se conectará directamente a la base de datos, sino que lo hará a través de esta API, es decir, nuestro programa en PHP será el que realmente se conecte a la base de datos y haga de intermediario entre ambos.
- utils.php, que contendrá alguna que otra función que simplifica el trabajo
- contactos.php, que contiene la funcionalidad del API

El fichero config.php tendrá el siguiente contenido:

```php
<?php
$db = [
    'host' => 'localhost',
    'username' => 'rest',
    'password' => 'Studium2020;',
    'db' => 'agenda'
];
?>
```

Aquí debemos destacar la inclusión de la IP del Host, es decir, del equipo al que se conectará nuestra App. En los ejemplos que estamos haciendo, la palabra "localhost" representará al equipo en el que está el mismo fichero. Si estuviéramos conectando a otro servidor MySQL, simplemente cambiar dicha palabra por la IP o URL oportunos.

El fichero utils.php contiene una serie de funciones que nos simplifican la conexión e intercambio de información entre el API y la Base de Datos.

```php
<?php
  // Abrir conexion a la base de datos
  function connect($db)
  {
      try
      {
          $conn = new PDO("mysql:host={$db['host']};dbname={$db['db']};charset=utf8",
           $db['username'], $db['password']);
          // Set the PDO error mode to exception
          $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
          return $conn;
      }
      catch (PDOException $exception)
      {
          exit($exception->getMessage());
      }
  }
  // Obtener parametros para updates
  function getParams($input)
  {
     $filterParams = [];
     foreach($input as $param => $value)
     {
          $filterParams[] = "$param=:$param";
     }
     return implode(", ", $filterParams);
  }
  // Asociar todos los parametros a un sql
  function bindAllValues($statement, $params)
  {
     foreach($params as $param => $value)
     {
         $statement->bindValue(':'.$param, $value);
     }
     return $statement;
  }
?>
```

Y, por último, el fichero que constituye el API propiamente dicho, el contactos.php:

```php
<?php
include "config.php";
include "utils.php";
$dbConn = connect($db);
/*
  Listar todos los registros o solo uno
 */
if ($_SERVER['REQUEST_METHOD'] == 'GET')
{
    if (isset($_GET['idContacto']))
    {
      //Mostrar un contacto
      $sql = $dbConn->prepare("SELECT * FROM contactos WHERE idContacto=:idContacto");
      $sql->bindValue(':idContacto', $_GET['idContacto']);
      $sql->execute();
      header("HTTP/1.1 200 OK");
      echo json_encode($sql->fetch(PDO::FETCH_ASSOC));
      exit();
    }
    else
    {
      //Mostrar lista de contactos
      $sql = $dbConn->prepare("SELECT * FROM contactos");
      $sql->execute();
      $sql->setFetchMode(PDO::FETCH_ASSOC);
      header("HTTP/1.1 200 OK");
      echo json_encode($sql->fetchAll());
      exit();
  }
}
```

```php
// Crear un nuevo contacto
if ($_SERVER['REQUEST_METHOD'] == 'POST')
{
    $input = $_POST;
    $sql = "INSERT INTO contactos VALUES (null,'".$_POST['nombreContacto']."', '".
    $_POST['apellidosContacto']."', '".$_POST['telefonoContacto']."')";
    $statement = $dbConn->prepare($sql);
    $statement->execute();
    $postId = $dbConn->lastInsertId();
    if($postId)
    {
      $input['idContacto'] = $postId;
      header("HTTP/1.1 200 OK");
      echo json_encode($input);
      exit();
   }
}
// Borrar
if ($_SERVER['REQUEST_METHOD'] == 'DELETE')
{
  $idContacto = $_GET['idContacto'];
  $statement = $dbConn->prepare("DELETE FROM contactos WHERE idContacto=:idContacto"
  );
  $statement->bindValue(':idContacto', $idContacto);
  $statement->execute();
  header("HTTP/1.1 200 OK");
  exit();
}
```

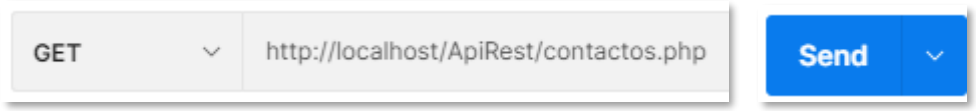
```php
// Actualizar
if ($_SERVER['REQUEST_METHOD'] == 'PUT')
{
  $sql = "UPDATE contactos SET nombreContacto='".$_GET['nombreContacto'].
  "',apellidosContacto='".$_GET['apellidosContacto']."',telefonoContacto='".$_GET[
  'telefonoContacto']."' WHERE idContacto=".$_GET['idContacto'];
  $statement = $dbConn->prepare($sql);
  $statement->execute();
  header("HTTP/1.1 200 OK");
  exit();
}
// En caso de que ninguna de las opciones anteriores se haya ejecutado
header("HTTP/1.1 400 Bad Request");
?>
```

Este fichero contactos.php, lo primero que hace es conectarse a la base de datos con las credenciales indicadas en config.php. Mediante sentencias condicionales, averigua qué tipo de petición le llega (GET, POST, PUT o DELETE) para realizar la tarea deseada y responder en consecuencia de lo establecido. Finalmente, en caso de recibir una petición no reconocida por la API, se devuelve una respuesta de **Error**.

Pruebas del API

Probemos el API con **Postman**. Hay que asegurarse que el servidor web (Apache) está operativo.

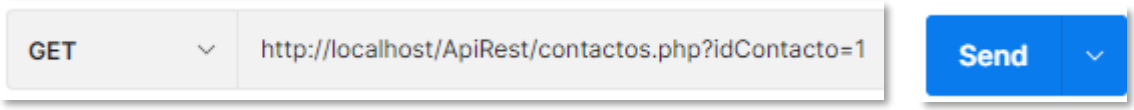
Una vez arrancada la aplicación, lanzamos una consulta a todos los datos:

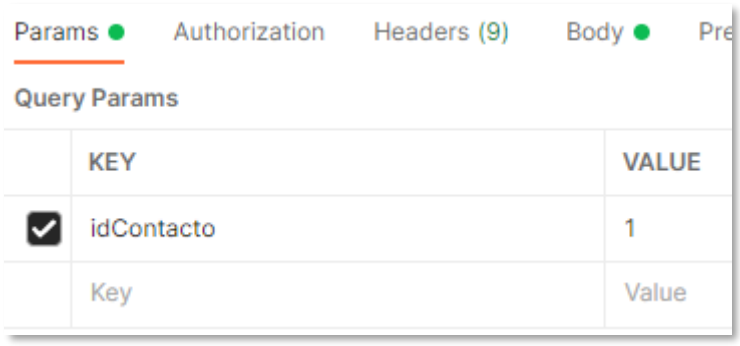


Y obtenemos:

[{"idContacto":"1","nombreContacto":"Elon","apellidosContacto":"Musk","telefonoContacto":"654654654"},{"idContacto":"2","nombreContacto":"Steve","apellidosContacto":"Jobs","telefonoContacto":"555666777"}]

Si enviamos un parámetro idContacto con valor 1, por ejemplo, deberíamos obtener solamente el contacto con idContacto = 1 de la siguiente manera:

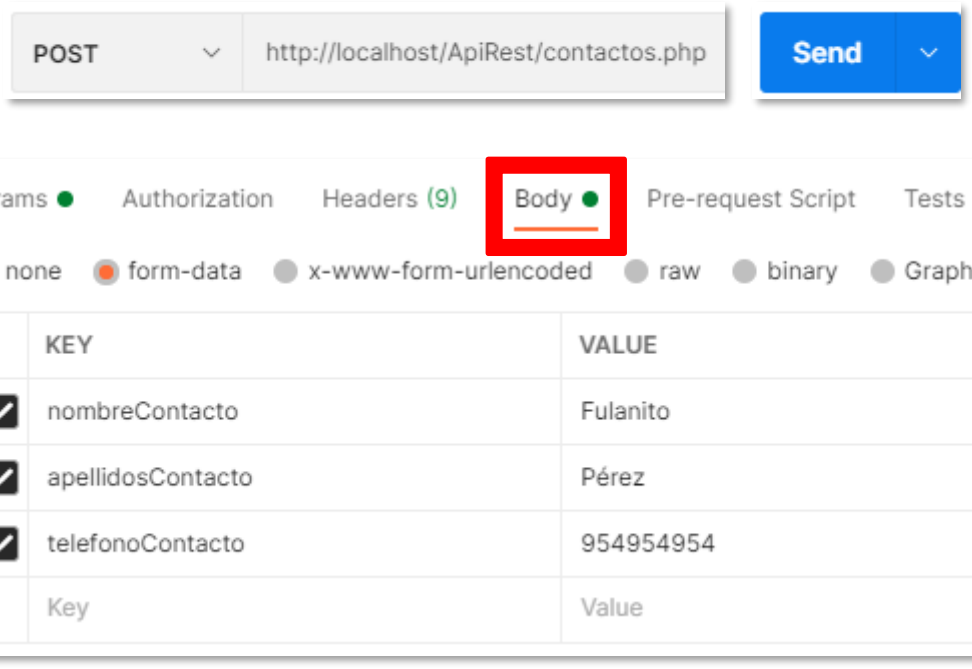




Obteniendo:

{"idContacto":"1","nombreContacto":"Elon","apellidosContacto":"Musk","telefonoContacto":"654654654"}

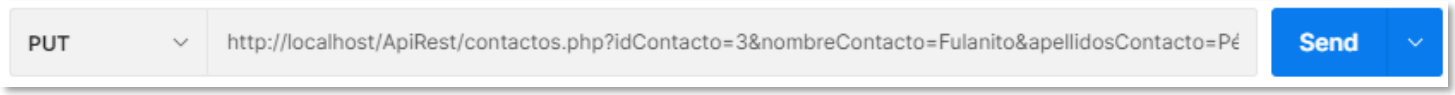
En la siguiente prueba, intentaremos dar de alta un nuevo contacto. Para ello, al API le tenemos que enviar en una petición POST los valores del alta (nombre, apellidos y número de teléfono). Para ello se lo enviaremos en el apartado de "Body" tal como se aprecia en la imagen:

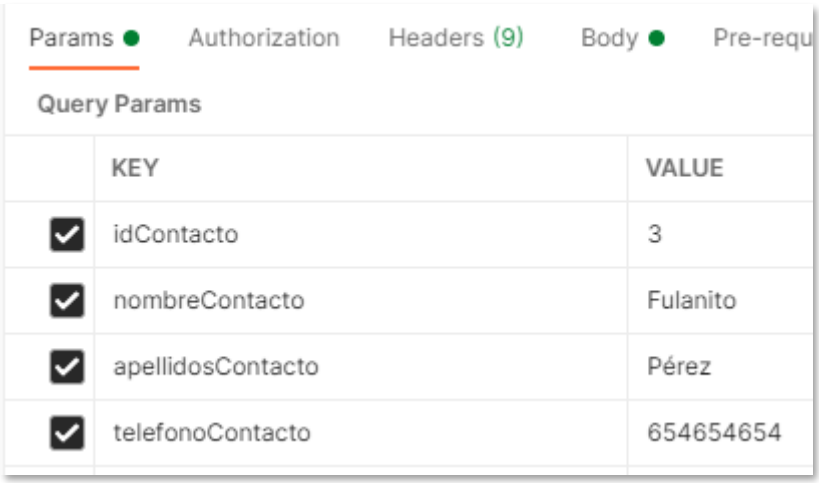


Como resultado de esta operación, si todo ha ido como debe, obtendremos el registro recién insertado, incluyendo el valor del idContacto, de tipo auto numérico generado por MySQL Server:

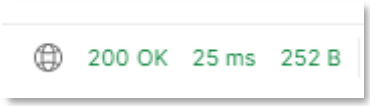
{"nombreContacto":"Fulanito","apellidosContacto":"P\u00e9rez","telefonoContacto":"954954954","idContacto":"3"}

Para hacer una modificación o petición PUT, los valores los debemos dar como parámetros GET, sin olvidar el idContacto que queremos modificar. Modificaremos el teléfono del recién creado contacto:



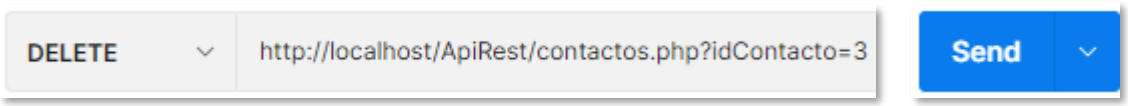


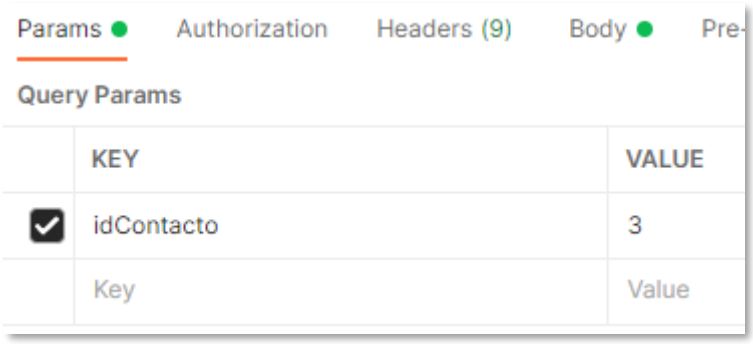
En este caso, no se devuelve nada excepto el Query OK:



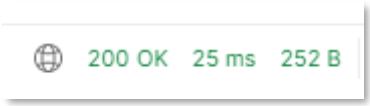
En la base de datos podemos comprobar el cambio realizado.

Por último, probaremos a borrar (Petición DELETE) este registro recién modificado:





Asimismo, solamente obtendremos el 200 OK:



Pero si nos vamos a la base de datos podremos comprobar que ya no existe tal registro.

App

Y ahora nos toca realizar el proyecto en Android, así que comenzamos creando uno nuevo tipo "Empty Views Activity" al que llamaremos "AccesoApi".

Esta aplicación se centrará en los aspectos de código para el acceso a la base de datos antes descrita a través del uso del servicio API para realizar las típicas tareas CRUD.

Lo primero que debemos hacer es incluir las dependencias para el uso de la librería OkHttp. En **build.gradle** de módulo:

```
dependencies {
    implementation libs.appcompat
    implementation libs.material
    implementation libs.activity
    implementation libs.constraintlayout
    testImplementation libs.junit
    androidTestImplementation libs.ext.junit
    androidTestImplementation libs.espresso.core
    implementation 'com.squareup.okhttp3:okhttp:3.0.1'
}
```

No podemos olvidar "Sincronizar".

Lo siguiente será dar permisos para acceso remoto a través de Internet y permitir intercambio sin cifrar en el AndroidManifest.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools">

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

    <application
        android:allowBackup="true"
        android:dataExtractionRules="@xml/data_extraction_rules"
        android:fullBackupContent="@xml/backup_rules"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.AccesoApi"
        tools:targetApi="31"
        android:usesCleartextTraffic="true">
        <activity
            android:name=".MainActivity"
            android:exported="true">
```

```
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

A continuación, el fichero de recursos strings.xml:

```
<resources>
    <string name="app_name">AccesoApi</string>
    <string name="lblId">id</string>
    <string name="lblNombre">Nombre</string>
    <string name="lblApellidos">Apellidos</string>
    <string name="lblTelefono">Teléfono</string>
    <string name="lblAnterior">Anterior</string>
    <string name="lblSiguiente">Siguiente</string>
    <string name="lblNuevo">Nuevo</string>
    <string name="lblEliminar">Eliminar</string>
    <string name="lblModificar">Modificar</string>
    <string name="lblAceptar">Aceptar</string>
</resources>
```

Seguidamente, la pantalla principal, es decir, 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/main"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/lblId"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintHorizontal_bias="0.179"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_bias="0.073" />

    <TextView
        android:id="@+id/textView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/lblNombre"
        app:layout_constraintBottom_toBottomOf="parent"
```

```
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.169"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_bias="0.153" />

    <TextView
        android:id="@+id/textView2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/lblApellidos"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.169"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_bias="0.235" />

    <TextView
        android:id="@+id/textView3"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/lblTelefono"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.169"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_bias="0.327" />

    <EditText
        android:id="@+id/editTextId"
        android:layout_width="212dp"
        android:layout_height="41dp"
        android:ems="10"
        android:focusable="false"
        android:inputType="none"
        android:textSize="18sp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.809"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_bias="0.057" />

    <EditText
        android:id="@+id/editTextNombre"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:ems="10"
        android:focusable="false"
        android:inputType="none"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.8"
        app:layout_constraintStart_toStartOf="parent"
```

```
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_bias="0.138" />

    <EditText
        android:id="@+id/editTextApellidos"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:ems="10"
        android:focusable="false"
        android:inputType="none"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.8"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_bias="0.224" />

    <EditText
        android:id="@+id/editTextTelefono"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:ems="10"
        android:focusable="false"
        android:inputType="none"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.81"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_bias="0.319" />

    <Button
        android:id="@+id/btnAnterior"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/lblAnterior"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.078"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_bias="0.43" />

    <Button
        android:id="@+id/btnSiguiente"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/lblSiguiente"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.949"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_bias="0.43" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

Que en modo "Diseño" quedaría algo así:

De momento, solamente conectaremos al API para solicitar (GET) los datos de la tabla *contactos* para mostrar el primer registro en los cuadros de texto oportunos de forma automática.

Añadimos la clase "AccesoRemoto" con el siguiente código[1]:

```
package es.studium.accesoapi;

import android.util.Log;
import org.json.JSONArray;
import org.json.JSONException;
import java.io.IOException;
import okhttp3.OkHttpClient;
import okhttp3.Request;
import okhttp3.Response;

public class AccesoRemoto
{
    // Crear una instancia de OkHttpClient
    OkHttpClient client = new OkHttpClient();
    Request request = new Request.Builder()
            .url("http://192.168.0.18/ApiRest/contactos.php")
            .build();

    AccesoRemoto(){} // Constructor

    public JSONArray obtenerListado()
    {
        JSONArray resultado = new JSONArray();
        try
```

[1] Cuidado con la URL del servidor. Hay que poner la del equipo anfitrión.

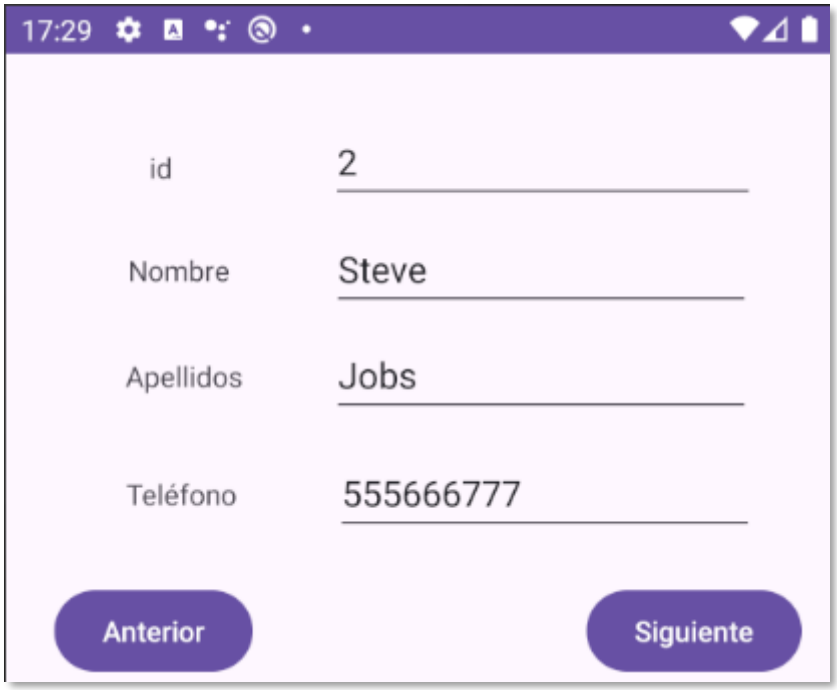
```
        {
            // Realizar la solicitud
            Response response = client.newCall(request).execute();
            // Procesar la respuesta
            if (response.isSuccessful())
            {
                resultado = new JSONArray(response.body().string());
            }
            else
            {
                Log.e("AccesoRemoto", response.message());
            }
        }
        catch (IOException e)
        {
            Log.e("AccesoRemoto", e.getMessage());
        }
        catch (JSONException e)
        {
            throw new RuntimeException(e);
        }
        return resultado;
    }
}
```

Por último, en MainActivity.java tendremos algo similar a lo que vimos en un ejemplo anterior:

```
package es.studium.accesoapi;

import android.os.Bundle;
import android.os.StrictMode;
import android.widget.TextView;
import androidx.appcompat.app.AppCompatActivity;
import org.json.JSONArray;
import org.json.JSONException;
import org.json.JSONObject;

public class MainActivity extends AppCompatActivity
{
    TextView txtId;
    TextView txtNombre;
    TextView txtApellidos;
    TextView txtTelefono;

    JSONArray result;
    JSONObject jsonobject;
    String idContacto = "";
    String nombreContacto = "";
    String apellidosContacto = "";
    String telefonoContacto = "";

    @Override
    protected void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        txtId = findViewById(R.id.editTextId);
        txtNombre = findViewById(R.id.editTextNombre);
        txtApellidos = findViewById(R.id.editTextApellidos);
        txtTelefono = findViewById(R.id.editTextTelefono);
        if (android.os.Build.VERSION.SDK_INT > 9)
        {
            StrictMode.ThreadPolicy policy = new
StrictMode.ThreadPolicy.Builder().permitAll().build();
            StrictMode.setThreadPolicy(policy);
        }
        AccesoRemoto accesoRemoto = new AccesoRemoto();
        result = accesoRemoto.obtenerListado();
        try
        {
            jsonobject = result.getJSONObject(0); // Obtenemos el Primer
registro
            idContacto = jsonobject.getString("idContacto");
            nombreContacto = jsonobject.getString("nombreContacto");
            apellidosContacto = jsonobject.getString("apellidosContacto");
            telefonoContacto = jsonobject.getString("telefonoContacto");
        }
        catch (JSONException e)
        {
```

```
                throw new RuntimeException(e);
            }
            txtId.setText(idContacto);
            txtNombre.setText(nombreContacto);
            txtApellidos.setText(apellidosContacto);
            txtTelefono.setText(telefonoContacto);
        }
}
```

Al ejecutar obtendremos lo siguiente:



Ahora daremos funcionalidad a los botones que aparecen en la aplicación, los que se "mueven" entre los registros, es decir, "Anterior" y "Siguiente" modificando simplemente el MainActivity.java:

```
package es.studium.accesoapi;

import android.os.Bundle;
import android.os.StrictMode;
import android.view.View;
import android.widget.TextView;
import androidx.appcompat.app.AppCompatActivity;
import org.json.JSONArray;
import org.json.JSONException;
import org.json.JSONObject;
import android.widget.Button;

public class MainActivity extends AppCompatActivity implements
View.OnClickListener {
    TextView txtId;
    TextView txtNombre;
    TextView txtApellidos;
    TextView txtTelefono;

    JSONArray result;
    JSONObject jsonobject;
    String idContacto = "";
    String nombreContacto = "";
    String apellidosContacto = "";
    String telefonoContacto = "";

    Button btnAnterior;
    Button btnSiguiente;
    int posicion = 0;

    @Override
    protected void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        txtId = findViewById(R.id.editTextId);
        txtNombre = findViewById(R.id.editTextNombre);
        txtApellidos = findViewById(R.id.editTextApellidos);
        txtTelefono = findViewById(R.id.editTextTelefono);

        btnAnterior = findViewById(R.id.btnAnterior);
        btnAnterior.setOnClickListener(this);
        btnSiguiente = findViewById(R.id.btnSiguiente);
        btnSiguiente.setOnClickListener(this);

        if (android.os.Build.VERSION.SDK_INT > 9)
        {
            StrictMode.ThreadPolicy policy = new
StrictMode.ThreadPolicy.Builder().permitAll().build();
            StrictMode.setThreadPolicy(policy);
```

```
        }
        AccesoRemoto accesoRemoto = new AccesoRemoto();
        result = accesoRemoto.obtenerListado();
        try
        {
            jsonobject = result.getJSONObject(0); // Obtenemos el
Primer registro
            idContacto = jsonobject.getString("idContacto");
            nombreContacto = jsonobject.getString("nombreContacto");
            apellidosContacto =
jsonobject.getString("apellidosContacto");
            telefonoContacto =
jsonobject.getString("telefonoContacto");
        }
        catch (JSONException e)
        {
            throw new RuntimeException(e);
        }
        txtId.setText(idContacto);
        txtNombre.setText(nombreContacto);
        txtApellidos.setText(apellidosContacto);
        txtTelefono.setText(telefonoContacto);
    }

    @Override
    public void onClick(View v)
    {
        if(v.getId() == R.id.btnAnterior)
        {
            if(posicion>0)
            {
                posicion--;
                try
                {
                    jsonobject = result.getJSONObject(posicion);
                    //Sacamos dato a dato obtenido
                    idContacto = jsonobject.getString("idContacto");
                    nombreContacto =
jsonobject.getString("nombreContacto");
                    apellidosContacto =
jsonobject.getString("apellidosContacto");
                    telefonoContacto =
jsonobject.getString("telefonoContacto");
                    // Actualizamos los cuadros de texto
                    txtId.setText(idContacto);
                    txtNombre.setText(nombreContacto);
                    txtApellidos.setText(apellidosContacto);
                    txtTelefono.setText(telefonoContacto);
                }
                catch (JSONException e)
                {
                    e.printStackTrace();
                }
            }
        }
        else if(v.getId() == R.id.btnSiguiente)
```

```
        {
            if(posicion<result.length()-1)
            {
                posicion++;
                try
                {
                    jsonobject = result.getJSONObject(posicion);
                    //Sacamos dato a dato obtenido
                    idContacto = jsonobject.getString("idContacto");
                    nombreContacto =
jsonobject.getString("nombreContacto");
                    apellidosContacto =
jsonobject.getString("apellidosContacto");
                    telefonoContacto =
jsonobject.getString("telefonoContacto");
                    // Actualizamos los cuadros de texto
                    txtId.setText(idContacto);
                    txtNombre.setText(nombreContacto);
                    txtApellidos.setText(apellidosContacto);
                    txtTelefono.setText(telefonoContacto);
                }
                catch (JSONException e)
                {
                    e.printStackTrace();
                }
            }
        }
    }
}
```

NOTAS: Se han incluido las siguientes partes de código:

1. implements View.OnClickListener
2. Dos Button y un int
3. Los Listeners correspondientes a los botones
4. Todo el método redefinido onClick()

Por último, se incluye todo el código para las clases para el Alta, para la Baja y para la Modificación.

Clase AltaRemota.java:

```
package es.studium.accesoapi;

import android.util.Log;
import java.io.IOException;

import okhttp3.Call;
import okhttp3.FormBody;
import okhttp3.OkHttpClient;
import okhttp3.Request;
import okhttp3.RequestBody;
import okhttp3.Response;

public class AltaRemota
{
    OkHttpClient client = new OkHttpClient();

    public AltaRemota(){};
    boolean darAlta(String nombre, String apellidos, int telefono)
    {
        boolean correcta = true;
        // Montamos la petición POST
        RequestBody formBody = new FormBody.Builder()
                .add("nombreContacto", nombre)
                .add("apellidosContacto", apellidos)
                .add("telefonoContacto", telefono+"")
                .build();
        Request request = new Request.Builder()
                .url("http://192.168.0.18/ApiRest/contactos.php")
                .post(formBody)
                .build();

        Call call = client.newCall(request);
        try
        {
            Response response = call.execute();
            Log.i("AltaRemota", String.valueOf(response));
            correcta = true;
        }
        catch (IOException e)
        {
            Log.e("AltaRemota", e.getMessage());
            correcta = false;
        }
        return correcta;
    }
}
```

Clase BajaRemota.java:

```
package es.studium.accesoapi;

import android.util.Log;
import java.io.IOException;

import okhttp3.Call;
import okhttp3.OkHttpClient;
import okhttp3.Request;
import okhttp3.Response;

public class BajaRemota
{
    OkHttpClient client = new OkHttpClient();

    BajaRemota() {}

    boolean darBaja(int id)
    {
        boolean correcta = true;
        Request request = new Request.Builder()

.url("http://192.168.0.18/ApiRest/contactos.php?idContacto="+id)
                .delete()
                .build();
        Call call = client.newCall(request);
        try
        {
            Response response = call.execute();
            Log.i("BajaRemota", String.valueOf(response));
            correcta = true;
        }
        catch (IOException e)
        {
            Log.e("BajaRemota", e.getMessage());
            correcta = false;
        }
        return correcta;
    }
}
```

Clase ModificacionRemota.java:

```
package es.studium.accesoapi;

import android.util.Log;
import java.io.IOException;
import java.net.URI;

import okhttp3.Call;
import okhttp3.FormBody;
import okhttp3.HttpUrl;
import okhttp3.OkHttpClient;
import okhttp3.Request;
import okhttp3.RequestBody;
import okhttp3.Response;

public class ModificacionRemota
{
    OkHttpClient client = new OkHttpClient();

    ModificacionRemota() {}

    boolean modificar(int id, String nombre, String apellidos, int
telefono)
    {
        boolean correcta = true;
        HttpUrl.Builder queryUrlBuilder =
HttpUrl.get(URI.create("http://192.168.0.18/ApiRest/contactos.php"))
.newBuilder();
        queryUrlBuilder.addQueryParameter("idContacto", id+"");
        queryUrlBuilder.addQueryParameter("nombreContacto", nombre);
        queryUrlBuilder.addQueryParameter("apellidosContacto",
apellidos);
        queryUrlBuilder.addQueryParameter("telefonoContacto",
telefono+"");

        // Las peticiones PUT requieren BODY, aunque sea vacío
        RequestBody formBody = new FormBody.Builder()
                .build();
        Log.i("ModificacionRemota", formBody.toString());
        Request request = new Request.Builder()
                .url(queryUrlBuilder.build())
                .put(formBody)
                .build();
        Log.i("ModificacionRemota", String.valueOf(request));
        Call call = client.newCall(request);
        try
        {
            Response response = call.execute();
            Log.i("ModificacionRemota", String.valueOf(response));
            correcta = true;
        }
        catch (IOException e)
        {
            Log.e("ModificacionRemota", e.getMessage());
            correcta = false;
```

```
        }
        return correcta;
    }
}
```

En el programa principal podríamos probar algo como:

```
AltaRemota altaRemota = new AltaRemota();
altaRemota.darAlta("Albert", "Einstein", 955955955);

BajaRemota bajaRemota = new BajaRemota();
bajaRemota.darBaja(26);

ModificacionRemota modificacionRemota = new ModificacionRemota();
modificacionRemota.modificar(37,"Albert", "Einstein", 955955988);
```

Pero lo suyo sería implementar un **menú** con el CRUD, diseñar pantallas donde hacer las **Altas**, las **Bajas** y las **Modificaciones**, mostrar **mensajes de feedback** indicando si las operaciones han sido correctas o no, …

¿Te atreves?

Resumen

Resumiendo, podríamos indicar los siguientes pasos:

1. Obtener IP del servidor, para usar en las URL
2. Montar la BD en MySQL
3. Crear el usuario de acceso a la base de datos
4. En el servidor web, crear un proyecto que albergue los ficheros que comprenden la API, en este caso en PHP, pero igualmente se podría hacer en JAVA, NodeJs, …
5. Probar con Postman el API
6. Realizar el proyecto en Android

Referencias

Develou. Com, en enlace.

{JSON} Placeholder, en enlace.

Android Developers, en enlace.

17/09/2024