

# CRUD usando MySQL, Express.js, Retrofit, Kotlin y Android Studio

Nota: La app se trabajo en servidor local XAMPP

Programas usados:

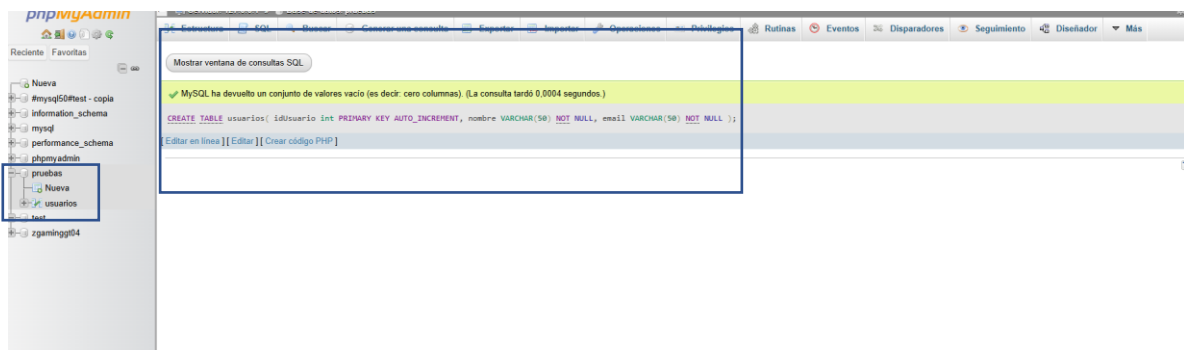
1. Visual studio code
2. Postman
3. Android Studio
4. Node
5. Xampp

**Paso 1.** Tenemos que tener instalado los programas anteriormente mencionados

**Paso2.** Creamos la base de datos en MySQL, para esto ocupamos el phpmyadmin que trae consigo xampp, la estructura de la base de datos debe quedar de la siguiente manera:

```
CREATE DATABASE pruebas;  
USE DATABASE pruebas;  
  
CREATE TABLE usuarios(  
    idUsuario int PRIMARY KEY AUTO_INCREMENT,  
    nombre VARCHAR(50) NOT NULL,  
    email VARCHAR(50) NOT NULL  
);
```

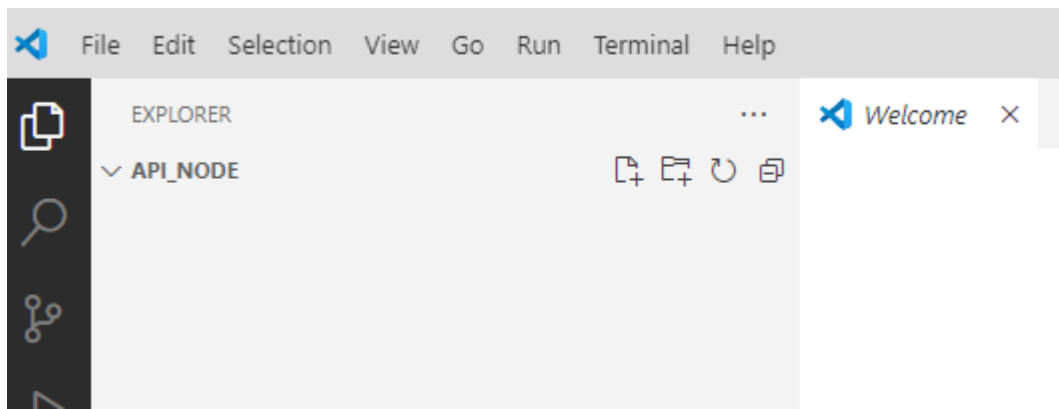
Debe quedar así:



**Paso 3.** En este caso se ha creado una carpeta que contendrá todos los archivos necesarios. Esta debe estar ubicada dentro de C:\xampp\htdocs\CRUD\_LABORATORIO\API\_NODE>

Nombre	Fecha de modificación	Tipo	Tamaño
API_NODE	8/6/2024 23:43	Carpeta de archivos	
BASE DE DATOS	8/6/2024 23:44	Carpeta de archivos	
PROYECTO_KOTLIN	8/6/2024 23:43	Carpeta de archivos	

**Paso 4.** Seguidamente procedemos a crear el api que vamos a usar, es decir el backend de nuestra aplicación, para ello debemos abrir la carpeta API\_NODE en visual studio code.



**Paso 5.** Como es primera vez debemos instalar todo lo necesario dentro de la API

Comandos: PS C:\xampp\htdocs\CRUD\_LABORATORIO\API\_NODE> npm init

Nos pedirá una serie de respuestas pero en este caso solo agregue el autor de la API

Una vez realizado se vera de la siguiente manera:

```

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

"author": "Patricia Flores",
"license": "ISC",
"description": ""
}

Is this OK? (yes) yes
npm notice
npm notice New minor version of npm available! 10.5.2 -> 10.8.1
npm notice Changelog: https://github.com/npm/cli/releases/tag/v10.8.1
npm notice Run npm install -g npm@10.8.1 to update!
npm notice
PS C:\xampp\htdocs\CRUD_LABORATORIO\API_NODE>

```

 A screenshot of the Visual Studio Code Explorer view. The 'API\_NODE' folder is expanded, and a file named 'package.json' is visible. The file icon is highlighted in red.

**Paso 6.** Seguidamente descargamos las dependencias que vamos a utilizar, para este caso sería express.js y MySQL.

Comando: PS C:\xampp\htdocs\CRUD\_LABORATORIO\API\_NODE> npm install express --save

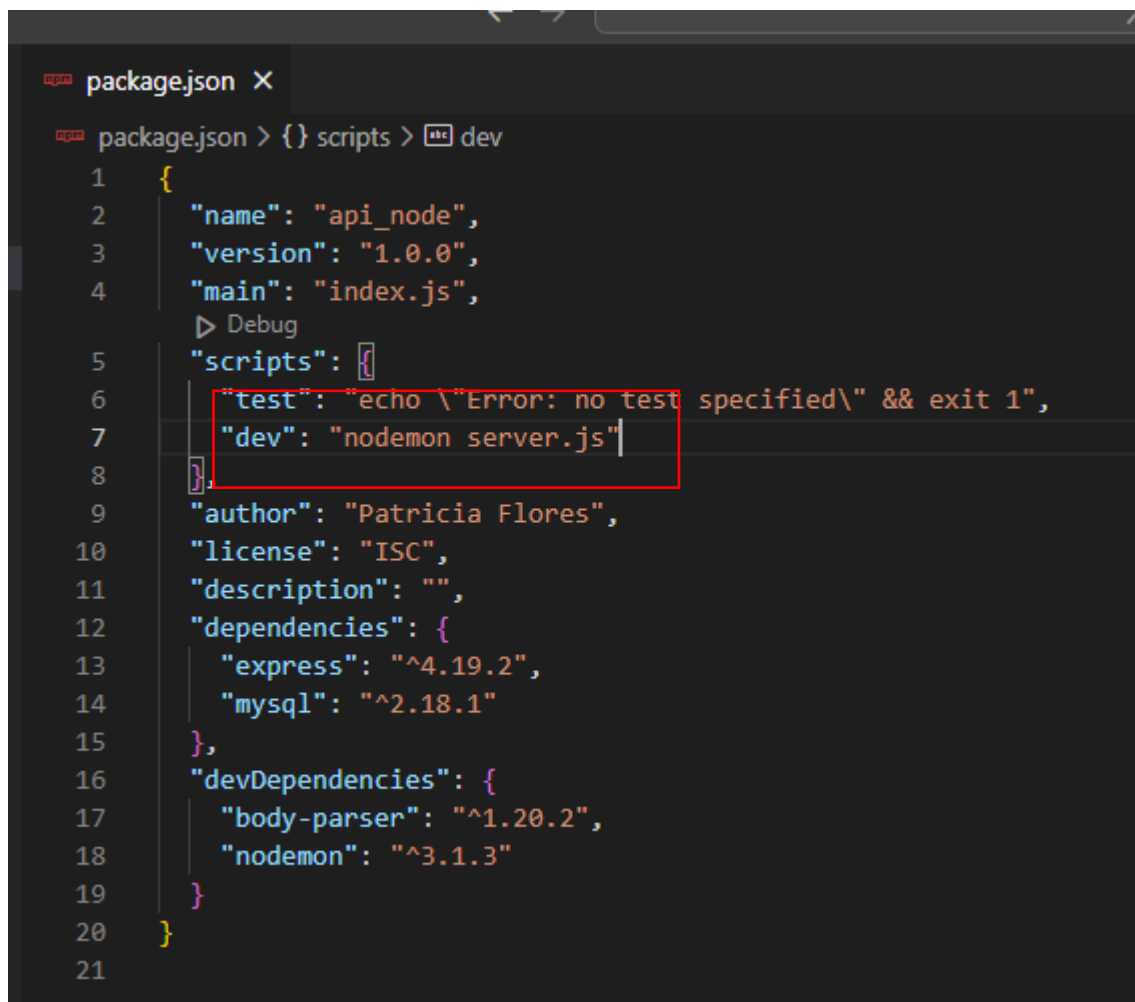
Comando: PS C:\xampp\htdocs\CRUD\_LABORATORIO\API\_NODE> npm install mysql --save

**Paso7.** Creamos las dependencias que vamos a usar para la parte del desarrollo.

PS C:\xampp\htdocs\CRUD\_LABORATORIO\API\_NODE> npm install body-parser --save-dev

PS C:\xampp\htdocs\CRUD\_LABORATORIO\API\_NODE> npm install nodemon --save-dev

**Paso 8.** Debemos crear un script para que pueda ejecutar el server con nodemon



```
package.json X
package.json > {} scripts > dev
1  {
2    "name": "api_node",
3    "version": "1.0.0",
4    "main": "index.js",
5    "scripts": {
6      "test": "echo \"Error: no test specified\" && exit 1",
7      "dev": "nodemon server.js"
8    },
9    "author": "Patricia Flores",
10   "license": "ISC",
11   "description": "",
12   "dependencies": {
13     "express": "^4.19.2",
14     "mysql": "^2.18.1"
15   },
16   "devDependencies": {
17     "body-parser": "^1.20.2",
18     "nodemon": "^3.1.3"
19   }
20 }
21
```

**Paso 9.** Creamos un archivo que se llame index.js

Una vez que tengamos listo el código debemos verificar si podemos levantar el servidor

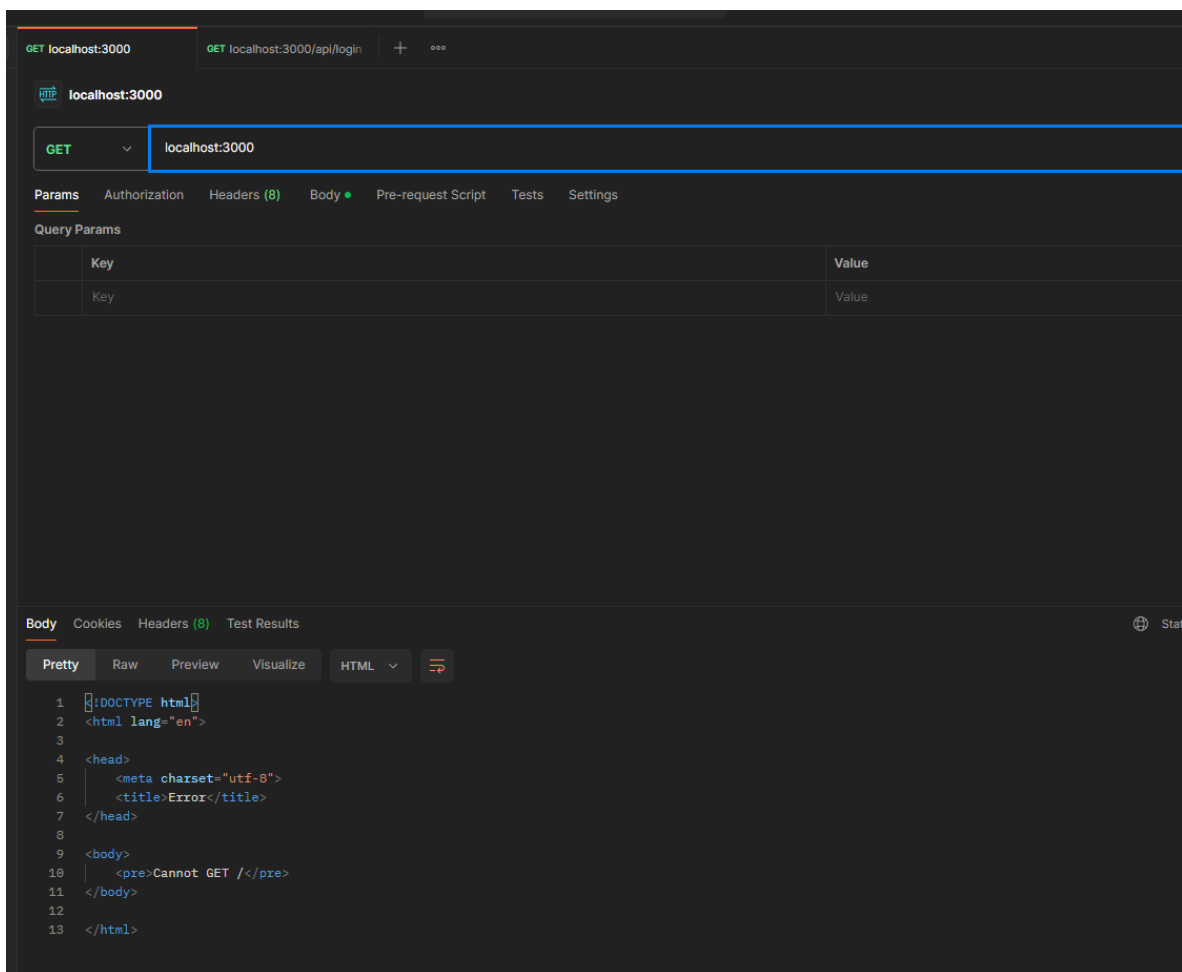
Comando: npm run dev

```
Found 0 vulnerabilities
PS C:\xampp\htdocs\CRUD_LABORATORIO\API_NODE> npm run dev

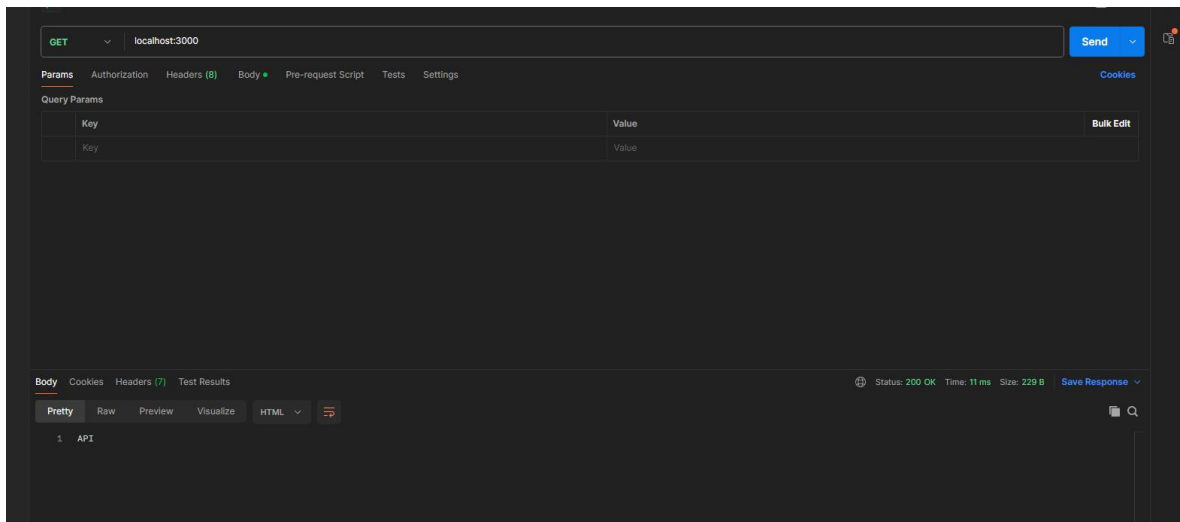
> api_node@1.0.0 dev
> nodemon server.js

[nodemon] 3.1.3
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,cjs,json
[nodemon] starting `node server.js index.js`
Servidor corriendo en el puerto 3000
```

Usamos postman para verificar que devuelve un GET



Una vez creada la raíz del backend debemos verificar que funcione, nuevamente usamos postman para verificar



**Paso 10.** Creamos el primer endpoint de listar los datos de la BD (GET)

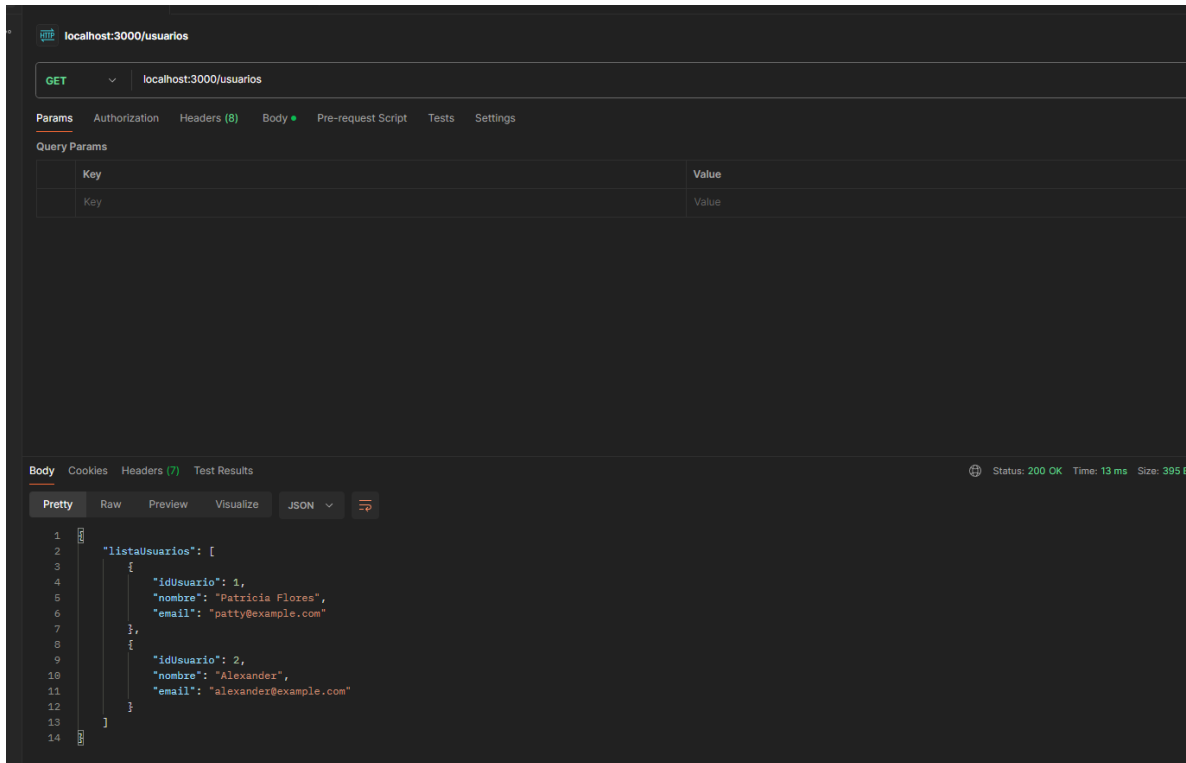
Código:

```
//PRIMER ENDPOINT -> CONSULTA A LA BASE DE DATOS GET
app.get('/usuarios', (req, res) => {

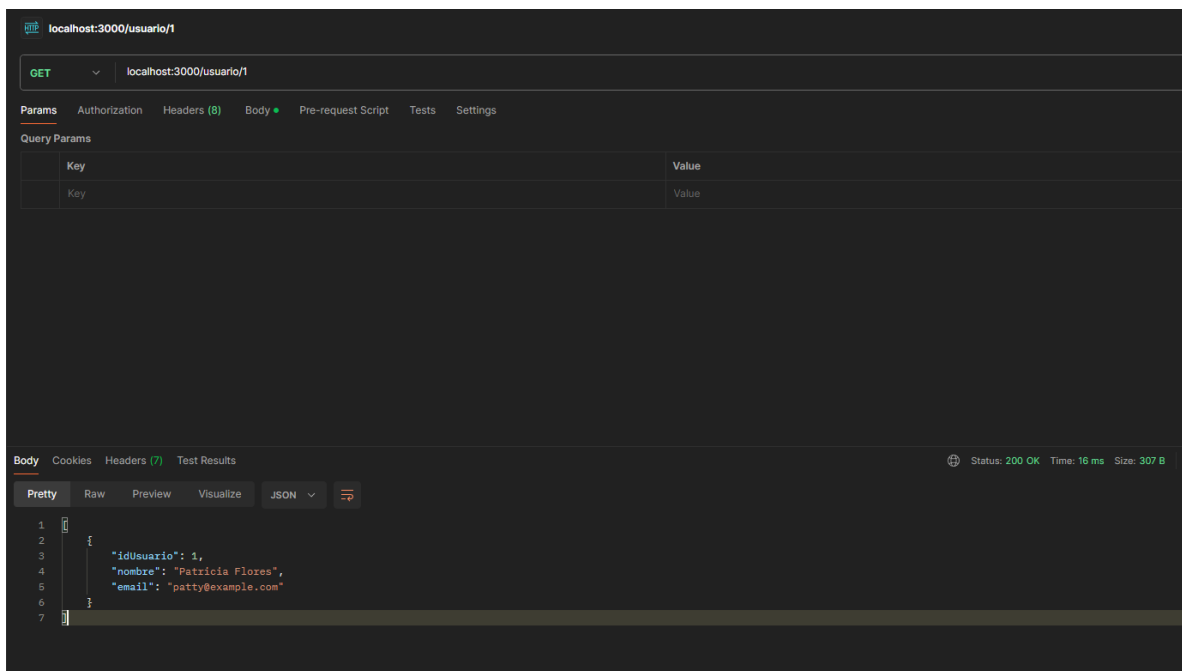
  const query = 'SELECT * FROM usuarios;'
  conexion.query(query, (error, resultado) => {
    if(error) return console.error(error.message)

    const obj = {}
    if(resultado.length > 0) {
      obj.listaUsuarios = resultado
      res.json(obj)
    } else {
      res.send('No hay registros')
    }
  })
})
})
```

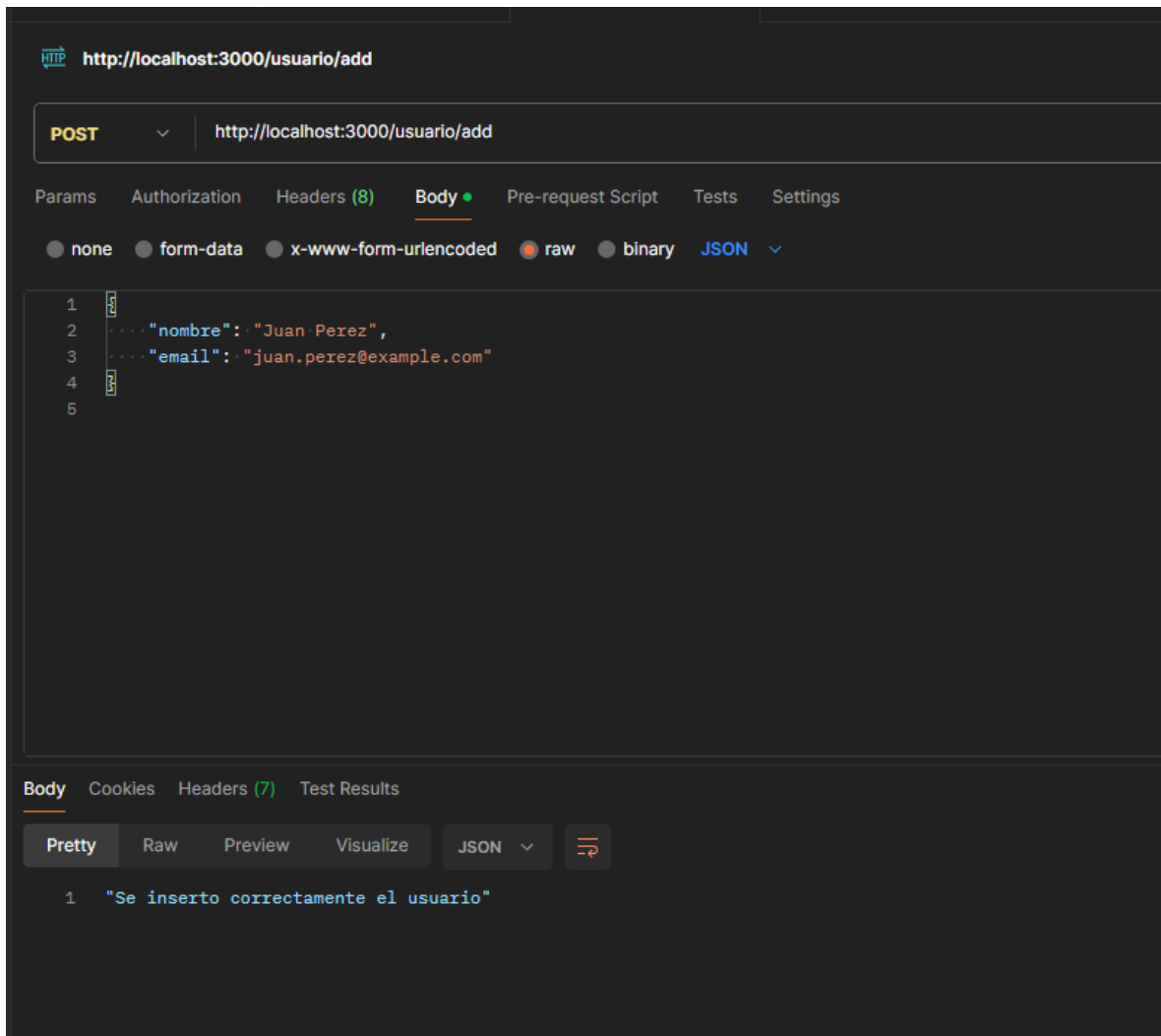
Nuestra BD se le agregaron dos registros, para verificar que funcionen haremos pruebas en el postman, debe verse así:



El segundo endpoint será de listar por id de usuario haremos una prueba y debe verse así:



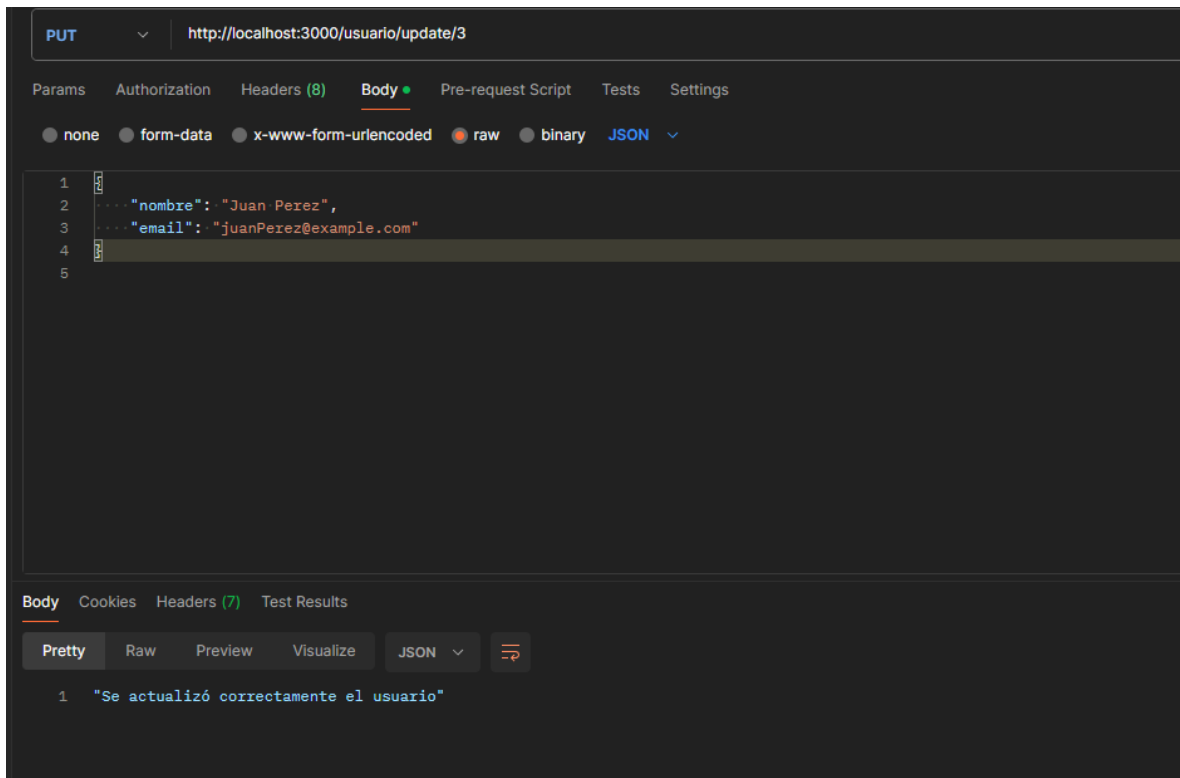
El tercer endpoint será de insertar y si comprobamos que funciona se debe ver asi



En la BD debe aparecer así:

				idUsuario	nombre	email
<input type="checkbox"/>		Editar		Copiar		Borrar
	1	Patricia Flores	patty@example.com			
<input type="checkbox"/>		Editar		Copiar		Borrar
	2	Alexander	alexander@example.com			
<input type="checkbox"/>		Editar		Copiar		Borrar
	3	Juan Perez	juan.perez@example.com			

El cuarto endpoint será el de actualizar, debemos corroborar que funcione debe ver así:

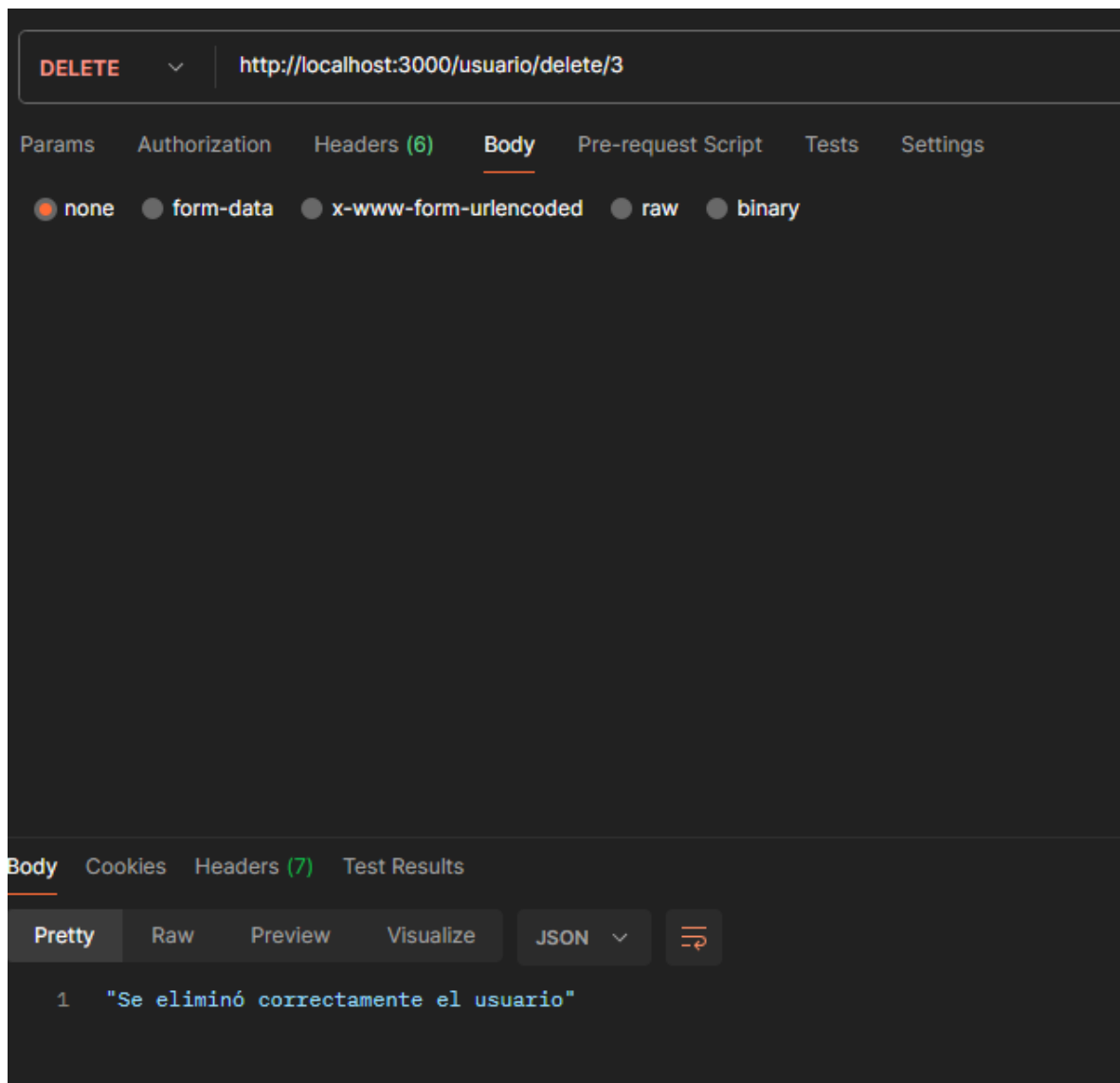


en la bd se debe reflejar:

		idUsuario	nombre	email
<input type="checkbox"/>	Editar	Copiar	Borrar	1 Patricia Flores patty@example.com
<input type="checkbox"/>	Editar	Copiar	Borrar	2 Alexander alexander@example.com
<input type="checkbox"/>	Editar	Copiar	Borrar	3 Juan Perez juanPerez@example.com

El quinto endpoint es de eliminar, para comprobar en postman se debe ver así:





En la bd se debe ver así:

	idUsuario	nombre	email
<input type="checkbox"/> Editar  Copiar  Borrar	1	Patricia Flores	patty@example.com
<input type="checkbox"/> Editar  Copiar  Borrar	2	Alexander	alexander@example.com

☐ Seleccionar todo Para los elementos que están marcados: Editar Copiar Borrar

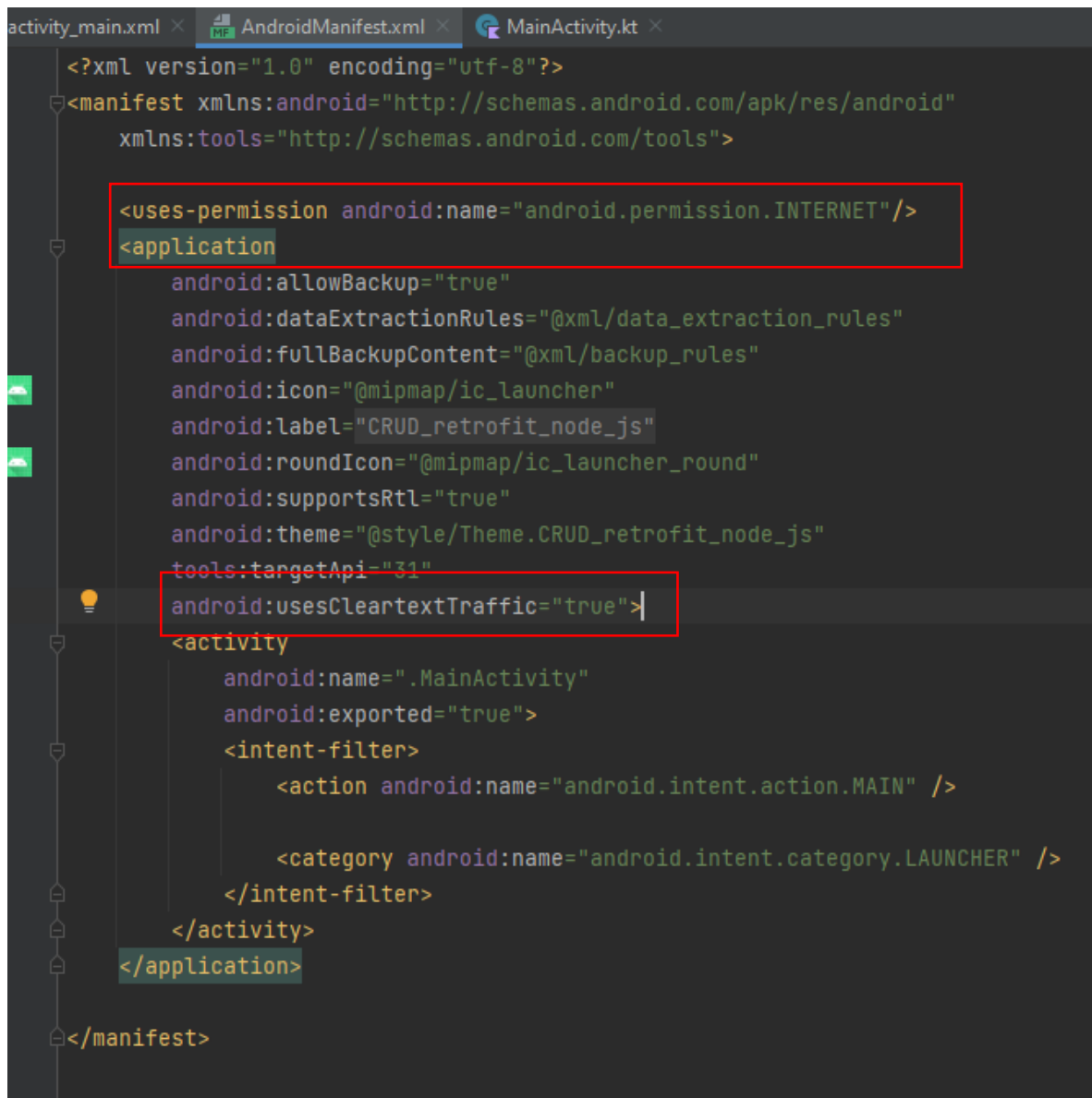
ESO ES TODO CON RESPECTO A LA API

## FRONTEND

**Paso 1.** Creamos un proyecto en Android studio vacío, en el gradle debemos agregar las dependencias de retrofit y la corutinas:

```
// Retrofit
implementation("com.squareup.retrofit2:retrofit:2.9.0")
// JSON Parsing
implementation("com.google.code.gson:gson:2.10.1")
implementation("com.squareup.retrofit2:converter-gson:2.9.0")
//Corutinas
implementation("org.jetbrains.kotlinx:kotlinx-coroutines-android:1.6.4")
```

**Paso 2.** Seguidamente nos dirigimos a nuestro AndroidManifest.xml, damos los permisos de internet y como todo será local debemos agregar que el tráfico este en true.



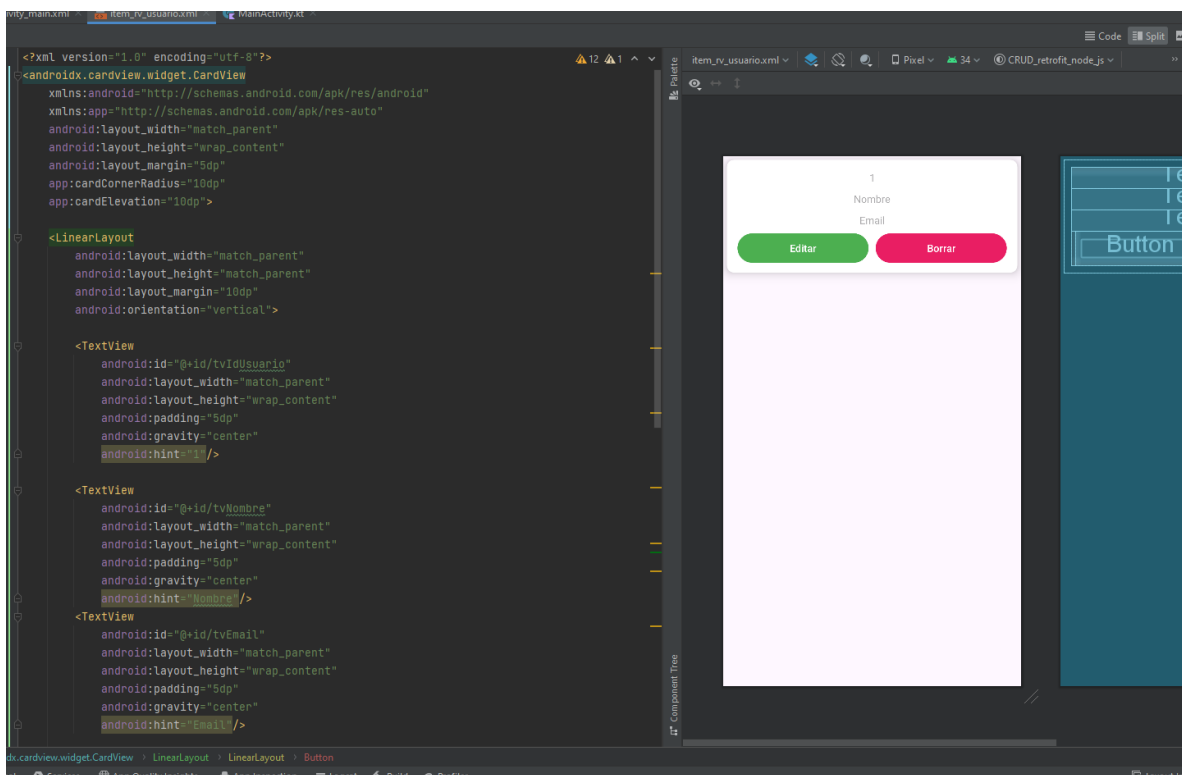
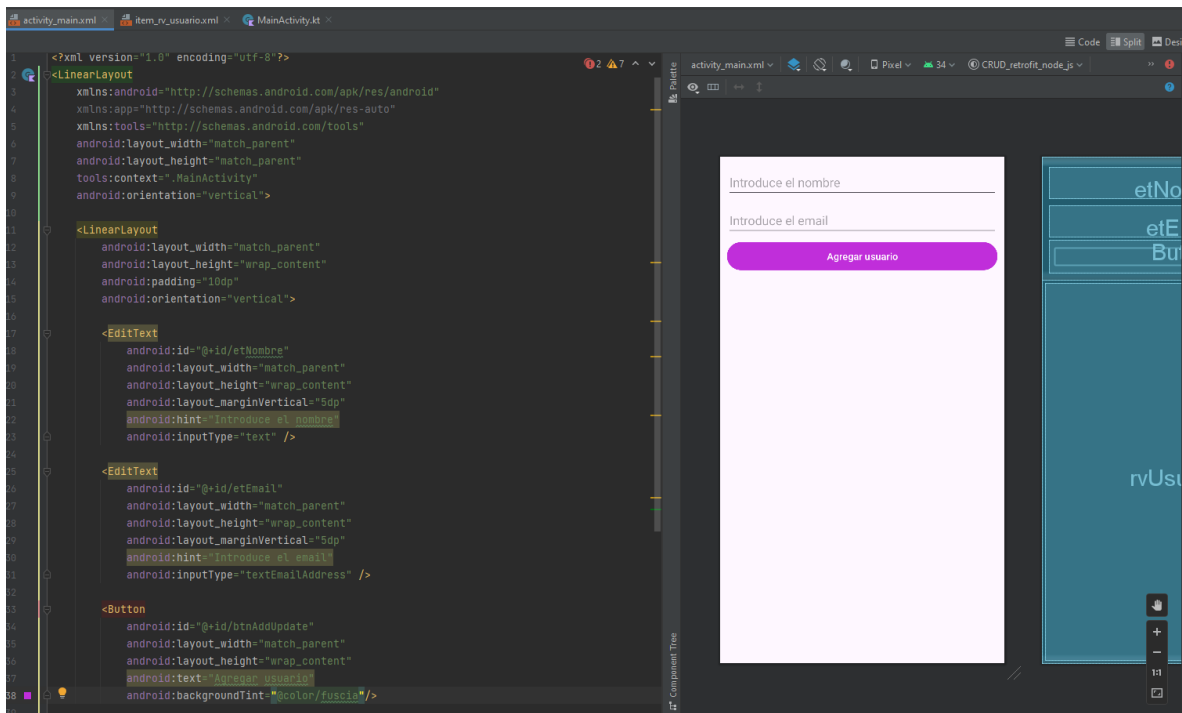
```
activity_main.xml x AndroidManifest.xml x MainActivity.kt x
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools">

    <uses-permission android:name="android.permission.INTERNET"/>
    <application
        android:allowBackup="true"
        android:dataExtractionRules="@xml/data_extraction_rules"
        android:fullBackupContent="@xml/backup_rules"
        android:icon="@mipmap/ic_launcher"
        android:label="CRUD_retrofit_node_js"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.CRUd_retrofit_node_js"
        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>
```

### Paso 3. Creamos nuestro diseño con xml

Quedando así:



### Paso 4. Creamos la interfaz para podernos conectar con la API

```
import retrofit2.http.Path
import retrofit2.Response
import com.google.gson.GsonBuilder

object AppConstantes {
    const val BASE_URL = "http://localhost:3000"
}

interface WebService {
    @GET("/usuarios")
    suspend fun obtenerUsuarios(): Response<UsuariosResponse>

    @GET("/usuario/{idUsuario}")
    suspend fun obtenerUsuario(
        @Path("idUsuario") idUsuario: Int
    ): Response<Usuario>

    @POST("/usuario/add")
    suspend fun agregarUsuario(
        @Body usuario: Usuario
    ): Response<String>

    @PUT("/usuario/update/{idUsuario}")
    suspend fun actualizarUsuario(
        @Path("idUsuario") idUsuario: Int,
        @Body usuario: Usuario
    ): Response<String>

    @DELETE("/usuario/delete/{idUsuario}")
    suspend fun borrarUsuario(
        @Path("idUsuario") idUsuario: Int
    ): Response<String>
}

object RetrofitClient {
    val webService: WebService by lazy {
        Retrofit.Builder()
            .baseUrl(AppConstantes.BASE_URL)
            .addConverterFactory(GsonConverterFactory.create(GsonBuilder().create()))
            .build().create(WebService::class.java)
    }
}
```

on Control ▶ Run ⚙️ TODO ⓘ Problems 📄 Terminal ⚙️ Services 💎 App Quality Insights 🐛 App Inspection 📋 Logcat 🏗️ Build 🔄 Pro

sync finished in 1 s 378 ms (today 08:47)

MXN 🔍 Buscar 📄 🏠

También debemos crear nuestra clase y el response

```
MainActivity.kt x WebService.kt x Usuario.kt x Usuar
package sv.edu.ues.crud_retrofit_node_js

data class Usuario(
    var idUsuario: Int,
    var nombre: String,
    var email: String
)
```

```
ainActivity.kt x WebService.kt x Usuario.kt x UsuariosResponse.kt x
package sv.edu.ues.crud_retrofit_node_js

import com.google.gson.annotations.SerializedName

data class UsuariosResponse (
    @SerializedName("listaUsuarios") var listaUsuarios: ArrayList<Usuario>
)
```

**Paso 5.** Implementamos la lógica en el main

```
MainActivity.kt | WebService.kt | Usuario.kt | UsuariosResponse.kt
1 package sv.edu.ves.crud_retrofit_node_js
2
3 import androidx.appcompat.app.AppCompatActivity
4 import android.os.Bundle
5 import android.widget.Toast
6 import androidx.recyclerview.widget.LinearLayoutManager
7 import kotlinx.coroutines.CoroutineScope
8 import kotlinx.coroutines.Dispatchers
9 import kotlinx.coroutines.launch
10 import sv.edu.ves.crud_retrofit_node_js.databinding.ActivityMainBinding
11
12 class MainActivity : AppCompatActivity(), UsuarioAdapter.OnItemClicked {
13
14     lateinit var binding: ActivityMainBinding
15
16     //Adaptador del recyclerView
17     lateinit var adaptador: UsuarioAdapter
18
19     var listaUsuarios = arrayListOf<Usuario>()
20
21
22     var usuario = Usuario( idUsuario: -1, nombre: "", email: "")
23
24     var isEditando = false
25
26     override fun onCreate(savedInstanceState: Bundle?) {
27         super.onCreate(savedInstanceState)
28         binding = ActivityMainBinding.inflate(layoutInflater)
29         setContentView(binding.root)
30
31         binding.rvUsuarios.layoutManager = LinearLayoutManager( context: this)
32         setupRecyclerView()
33
34         obtenerUsuarios()
35
36         binding.btnAddUpdate.setOnClickListener { it: View?
37             var isValidado = validarCampos()
38             if (isValidado) {
39                 if (!isEditando) {
```

**Paso 6.** Implementamos la lógica del recyclerView

```
MainActivity.kt x WebService.kt x Usuario.kt x UsuariosResponse.kt x UsuarioAdapter.kt x
package sv.edu.ues.crud_retrofit_node_js

import ...

class UsuarioAdapter(
    var context: Context,
    var listausuarios: ArrayList<Usuario>
): RecyclerView.Adapter<UsuarioAdapter.UsuarioViewHolder>() {

    private var onClick: OnItemClickListener? = null

    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): UsuarioViewHolder {
        val vista = LayoutInflater.from(parent.context).inflate(R.layout.item_rv_usuario, parent, attachToRoot = false)
        return UsuarioViewHolder(vista)
    }

    override fun onBindViewHolder(holder: UsuarioViewHolder, position: Int) {
        val usuario = listausuarios.get(position)

        holder.tvIdUsuario.text = usuario.idUsuario.toString()
        holder.tvNombre.text = usuario.nombre
        holder.tvEmail.text = usuario.email

        holder.btnEditar.setOnClickListener { it: View?
            onClick?.editarUsuario(usuario)
        }

        holder.btnBorrar.setOnClickListener { it: View?
            onClick?.borrarUsuario(usuario.idUsuario)
        }
    }

    override fun getItemCount(): Int {
        return listausuarios.size
    }

    inner class UsuarioViewHolder(itemView: View): RecyclerView.ViewHolder(itemView) {
        val tvIdUsuario = itemView.findViewById(R.id.tvIdUsuario) as TextView
        val tvNombre = itemView.findViewById(R.id.tvNombre) as TextView
        val tvEmail = itemView.findViewById(R.id.tvEmail) as TextView
        val btnEditar = itemView.findViewById(R.id.btnEditar) as Button
    }
}
```

**Paso 7.** Corremos

