



GUÍA PRÁCTICA

1. Datos Generales

Carrera:	Tecnología Superior en Desarrollo de Software
Período académico:	Diciembre 2021 – Abril 2022
Asignatura:	Desarrollo de Aplicaciones Móviles
Unidad N°:	4 Consumiendo Web Services, Acceso a Datos, Sincronización de Información
Tema:	Consumiendo WebServices
Ciclo-Paralelo:	M4A
Fecha de inicio de la Unidad:	15/02/2022
Fecha de fin de la Unidad	16/03/2022
Práctica N°:	4
Horas:	12
Docente:	Ing. Patricio Pacheco
Estudiante:	Jonnathan Gallegos

2. Contenido

2.1 Introducción

Esta guía práctica está centrada en cómo consumir una API o servicios web desde una aplicación Android.

Si por ejemplo tienes una base de datos, pero no tienes una API creada.

Entonces primero deberías definir una API.

Una API es un intermediario entre una base de datos y una aplicación móvil

2.2 Objetivo de la Guía

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

2.3 Materiales, herramientas, equipos y software

- Equipos de computación,
- Android Studio
- Internet,
- Material Guía (Talleres, ejercicios prácticos).

2.4 Procedimiento

1. Añadir la librería de Retrofit a nuestro proyecto.

Existen varias formas de añadir dependencias a nuestro proyecto.

En este caso usaremos el método más común y recomendado: añadiremos Retrofit vía Gradle.

Eso significa que debemos ir a nuestro archivo build.gradle y añadir las siguientes líneas:

```
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'
```

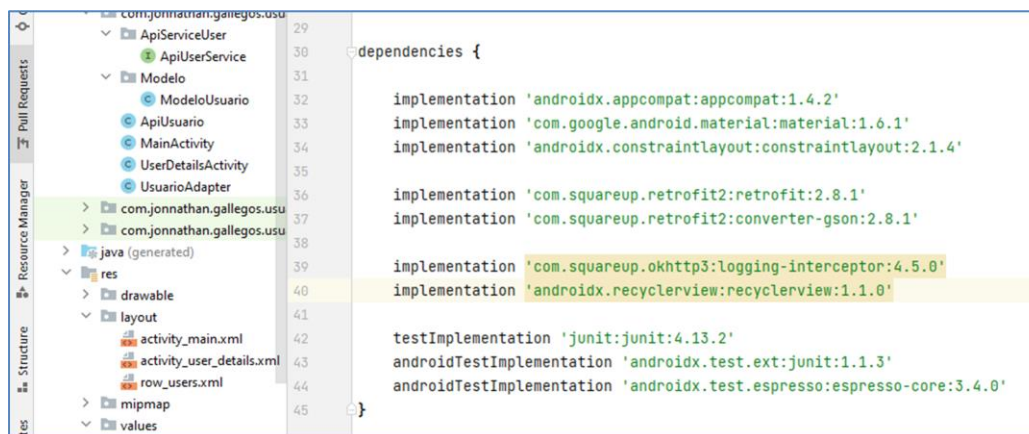
Debes añadir la dependencia en el archivo build.gradle a nivel de módulo.

En el lado izquierdo de Android Studio encontrarás dentro de Gradle Scripts:



Dentro del archivo debes agregar 2 dependencias. Una para Retrofit y otra para GSON.

La tercera dependencia, la del logging interceptor es opcional. Pero te recomiendo añadirla para poder debuggear las peticiones.



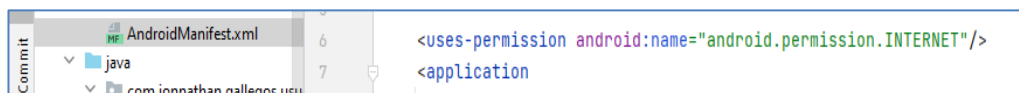
JSON es un formato de respuesta que usan las API. Eso es lo que vamos a obtener y procesar.

Pero GSON es una dependencia adicional, que funciona en conjunto con Retrofit para "convertir las respuestas JSON obtenidas en objetos Java". Retrofit los llama "converters", y existen varios de ellos. Incluso para "mapear" respuestas obtenidas en formato XML.

2. Solicitar Permisos

Antes de empezar a configurar Retrofit en nuestro proyecto, es importante que nuestra aplicación se pueda conectar a internet.

Para solicitar este permiso debemos añadir la siguiente línea a nuestro archivo manifest



3. Crear Una clase y una interfaz

En el siguiente ejemplo de ApiService se han considerado 4 métodos abstractos.

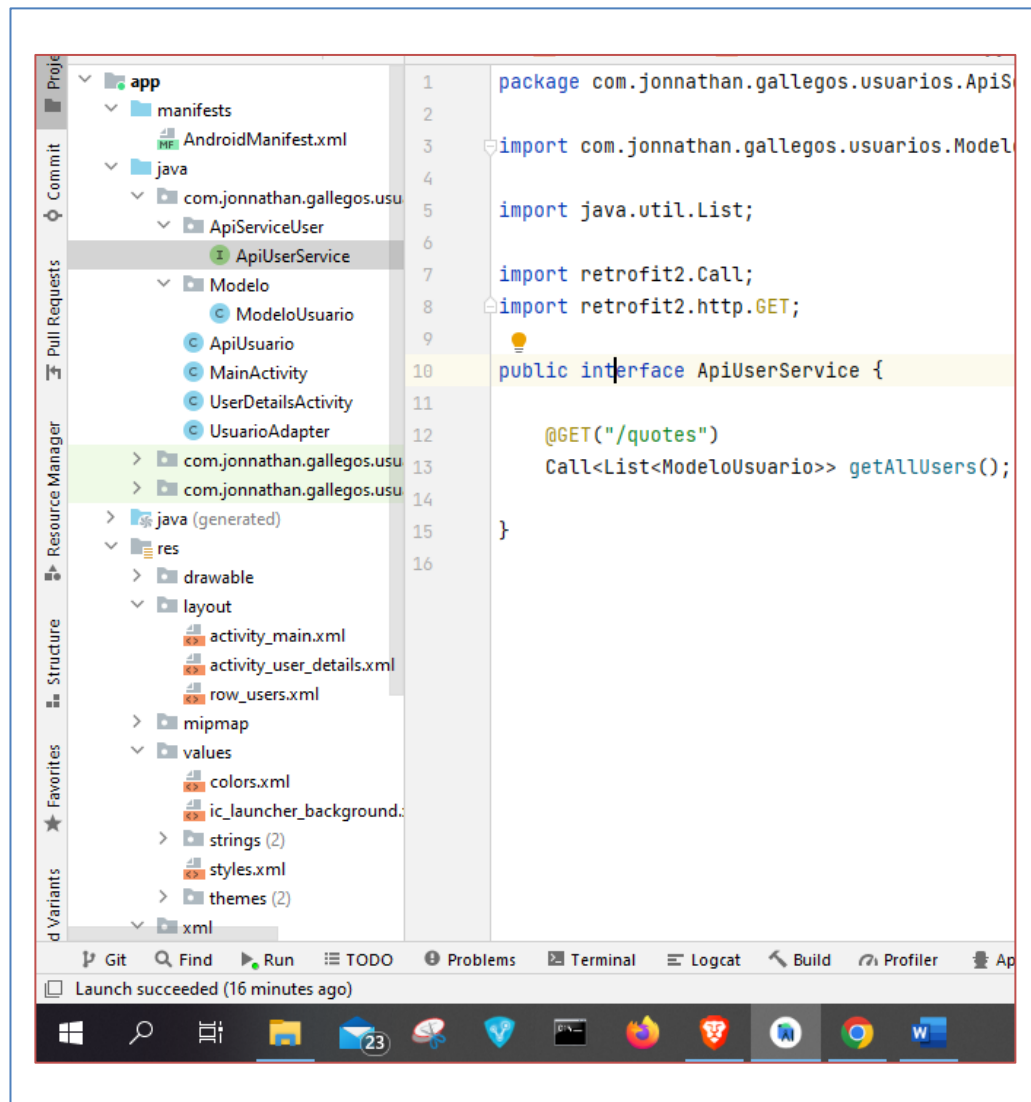
Cada método define una ruta, y especifica qué clase se encargará de procesar la respuesta obtenida. Esto te lo explicaré con más detalle en un momento. Vamos de a pocos.

El primer método representa una petición GET a la ruta diseases. La respuesta será un listado de enfermedades. Y esta respuesta se va a procesar gracias a la clase DiseasesResponse.

El segundo método es una petición POST a la ruta upload/photo. Esta petición se hace enviando ciertos parámetros. Entre ellos, una variable String que representa una imagen codificada en base64. Se asume que la API está lista para subir la foto a través de esta ruta.

El tercer método permite iniciar sesión en una aplicación. Se asume que LoginResponse indica el formato para procesar la respuesta de esta ruta. Debería encargarse de parsear el posible token recibido, si el login fue exitoso.

El último método permite registrar un producto a través de una petición POST. Se asume que la respuesta devolverá un arreglo con mensajes de error, en caso de que el servidor así lo considere. Todo depende de la API. Aquí solo estamos viendo cómo consumirla.



Ejemplo de API Adapter:

```

package com.jonnathan.gallegos.usuarios;

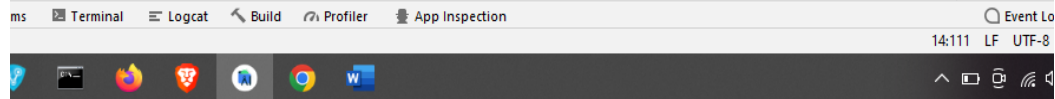
import com.jonnathan.gallegos.usuarios.ApiServiceUser.ApiUserService;
import okhttp3.OkHttpClient;
import okhttp3.logging.HttpLoggingInterceptor;
import retrofit2.Retrofit;
import retrofit2.converter.gson.GsonConverterFactory;

public class ApiUsuario {

    private static Retrofit getRetrofit(){
        HttpLoggingInterceptor httpLoggingInterceptor = new HttpLoggingInterceptor();
        httpLoggingInterceptor.setLevel(HttpLoggingInterceptor.Level.BODY);
        OkHttpClient okHttpClient = new OkHttpClient.Builder().addInterceptor(httpLoggingInterceptor).build();
        Retrofit retrofit = new Retrofit.Builder()
            .baseUrl("https://programming-quotes-api.herokuapp.com")
            .addConverterFactory(GsonConverterFactory.create())
            .client(okHttpClient)
            .build();
        return retrofit;
    }

    public static ApiService getUserService(){
        ApiService userService = getRetrofit().create(ApiUserService.class);
        return userService;
    }
}

```



Con el fin de tener nuestro proyecto organizado por carpetas, vamos a crear una carpeta `io` y a situar allí los 2 archivos antes mencionados.

El nombre de `IO` hace referencia a `input/output`.

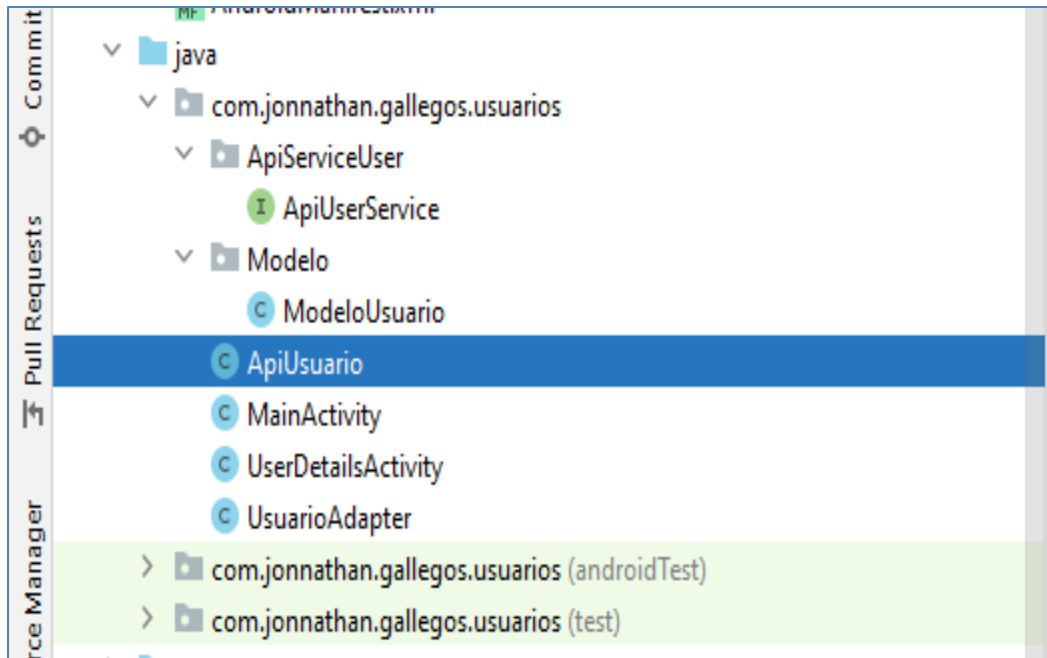
Ten en cuenta que no he puesto los `import` en los ejemplos. Pero puedes importar las clases fácilmente en Android Studio.

Adicional a ello, necesitamos otro grupo de clases que nos permitirán "parsear" las respuestas JSON obtenidas. Estas clases las guardaremos en una carpeta `model`.

Por ejemplo, si tenemos una entidad `Disease` (con los datos de una enfermedad), entonces vamos a crear esta clase dentro de la carpeta `model`.

Esta carpeta contendrá todo nuestro modelo de datos. Es decir, existirá una clase por cada entidad que recibamos desde la API.

A estas alturas nuestro proyecto se verá de la siguiente forma:



En la carpeta response, ubicada dentro del paquete io se encontrarán nuestras clases que sirven para determinar el formato a usar en el "parse" de la respuesta JSON a objetos.

Lista de APIs Públicas para Realizar pruebas.

[GitHub - public-apis/public-apis: A collective list of free APIs](#)

Service Interface:



Modelo:

```
package com.jonnathan.gallegos.usuarios.Modelo;

import java.io.Serializable;

public class ModeloUsuario implements Serializable {

    private String id;
    private String author;
    private String en;

    public String getId() { return id; }

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

    public String getAuthor() { return author; }

    public void setAuthor(String author) { this.author = author; }

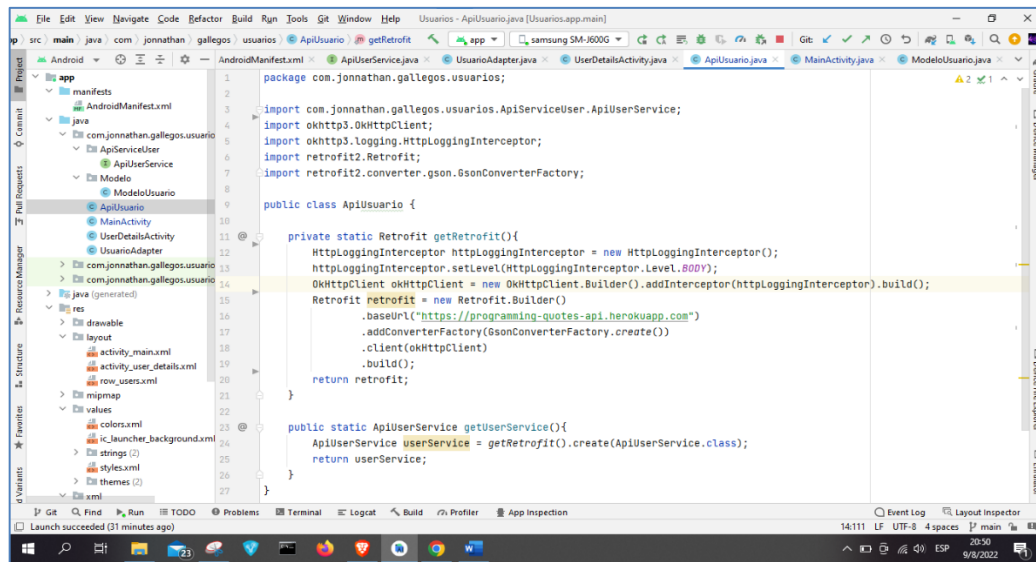
    public String getEn() { return en; }

    public void setEn(String en) { this.en = en; }

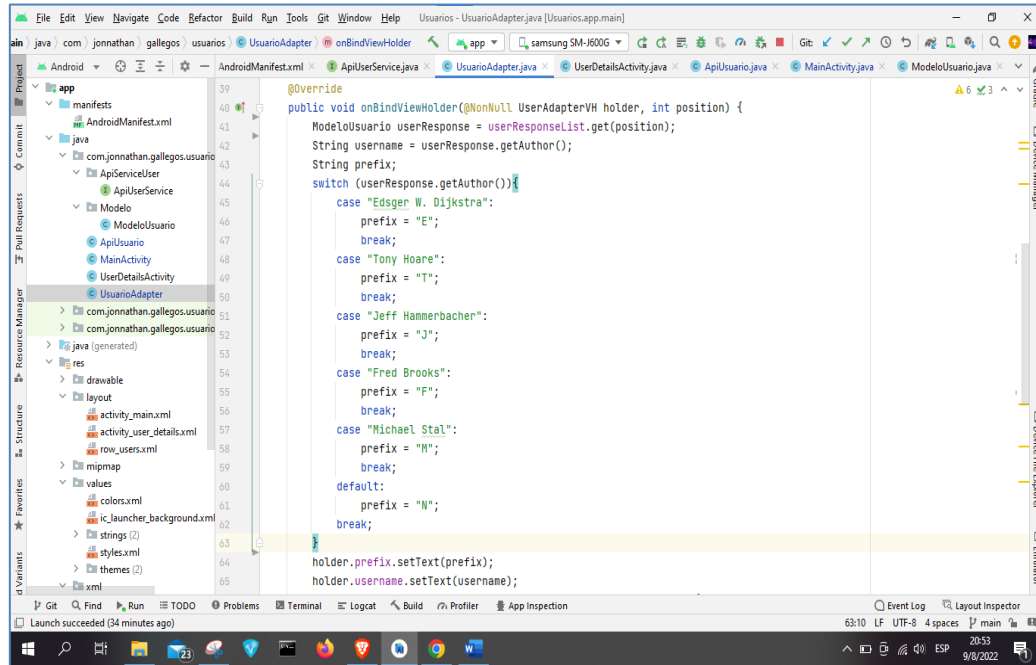
    @Override
    public String toString() {
        return "UserResponse{" +
            "id='" + id + '\'' +
            ", author='" + author + '\'' +

```

Consumo de API:

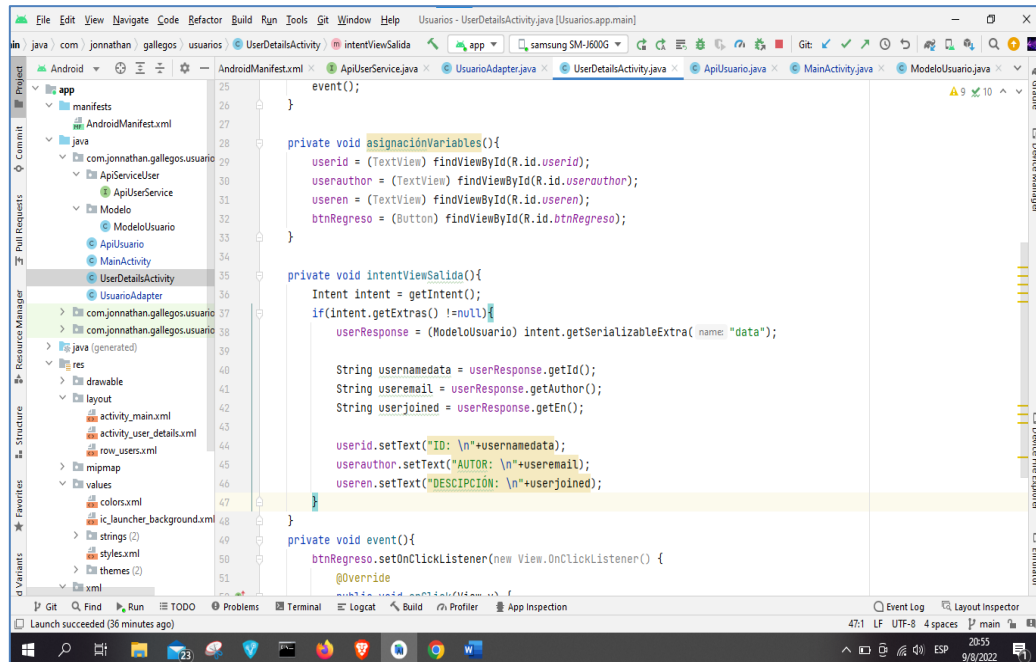


Adapter:



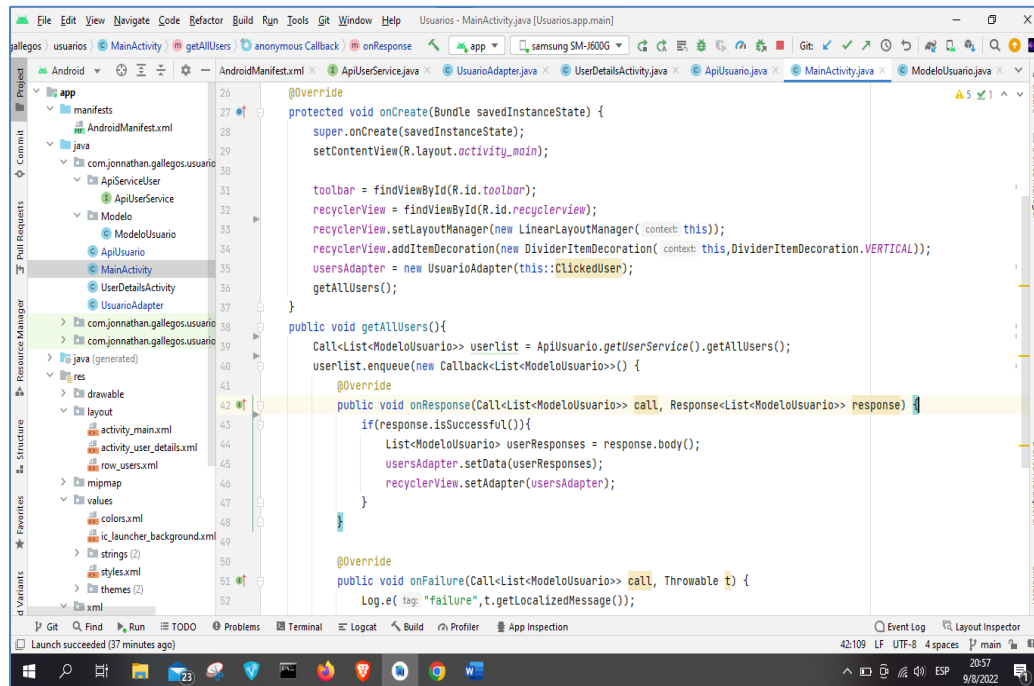
```
File Edit View Navigate Code Refactor Build Run Tools Git Window Help Usuarios - UsuarioAdapter.java [Usuarios.app.main]
main java com jonathan gallegos usuarios UsuarioAdapter onBindViewHolder app samsung SM-J600G Git
Project app manifests AndroidManifest.xml java com.jonathan.gallegos.usuarios ApiUserService ModeloUsuario ApiUsuario MainActivity UserDetailsActivity UsuarioAdapter
Pull Requests Resource Manager Structure Favorites
39 @Override
40 public void onBindViewHolder(@NonNull UsuarioAdapterVH holder, int position) {
41     ModeloUsuario userResponse = userResponseList.get(position);
42     String username = userResponse.getAuthor();
43     String prefix;
44     switch (userResponse.getAuthor()) {
45         case "Edsger W. Dijkstra":
46             prefix = "E";
47             break;
48         case "Tony Hoare":
49             prefix = "T";
50             break;
51         case "Jeff Hammerbacher":
52             prefix = "J";
53             break;
54         case "Fred Brooks":
55             prefix = "F";
56             break;
57         case "Michael Stal":
58             prefix = "M";
59             break;
60         default:
61             prefix = "N";
62             break;
63     }
64     holder.prefix.setText(prefix);
65     holder.username.setText(username);
66 }
Git Find Run TODO Problems Terminal Logcat Build Profiler App Inspection
Launch succeeded (34 minutes ago) 63:10 LF UTF-8 4 spaces main 20:53 9/8/2022
```

View controller:



```
File Edit View Navigate Code Refactor Build Run Tools Git Window Help Usuarios - UserDetailsActivity.java [Usuarios.app.main]
main java com jonathan gallegos usuarios UserDetailsActivity getIntentValida app samsung SM-J600G Git
Project app manifests AndroidManifest.xml java com.jonathan.gallegos.usuarios ApiUserService ModeloUsuario ApiUsuario MainActivity UserDetailsActivity UsuarioAdapter
Pull Requests Resource Manager Structure Favorites
25 event();
26 }
27
28 private void asignaciónVariables(){
29     userid = (TextView) findViewById(R.id.userid);
30     userauthor = (TextView) findViewById(R.id.userauthor);
31     useren = (TextView) findViewById(R.id.useren);
32     btnRegreso = (Button) findViewById(R.id.btnRegreso);
33 }
34
35 private void getIntentValida(){
36     Intent intent = getIntent();
37     if(intent.getExtras() != null){
38         userResponse = (ModeloUsuario) intent.getSerializableExtra("data");
39
40         String usernamedata = userResponse.getId();
41         String useremail = userResponse.getAuthor();
42         String userjoined = userResponse.getEn();
43
44         userid.setText("ID: \n"+usernamedata);
45         userauthor.setText("AUTOR: \n"+useremail);
46         useren.setText("DESCRIPCIÓN: \n"+userjoined);
47     }
48 }
49 private void event(){
50     btnRegreso.setOnClickListener(new View.OnClickListener() {
51         @Override
52         public void onClick(View v) {
53             // TODO: Implement your click logic here
54         }
55     });
56 }
Git Find Run TODO Problems Terminal Logcat Build Profiler App Inspection
Launch succeeded (36 minutes ago) 47:11 LF UTF-8 4 spaces main 20:55 9/8/2022
```

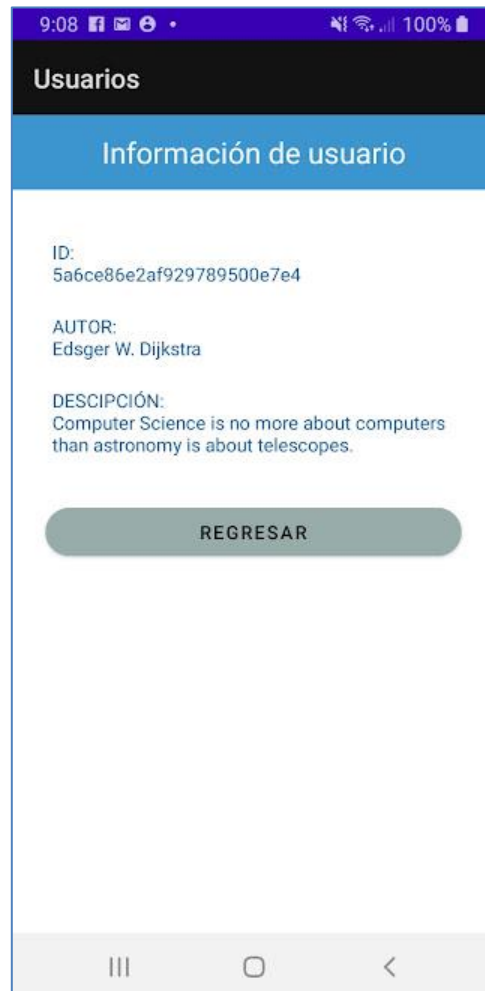

Main Activity:



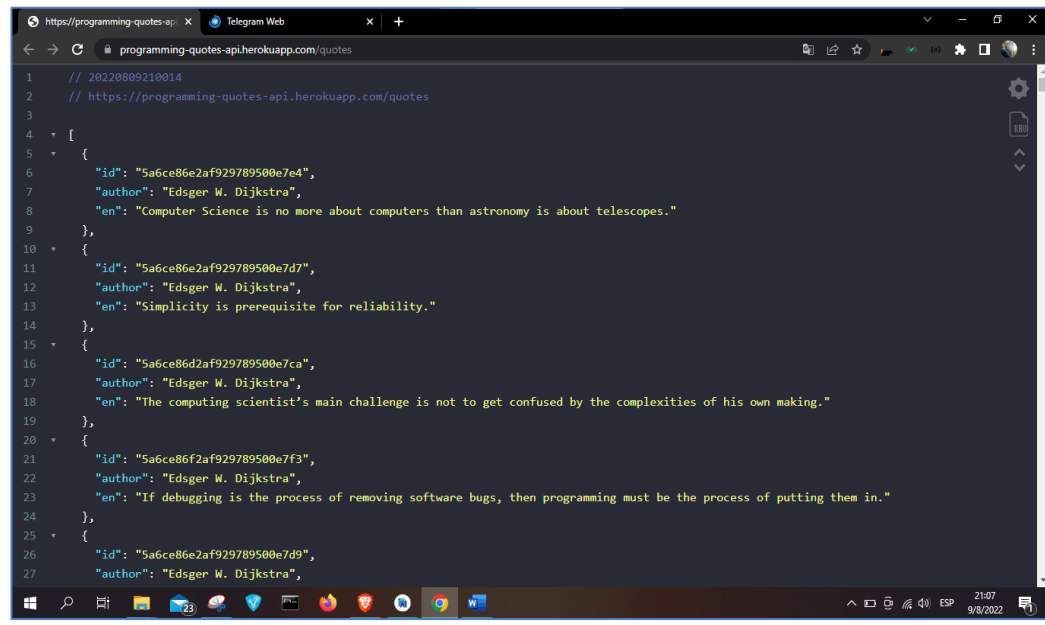
Resultado:



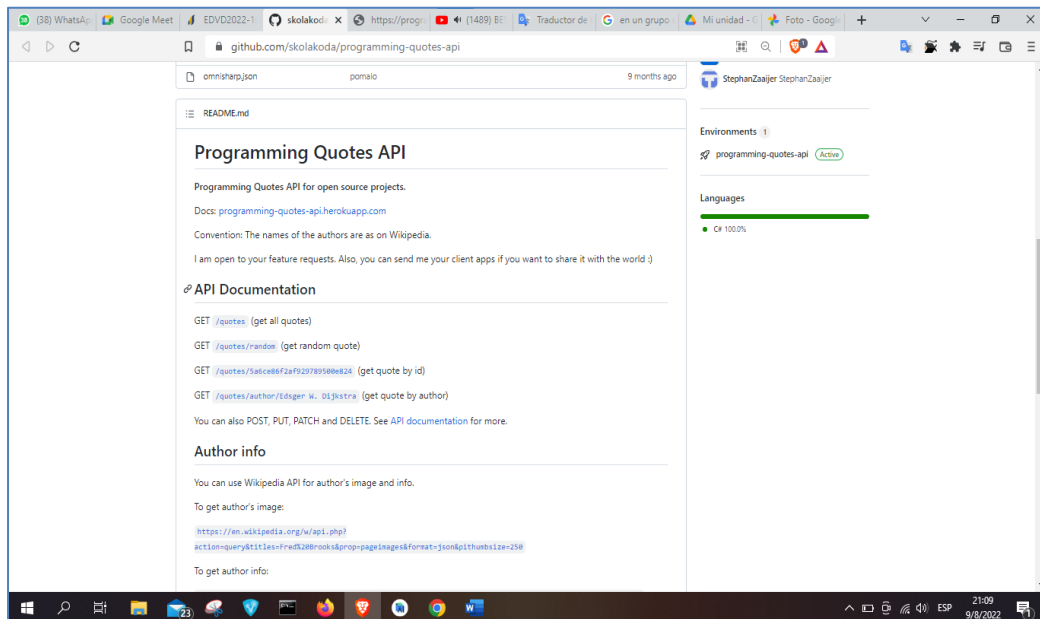
Resultado detalle:



API:



API Usada:



URL API:

<https://github.com/skolakoda/programming-quotes-api>

Consumo:

<https://programming-quotes-api.herokuapp.com/quotes>

2.5 Resultados esperados

- .Al final de la guía, el estudiante está en capacidad de consumir cualquier web service de tipo REST API.

2.6 Bibliografía

Descripción en norma APA

Cómo instalar Android Studio | Desarrolladores de Android. (s. f.). Android Developers. Recuperado 6 de enero de 2020, de <https://developer.android.com/studio/install?hl=es>

Flutter Inicio. (s. f.). Recuperado 18 de febrero de 2021, de <https://flutter.dev/docs/get-started/install-de-dispositius-mobils/>

ESTUDIANTE	DOCENTE	COORDINADOR DE CARRERA
Nombre:	Nombre: Ing. Patricio Pacheco	Nombre: Ing. Jessica Herrera
Firma	Firma	Firma
Fecha:	Fecha: 11/02/2022	Fecha: