ESCUELA SUPERIOR DE INGENIERÍA

Programación en Internet

Grado en Ingeniería Informática

Aplicación Android

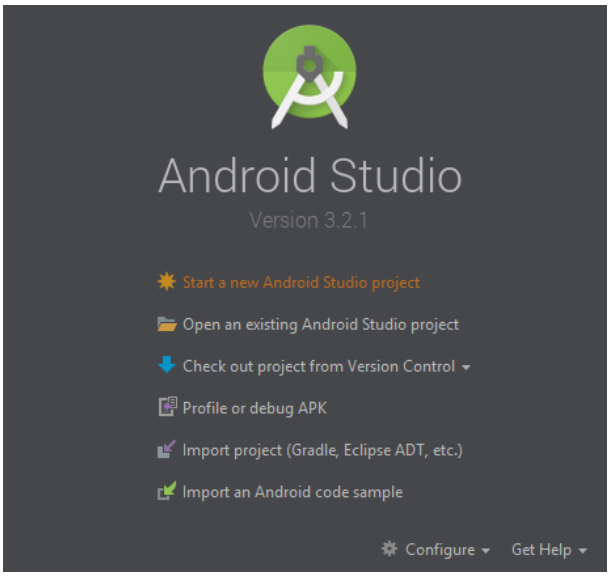
Autores:
José Manuel Periñán Freire
Iñaki Urrutia Sanchez

Supervisores:
David Corral Plaza

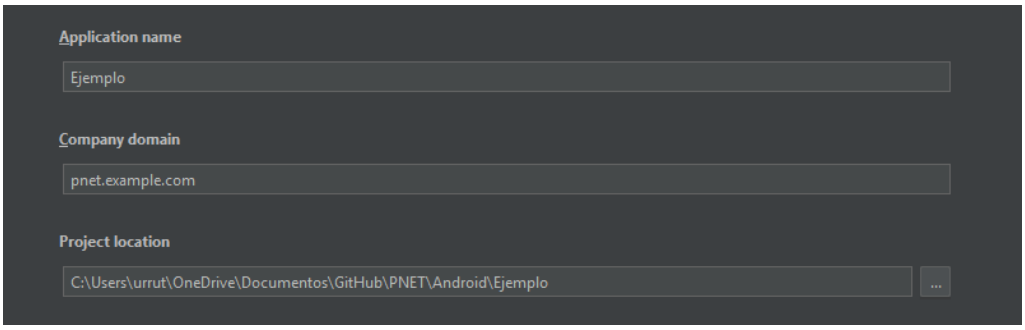
Cádiz, 14 de diciembre de 2019

1. Creación del proyecto

Para crear el proyecto de nuestra aplicación, abrimos Android Studio, y damos click en “Start a new Android Studio project”.



Ahora especificamos el nombre de nuestra aplicación, el nombre de nuestra empresa y la localización del proyecto en el que vamos a trabajar. En este ejemplo los campos quedarían rellenos de la siguiente manera:



Pulsamos next, y a continuación marcamos la casilla *Phone and Tablet*. En el desplegable que está justo abajo seleccionamos la API mínima compatible con nuestra aplicación. Es recomendable escoger una que incluya el mayor número de dispositivos posible, como la API 15. Le damos a next, y elegimos una *Empty Activity*. Dejamos las opciones por defecto y pulsamos *Finish* para crear nuestro proyecto.

2. Creando la interfaz para usar la API

Primero vamos a crear la interfaz a partir de la cuál consumiremos de nuestra API, ésta constará de 4 botones:

- Get All
- Get one
- Delete one
- Post

Nos situamos ahora en el layout que es donde vamos a editar la interfaz de usuario de la actividad, llamada *activity_main.xml* y dentro creamos los botones y un EditText para introducir el dni para poder usar el getone y el delete one. A continuación encontramos el código que usaremos para el mismo:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:http="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/UsersActivity"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context=".MainActivity">

    <EditText
        android:id="@+id/dni"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:hint="Set dni"
        />

    <Button
        android:id="@+id/Getall"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Get users" />

    <Button
        android:id="@+id/Getone"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Get user"/>

    <Button
        android:id="@+id/Deleteone"
        android:layout_width="wrap_content"
```

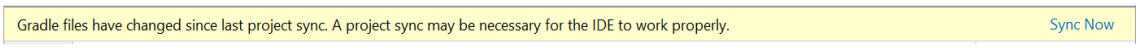
```
            android:layout_height="wrap_content"
            android:text="Delete user"/>

        <Button
            android:id="@+id/Post"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Add user"/>


        <android.support.v7.widget.RecyclerView
            android:id="@+id/my_recycler_view"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:scrollbars="vertical" />


    </LinearLayout>
```

Añadir que para mostrar los usuarios usaremos un RecyclerView donde situaremos los usuarios a mostrar, además para usar esto así como para realizar las peticiones será necesario incluir algunas dependencias al proyecto.

Nos vamos a Gradle Scripts/build.gradle (el segundo que aparece) y añadimos las siguientes dependencias:

```
implementation 'com.android.support:appcompat-v7:28.0.0'
implementation 'com.android.support:recyclerview-v7:28.0.0'
implementation 'com.android.support:cardview-v7:28.0.0'
implementation 'com.squareup.okhttp3:okhttp:3.12.0'
```

Una vez realizado esto nos aparecerá una ventana como la siguiente:



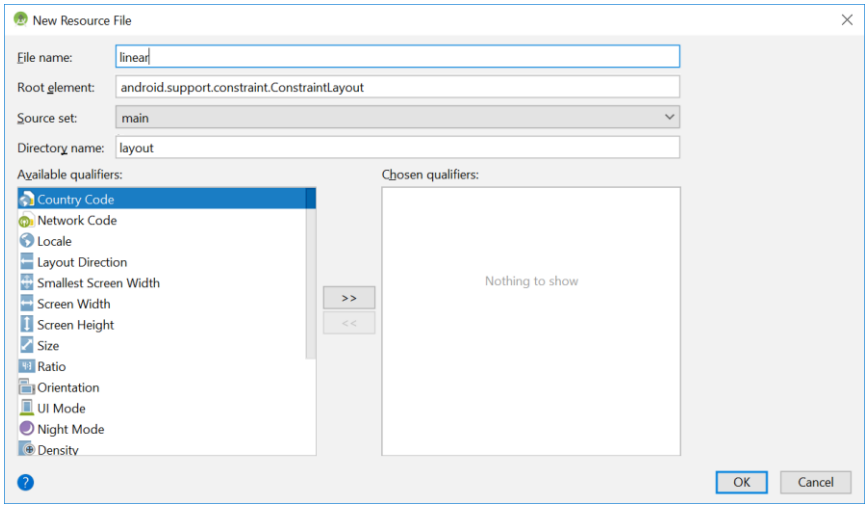
Pulsamos en *Sync Now*

También para poder realizar las peticiones será necesario añadir esta línea al *Android.Manifest.xml*

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

Una vez terminado este proceso, vamos a empezar a realizar las acciones necesarias para adaptar nuestro RecyclerView a nuestro proyecto y podemos mostrar los usuarios.

Primero vamos a crear el Layout que mostrará los distintos campos del RecyclerView: new > Layout resources file

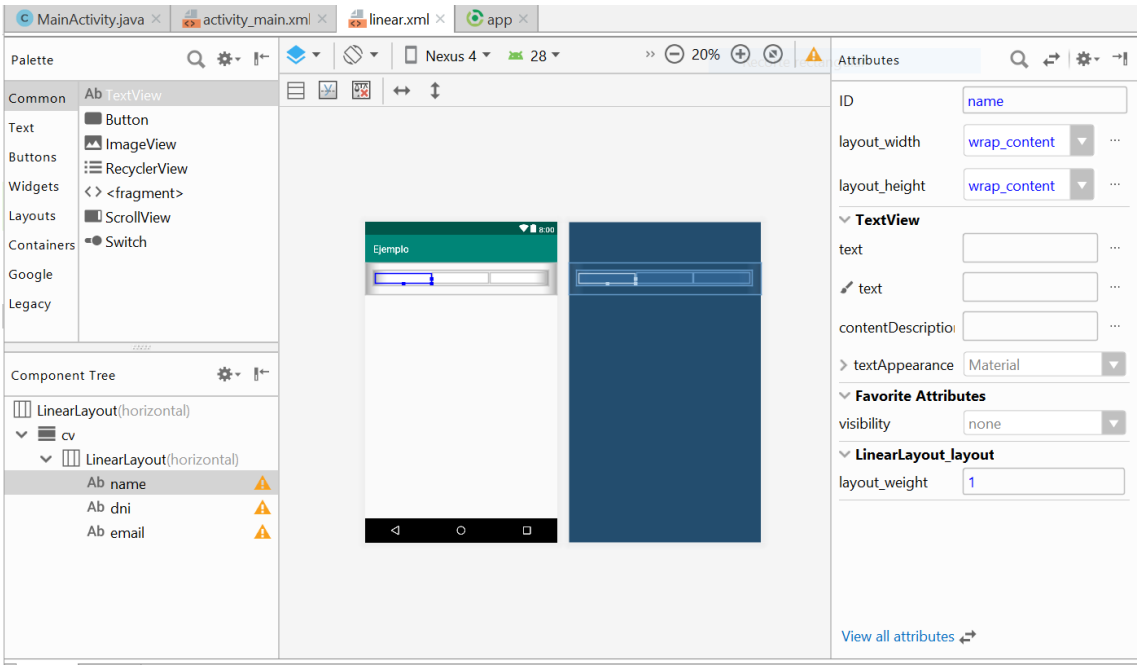


Y una vez situado dentro del mismo incluimos el siguiente código:

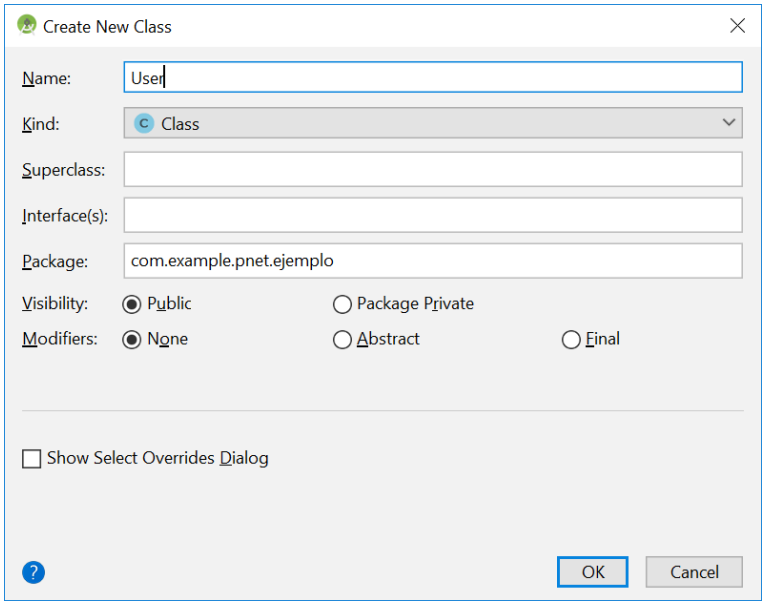
```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"


    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:padding="16dp">

    <android.support.v7.widget.CardView
        android:id="@+id/cv"
        android:layout_width="match_parent"
        android:layout_height="wrap_content">

        <LinearLayout
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:orientation="horizontal"
            android:padding="5dp">

            <TextView
                android:id="@+id/name"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_weight="1"
                android:padding="5dp"
                android:textColor="@color/colorPrimaryDark"
                android:textSize="8sp" />
```

```
            <TextView
                android:id="@+id/dni"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_weight="1"
                android:padding="5dp"
                android:textColor="@color/colorPrimary"
                android:textSize="10sp" />

            <TextView
                android:id="@+id/email"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_weight="1"
                android:padding="5dp"
                android:textColor="@color/colorPrimary"
                android:textSize="10sp" />

        </LinearLayout>
    </android.support.v7.widget.CardView>
</LinearLayout>
```

Dentro del segundo LinearLayout lo podemos adaptar según los campos que necesitemos, en nuestro caso vamos a mostrar el nombre, el dni y el email, que son los 3 TextView que encontramos dentro. Dentro de design podremos ver como ira quedando nuestro RecyclerView:



Hecho esto crearemos la clase User para usarla sobre el ReclyclerView: new > Java class




```
public class User {
    private String name;
    private String dni;
    private String email;

    public User(String name, String dni, String email)
    {
        this.name = name;
        this.dni = dni;
        this.email = email;
    }

    public String getName() {
        return name;
    }

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

    public String getDni() {
        return dni;
    }

    public void setDni(String dni) {
        this.dni = dni;
    }

    public String getEmail() {
        return email;
```

```
    }

    public void setEmail(String email) {
        this.email = email;
    }
}
```

Esta clase tendrá los atributos y los tipos de los parámetros que vayamos a mostrar en la aplicación obtenidos de la API.

Ahora crearemos la clase UserAdapter para unir nuestra clase User con el RecyclerView, como antes, new > Java class.

```
import android.content.Context;
import android.support.v7.widget.RecyclerView;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.TextView;

import java.util.ArrayList;

public class UserAdapter
        extends RecyclerView.Adapter<UserAdapter.MyViewHolder> {

    private ArrayList<User> users;
    private Context context;


    public static class MyViewHolder extends RecyclerView.ViewHolder {
        TextView name;
        TextView dni;
        TextView email;


        public MyViewHolder(View v) {
            super(v);
            name = (TextView) v.findViewById(R.id.name);
            dni = (TextView) v.findViewById(R.id.dni);
            email = (TextView) v.findViewById(R.id.email);
        }
    }

    public UserAdapter(ArrayList<User> myDataset) {
        users = myDataset;
    }

    @Override
    public UserAdapter.MyViewHolder onCreateViewHolder(ViewGroup
parent, int viewType) {
        View v =
LayoutInflater.from(parent.getContext()).inflate(R.layout.linear,
parent, false);
```

```
        MyViewHolder vh = new MyViewHolder(v);
        context = parent.getContext();
        return vh;
    }

    @Override
    public void onBindViewHolder(MyViewHolder holder, final int
position) {

holder.name.setText(String.valueOf(users.get(position).getName()));

holder.dni.setText(String.valueOf(users.get(position).getDni()));;

holder.email.setText(String.valueOf(users.get(position).getEmail()));
    }

    @Override
    public int getItemCount() {
        return users.size();
    }

}
```

De aquí quedarnos que nuevamente tendremos que definir los TextView como al principio para los atributos que usaremos para mostrar, y tomando como referencia los creados anteriormente con *v.findViewById(R.id.nombredelidquehemoscreadoenelxml)*

Tenemos que adaptarlo siempre a la clase que queramos usar, en nuestro caso es User pero si tuviéramos otra tendríamos adaptar esta clase a la misma, con los mismos TextView que en el activity_main.xml y la clase que usemos.

3. MainActivity

Ahora empezaremos a crear la lógica asociada a nuestra MainActivity con los botones y el RecyclerView.

Creamos primero las variables globales a utilizar dentro de la misma:

```
private RecyclerView mRecyclerView;
private RecyclerView.Adapter mAdapter;
private RecyclerView.LayoutManager mLayoutManager;
private Button getAll;
private Button getOne;
private Button deleteOne;
private Button post;
private static ArrayList<User> users;
```

Los 3 primeros asociados al RecyclerView, luego los 4 botones para hacer las distintas peticiones y el ArrayList para el RecyclerView.

Y continuación tenemos el código inicial del OnCreate aún sin las peticiones implementadas.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    // Referenciamos al RecyclerView
    mRecyclerView = (RecyclerView)
findViewById(R.id.my_recycler_view);

    //Referenciamos los botones para las peticiones
    getAll = (Button) findViewById(R.id.Getall);
    getOne = (Button) findViewById(R.id.Getone);
    deleteOne = (Button) findViewById(R.id.Deleteone);
    post = (Button) findViewById(R.id.Post);

    // Mejoramos rendimiento con esta configuración
    mRecyclerView.setHasFixedSize(true);

    users = new ArrayList<User>();

    // Creamos un LinearLayoutManager para gestionar el item.xml
creado antes
    mLayoutManager = new LinearLayoutManager(this);
    // Lo asociamos al RecyclerView
    mRecyclerView.setLayoutManager(mLayoutManager);

    getAll.setOnClickListener(new View.OnClickListener() {
```

```
        @Override
        public void onClick(View v) {

        }
    });

    getOne.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {

        }
    });

    deleteOne.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {

        }
    });

    post.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {

        }
    });

}
```

4. GET ALL

Ahora empezaremos a crear las clases necesarias para las peticiones, empezaremos por el getAll, creamos la clase que se encargará de hacer la petición:

```
public class LongRunningGetIO extends AsyncTask<Void, Void, String> {

    @Override
    protected String doInBackground(Void... params) {
        String text = null;
        OkHttpClient client = new OkHttpClient();
        Request request = new
Request.Builder().url("http://192.168.1.18:8080/users")
                .build();
        try {
            Response res = client.newCall(request).execute();
            return res.body().string();
        } catch (Exception e) {
            return e.toString();
        }
    }

    @Override
    protected void onPostExecute(String results) {
        //JSONObject respJson = null;
        JSONArray json = null;
        if (results == null) {
        }
        if (results != null) {
            try {
                json = new JSONArray(results);
                for (int i = 0; i < json.length(); i++) {
                    users.add(new
User(json.getJSONObject(i).get("firstname").toString(),

json.getJSONObject(i).get("DNI").toString(),

json.getJSONObject(i).get("email").toString()
                            ));
                }

                mAdapter = new UserAdapter(users);

                mRecyclerView.setAdapter(mAdapter);
            } catch (JSONException e) {
                e.printStackTrace();
            }
        }
    }

}
```

En la línea: `Request.Builder().url("http://192.168.1.18:8080/users")` tendremos que adaptarla a la IP donde realizaremos las peticiones . Luego dentro de OnPostExecute, que es el método ejecutado tras realizar la petición recibiremos el string que contendrá el JSON que hemos obtenido de la petición:

```
try {
            Response res = client.newCall(request).execute();
            return res.body().string();
        } catch (Exception e) {
            return e.toString();
        }
```

Y ese string lo convertimos a JSON:

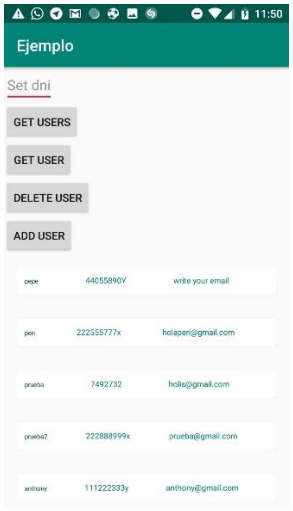
```
json = new JSONArray(results);
```

Y ahora simplemente creamos un bucle donde recorremos el mismo y vamos creando nuevos usuarios en cada iteración en base a los parámetros que recibimos del JSON, en nuestro caso los campos del JSON que queremos son: firstname, DNI y email. Al finalizar el bucle instanciamos el *adapter* con el array que hemos rellenado con cada uno de los usuarios.

Añadimos la llamada a la petición al pulsar el botón del get users:

```
getAll.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        new LongRunningGetIO().execute();
    }
});
```

Esto es lo que se mostraría en el móvil tras pulsar GET USERS



5. GET ONE

Para el getOne añadiremos la lógica del EditText que creamos al principio para que podamos que escribir el dni del usuario que queremos obtener:

```
private TextView DNI;
```

Y dentro del onCreate como hemos hecho anteriormente:

```
DNI = (TextView) findViewById(R.id.dni);
```

Y ahora cremoas la clase que se encarga realizar la petición para obtener un único usuario, muy parecida a la anterior:

```
public class LongRunningGetOneIO extends AsyncTask<Void, Void, String>
{

    @Override
    protected String doInBackground(Void... params) {

        String text = null;
        OkHttpClient client = new OkHttpClient();
        String dni = DNI.getText().toString();
        Request request = new
Request.Builder().url("http://192.168.1.18:8080/users/" + dni)
                .build();
        try {
            Response res = client.newCall(request).execute();
            return res.body().string();
        } catch (Exception e) {
            return e.toString();
        }
    }

    @Override
    protected void onPostExecute(String results) {
        //JSONObject respJson = null;
        JSONArray json = null;
        if (results == null) {
        }
        if (results != null) {
            try {
                json = new JSONArray(results);
                    users.add(new
User(json.getJSONObject(0).get("firstname").toString(),

json.getJSONObject(0).get("DNI").toString(),

json.getJSONObject(0).get("email").toString()
```

```
                    ));


                mAdapter = new UserAdapter(users);

                mRecyclerView.setAdapter(mAdapter);
            } catch (JSONException e) {
                e.printStackTrace();
            }
        }
    }

}
```

A la URL le añadimos el parámetro del dni que obtenemos de haber escrito en el EditText :

```
String dni = DNI.getText().toString();

Request.Builder().url("http://192.168.1.18:8080/users/" + dni)
```

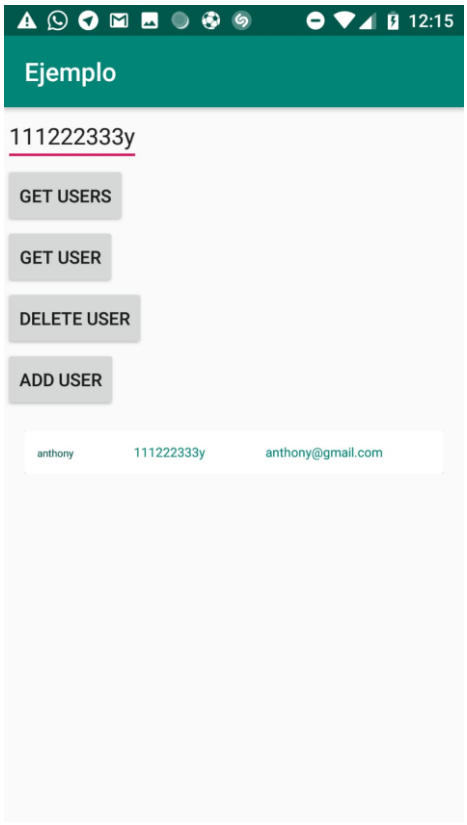
Y el onPostExecute es exactamente igual al anterior cambiando que en vez de necesitar un bucle para recorrer todo el JSON que nos devolvía el GET, aquí simplemente accedemos al índice del primero pues solo nos debe devolver uno:

```
try {
    json = new JSONArray(results);
        users.add(new
User(json.getJSONObject(0).get("firstname").toString(),
                json.getJSONObject(0).get("DNI").toString(),
                json.getJSONObject(0).get("email").toString()
        ));


    mAdapter = new UserAdapter(users);

    mRecyclerView.setAdapter(mAdapter);
} catch (JSONException e) {
    e.printStackTrace();
}
```

Añadimos en el onCreate la funcionalidad del getOne:

```
getOne.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        new LongRunningGetOneIO().execute();
    }
});
```

Y este es el resultado que obtenemos dentro de la aplicación:



6. DELETE ONE

Para el deleteOne usaremos un código más sencillo que los anteriores pues únicamente vamos a hacer la petición y a mostrar si la misma se ha realizado correctamente o no.

Empezamos como en los anteriores ejemplos a crear la clase que se encargará de ello:

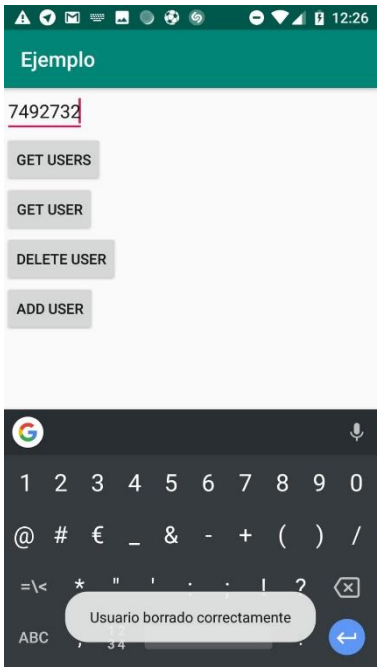
```
public class LongRunningGetIODELETE extends AsyncTask<Void, Void,
Boolean> {
    public final MediaType JSON
            = MediaType.parse("application/json; charset=utf-8");
    @Override
    protected Boolean doInBackground(Void... params) {

        String text = null;
        OkHttpClient client = new OkHttpClient();
        String dni = DNI.getText().toString();
        Request request = new
Request.Builder().url("http://192.168.1.18:8080/users/"+ dni)
                .delete()
                .build();
        try {
            Response res = client.newCall(request).execute();
            if(!res.isSuccessful())
                return false;
            else
                return true;
        } catch (Exception e) {
            return false;
        }
    }

    @Override
    protected void onPostExecute(Boolean results) {
        if(!results)
            Toast.makeText(getApplicationContext(), "Fallo al
borrar",Toast.LENGTH_LONG).show();
        else
            Toast.makeText(getApplicationContext(), "Usuario borrado
correctamente",Toast.LENGTH_LONG).show();

    }

}
```

Simplemente indicamos en la URL que queremos borrar de la siguiente manera:

```
Request.Builder().url("http://192.168.1.18:8080/users/"+ dni)
                 .delete()
                 .build();
```

Obteniendo el dni igual que hemos hecho en el getOne y devolvemos true si la operación se ha realizado correctamente y false si no y esto lo mostramos en un Toast:

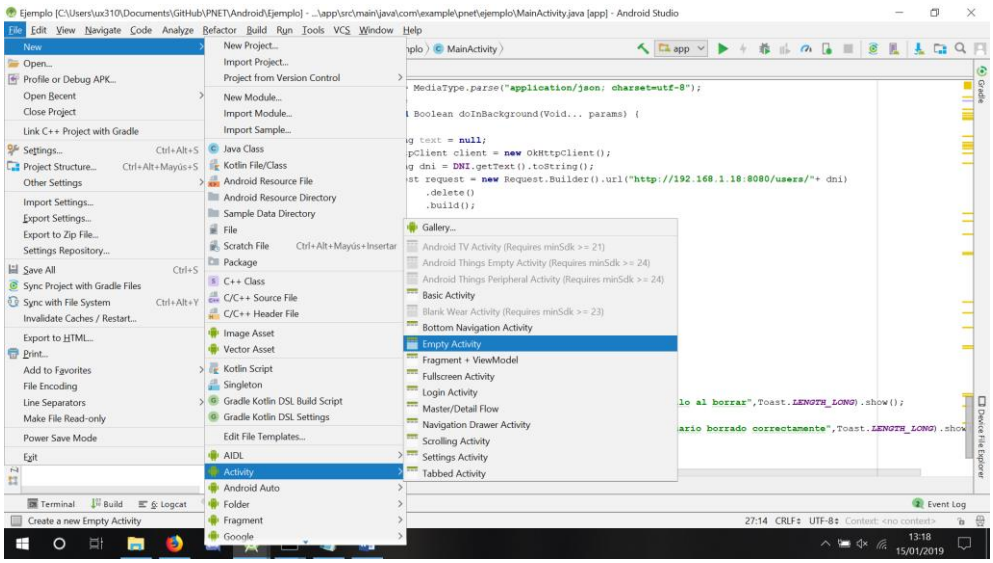
```
if(!results)
            Toast.makeText(getApplicationContext(), "Fallo al
borrar",Toast.LENGTH_LONG).show();
        else
            Toast.makeText(getApplicationContext(), "Usuario borrado
correctamente",Toast.LENGTH_LONG).show();
```

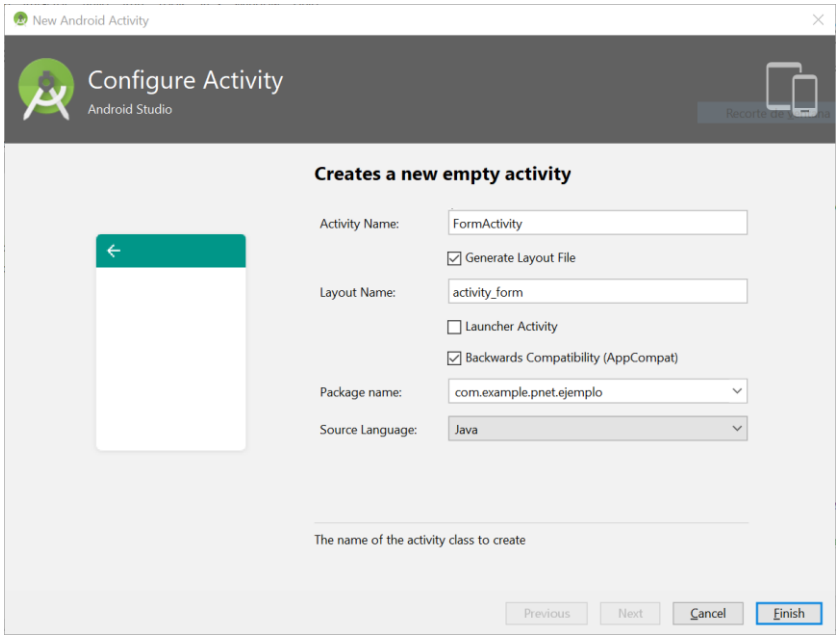


7. POST

Para el post vamos a añadir una nueva Actividad en la que creamos el formulario con unos campos para añadir un nuevo usuario por lo que añadiremos la funcionalidad al botón del POST para que nos mande a esta nueva.

Primero creamos la actividad:





Como hemos creado una actividad nueva tendremos que editar el nuevo layout que nos ha generado según la vista que queramos crear, en nuestro caso, tendremos 3 EditText para meter: Nombre, DNI y correo; y un botón para realizar el envío del nuevo usuario. Este layout lo encontraremos en res > layout > *activity_form.xml* y dentro de el añadiremos el siguiente código:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
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=".FormActivity">

    <EditText
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/name"
        android:hint="Set name"
        />

    <EditText
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/dni"
        android:layout_below="@id/name"
        android:hint="Set dni"
        />

    <EditText
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/email"
        android:layout_below="@id/dni"
        android:hint="Set email"
        />

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/add"
        android:layout_below="@id/email"
        android:text="ADD USER"/>

</RelativeLayout>
```

Y ahora nos situamos en FormActivity.java y empezaremos a desarrollar la lógica de nuestro POST. Referenciamos los elementos que hemos creado en el layout de la siguiente manera:

```
public class FormActivity extends AppCompatActivity {

    private EditText name;
    private EditText dni;
    private EditText email;

    private Button send;

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

        name = (EditText) findViewById(R.id.name);
        dni = (EditText) findViewById(R.id.dni);
        email = (EditText) findViewById(R.id.email);

        send = (Button) findViewById(R.id.add);

        send.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {

            }
        });
    }
}
```

Y por ahora tendremos el botón sin funcionalidad y nos dedicaremos ahora a crear la clase encargada de hacer el POST:

```
public class LongRunningGetIOPOST extends AsyncTask<Void, Void,
Boolean> {
    public final MediaType JSON
            = MediaType.parse("application/json; charset=utf-8");
    @Override
    protected Boolean doInBackground(Void... params) {

        String text = null;
        OkHttpClient client = new OkHttpClient();
        JSONObject json = new JSONObject();

        try{
            json.put("firstname", name.getText().toString());
            json.put("DNI", dni.getText().toString());
            json.put("email", email.getText().toString());
        }catch (JSONException e) {
            e.printStackTrace();
        }

        RequestBody body = RequestBody.create(JSON, json.toString());
        Request request = new
Request.Builder().url("http://192.168.1.23:8080/users")
                .post(body)
```

```
                .build();
        try {
            Response res = client.newCall(request).execute();
            if(!res.isSuccessful())
                return false;
            else
                return true;
        } catch (Exception e) {
            return false;
        }
    }
    //return text

    @Override
    protected void onPostExecute(Boolean results) {
        if(!results)
            Toast.makeText(getApplicationContext(), "Fallo al
insertar",Toast.LENGTH_LONG).show();
        else
            Toast.makeText(getApplicationContext(), "Usuario insertado
correctamente",Toast.LENGTH_LONG).show();
    }

}
```

Esta clase es parecida a las anteriores pero ahora el JSON no lo tenemos que obtener si no generarlo nosotros para enviárselo a la base de datos. Primero nos creamos un JSON vacío:

```
JSONObject json = new JSONObject();
```

Y ahora por medio de los EditText que hemos creado recogeremos la información de los mismos y la añadiremos al JSON, siendo el primer parámetro del put la etiqueta y el segundo son las variables de tipo EditText que creamos anteriormente de las cuales obtenemos los distintos valores que hemos modificado al escribir sobre ellos:

```
try{
    json.put("firstname", name.getText().toString());
    json.put("DNI", dni.getText().toString());
    json.put("email", email.getText().toString());
}catch (JSONException e) {
    e.printStackTrace();
}
```

Creamos el body que añadiremos a la petición:

```
RequestBody body = RequestBody.create(JSON, json.toString());
```

Convirtiendo ese JSON en un string para poder realizarlo y realizamos la petición:

```
Request request = new
Request.Builder().url("http://192.168.1.23:8080/users")
        .post(body)
        .build();
```

Según nuestra IP a la que queramos mandarlo y añadiéndole el campo post a la misma así como el body por parámetros que será lo que se insertará en dicha URL en nuestro caso, el formulario con los valores que hemos escrito.

Comprobamos que la petición se haya realizado correctamente:

```
try {
    Response res = client.newCall(request).execute();
    if(!res.isSuccessful())
        return false;
    else
        return true;
} catch (Exception e) {
    return false;
}
```

Y por último sobrecargamos el método que se ejecuta tras la ejecución de la petición, como en los casos anteriores:

```
@Override
protected void onPostExecute(Boolean results) {
    if(!results)
        Toast.makeText(getApplicationContext(), "Fallo al
insertar",Toast.LENGTH_LONG).show();
    else
        Toast.makeText(getApplicationContext(), "Usuario insertado
correctamente",Toast.LENGTH_LONG).show();
}
```

Mostrando un Toast según si ha podido realizarse correctamente o no.

Y añadimos la funcionalidad del botón que antes teníamos vacío:

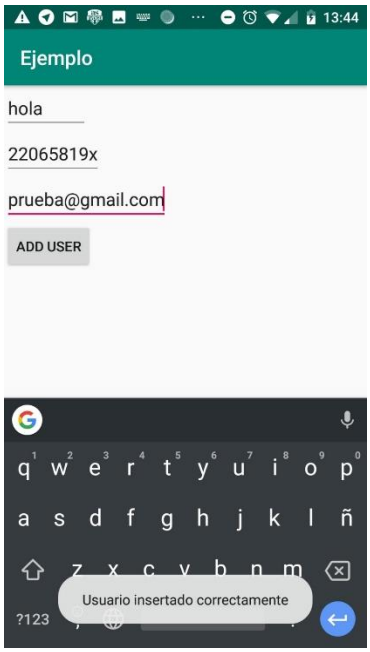
```
send.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        new LongRunningGetIOPOST().execute();
    }
});
```

Instanciando la clase que acabamos de crear.

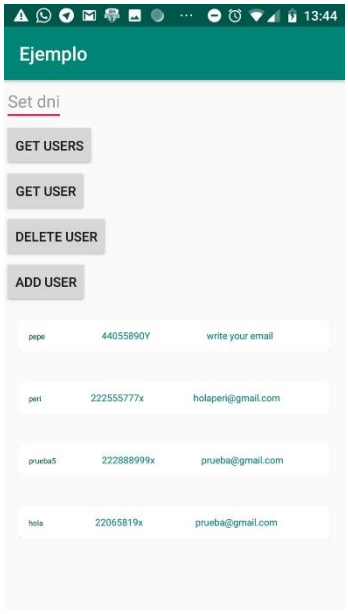
Nos faltaría poder movernos a esta nueva actividad que hemos creado así que nos situamos en el MainActivity.java y sobre el botón del post hacemos lo siguiente:

```
post.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Intent intent = new Intent(getApplicationContext(),
FormActivity.class);
        startActivity(intent);
    }
});
```

Esto hará que cuando pulsemos el botón del post nos cambie a la nueva actividad que hemos creado. Dentro de dicha actividad hacemos la prueba:



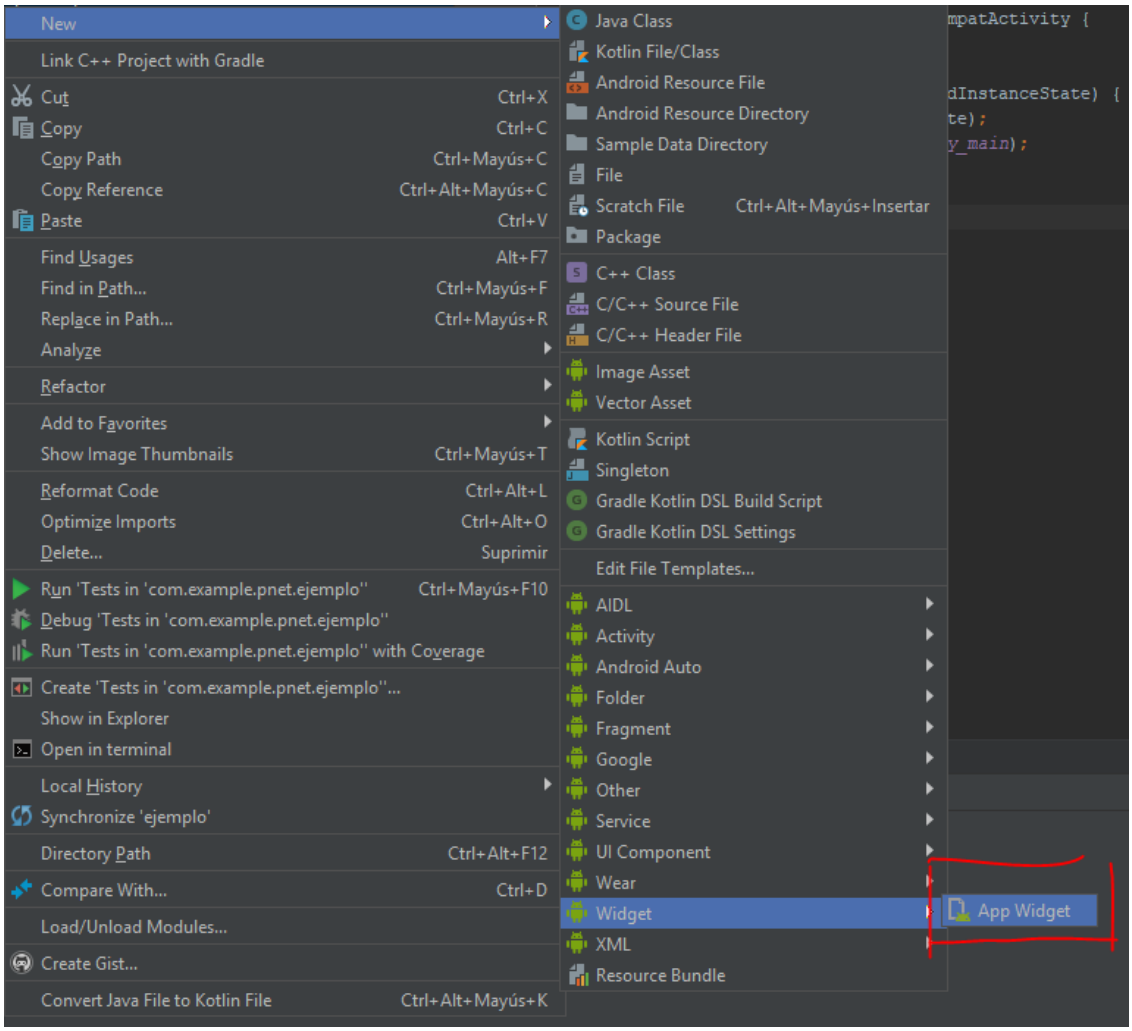
Y si volvemos a la actividad principal y pulsamos para obtener todos los usuarios de nuevo:



Vemos que se añadido ese último al mismo.

8. Widgets

Para crear un widget seleccionamos *New* > *Widget* > *App Widget* tal y como se muestra en la siguiente imagen:



Indicamos el nombre de la clase que va a controlar su comportamiento, su tamaño mínimo, controlamos su posible colocación, seleccionamos Java como lenguaje del código fuente y pulsamos en *Finish*.

Una vez hagamos esto, Android Studio nos creará los correspondientes archivos .xml y .java para programar nuestro widget.

En este ejemplo vamos a hacer que nuestro widget abra nuestro *Main Activity*, y también vamos a personalizar su color. Para lo primero, antes de nada, tenemos que añadir un id único al layout que nos ha creado Android Studio:

```
<RelativeLayout mlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/widget"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="#09C"
    android:padding="@dimen/widget_margin">

    <TextView
        android:id="@+id/appwidget_text"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:layout_centerVertical="true"
        android:layout_margin="8dp"
        android:background="#09C"
        android:contentDescription="@string/appwidget_text"
        android:text="@string/appwidget_text"
        android:textColor="#ffffff"
        android:textSize="24sp"
        android:textStyle="bold|italic" />

</RelativeLayout>
```

Ahora nos vamos a la clase Java de nuestro widget, y sustituimos el método onUpdate(…) por la siguiente versión:

```
public void onUpdate(Context context, AppWidgetManager appWidgetManager,
int[] appWidgetIds) {
        // There may be multiple widgets active, so update all of them
        for (int appWidgetId : appWidgetIds) {
            Intent intent = new Intent(context, MainActivity.class);
            PendingIntent pendingIntent =
                    PendingIntent.getActivity(context, 0, intent, 0);
            RemoteViews views = new RemoteViews(context.getPackageName(),
                    R.layout.new_app_widget);
            views.setOnClickPendingIntent(R.id.widget, pendingIntent);
            appWidgetManager.updateAppWidget(appWidgetId, views); }
    }
```

Con el código anterior:

- ➔ Creamos un *Intent* que abre la *Activity* que queramos de nuestra aplicación, y la incluye en un *PendingIntent.*
- ➔ Creamos un *RemoteViews* que establece un canal de comunicación entre nuestra aplicación y nuestro Widget.
- ➔ Indicamos que al pulsar el Widget se lance el *Intent.*
- ➔ A través de *AppWidgetMannager*, actualizamos nuestro Widget.

Ahora vamos a personalizar sus colores, para ello nos vamos al xml y asignamos a los dos ***android:background*** el color que queramos, en este caso el verde (#00ff00).

```
        …
        …
        …
    android:background="#00ff00"

    <TextView
        android:id="@+id/appwidget_text"
        android:background="#00ff00"
        …
        …
        …
</RelativeLayout>
```

Así quedaría nuestro Widget: