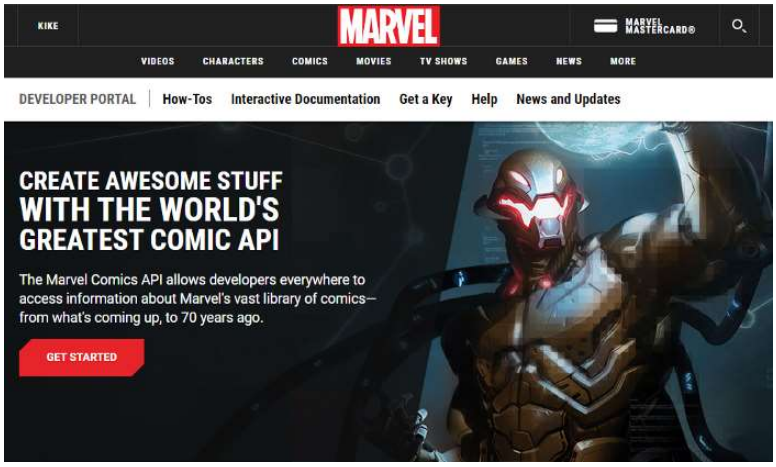
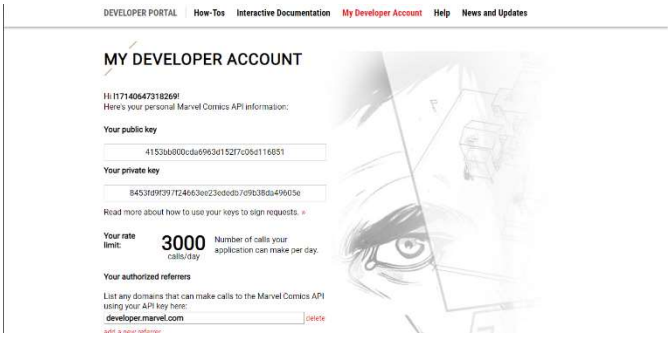
Marvel API

The first thing is to sign up on the official Marvel website for developers:

https://developer.marvel.com/





The second and essential requirement is to obtain an API Key.

In order to invoke the endpoints we need some additional parameters that I explain below:

1. ts: a timestamp
2. hash: an md5 with the following structure md5(ts+privateKey+publicKey)

Example: We have the following information

ts:3000, privateKey: 8453fd9f397f24663ee23ededb7d9b38da49605e, publicKey: 4153bb800cda6963d152f7c06d116851

For this demo at

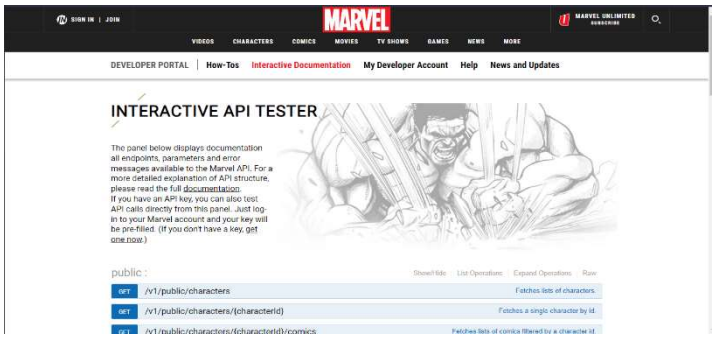
http://www.md5.cz/

After generating the hash with

md5(ts+privateKey+publicKey)

The call would be as follows:

http://gateway.marvel.com/v1/public/comics?ts=3000&apikey=4153bb800cda6963d152f7c06d1168516&hash=ed3ebe17a7443217d3cc803de8fda7dc9b



Then all that remains is to review the documentation and get started.

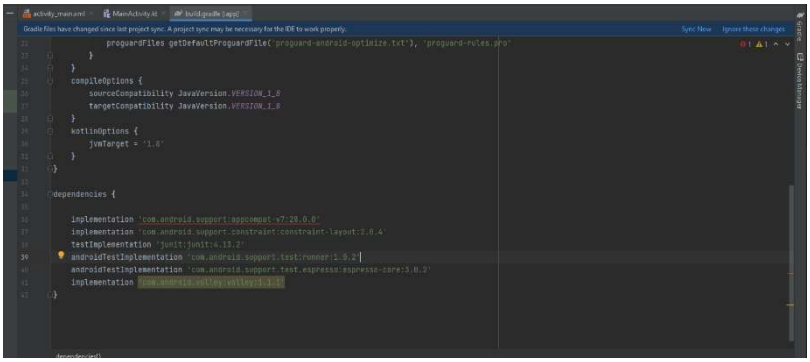
https://developer.marvel.com/docs

In order to connect to the Internet on Android, we need to add the following line to the manifest file

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

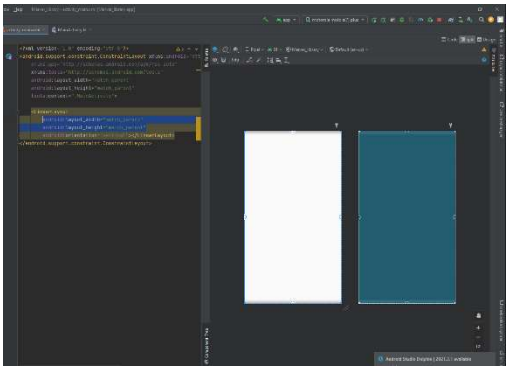
```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.aoz.marver_library">

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

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="Marver_library"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.Marver_library">
        <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>
```

We enter the official Android page and look for the Volley library: https://developer.android.com/training/volley?hl=es-419 and look for the dependency and paste it into the build.gradle(Module:) file.



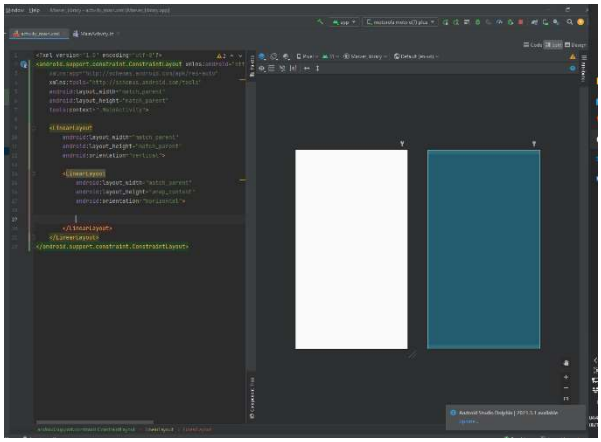
for it to be applied, you must click on “Sync Now”

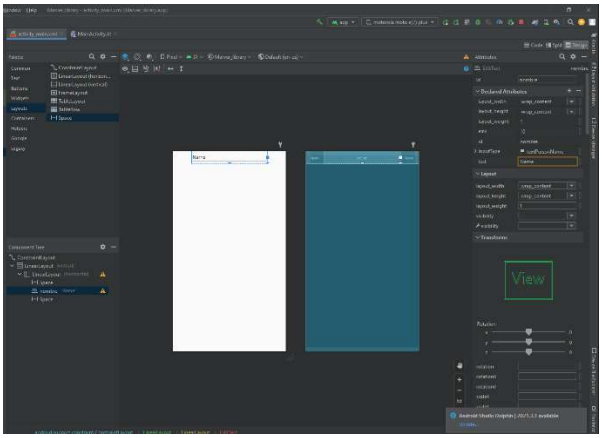
in the mainActivity we create a linearLayout that will be the vertical base to add the elements in rows, we edit the layout_with and layout_height in match_parent so that it adjusts to the maximum size of the screen either as length and width

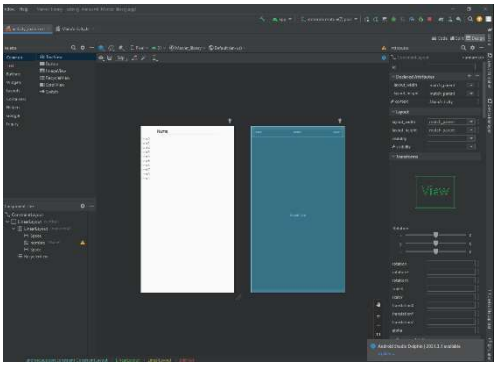


We add another horizontal linearLayout to add the elements in columns, edit the layout_with match_parent for screen length and layout_height in wrap_content for maximum objects.

In the horizontal linearLayout to add a paintext changing the id to the names and two spaces on the sides to make it centered.

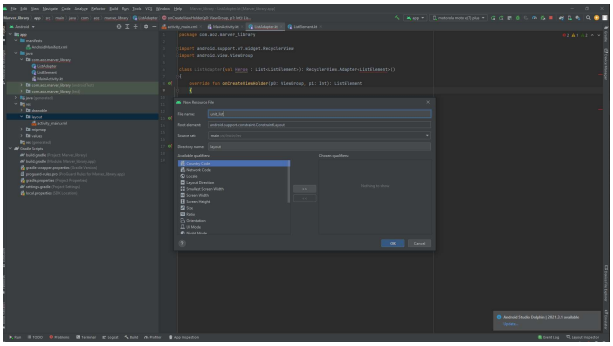




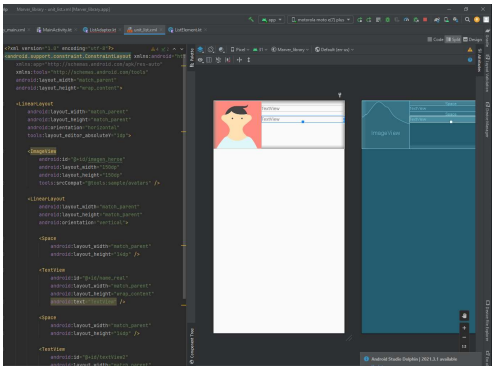


We add a RecyclerView inside the vertical linearLayout and outside the horizontal linearLayout so we don't have problems with these two. We change the id to list.

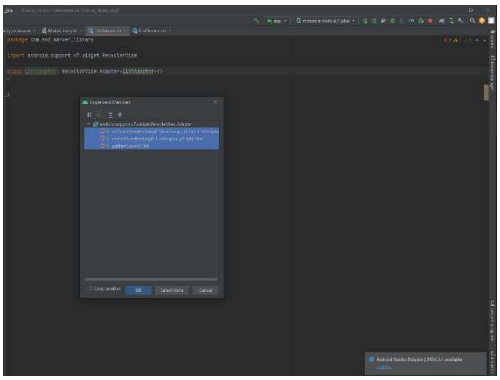
We create a Resource File in the layout folder and name it unit_list. This will be the pattern for each cell in the list.



We edit it to our liking, just like the mainActivity, making sure that the layout_height is wrap_content so that it doesn't cover the screen size and only shows the elements that we want.

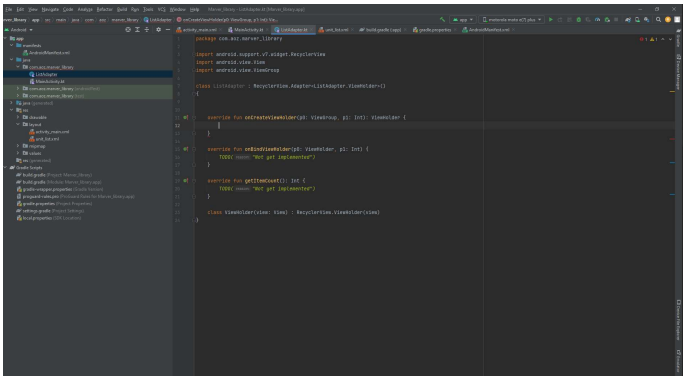


In order to use the RecyclerView we have to create 2 classes called ListAdapter and ListElement



In the list adapter we extend the RecyclerView.Adapter<ListElement> if we mark it as error, we click on implement member and we select everything

We will get the following methods and delete the content of each of them. We add at the end the class: class ViewHolder(view: View) : RecyclerView.ViewHolder(view)



In the onCreateViewHolder method we add the code:

```
val layoutInflater =
LayoutInflater.from(p0.context).inflate(R.layout.unit_list, p0, false)
return  ViewHolder(layoutInflater)
```

in this the unit_list is loaded.

In the onBindViewHolder method we add the code:

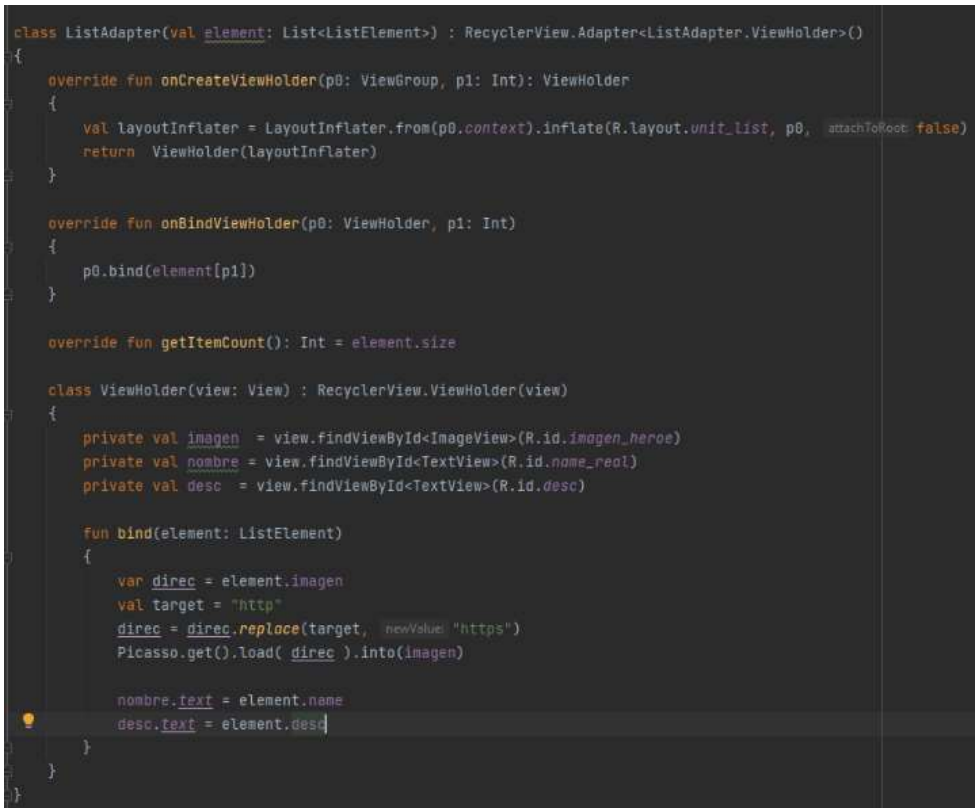
```
p0.bind(element[p1])
```

will ask you to create a method. You click create and it generates it in the ViewHolder(view: View) class: RecyclerView.ViewHolder(view)

```
element.size
```

en la clase ViewHolder(view: View) : RecyclerView.ViewHolder(view) se podra utilizer los elementos del unit_list.

This is how it looks:



```
class ListAdapter(val element: List<ListElement>) : RecyclerView.Adapter<ListAdapter.ViewHolder>()
{
    override fun onCreateViewHolder(p0: ViewGroup, p1: Int): ViewHolder
    {
        val layoutInflater = LayoutInflater.from(p0.context).inflate(R.layout.unit_list, p0, attachToRoot: false)
        return ViewHolder(layoutInflater)
    }

    override fun onBindViewHolder(p0: ViewHolder, p1: Int)
    {
        p0.bind(element[p1])
    }

    override fun getItemCount(): Int = element.size

    class ViewHolder(view: View) : RecyclerView.ViewHolder(view)
    {
        private val imagen  = view.findViewById<ImageView>(R.id.imagen_heroe)
        private val nombre = view.findViewById<TextView>(R.id.name_real)
        private val desc  = view.findViewById<TextView>(R.id.desc)

        fun bind(element: ListElement)
        {
            var direc = element.imagen
            val target = "http"
            direc = direc.replace(target, newValue: "https")
            Picasso.get().load( direc ).into(imagen)

            nombre.text = element.name
            desc.text = element.desc
        }
    }
}
```

The Picasso library will be used in the build.gradle file, in the dependencies part the line is added

```
implementation 'com.squareup.picasso:picasso:2.71828'
```

the library loads the images through the url but does not accept the http format so this code is added to replace the “http” part with “https” and implement it to Picasso:

```
var direc = element.imagen
val target = "http"
direc = direc.replace(target, "https")
Picasso.get().load( direc ).into(imagen)
```

In the ListElement it is changed to data type and the attributes are added to show us the images

```
package com.aoz.marver_library

data class ListElement( var id: Int,
                        val name : String,
                        val desc: String,
                        val modif : String,
                        val imagen : String
                        )
```

In the mainActivity a method is added to load the JSON

```
private fun acceda_a_db()
{
    val queue = Volley.newRequestQueue(this)
    val stringRequest =  StringRequest(Request.Method.GET, links(),
        Response.Listener<String>
        { response ->
            extracionjason(response)
        },
        Response.ErrorListener
        { it: VolleyError!
            mensaje( mensaje: "error. ")
        })
    queue.add(stringRequest)
}
```

In the response part -> the jason is received in this case another method was created to extract the data from JSON

```
private fun extracionjason(dato : String)
{
    try
    {
        val data = JSONObject(dato)

        val datas = data.getJSONObject( name: "data")

        val elementheroes = datas.getJSONArray( name: "results")

        for ( i in ( 0 until elementheroes.length() ) )
        {
            val obj = elementheroes.getJSONObject(i)

            val imagen = obj.getJSONObject( name: "thumbnail")
            val linkimag = imagen["path"].toString() + "." + imagen["extension"].toString()

            Element.add(
                ListElement(obj.optInt( name: "id"),
                            obj.optString( name: "name"),
                            obj.optString( name: "description"),
                            obj.optString( name: "modified"),
                            linkimag),
                )

        }
        listo.layoutManager = LinearLayoutManager( context: this)
        val adapter = ListAdapter(Element)
        listo.adapter = adapter

    }
    catch (e: JSONException)
    {
        mensaje( mensaje: "JSON Error. ${e.message.toString()}")
    }
}
```

the package received is a String so it is converted into Json and we extract the object called "data" and the list called "result" at that point the characteristics of each hero can be extracted through the numerical indices and thus be able to automate it with a

for as well We extract another object called “thumbnall” to extract the image link and its extension. We create a variable of type ArrayList<ListElement> To be able to add the data to the RecycleView

```
var Element = ArrayList<ListElement>()
```

```
for ( i in ( 0 until elementheroes.length() ) )
{
    val obj = elementheroes.getJSONObject(i)

    val imagen = obj.getJSONObject( name: "thumbnail")
    val linkimag = imagen["path"].toString() + "." + imagen["extension"].toString()

    Element.add(
        ListElement(obj.optInt( name: "id"),
                    obj.optString( name: "name"),
                    obj.optString( name: "description"),
                    obj.optString( name: "modified"),
                    linkimag),
    )

}
```

At the end of the "for" loop we start the recycleView to show us the data

So that it can be updated, we add at the beginning of the method an Element.clear() To empty the list and start another one

```
lista.layoutManager = LinearLayoutManager( context: this)
val adapter = ListAdapter(Element)
lista.adapter = adapter
```

It would look like this:

To do a search we add a global variable type editex and assign it the editex of the main activity

```
lateinit var lista : RecyclerView
lateinit var  entratotexto : EditText

class MainActivity : AppCompatActivity()
{
    override fun onCreate(savedInstanceState: Bundle?)
    {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        entratotexto = findViewById(R.id.nombre)
        lista = findViewById(R.id.lista)

        entratotexto.addTextChangedListener(object : TextWatcher
        {

            override fun afterTextChanged(s: Editable) {}

            override fun beforeTextChanged(s: CharSequence, start: Int,
                                           count: Int, after: Int)
            {

            }

            override fun onTextChanged(s: CharSequence, start: Int,
                                       before: Int, count: Int)
            {
                if(entratotexto.text == null)
                    filtro = ""
                else
                    filtro = "nameStartsWith=${entratotexto.text.toString()}&"

                mensaje(entratotexto.text.toString())
                acceda_a_db()
            }
        })
        acceda_a_db()
    }
```

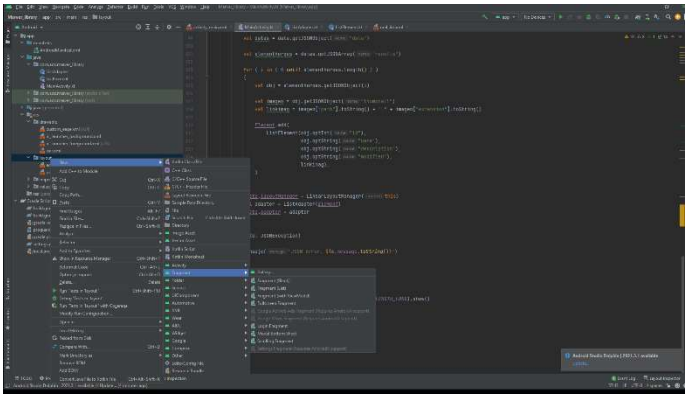
So that it is updated when adding text, we assign the method `addTextChangedListener`

So that each item in the list can be clicked, a `setOnClickListener`

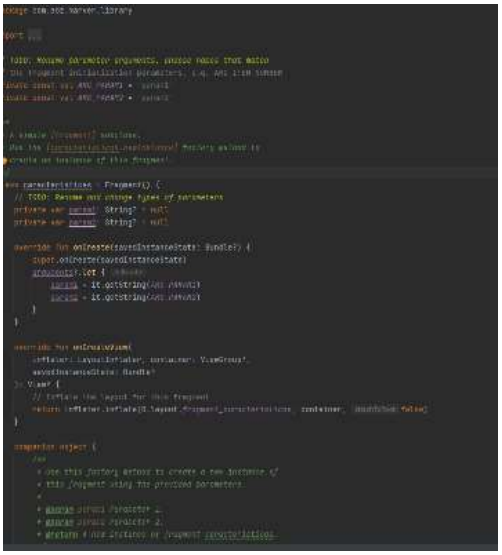
```
class ViewHolder(val view: View) : RecyclerView.ViewHolder(view)
{

    private val imagen  = view.findViewById<ImageView>(R.id.imagen_heroe)
    private val nombre = view.findViewById<TextView>(R.id.name_real)
    private val desc  = view.findViewById<TextView>(R.id.desc)

    fun bind(element: ListElement)
    {
        var direc = element.imagen
        val target = "http"
        direc = direc.replace(target, newValue: "https")
        Picasso.get().load( direc ).into(imagen)

        nombre.text = element.name
        desc.text = element.desc

        view.setOnClickListener {  }
    }
}
```

here we tell the program to do an action when an element in the recycleview is clicked

We create a fragment, to be called by the recycleview. In this case I will call it characteristics



It generates a design with the name of fragment_characteristics and a class characteristics.kt



We create a global variable rootView:View in the kt class of features. We delete everything that is not needed in the fragment and leave it in onCreateView with the return

```
Return inflater.inflate(R.layout.fragment_caracteristicas, container,
false)
```

We assign the return value to the created variable rootVew and add it to the return

```
rootView = inflater.inflate(R.layout.fragment_caracteristicas, container,
false)
return rootView
```

this is to be able to use the elements of the fragment_characteristics. Example:

```
cerrar = rootView.findViewById(R.id.cerrar)
```

From here we can edit the fragment and use the elements.

To invoke the fragment from the RecycleView we go to the ListAdapter class where we create the setOnclickListener. In the setOnclickListener we add the option to invoke the fragment, for this we create an object with the fragment class.

```
val fragmento = caracteristicas()
```

We create an object of type Bundle. To send data to the fragment.

```
val data = Bundle()
```

To save the data in the Bundle object just follow the codes:

```
data.putInt("id", element.id)
data.putString("name", element.name)
data.putString("desc", element.desc)
data.putString("mod", element.modif)
data.putString("imagen", direc)
```

We add the data as arguments to the fragment object that was created.

```
fragmento.arguments = data
```

All that remains is to invoke the fragment. For this we create a variable type mainActivity. To be able to invoke it in that context.

```
val activity = view.context as MainActivity
```

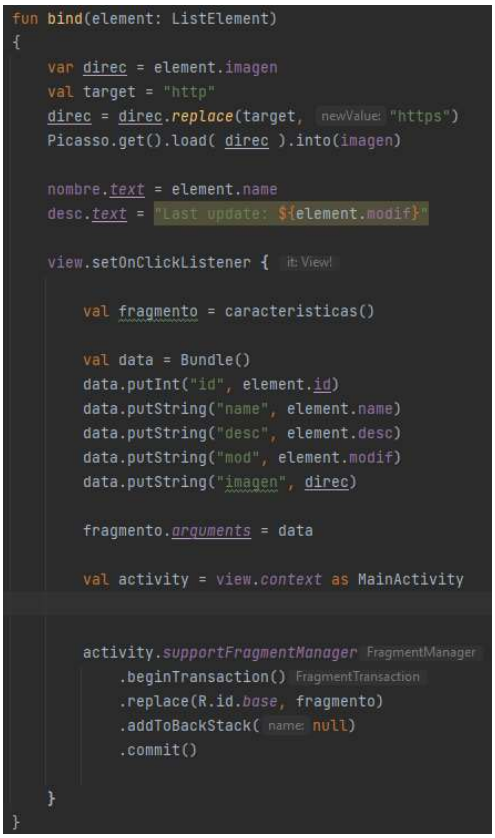
We then use this line of code to call it.

```
activity.supportFragmentManager
    .beginTransaction()
    .replace(R.id.base, fragmento)
    .addToBackStack(null)
    .commit()
```

the replace part replaces an element of the mainActiviti with the fragment

addToBackStack(null) allows to return to the state before the fragment called

the final part of the commit is the most important to perform the above actions.



```
fun bind(element: ListElement)
{
    var direc = element.imagen
    val target = "http"
    direc = direc.replace(target, newValue: "https")
    Picasso.get().load( direc ).into(imagen)

    nombre.text = element.name
    desc.text = "Last update: ${element.modif}"

    view.setOnClickListener {  it: View!

        val fragmento = caracteristicas()

        val data = Bundle()
        data.putInt("id", element.id)
        data.putString("name", element.name)
        data.putString("desc", element.desc)
        data.putString("mod", element.modif)
        data.putString("imagen", direc)

        fragmento.arguments = data

        val activity = view.context as MainActivity


        activity.supportFragmentManager FragmentManager
            .beginTransaction() FragmentTransaction
            .replace(R.id.base, fragmento)
            .addToBackStack( name: null)
            .commit()

    }
}
```

To extract the data that was sent to the fragment we create a variable in the

fragment of type argument.

```
val bundle = arguments
```

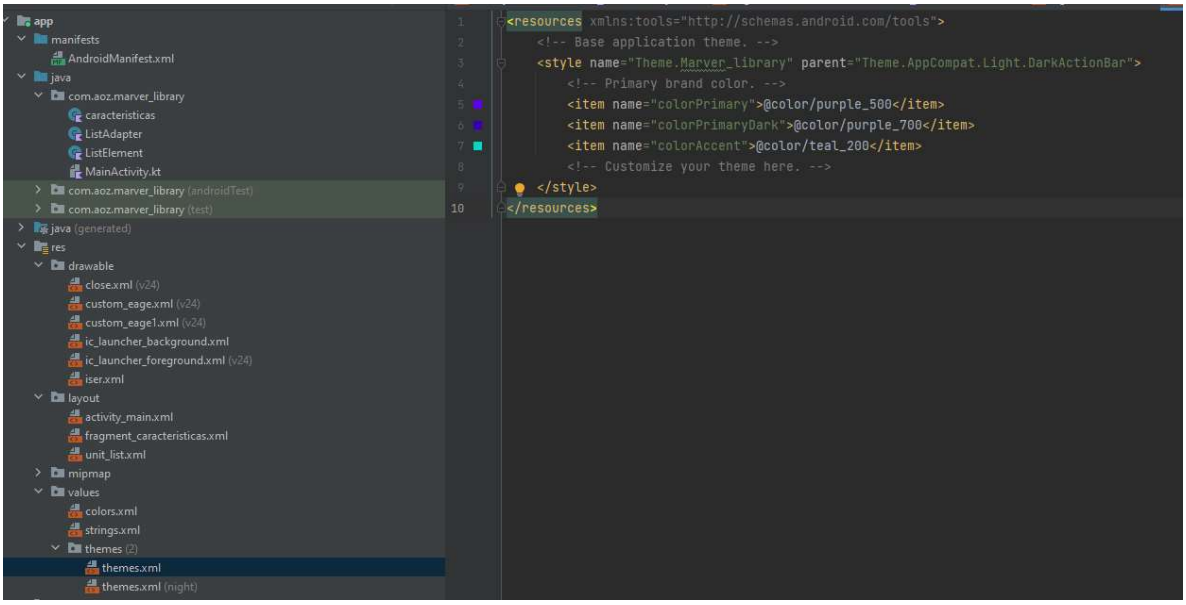
and we extract the data as in the example:

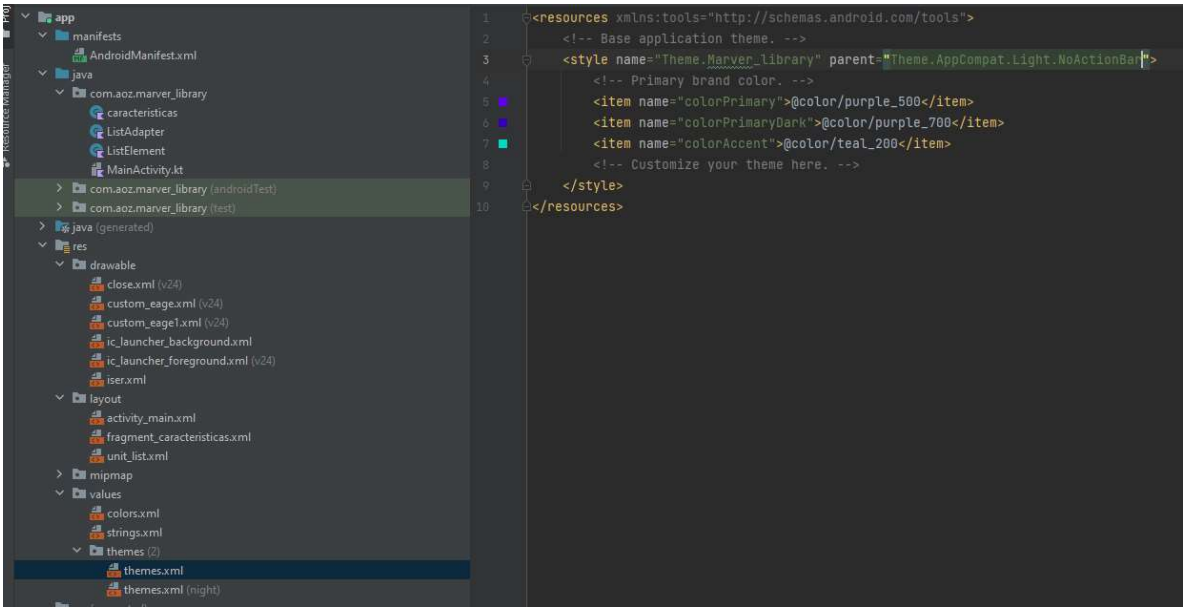
```
val ids = bundle!!.getInt("id")
val names = bundle.getString("name")
val descs = bundle.getString("desc")
val mods = bundle.getString("mod")
val imagens = bundle.getString("imagen")
```

after this you already have the data to use in the fragment.

```
override fun onCreateView(
    inflater: LayoutInflater, container: ViewGroup?,
    savedInstanceState: Bundle?
): View
{
    rootView = inflater.inflate(R.layout.fragment_caracteristicas, container,  attachToRoot: false)

    cerrar = rootView.findViewById(R.id.cerrar)
    base_transp = rootView.findViewById(R.id.base_trassp)

    id = rootView.findViewById(R.id.id)
    nombre = rootView.findViewById(R.id.name)
    desc = rootView.findViewById(R.id.description_unit)
    mod = rootView.findViewById(R.id.last_update)
    imagen = rootView.findViewById(R.id.imagen_heroe_unit)

    val bundle = arguments
    val ids = bundle!!.getInt( key: "id")
    val names = bundle.getString( key: "name")
    val descs = bundle.getString( key: "desc")
    val mods = bundle.getString( key: "mod")
    val imagens = bundle.getString( key: "imagen")

    id.text = ids.toString()
    nombre.text = names
    desc.text = descs
    mod.text = mods

    Picasso.get().load( imagens ).into(imagen)

    cerrar.setOnClickListener {  it: View!

        val activity = context as MainActivity
        activity.supportFragmentManager.beginTransaction().remove( fragment: this).commit()
    }

    base_transp.setOnClickListener {  it: View!
        val activity = context as MainActivity
        activity.supportFragmentManager.beginTransaction().remove( fragment: this).commit()
    }
    return rootView
}
```

If you want to remove the top bar of the app go to values/themes and change the first line of style to no actionbar.


app
manifests
AndroidManifest.xml
java
com.aoz.marver_library
caracteristicas
ListAdapter
ListElement
MainActivity.kt
com.aoz.marver_library (androidTest)
com.aoz.marver_library (test)
java (generated)
res
drawable
close.xml (v24)
custom_eage.xml (v24)
custom_eage1.xml (v24)
ic_launcher_background.xml
ic_launcher_foreground.xml (v24)
iser.xml
layout
activity_main.xml
fragment_caracteristicas.xml
unit_list.xml
mipmap
values
colors.xml
strings.xml
themes (2)
themes.xml
themes.xml (night)
<resources xmlns:tools="http://schemas.android.com/tools">
<!-- Base application theme. -->
<style name="Theme.Marver_library" parent="Theme.AppCompat.Light.DarkActionBar">
<!-- Primary brand color. -->
<item name="colorPrimary">@color/purple_500</item>
<item name="colorPrimaryDark">@color/purple_700</item>
<item name="colorAccent">@color/teal_200</item>
<!-- Customize your theme here. -->
</style>
</resources>


app
manifests
AndroidManifest.xml
java
com.aoz.marver_library
caracteristicas
ListAdapter
ListElement
MainActivity.kt
com.aoz.marver_library (androidTest)
com.aoz.marver_library (test)
java (generated)
res
drawable
close.xml (v24)
custom_eage.xml (v24)
custom_eage1.xml (v24)
ic_launcher_background.xml
ic_launcher_foreground.xml (v24)
iser.xml
layout
activity_main.xml
fragment_caracteristicas.xml
unit_list.xml
mipmap
values
colors.xml
strings.xml
themes (2)
themes.xml
themes.xml (night)
<resources xmlns:tools="http://schemas.android.com/tools">
<!-- Base application theme. -->
<style name="Theme.Marver_library" parent="Theme.AppCompat.Light.NoActionBar">
<!-- Primary brand color. -->
<item name="colorPrimary">@color/purple_500</item>
<item name="colorPrimaryDark">@color/purple_700</item>
<item name="colorAccent">@color/teal_200</item>
<!-- Customize your theme here. -->
</style>
</resources>