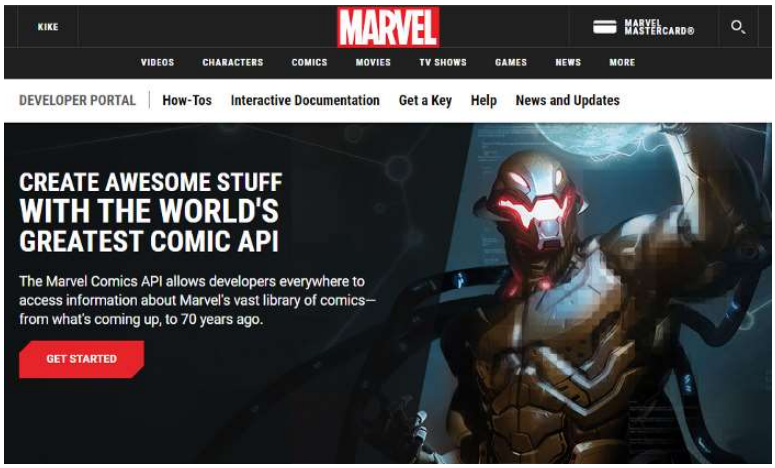
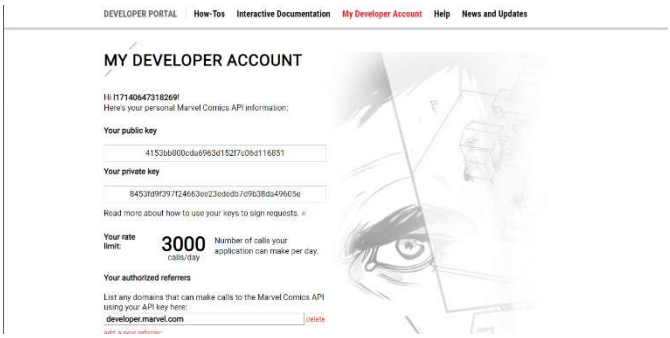
API de Marvel

Lo primero es darte de alta en la web oficial de Marvel para desarrolladores: https://developer.marvel.com/





Lo segundo y requisito indispensable es obtener una API Key.

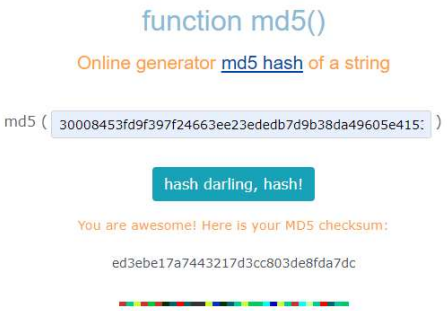
Para poder invocar los endpoints necesitamos unos parámetros adicionales que te explico a continuación:

1. **ts:** un timestamp
2. **hash:** un md5 con la siguiente estructura `md5(ts+privateKey+publicKey)`

Ejemplo: Tenemos la siguiente información

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

Para este demo en http://www.md5.cz/

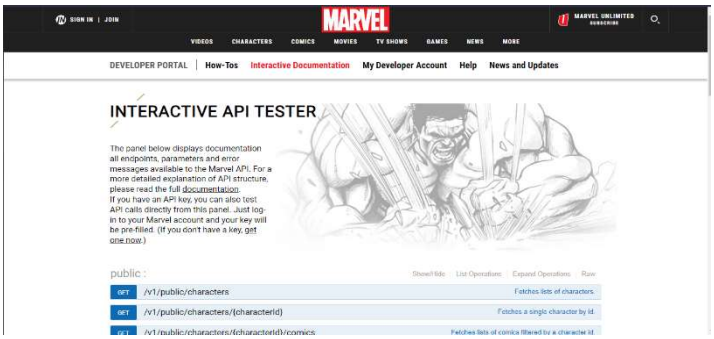


Luego de generar el hash con

md5(**ts+privateKey+publicKey**)

La llamada quedaría como sigue:

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



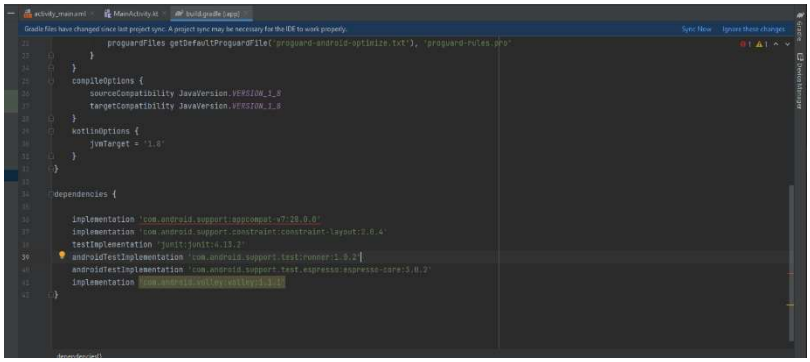
Luego solo queda revisar la documentación y empezar. https://developer.marvel.com/docs

Para podernos conectar en Android a internet se necesita agregar la siguiente línea al archivo manifestó

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

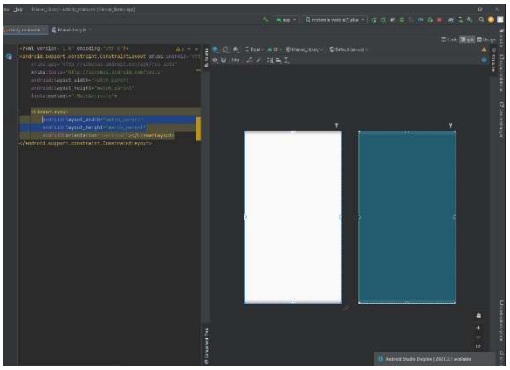
```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.aoz.marver_library">

    <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>
```

Entramos en la página oficial de Android y buscamos la librería Volley: https://developer.android.com/training/volley?hl=es-419 y buscamos la dependencia y la pegamos en el archivo build.gradle(Module:)



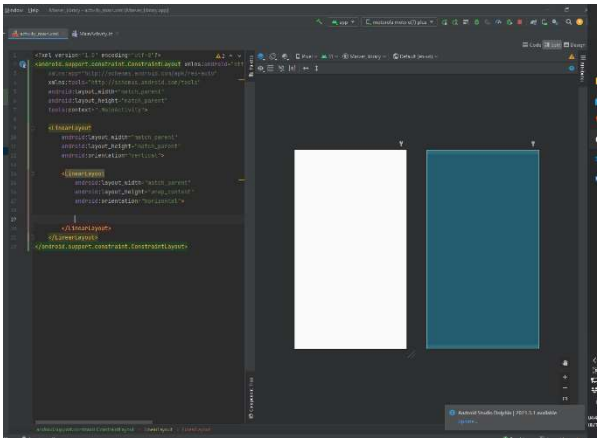
para que se aplique se debe dar clic en “Sync Now”

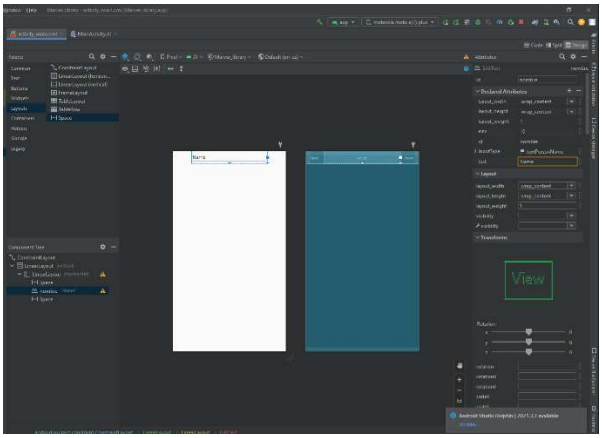
en el mainActivity creamos un linearLayout que será la base en vertical para agregar los elementos en filas, editamos el layout_with y layout_height en match_parent para que se ajuste al tamaño máximo de la pantalla ya sea como largo y ancho

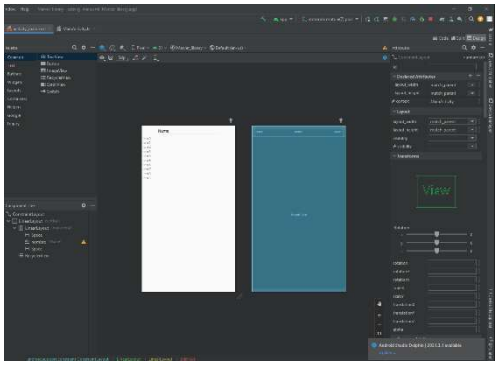


Agregamos otro linearLayout horizontalpara agregar los elementos en columnas, editamos el layout_with match_parent para el largo de la pantalla y layout_height en wrap_content para el máximo de objetos.

En el linearLayout horizontal para agregamos un paintext cambiando el id a nombres y dos espacions a los lados para que se centre.

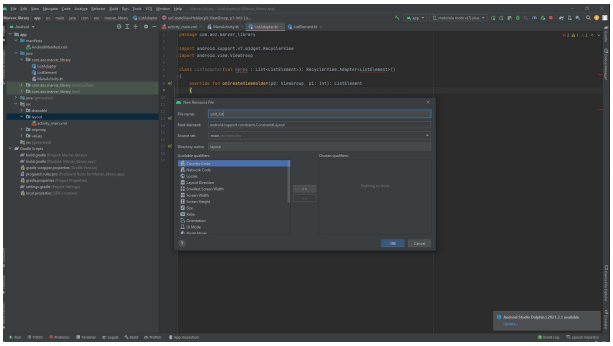




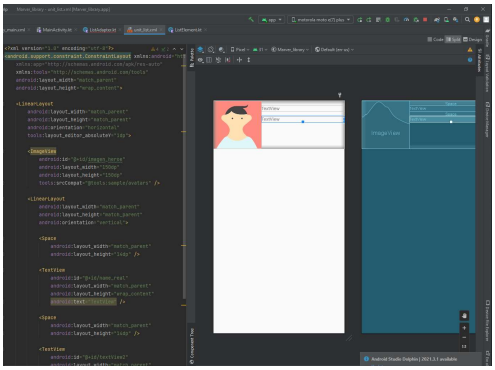


Agregamos un RecyclerView dentro del linearLayout vertical y fuera de linearLayout horizontal para no tener problemas con estos dos. Le cambiamos el id a lista.

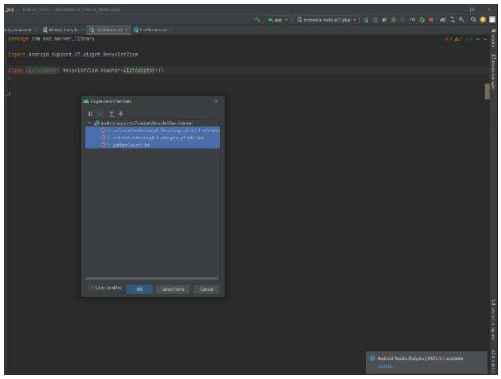
Creamos un Resource File en la carpeta layout y la nombraremos unit_list. Esta será el modelo de cada celda de la lista.



Lo editamos a nuestro gusto, igual que el mainActivity, asegurándonos que el layouy_height sea wrap_content para que no cubra el tamaño de la pantalla y solo muestre los elementos que se desea.



Para poder utilizar el RecyclerView tenemos que crear 2 clases llamadas ListAdapter y ListElement

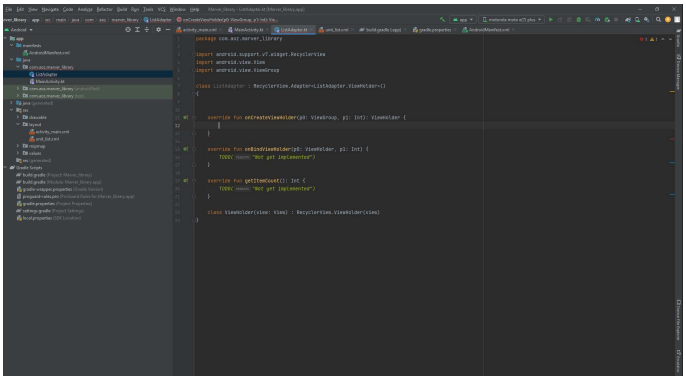


En el list adapter extendemos el RecyclerView.Adapter<ListElement> si lo marca como error, damos en implementar miembro y seleccionamos todo

Nos saldrá los siguientes métodos y borramos el contenido de cada uno de ellos.

Agregamos al final la clase: class ViewHolder(view: View) :

RecyclerView.ViewHolder(view)



En el metodo onCreateViewHolder agregamos el código:

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

en este se carga el unit_list.

En el metodo onBindViewHolder agregamos el código:

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

te pedirá que crees un método. le das en crear y lo genera en la clase

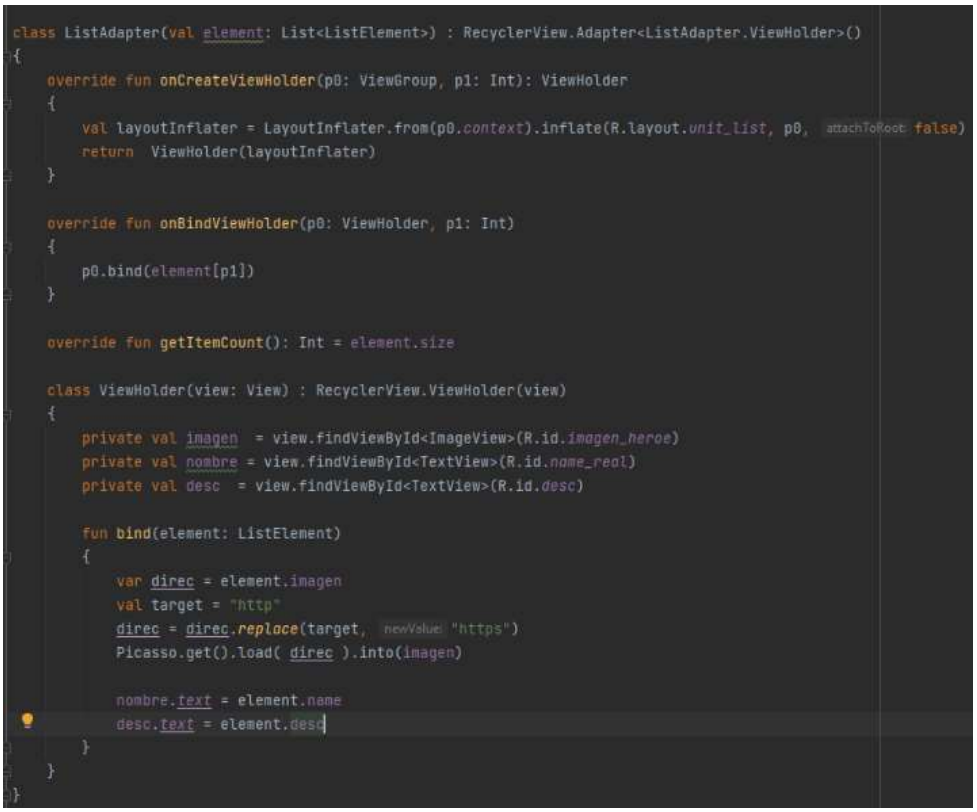
ViewHolder(view: View) : RecyclerView.ViewHolder(view)

```
element.size
```

en la clase ViewHolder(view: View) : RecyclerView.ViewHolder(view) se podra

utilizer los elementos del unit_list.

Así es como queda:



```
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
        }
    }
}
```

Se utilizara la librería de Picasso en el archivo build.gradle en la parte de dependencias se agrega la línea

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

la librería carga las imágenes mediate el url pero no acepta el formato http a si que se agrega este código para remplazar la parte de “http” a “https” e implementarlo al Picasso:

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

En el ListElement se cambia a tipo data y se le agregan los atributos para que nos muestre las imágenes.

```
package com.aoz.marver_library

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

En el mainActivity se agrega un método para cargar el 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)
}
```

En la parte de respose -> se recibe el jason en este caso se creó otro método para extraer los datos de 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),
                )

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

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

el paquete recibito es un String asi que se convierte en Json y extraemos el objeto llamado “data” y la lista llamada “result” en ese punto se puede extraer las características de cada héroe mediante los índices numéricos y así poderlo automatizar con un for también

extraemos otro objeto llamado “thumbnall” para extraer el link de la imagen y su extensión. Creamos una variable de tipo ArrayList<ListElement> Para poder agregarles los datos al 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),
    )

}
```

Al final del ciclo “for” iniciamos el recicleView para que nos muestre los datos:

Para que se pueda actualizar agregamos al inicio del método un *Element*.clear() Para vaciar la lista e inicie otra

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

Así quedaría:

Para hacer una búsqueda agregamos una variable global tipo editex y le asignamos el editex del main activiti

```
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.listo)

        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()
    }
```

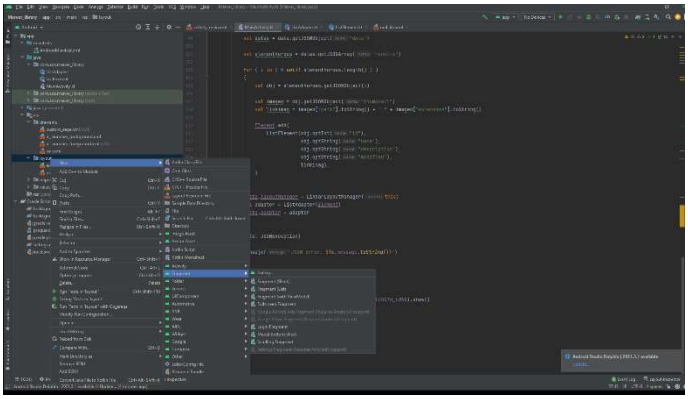
Para que se actualice al momento de agregar texto le asignamos el método `addTextChangedListener`

Para que se pueda dar clic a cada elemento de la lista se agregara un `setOnClickListener`

aquí le decimos al programa que haga una acción cuando se le de clic a un elemento del recycleview.

```
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 {  }
    }
}
```

Creamos un fragment, para que sea invocado por el recicleview. En este caso lo llamare características.



Nos generara un layout con el nombre de fragmen_caracteristicas y una clase características.kt

Creamos una variable global rootView : View en la clase kt de características.

Borramos todo lo que no se necesita del fragmento y dejamos en onCreateView con el return

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

El valor del return se lo asignamos a la variable creada rootVew y lo agregamos al return

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

esto es para poder utilizar los elementos del fragment_caracteristicas. Ejemplo:

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

A partir de aquí podremos editar el fragment y utilizar los elementos.

Para invocar el fragment desde el RecycleView nos vamos en la clase ListAdapter donde creamos el setOnclickListener. En el setOnclickListener agregamos la opción de invocar el fragment para ello creamos un objeto con la clase del fragment.

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

Creamos un objeto de tipo Bundle. Para mandar datos al fragment.

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

Para guardar los datos en el objeto Bundle solo sigue los códigos:

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

Agregamos los datos como argumentos al objeto fragment que se creo.

```
fragmento.arguments = data
```

ya solo falta invocar el fragment. Para ello creamos una variable tipo mainActivity.

Para poderlo invocar en ese contexto.

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

Después utilizamos esta linea de código para invocarlo.

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

la parte de replace remplaza un elemento del mainActiviti por el fragmen

el addToBackStack(null) permite regresar al estado antes del fragment invocado

la parte final del commit es la más importante para que realice las acciones anteriores.

```
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()

    }
}
```

Para extraer los datos que se enviaron al fragment creamos una variable en el fragmen de tipo argument.

```
val bundle = arguments
```

y extraemos los datos como en el ejemplo:

```
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")
```

después de esto ya tienes los datos para utilizarlos en el 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
}
```

Si quieres quitar la barra superior de la app se dirigen a values/themes y cambia la primera línea de style a no actionbar