REST API VOLLEY

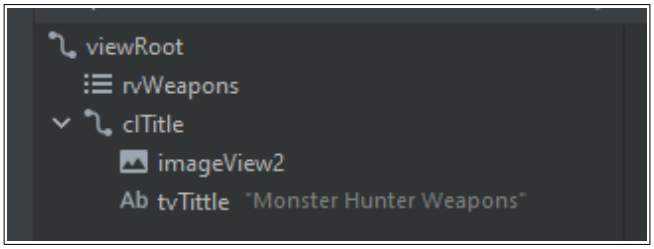
1-Lo primero que haremos será agregar las dependencias necesarias + viewBinding

```
    buildFeatures{
        viewBinding = true
    }
}

dependencies {

    implementation 'androidx.core:core-ktx:1.7.0'
    implementation 'androidx.appcompat:appcompat:1.4.0'
    implementation 'com.google.android.material:material:1.4.0'
    implementation 'androidx.constraintlayout:constraintlayout:2.1.2'
    testImplementation 'junit:junit:4.+'
    androidTestImplementation 'androidx.test.ext:junit:1.1.3'
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.4.0'

    //Volley
    implementation("com.android.volley:volley:1.2.1")

    //Glide
    implementation 'com.github.bumptech.glide:glide:4.12.0'
}
```

2-Ahora Empezaremos por el MainActivity que tenemos al crear el proyecto Empty,para empezar tenemos unas cuantas variables relacionadas con Recycleviewer ,contexto y binding

```
class MainActivity : AppCompatActivity() {
    lateinit var rvWeapons: RecyclerView//recycleviewer

    var context: Context = this
    lateinit var wList: ArrayList<Weapon> //Lista donde guardaremos las arma
    lateinit var adapter: WeaponAdapter//adaptador del recycleviewer

    private lateinit var binding: ActivityMainBinding
```

2-2 Acuerdate de modificar el XML del MainActivity y ponerle los elementos necesarios con las ids necesarias



3-Como hemos podido ver creamos un ArrayList de weapons,lo que haremos para quitar el error es crear la data class weapon (es donde almacenaremos todos los atributos que cogeremos de la API)

```
data class Weapon(
    var name: String,
    var type: String,
    var rarity: Int,
    val assets: Assets
)
```

4-Como podemos ver hay otro atributo llamado assets que en si es una nueva DataClass,esto es por que esta API en concreto tiene un atributo llamado assets el cual esta compuesta de dos imagenes(img + ico) entonces hay que crear una nueva clase para poder almacenar las dos imagenes

```
data class Assets(
        var icon: String,
        var image: String
        )
```

5-Ahora continuamos con el MainActivity,tenemos el oncreate comentado donde relizamos las tareas de inflate,inicializamos el recycleviewer y le asignamos un layout y un adapter.
Por último invocamos un metodo el cual es el encargado de realizar la tarea de obtención de datos de la API

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    binding = ActivityMainBinding.inflate(layoutInflater)
    setContentView(binding.root)

    wList = ArrayList<Weapon>() //Inicializamos la lista de armas

    //Atributos del recycleviewer
    rvWeapons = binding.rvWeapons
    rvWeapons.setHasFixedSize(true)
    rvWeapons.visibility = View.VISIBLE
    rvWeapons.layoutManager = LinearLayoutManager( context: this,LinearLayoutManager.VERTICAL, reverseLayout: true) //layoutManager
    adapter = WeaponAdapter(context, wList)//Creamos adaptador
    rvWeapons.adapter = adapter//asignamos adaptador

    getData()//invocamos funcion de coger datos
}
```

6-Antes de seguir con el metodo principal getData vamos a seguir el camino del recycleviewer viendo primero su adaptador,que es una clase llamada WeaponAdapter.
Aunque ya esta explicado en si voy a comentar mas o menos de que trata.
Para crear un Adapter le tenemos que pasar un ViewHolder que luego veremos.
Luego tenemos 3 metodos:

-1 → Encargado de inflar el ViewHolder con el layout del item a repetir
-2 → Cuenta el numero de items
-3→ Le pasa todos los weapons al viewHolder

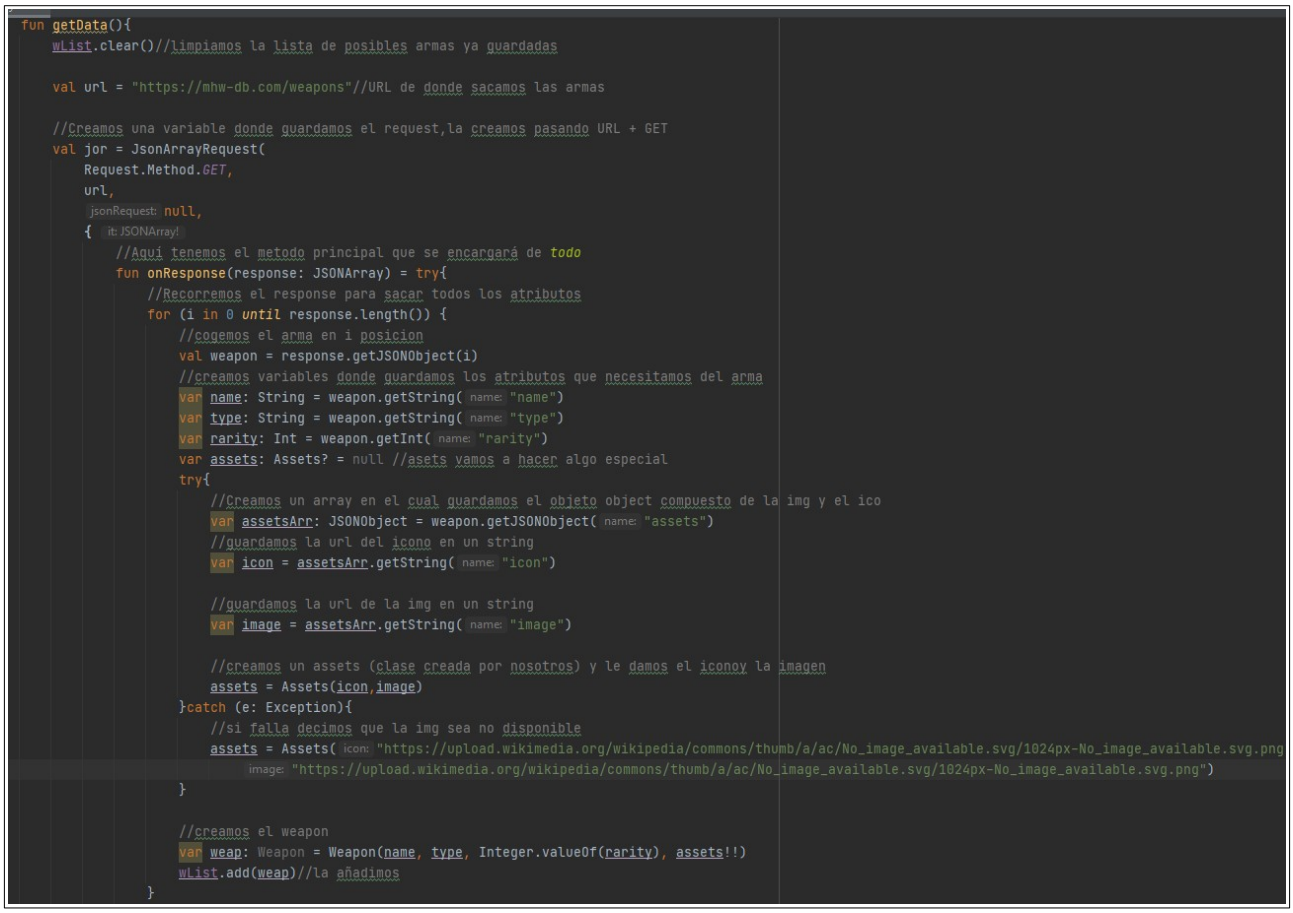
```
class WeaponAdapter(val context: Context, val wList: List<Weapon>):
    //Esta linea crea en si el adapter diciendole que utilice la clase WeaponViewHolder
    RecyclerView.Adapter<WeaponViewHolder>() {

    //Este metodo infla el ViewHolder con el Weapon_item que se debe repetir
    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): WeaponViewHolder {
        val inflater = LayoutInflater.from(parent.context)
        return WeaponViewHolder(inflater.inflate(R.layout.weapon_item,parent, attachToRoot: false))
    }

    //ver todos los items contados
    override fun getItemCount(): Int = wList.size

    //asignamos al holder los objetos de la lista
    override fun onBindViewHolder(holder: WeaponViewHolder, position: Int) {
        val weapon = wList[position]
        holder.bind(weapon)
    }
}
```

6-2 Antes de seguir con el ViewHolder tienes que crear el weapon_item,simplemente copialo,te dejo una imagen de su aspecto(en este caso necesirás algunos archivos como un xml llamado imageborder)

7-Ahora vamos a ver la clase WeaponViewHolder que es la clase que en si es el ViewHolder del adaoter.
Esta ya explicado en la IMG que hace cada cosa

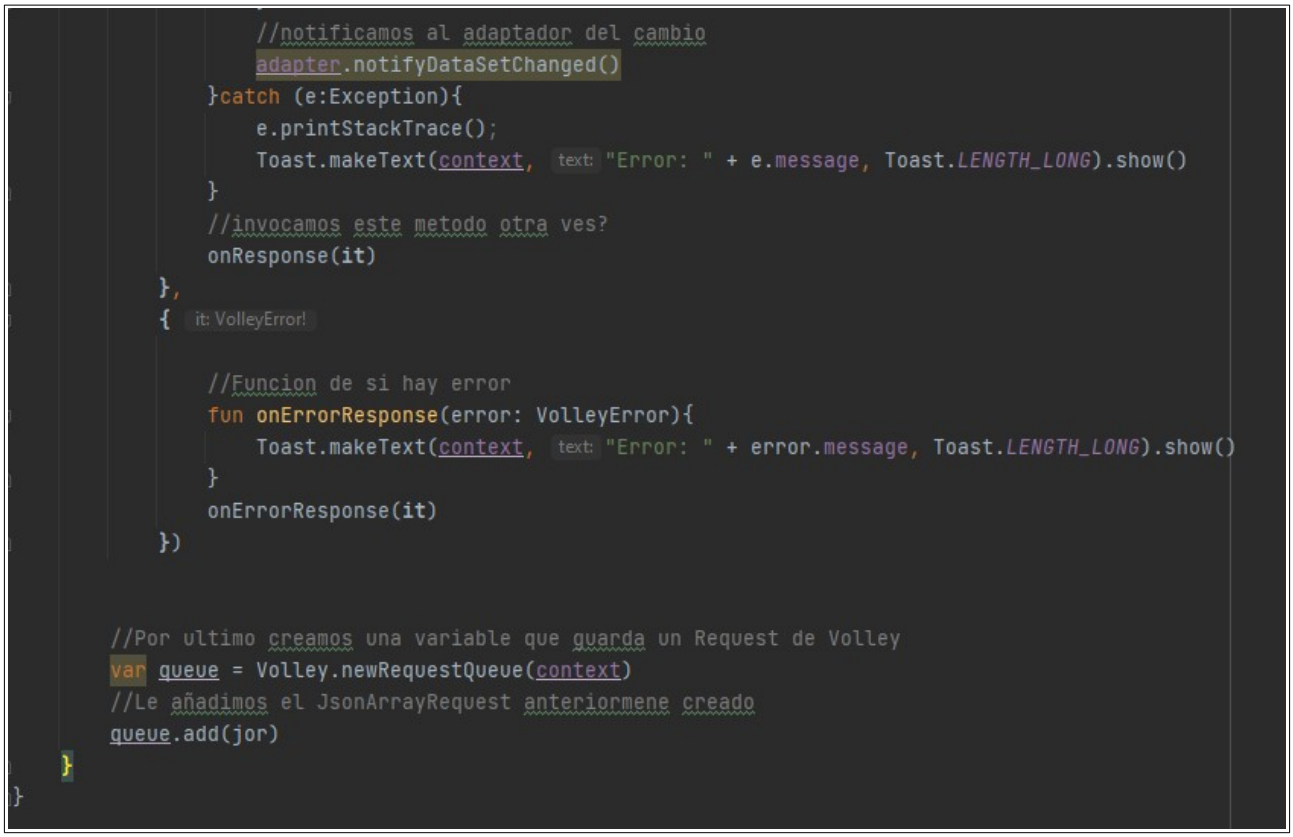
```
class WeaponViewHolder(view: View): RecyclerView.ViewHolder(view){
    private val binding = WeaponItemBinding.bind(view)//binding
    private val view: View = view//vista que recibe

    //campos el  wapon_item(oobjeto repetido)
    var tvTitle: TextView = binding.tvTitle
    var tvType: TextView = binding.tvType
    var tvRarity: TextView = binding.tvRarity
    var ivWeapon: ImageView = binding.ivWeapon
    var ivType: ImageView = binding.ivType

    //en este metodo recibimos todas las armas
    fun bind(weapon: Weapon){
        //Asignamos los atributos a los campos visibles
        tvTitle.text = weapon.name
        tvType.text = weapon.type
        tvRarity.text = weapon.rarity.toString()

        //Utilizamos glide accediendo a los assets de la weapon para asignar las imagenes donde tocan
        Glide.with(view).load(weapon.assets.image).fitCenter().override( width: 350, height: 350).into(ivWeapon)
        Glide.with(view).load(weapon.assets.icon).fitCenter().override( width: 50, height: 50).into(ivType)

    }
}
```

8-Aqui voy a mostrar el metodo principal el cual esta explicado de p a pa



```
fun getData(){
    wList.clear()//limpiamos la lista de posibles armas ya guardadas

    val url = "https://mhw-db.com/weapons"//URL de donde sacamos las armas

    //Creamos una variable donde guardamos el request,la creamos pasando URL + GET
    val jor = JsonArrayRequest(
        Request.Method.GET,
        url,
        jsonRequest: null,
        { it: JSONArray!
            //Aqui tenemos el metodo principal que se encargará de todo
            fun onResponse(response: JSONArray) = try{
                //Recorremos el response para sacar todos los atributos
                for (i in 0 until response.length()) {
                    //cogemos el arma en i posicion
                    val weapon = response.getJSONObject(i)
                    //creamos variables donde guardamos los atributos que necesitamos del arma
                    var name: String = weapon.getString( name: "name")
                    var type: String = weapon.getString( name: "type")
                    var rarity: Int = weapon.getInt( name: "rarity")
                    var assets: Assets? = null //asets vamos a hacer algo especial
                    try{
                        //Creamos un array en el cual guardamos el objeto object compuesto de la img y el ico
                        var assetsArr: JSONObject = weapon.getJSONObject( name: "assets")
                        //guardamos la url del icono en un string
                        var icon = assetsArr.getString( name: "icon")

                        //guardamos la url de la img en un string
                        var image = assetsArr.getString( name: "image")

                        //creamos un assets (clase creada por nosotros) y le damos el iconoy la imagen
                        assets = Assets(icon,image)
                    }catch (e: Exception){
                        //si falla decimos que la img sea no disponible
                        assets = Assets( icon: "https://upload.wikimedia.org/wikipedia/commons/thumb/a/ac/No_image_available.svg/1024px-No_image_available.svg.png
                            image: "https://upload.wikimedia.org/wikipedia/commons/thumb/a/ac/No_image_available.svg/1024px-No_image_available.svg.png")
                    }

                    //creamos el weapon
                    var weap: Weapon = Weapon(name, type, Integer.valueOf(rarity), assets!!)
                    wList.add(weap)//la añadimos
                }
```

ESTO VA JUSTO DESPUÉS DE LO DE ARRIBA



```
                    //notificamos al adaptador del cambio
                    adapter.notifyDataSetChanged()
                }catch (e:Exception){
                    e.printStackTrace();
                    Toast.makeText(context, text: "Error: " + e.message, Toast.LENGTH_LONG).show()
                }
                //invocamos este metodo otra ves?
                onResponse(it)
            },
            { it: VolleyError!

                //Funcion de si hay error
                fun onErrorResponse(error: VolleyError){
                    Toast.makeText(context, text: "Error: " + error.message, Toast.LENGTH_LONG).show()
                }
                onErrorResponse(it)
            })


        //Por ultimo creamos una variable que guarda un Request de Volley
        var queue = Volley.newRequestQueue(context)
        //Le añadimos el JsonArrayRequest anteriormene creado
        queue.add(jor)
    }
}
```