



cpi'fp

Los Enlaces

PROGRAMACIÓN MULTIMEDIA Y DISPOSITIVOS MÓVILES

2º DESARROLLO DE APLICACIONES MULTIPLATAFORMA

TEMA 6. APIs

0. DEFINICIONES

El término API es una abreviatura de Application Programming Interface.

Se trata de un conjunto de definiciones y protocolos que se utiliza para desarrollar e integrar el software de las aplicaciones, permitiendo la comunicación entre dos aplicaciones de software a través de un conjunto de reglas.

Una API es un intermediario entre dos sistemas, que permite que una aplicación se comuniquen con otra y pida datos o acciones específicas. Un conjunto de funciones y procedimientos que proporciona una biblioteca para uso de otro software como una capa de abstracción.

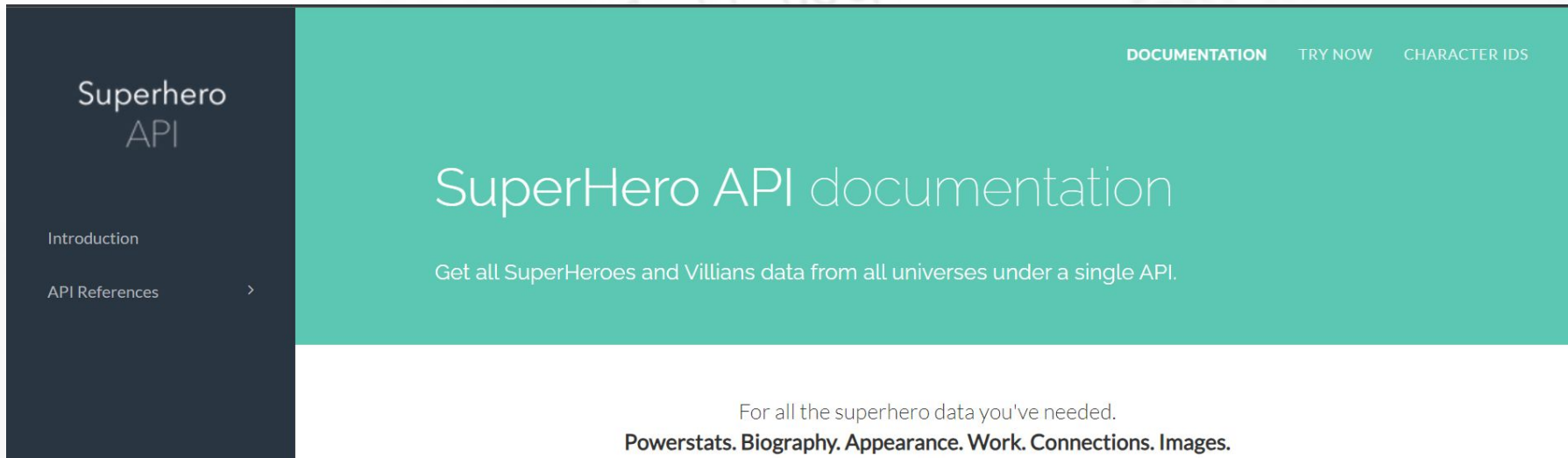
0. DEFINICIONES

Ejemplos de APIs:

- **REST Countries v1**, que provee datos sobre todos los países del mundo.
- **Skyscanner Flight Search**, que es un motor de búsqueda sobre vuelos, hoteles, alquiler de coches,...
- **RAWG**, que es una base de datos de videojuegos.
- **BGG XML API**, que contiene toda la información de la página BoardGameGeek sobre juegos de mesa.

1. APLICACIÓN CON CONSUMO DE API

Vamos a diseñar una aplicación sobre superhéroes utilizando la siguiente API de acceso gratuito para desarrolladores:



The screenshot shows the documentation page for the SuperHero API. The page has a dark sidebar on the left with the title 'Superhero API' and two menu items: 'Introduction' and 'API References' with a right-pointing arrow. The main content area has a teal header with navigation links 'DOCUMENTATION', 'TRY NOW', and 'CHARACTER IDS'. Below the header, the title 'SuperHero API documentation' is displayed in large white text, followed by the subtitle 'Get all SuperHeroes and Villians data from all universes under a single API.' in smaller white text. At the bottom of the page, on a white background, is the text 'For all the superhero data you've needed. Powerstats. Biography. Appearance. Work. Connections. Images.'


1. APLICACIÓN CON CONSUMO DE API

El único requisito para utilizar esta API es el uso de una cuenta de Facebook:

Superhero
API

Introduction

API References >


 Introduction

What is this?
The superhero API, is a quantified and programmatically accessible data source of all superheroes from both the comic universe. We've taken all the stuff and put it together in a form that is easier to consume with software. Then we made an API so you can consume it in a hassle free manner.

How can I use it?
The data is accessible through a REST API. Consult our documentation if you'd like to get started. Helper libraries are also provided so you can consume the API in the choice of your language.

Getting your Access Token
You need a facebook account to get your access token. You can generate your access token below

Access Token Here

 Continue with Facebook

1. APLICACIÓN CON CONSUMO DE API

Si nos identificamos con cualquier cuenta de Facebook, ya sea real o ficticia, se generará un token que nos permitirá acceder a la base de datos e interactuar con ella desde la aplicación que estemos desarrollando.

También nos permite utilizar la pestaña TRY NOW del sitio web para visualizar la información que recibirá la aplicación en formato JSON (JavaScript Object Notation).

Éste es un formato de texto pensado para el intercambio de datos, con una estructura en forma de árbol con dos tipos de elementos únicamente (arrays y objects).

1. APLICACIÓN CON CONSUMO DE API

Los arrays son listas de valores separados por comas y se escriben entre corchetes [].

```
[1, "pepe", 3.14, "Hola mundo"]
```

Los objetos son listas de parejas nombre/valor. El nombre y el valor están separados por dos puntos : y las parejas están separadas por comas. Los objetos se escriben entre llaves { } y los nombres de las parejas se escriben siempre entre comillas dobles.

```
{"nombre": "pepe", "edad": 25, "carnet conducir": true}
```

1. APLICACIÓN CON CONSUMO DE API

Un documento JSON está formado por un único elemento (un objeto o una matriz). Tanto en los objetos como en las matrices, el último elemento no puede ir seguido de una coma. Los espacios en blanco y los saltos de línea no son significativos.

```
[  
  {  
    "nombre": "Ana Barberá",  
    "edad": 25,  
    "carnet de conducir": true  
  },  
  {  
    "nombre": "José Esteban",  
    "edad": 90,  
    "carnet de conducir": false  
  }  
]
```

```
{  
  "nombre": "José Esteban",  
  "edad": 90,  
  "carnet de conducir": false  
}
```


1. APLICACIÓN CON CONSUMO DE API

Así, si entramos a la pestaña TRY NOW de <https://superheroapi.com> podemos ver la estructura de información de cualquier superhéroe:

Get all SuperHeroes and Villians data from all universes under a single API.

`https://superheroapi.com/api/access-token/` `search/iron%20man`

request

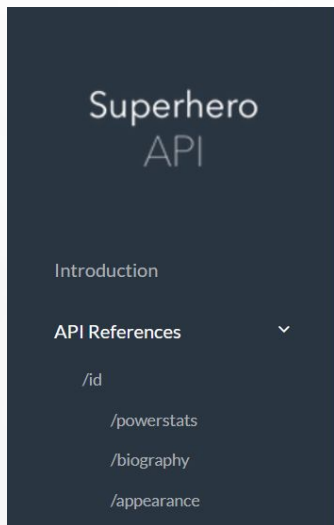
Need a hint? try `search/ironman` or `644`

```
1 {
2   "response": "success",
3   "results-for": "iron man",
4   "results": [
5     {
6       "id": "346",
7       "name": "Iron Man",
8       "powerstats": {
9         "intelligence": "100",
10        "strength": "85",
11        "speed": "73",
12        "durability": "100",
13        "energy": "100",
14        "combat": "85"
15      }
16    }
17  ]
18 }
```

JSON

1. APLICACIÓN CON CONSUMO DE API

Ésta será la información que podremos insertar en nuestra aplicación a través de llamadas a la API por URL facilitadas en la documentación:



API References

`https://superheroapi.com/api/``access-token`

BASE URL

API HEIRARCHY

- `/id`
- `/powerstats`
- `/biography`
- `/appearance`
- `/work`
- `/connections`

1. APLICACIÓN CON CONSUMO DE API

Empezamos la aplicación como siempre creando un nuevo *Package* en el directorio principal de nuestro proyecto, de nombre *SuperheroApp*.

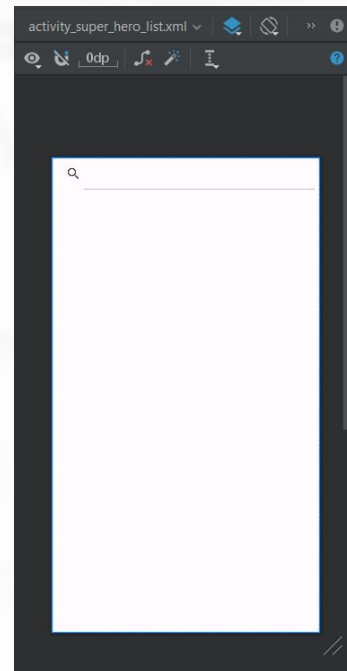
En este directorio creamos una nueva *Empty Views Activity* llamada *SuperheroListActivity* e insertamos el botón necesario para acceder a ella en *MenuActivity*.

Más adelante crearemos otra pantalla llamada *SuperheroDetailActivity* a la que accederemos desde el listado de superhéroes que vamos a diseñar y que ofrecerá una vista con los detalles del superhéroe seleccionado por el usuario.

2. LAYOUT PRINCIPAL

Vamos a insertar en la parte superior una barra de búsqueda para que el usuario pueda escribir el nombre de un superhéroe que mostrar en pantalla.

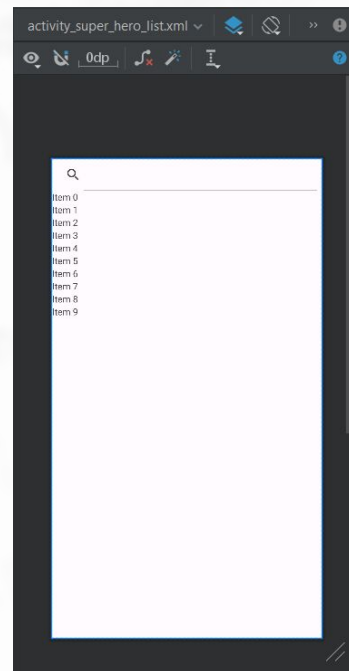
```
<androidx.appcompat.widget.SearchView
    android:id="@+id/searchView"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    app:iconifiedByDefault="false" />
<!--Este último atributo muestra el cuadro de texto completo en
vez de únicamente la lupa-->
```



2. LAYOUT PRINCIPAL

En SuperHero API, para un nombre de superhéroe, se puede dar el caso de que aparezcan varios resultados, por lo que mostraremos éstos mediante un RecyclerView.

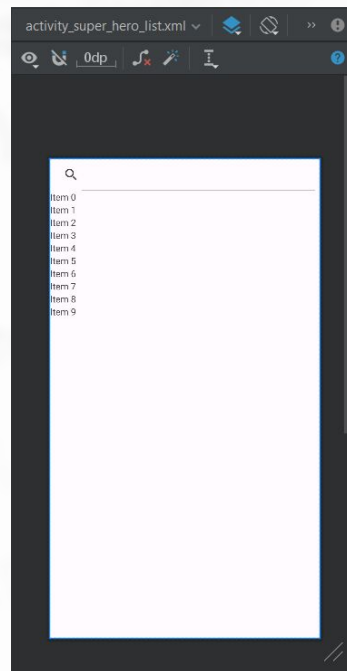
```
<androidx.recyclerview.widget.RecyclerView  
    android:id="@+id/rvSuperhero"  
    android:layout_width="0dp"  
    android:layout_height="0dp"  
    app:layout_constraintBottom_toBottomOf="parent"  
    app:layout_constraintEnd_toEndOf="parent"  
    app:layout_constraintStart_toStartOf="parent"  
    app:layout_constraintTop_toBottomOf="@+id/searchView" />
```



2. LAYOUT PRINCIPAL

Y como nuestra aplicación va a realizar búsquedas sobre un servidor externo, cuyo tiempo de respuesta no podemos controlar, vamos a añadir una ProgressBar.

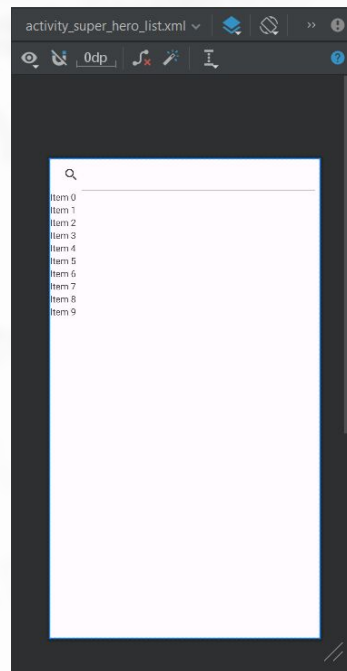
```
<ProgressBar
    android:id="@+id/progressBar"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:visibility="gone"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent" />
```



2. LAYOUT PRINCIPAL

El atributo *visibility* puede tomar los valores *visible*, *invisible* o *gone*. Este último sirve para que la *ProgressBar* sea invisible y además no interfiera con otros Views.

```
<ProgressBar  
    android:id="@+id/progressBar"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:visibility="gone"  
    app:layout_constraintBottom_toBottomOf="parent"  
    app:layout_constraintEnd_toEndOf="parent"  
    app:layout_constraintStart_toStartOf="parent"  
    app:layout_constraintTop_toTopOf="parent" />
```





3. VIEWBINDING

Google proporciona una librería que permite acceder a los Views de los layouts de una forma directa, sin necesidad de utilizar el método *findViewById*. Para ello, en **build.gradle(Module:app)** añadimos:

```
kotlinOptions {  
    jvmTarget = "1.8"  
}  
buildFeatures{  
    viewBinding = true //Además, hay que darle a sincronizar  
}  
  
dependencies {  
    implementation("androidx.core:core-ktx:1.9.0")  
}
```




3. VIEWBINDING

Y en la actividad principal definiremos un objeto tipo *binding* (empezando a escribir *Activity...* en la clase asignada a la variable) que nos permitirá acceder a las Views del layout asociado a la actividad en la que estemos.

```
class SuperHeroListActivity : AppCompatActivity() {  
    private lateinit var binding: ActivitySuperHeroListBinding  
  
    override fun onCreate(savedInstanceState: Bundle?) {  
        super.onCreate(savedInstanceState)  
        setContentView(R.layout.activity_super_hero_list)  
    }  
}
```



3. VIEWBINDING

Pero además, en el método *onCreate*, debemos cambiar la forma de acceder al layout definiendo un *LayoutInflater* y modificando la función *setContentView*.

```
class SuperHeroListActivity : AppCompatActivity() {  
  
    private lateinit var binding: ActivitySuperHeroListBinding  
  
    override fun onCreate(savedInstanceState: Bundle?) {  
        super.onCreate(savedInstanceState)  
        binding = ActivitySuperHeroListBinding.inflate(layoutInflater)  
        setContentView(binding.root)  
    }  
}
```



3. VIEWBINDING

Todo esto nos permite ahora implementar métodos, cambiar atributos, etc..., en cualquiera de nuestras Views sin necesidad de definir variables e inicializar componentes para cada una.

```
override fun onCreate(savedInstanceState: Bundle?) {  
    super.onCreate(savedInstanceState)  
    binding = ActivitySuperHeroListBinding.inflate(layoutInflater)  
    setContentView(binding.root)  
  
    binding.searchView.setOnClickListener{  
        binding.rvSuperhero.adapter = ...  
        binding.progressBar.visibility = ...  
    }  
}
```

3. VIEWBINDING

Lo primero que vamos a preparar es el *SearchView*. Invocamos a la función *initUI()* dentro del método *onCreate*, borrando las líneas de ejemplo anteriores, y la creamos fuera de este método.

```
override fun onCreate(savedInstanceState: Bundle?) {  
    super.onCreate(savedInstanceState)  
    binding = ActivitySuperHeroListBinding.inflate(layoutInflater)  
    setContentView(binding.root)  
  
    initUI()  
}  
  
private fun initUI() {  
    TODO("Not yet implemented")  
}
```

3. VIEWBINDING

El *Listener* que vamos a utilizar actúa cuando el usuario pulsa la lupa que aparece en el teclado al introducir texto en este tipo de View. Su sintaxis es distinta a los métodos utilizados hasta ahora.

```
private fun initUI() {  
    binding.searchView.setOnQueryTextListener(object: SearchView.OnQueryTextListener {  
          
    })  
}
```

3. VIEWBINDING

Se le ha de pasar un objeto como parámetro y las llaves van en el método *OnQueryTextListener* de dicho objeto. Una vez configurado este método, AndroidStudio nos facilita los métodos que debe contener (Alt + Intro).

```
private fun initUI() {  
    binding.searchView.setOnQueryTextListener(object: SearchView.OnQueryTextListener  
    {  
        override fun onQueryTextSubmit(query: String?): Boolean {  
            TODO("Not yet implemented")  
        }  
        override fun onQueryTextChange(newText: String?): Boolean {  
            TODO("Not yet implemented")  
        }  
    })  
}
```

3. VIEWBINDING

El primer método es el *Listener* que muestra el resultado cuando el usuario le da a la lupa de búsqueda, mientras que el segundo muestra la búsqueda conforme se está escribiendo. Éste lo dejaremos en `false`.

```
private fun initUI() {  
    binding.searchView.setOnQueryTextListener(object: SearchView.OnQueryTextListener  
    {  
        override fun onQueryTextSubmit(query: String?): Boolean {  
            TODO("Not yet implemented")  
        }  
        override fun onQueryTextChange(newText: String?) = false  
    })  
}
```

3. VIEWBINDING

Dentro del primer método vamos a crear una nueva función *searchByName* que definiremos fuera de la función *initUI()*. En este método siempre se devuelve un *false*.

```
private fun initUI() {  
    binding.searchView.setOnQueryTextListener(object: SearchView.OnQueryTextListener  
    {  
        override fun onQueryTextSubmit(query: String?): Boolean {  
            searchByName(query)  
            return false  
        }  
        override fun onQueryTextChange(newText: String?) = false  
    })  
}
```


3. VIEWBINDING

Pero si creamos esta función con las acciones contextuales (Alt + Intro) nos pasa la consulta como parámetro *nullable*, lo que puede provocar errores en la ejecución, por lo que indicaremos...

```
private fun initUI() {  
    binding.searchView.setOnQueryTextListener(object: SearchView.OnQueryTextListener  
    {  
        override fun onQueryTextSubmit(query: String?): Boolean {  
            searchByName(query.orEmpty())  
        }  
        override fun onQueryTextChange(newText: String?) = false  
    })  
}  
private fun searchByName(query: String) {}
```

4. CONSUMO DE API

Ahora sí, necesitamos acceder a la API que queremos consumir para mostrar los resultados de búsqueda de nuestra aplicación.

Si leemos la documentación facilitada en <https://superheroapi.com> vemos que podemos hacer distintas llamadas a la base de datos para obtener cierta información.

Por ejemplo, si en la barra de búsqueda del navegador insertamos la *baseURL*, `https://superheroapi.com/api/{access-token}`, con el fragmento `/search/batman`, la API nos devuelve un archivo de texto plano en formato JSON.

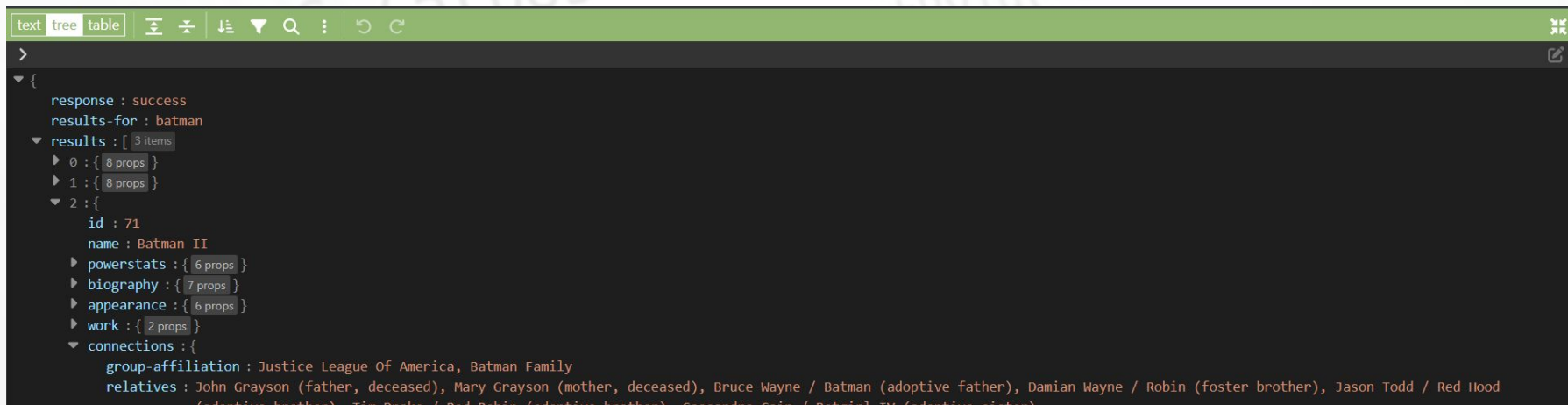
4. CONSUMO DE API

El problema es que este código está sin formatear y es casi ilegible. Podemos utilizar un editor de código como VSCode, el cual nos permite formatearlo con Alt + Shift + F, buscar un editor JSON online o utilizar la pestaña TRY NOW que facilita la propia API.

```
{
  "response": "success",
  "results-for": "batman",
  "results": [
    {
      "id": "69",
      "name": "Batman",
      "powerstats": {
        "intelligence": "81",
        "strength": "40",
        "speed": "29",
        "durability": "55",
        "power": "63",
        "combat": "90"
      },
      "biography": {
        "full-name": "Terry McGinnis",
        "alter-egos": "No alter egos found.",
        "aliases": ["Batman II", "The Tomorrow Knight", "The second Dark Knight", "The Dark Knight of Tomorrow", "Batman Beyond"],
        "place-of-birth": "Gotham City, 25th Century",
        "first-appearance": "Batman Beyond #1",
        "publisher": "DC Comics",
        "alignment": "good",
        "appearance": {
          "gender": "Male",
          "race": "Human",
          "height": ["5'10", "178 cm"],
          "weight": ["170 lb", "77 kg"],
          "eye-color": "Blue",
          "hair-color": "Black"
        },
        "work": {
          "occupation": "-",
          "base": "21st Century Gotham City"
        },
        "connections": {
          "group-affiliation": "Batman Family, Justice League Unlimited",
          "relatives": "Bruce Wayne (biological father), Warren McGinnis (father, deceased), Mary McGinnis (mother), Matt McGinnis (brother)"
        },
        "image": {
          "url": "https://www.superherodb.com/pictures2/portraits/10/100/10441.jpg"
        }
      },
      "id": "70",
      "name": "Batman",
      "powerstats": {
        "intelligence": "100",
        "strength": "26",
        "speed": "27",
        "durability": "50",
        "power": "47",
        "combat": "100"
      },
      "biography": {
        "full-name": "Bruce Wayne",
        "alter-egos": "No alter egos found.",
        "aliases": ["Insider", "Matches Malone"],
        "place-of-birth": "Crest Hill, Bristol Township; Gotham County",
        "first-appearance": "Detective Comics #27",
        "publisher": "DC Comics",
        "alignment": "good",
        "appearance": {
          "gender": "Male",
          "race": "Human",
          "height": ["6'2", "188 cm"],
          "weight": ["210 lb", "95 kg"],
          "eye-color": "blue",
          "hair-color": "black"
        },
        "work": {
          "occupation": "Businessman",
          "base": "Batcave, Stately Wayne Manor, Gotham City, Hall of Justice, Justice League Watchtower",
          "connections": {
            "group-affiliation": "Batman Family, Batman Incorporated, Justice League, Outsiders, Wayne Enterprises, Club of Heroes, formerly White Lantern Corps, Sinestro Corps",
            "relatives": "Damian Wayne (son), Dick Grayson (adopted son), Tim Drake (adopted son), Jason Todd (adopted son), Cassandra Cain (adopted ward)\nMartha Wayne (mother, deceased), Thomas Wayne (father, deceased), Alfred Pennyworth (former guardian), Roderick Kane (grandfather, deceased), Elizabeth Kane (grandmother, deceased), Nathan Kane (uncle, deceased), Simon Hurt (ancestor), Wayne Family"
          },
          "image": {
            "url": "https://www.superherodb.com/pictures2/portraits/10/100/639.jpg"
          }
        },
        "id": "71",
        "name": "Batman II",
        "powerstats": {
          "intelligence": "88",
          "strength": "11",
          "speed": "33",
          "durability": "28",
          "power": "36",
          "combat": "100"
        },
        "biography": {
          "full-name": "Dick Grayson",
          "alter-egos": "Nightwing, Robin",
          "aliases": ["Dick Grayson"],
          "place-of-birth": "-",
          "first-appearance": "-",
          "publisher": "Nightwing",
          "alignment": "good",
          "appearance": {
            "gender": "Male",
            "race": "Human",
            "height": ["5'10", "178 cm"],
            "weight": ["175 lb", "79 kg"],
            "eye-color": "Blue",
            "hair-color": "Black"
          },
          "work": {
            "occupation": "-",
            "base": "Gotham City; formerly Bludhaven, New York City",
            "connections": {
              "group-affiliation": "Justice League Of America, Batman Family",
              "relatives": "John Grayson (father, deceased), Mary Grayson (mother, deceased), Bruce Wayne \\/ Batman (adoptive father), Damian Wayne \\/ Robin (foster brother), Jason Todd \\/ Red Hood (adoptive brother), Tim Drake \\/ Red Robin (adoptive brother), Cassandra Cain \\/ Batgirl IV (adoptive sister)"
            },
            "image": {
              "url": "https://www.superherodb.com/pictures2/portraits/10/100/1496.jpg"
            }
          }
        }
      ]
    }
  ]
}
```

4. CONSUMO DE API

De esta manera, podremos estudiar la disposición de los distintos arrays y objetos de una determinada búsqueda para utilizar aquellos atributos que queramos mostrar en nuestra aplicación.



```
>
{
  response : success
  results-for : batman
  results : [ 3 items
    ▶ 0 : { 8 props }
    ▶ 1 : { 8 props }
    ▼ 2 : {
      id : 71
      name : Batman II
      ▶ powerstats : { 6 props }
      ▶ biography : { 7 props }
      ▶ appearance : { 6 props }
      ▶ work : { 2 props }
      ▼ connections : {
        group-affiliation : Justice League Of America, Batman Family
        relatives : John Grayson (father, deceased), Mary Grayson (mother, deceased), Bruce Wayne / Batman (adoptive father), Damian Wayne / Robin (foster brother), Jason Todd / Red Hood (adoptive brother), Tim Drake / Red Robin (adoptive brother), Cassandra Cain / Batgirl IV (adoptive sister)
```

4. CONSUMO DE API

Lo primero que debemos hacer es darle permisos a nuestra aplicación para que pueda conectarse a internet. En el archivo *AndroidManifest.xml* utilizamos la etiqueta `<uses-permission.../>`.

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

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

    <application
        android:allowBackup="true"
        android:dataExtractionRules="@xml/data_extraction_rules"
        android:fullBackupContent="@xml/backup_rules"
```

4. CONSUMO DE API

Ahora necesitamos añadir una librería para consumo de APIs llamada *Retrofit* en el archivo *build.gradle.kts(Module:app)*.

<https://programacionymas.com/blog/consumir-una-api-usando-retrofit>

```
dependencies {  
    implementation("androidx.core:core-ktx:1.9.0")  
    implementation("androidx.appcompat:appcompat:1.6.1")  
    implementation("com.google.android.material:material:1.9.0")  
    implementation("androidx.constraintlayout:constraintlayout:2.1.4")  
    implementation("androidx.core:core-ktx:")  
  
    //Retrofit  
    implementation("com.squareup.retrofit2:retrofit:2.9.0")  
    implementation("com.squareup.retrofit2:converter-gson:2.9.0")  
}
```



4. CONSUMO DE API

Ésta nos permite convertir archivos JSON en clases propias de Java y transformar las URLs en objetos Retrofit. Creamos en la actividad principal la función *getRetrofit()*.

```
private fun searchByName(query: String) {}  
  
private fun getRetrofit(): Retrofit {  
    return Retrofit  
        .Builder()  
        .baseUrl("https://superheroapi.com/")  
        .addConverterFactory(GsonConverterFactory.create())  
        .build()  
}
```

4. CONSUMO DE API

Podemos crear entonces el objeto *retrofit* dentro del método *onCreate*, declarándolo previamente como cualquier otra variable, y así implementar sus métodos desde cualquier parte del código.

```
private lateinit var binding: ActivitySuperHeroListBinding
private lateinit var retrofit: Retrofit

override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    binding = ActivitySuperHeroListBinding.inflate(layoutInflater)
    setContentView(binding.root)
    retrofit = getRetrofit()
    initUI()
}
```


5. INTERFACES Y CORRUTINAS

Una **interfaz** es un conjunto de métodos que indican a los objetos qué hacer y cómo hacerlo de forma predeterminada. Con click derecho elegimos *New* → *Kotlin Class/File* → *Interface*, y creamos el archivo *ApiService.kt*.

Dentro definiremos dos métodos para realizar peticiones a la API por GET, es decir, a través de fragmentos de URL.

```
package com.ruben.aplicacionespmdm.SuperHeroApp

interface ApiService {
    @GET
}
```

5. INTERFACES Y CORRUTINAS

Pero antes tenemos que entender el funcionamiento de una aplicación en cualquier dispositivo. Nuestro código es secuencial y todo proceso que deba mostrarse por pantalla debe ejecutarse desde el hilo principal del procesador.

En ocasiones, como en el consumo de APIs, necesitaremos ejecutar tareas secundarias que podrían bloquear el subproceso principal.

Una *corrutina* es un patrón de diseño de simultaneidad que puedes usar en Android para simplificar el código que se ejecuta de forma asíncrona.

<https://developer.android.com/kotlin/coroutines?hl=es-419>

5. INTERFACES Y CORRUTINAS

Cuando vamos a usar corrutinas en una función, ésta debe empezar por la palabra `suspend`. Vamos a crear dos funciones para buscar superhéroes por nombre y por su ID.

```
package com.ruben.aplicacionespmdm.SuperHeroApp

interface ApiService {
    @GET("/api/10229233666327556/search/{name}")
    suspend fun getSuperheroes(@Path("name") superheroName:String):
    Response<SuperHeroDataResponse>

    @GET("/api/10229233666327556/{id}")
    suspend fun getSuperheroDetail(@Path("id")
    superheroId:String):Response<SuperHeroDetailResponse>
```

5. INTERFACES Y CORRUTINAS

En el método GET especificamos el fragmento que se le ha de añadir a *baseURL* y a la función le pasamos por parámetro una ruta como *string* que sustituya la variable {name} que hemos especificado en el fragmento.

Por otro lado, indicamos la respuesta que va a devolver la función, que es la transformación que realizará *Retrofit* del archivo JSON a una clase que debemos configurar.

GET, Path y Response son las herramientas que nos facilita *Retrofit*.

```
@GET("/api/10229233666327556/search/{name}")
suspend fun getSuperheroes(@Path("name") superheroName:String) :
Response<SuperHeroDataResponse>
```

5. INTERFACES Y CORRUTINAS

Creamos entonces una nueva clase llamada *SuperHeroDataResponse.kt*. Esta *data class* debe tener la misma estructura de árbol que el JSON con el que vamos a trabajar, con clases anidadas siguiendo esa estructura.

```
data class SuperHeroDataResponse(  
    @SerializedName("response") val response: String,  
    @SerializedName("results") val superheroes: List<SuperheroItemResponse>  
)
```

5. INTERFACES Y CORRUTINAS

Para tomar datos del archivo JSON debemos indicar los campos a los que queremos acceder con `@SerializedName` y almacenar su valor en una variable. El campo `url` está dentro de un objeto `image` en este JSON.

```
data class SuperHeroDataResponse(  
    @SerializedName("response") val response: String,  
    @SerializedName("results") val superheroes: List<SuperheroItemResponse>  
)  
data class SuperheroItemResponse(  
    @SerializedName("id") val superheroId: String,  
    @SerializedName("name") val name: String,  
    @SerializedName("image") val superheroImage: SuperheroImageResponse  
)  
data class SuperheroImageResponse(@SerializedName("url") val url: String)
```

5. INTERFACES Y CORRUTINAS

Ahora podemos comprobar el funcionamiento de la aplicación definiendo en la función de consulta `searchByName()` de la actividad principal la forma en la que queremos que se realice la consulta.

```
// Existen varios Dispatchers pero para cualquier acción como peticiones de red
// o BBDD utilizamos IO para que se ejecuten en un hilo secundario.
private fun searchByName(query: String) {
    CoroutineScope(Dispatchers.IO).launch {
        val myResponse: Response<SuperHeroDataResponse> =
            retrofit.create(ApiService::class.java).getSuperheroes(query)
    }
}
```

5. INTERFACES Y CORRUTINAS

Lo que estaremos haciendo es implementar la interfaz *ApiService* que hemos creado anteriormente. Podemos mostrarlo por consola evaluando si la respuesta ha tenido éxito.

```
private fun searchByName(query: String) {  
    CoroutineScope(Dispatchers.IO).launch {  
        val myResponse: Response<SuperHeroDataResponse> =  
            retrofit.create(ApiService::class.java).getSuperheroes(query)  
        if (myResponse.isSuccessful) {  
            Log.i("Consulta", "Funciona :)")  
        } else {  
            Log.i("Consulta", "No funciona :(")  
        }  
    }  
}
```

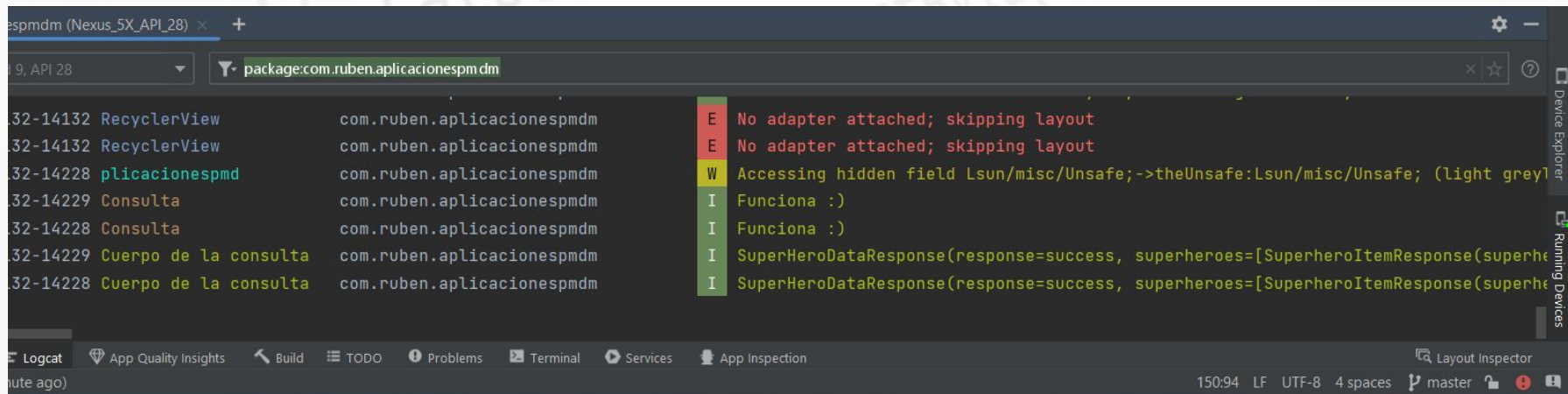

5. INTERFACES Y CORRUTINAS

Pero también podemos mostrar todo el cuerpo de la consulta creando una nueva variable que lo almacene y transformándolo a *string* en el comando `Log.i()`.

```
if (myResponse.isSuccessful) {  
    Log.i("Consulta", "Funciona :)")  
    val response: SuperHeroDataResponse? = myResponse.body()  
    if (response != null) {  
        Log.i("Cuerpo de la consulta", response.toString())  
    }  
}  
else {  
    Log.i("Consulta", "No funciona :(")  
}
```

5. INTERFACES Y CORRUTINAS

Y nos devuelve la estructura de datos que hayamos conformado en la clase *SuperHeroDataResponse.kt* a través de la interfaz *ApiService.kt*. Es decir, elementos de arrays y objetos del archivo JSON generado.



```
32-14132 RecyclerView      com.ruben.aplicacionespmdm  E No adapter attached; skipping layout
32-14132 RecyclerView      com.ruben.aplicacionespmdm  E No adapter attached; skipping layout
32-14228 plicacionespmdm        com.ruben.aplicacionespmdm  W Accessing hidden field Lsun/misc/Unsafe; ->theUnsafe:Lsun/misc/Unsafe; (light grey)
32-14229 Consulta         com.ruben.aplicacionespmdm  I Funciona :)
32-14228 Consulta         com.ruben.aplicacionespmdm  I Funciona :)
32-14229 Cuerpo de la consulta com.ruben.aplicacionespmdm  I SuperHeroDataResponse(response=success, superheroes=[SuperheroItemResponse(superhe
32-14228 Cuerpo de la consulta com.ruben.aplicacionespmdm  I SuperHeroDataResponse(response=success, superheroes=[SuperheroItemResponse(superhe
```

Logcat App Quality Insights Build TODO Problems Terminal Services App Inspection

Layout Inspector 150:94 LF UTF-8 4 spaces master

5. INTERFACES Y CORRUTINAS

Si además, mientras nuestra aplicación accede a la API para obtener la consulta, queremos mostrar la *ProgressBar*, deberemos mostrarla modificando su atributo *isVisible* a `true`.

```
private fun searchByName(query: String) {  
    binding.progressBar.isVisible = true  
    CoroutineScope(Dispatchers.IO).launch {  
        ...  
        val response: SuperHeroDataResponse? = myResponse.body()  
        if (response != null) {  
            Log.i("Cuerpo de la consulta", response.toString())  
            binding.progressBar.isVisible = false  
        }  
    }  
}
```

5. INTERFACES Y CORRUTINAS

ERROR: Es el hilo principal el que se encarga de pintar las vistas, y hemos configurado la visibilidad de la *ProgressBar* en una corrutina. Dentro de ésta, podemos ejecutar comandos en el hilo principal con *runOnUiThread*.

```
private fun searchByName(query: String) {  
    binding.progressBar.isVisible = true  
    CoroutineScope(Dispatchers.IO).launch {  
        ...  
        val response: SuperHeroDataResponse? = myResponse.body()  
        if (response != null) {  
            Log.i("Cuerpo de la consulta", response.toString())  
            runOnUiThread {  
                binding.progressBar.isVisible = false  
            }  
        }  
    }  
}
```

6. RECYCLERVIEW

Es momento de crear el Adapter del RecyclerView. Creamos la clase *SuperheroAdapter.kt* al que pasarle una lista (inicialmente vacía) de objetos *SuperheroItemResponse*.

```
class SuperheroAdapter( var superheroList: List<SuperheroItemResponse> = emptyList() )
: RecyclerView.Adapter<SuperheroViewHolder>() {
    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int):
SuperheroViewHolder {
    }
    override fun onBindViewHolder(holder: SuperheroViewHolder, position: Int) {
    }
    override fun getItemCount(): Int{
    }
}
```

6. RECYCLERVIEW

Rellenamos los métodos asociados al Adapter teniendo en cuenta que aún no hemos creado el layout de los ítems que va a recibir. La función *bind* es la función *render* que implementamos en el tema anterior.

```
override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): SuperheroViewHolder {  
    return SuperheroViewHolder(  
        LayoutInflater.from(parent.context).inflate(R.layout.item_superhero, parent, false)  
    )  
}  
  
override fun onBindViewHolder(holder: SuperheroViewHolder, position: Int) {  
    holder.bind(superheroList[position])  
}  
  
override fun getItemCount() = superheroList.size
```

6. RECYCLERVIEW

Creamos también la clase *SuperheroViewHolder.kt* con la función *bind* definida en el Adapter. Ésta recibirá de la clase *SuperHeroDataResponse* solo la lista de ítems, es decir, objetos tipo *SuperheroItemResponse*.

```
class SuperheroViewHolder(view: View) : RecyclerView.ViewHolder(view) {  
    fun bind(superheroItemResponse: SuperheroItemResponse) {  
  
    }  
}
```



6. RECYCLERVIEW

Y en la actividad principal definimos el Adapter creando la variable correspondiente de tipo *SuperheroAdapter*.

```
class SuperHeroListActivity : AppCompatActivity() {  
  
    private lateinit var binding: ActivitySuperHeroListBinding  
    private lateinit var retrofit: Retrofit  
  
    private lateinit var adapter: SuperheroAdapter
```


6. RECYCLERVIEW

Asociamos el Adapter al RecyclerView dentro de la función *initUI*. El método *setHasFixedSize*, aunque no es obligatorio, sí es conveniente usarlo (<https://www.cartagena99.com/recursos/alumnos/apuntes/UF4.1%20RecyclerView.pptx.pdf>).

```
        override fun onQueryTextChange(newText: String?) = false
    })
    adapter = SuperheroAdapter ()
    binding.rvSuperhero.setHasFixedSize(true)
    binding.rvSuperhero.layoutManager = LinearLayoutManager(this)
    binding.rvSuperhero.adapter = adapter
}
```

6. RECYCLERVIEW

El layout de los ítems constará de un ImageView y un TextView dentro de un CardView. El atributo *cardElevation* crea el efecto de una pequeña sombra bajo el CardView.

```
<androidx.cardview.widget.CardView
    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="200dp"
    android:layout_marginHorizontal="16dp"
    android:layout_marginVertical="8dp"
    app:cardCornerRadius="22dp"
    app:cardElevation="8dp">
```

6. RECYCLERVIEW

Para el `ImageView` aplicamos el atributo `scaleType` que se encarga de ajustar la imagen al tamaño del `View` según el valor que le demos:

<https://thoughtbot.com/blog/android-imageview-scaletype-a-visual-guide>

```
<ImageView  
    android:id="@+id/ivSuperhero"  
    android:layout width="match parent"  
    android:layout height="match parent"  
    android:scaleType="centerCrop" />
```



6. RECYCLERVIEW

Y el TextView podemos situarlo en la parte inferior del CardView mediante un FrameLayout con la misma anchura que el CardView pero de una altura determinada.

```
<FrameLayout
    android:layout width="match parent"
    android:layout height="30dp"
    android:layout_gravity="bottom">

    <TextView
        ...
        tools:text="SUPERHÉROE" />

</FrameLayout>
```

6. RECYCLERVIEW

Este TextView lo situaremos centrado en el FrameLayout que lo contiene y le aplicaremos un color de fondo con un 35% de opacidad para que se vea la parte de imagen que hay por debajo.

```
<TextView
    android:id="@+id/tvSuperheroName"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:gravity="center"
    android:background="@color/violet35"
    android:textColor="@color/white"
    android:textSize="20sp"
    tools:text="SUPERHEROE" />
```

6. RECYCLERVIEW

Si queremos empezar a mostrar los elementos de una consulta deberemos actualizar la lista que recibe el Adapter. Para ello, podemos crear una función *updateList* dentro del propio Adapter.

```
class SuperheroAdapter( var superheroList: List<SuperheroItemResponse> = emptyList())
: RecyclerView.Adapter<SuperheroViewHolder>() {

    fun updateList(list: List<SuperheroItemResponse>) {
        superheroList = list
        notifyDataSetChanged()
    }

    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int):
    SuperheroViewHolder {
```

6. RECYCLERVIEW

E invocarla desde el hilo principal, dentro de la función *searchByName*, pasándole la lista de superhéroes recibida en la consulta realizada a la API, que se almacena en el objeto *response*.

```
val response: SuperHeroDataResponse? = myResponse.body()
if (response != null) {
    Log.i("Cuerpo de la consulta", response.toString())
    runOnUiThread {
        adapter.updateList(response.superheroes)
        binding.progressBar.isVisible = false
    }
}
```

6. RECYCLERVIEW

Nos falta definir la función *bind* del ViewHolder. Desde ésta, pintaremos el layout anterior tomando cada uno de los elementos que queremos mostrar con ViewBinding.

```
class SuperheroViewHolder(view: View) : RecyclerView.ViewHolder(view) {  
  
    private val binding = ItemSuperheroBinding.bind(view)  
  
    fun bind(superheroItemResponse: SuperheroItemResponse) {  
        binding.tvSuperheroName.text = superheroItemResponse.name  
    }  
}
```


6. RECYCLERVIEW

El problema es que la imagen del superhéroe viene dada por la API mediante una URL. Necesitamos entonces una librería que nos permita renderizar imágenes de internet (<https://square.github.io/picasso/>).

Picasso

[Download Latest](#)

A powerful image downloading and caching library for Android

Introduction

Images add much-needed context and visual flair to Android applications. Picasso allows for hassle-free image loading in your application—often in one line of code!

```
Picasso.get().load("https://i.imgur.com/DvpvklR.png").into(imageView);
```

Many common pitfalls of image loading on Android are handled automatically by Picasso:

[Introduction](#)[Features](#)[Download](#)[Contributing](#)[License](#)

6. RECYCLERVIEW

En el apartado Download encontraremos la manera de importarla y el enlace a GitHub que nos facilita la última versión de la misma. Insertaremos la siguiente línea de código en *build.gradle.kts(Module: app)*.

```
implementation("androidx.constraintlayout:constraintlayout:2.1.4")
implementation("androidx.core:core-ktx:+")

//Retrofit
implementation("com.squareup.retrofit2:retrofit:2.9.0")
implementation("com.squareup.retrofit2:converter-gson:2.9.0")

//Picasso
implementation("com.squareup.picasso:picasso:2.8")
```