



Reskilling 4Employment Software Developer

Acesso móvel a sistemas de informação

Bruno Santos

bruno.santos@cesae.pt

Tópicos

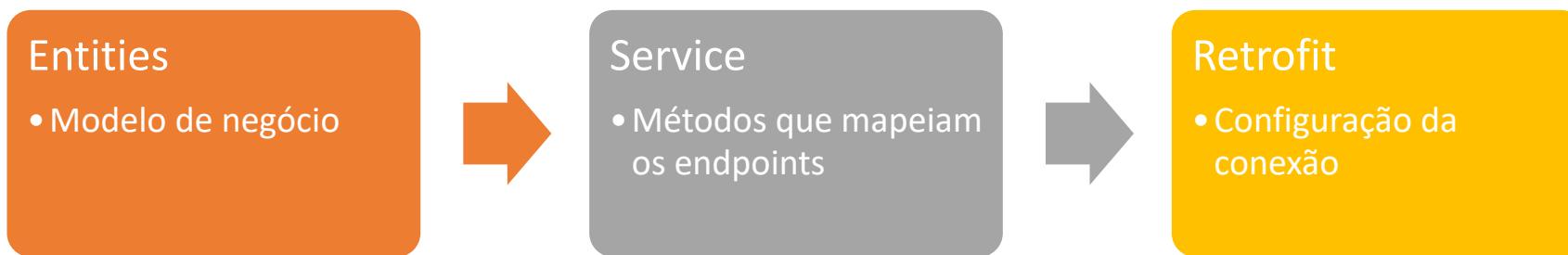
- Retrofit



Centro para o Desenvolvimento
de Competências Digitais

Retrofit

- Camada de abstração para chamadas a API



Retrofit

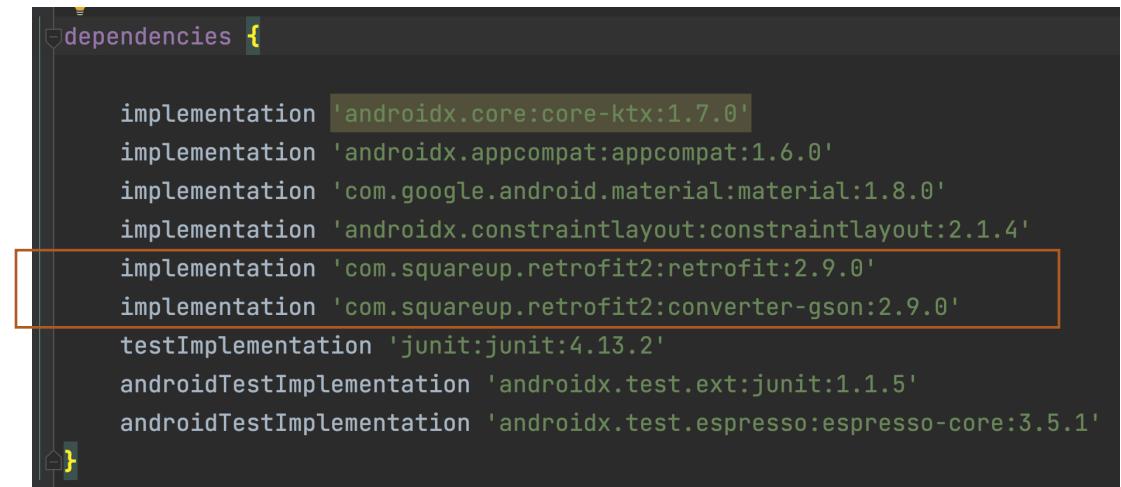


Centro para o Desenvolvimento
de Competências Digitais

- Para adicionar o Retrofit a um projeto necessitamos adicionar o seguinte código ao Gradle

Retrofit

```
dependencies {  
  
    implementation 'androidx.core:core-ktx:1.7.0'  
    implementation 'androidx.appcompat:appcompat:1.6.0'  
    implementation 'com.google.android.material:material:1.8.0'  
    implementation 'androidx.constraintlayout:constraintlayout:2.1.4'  
    testImplementation 'junit:junit:4.13.2'  
    androidTestImplementation 'androidx.test.ext:junit:1.1.5'  
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.5.1'  
}
```



```
dependencies {  
  
    implementation 'androidx.core:core-ktx:1.7.0'  
    implementation 'androidx.appcompat:appcompat:1.6.0'  
    implementation 'com.google.android.material:material:1.8.0'  
    implementation 'androidx.constraintlayout:constraintlayout:2.1.4'  
    implementation 'com.squareup.retrofit2:retrofit:2.9.0' implementation 'com.squareup.retrofit2:converter-gson:2.9.0'  
    testImplementation 'junit:junit:4.13.2'  
    androidTestImplementation 'androidx.test.ext:junit:1.1.5'  
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.5.1'  
}
```

```
implementation 'com.squareup.retrofit2:retrofit:2.9.0'  
implementation 'com.squareup.retrofit2:converter-gson:2.9.0'
```

Retrofit



Centro para o Desenvolvimento
de Competências Digitais

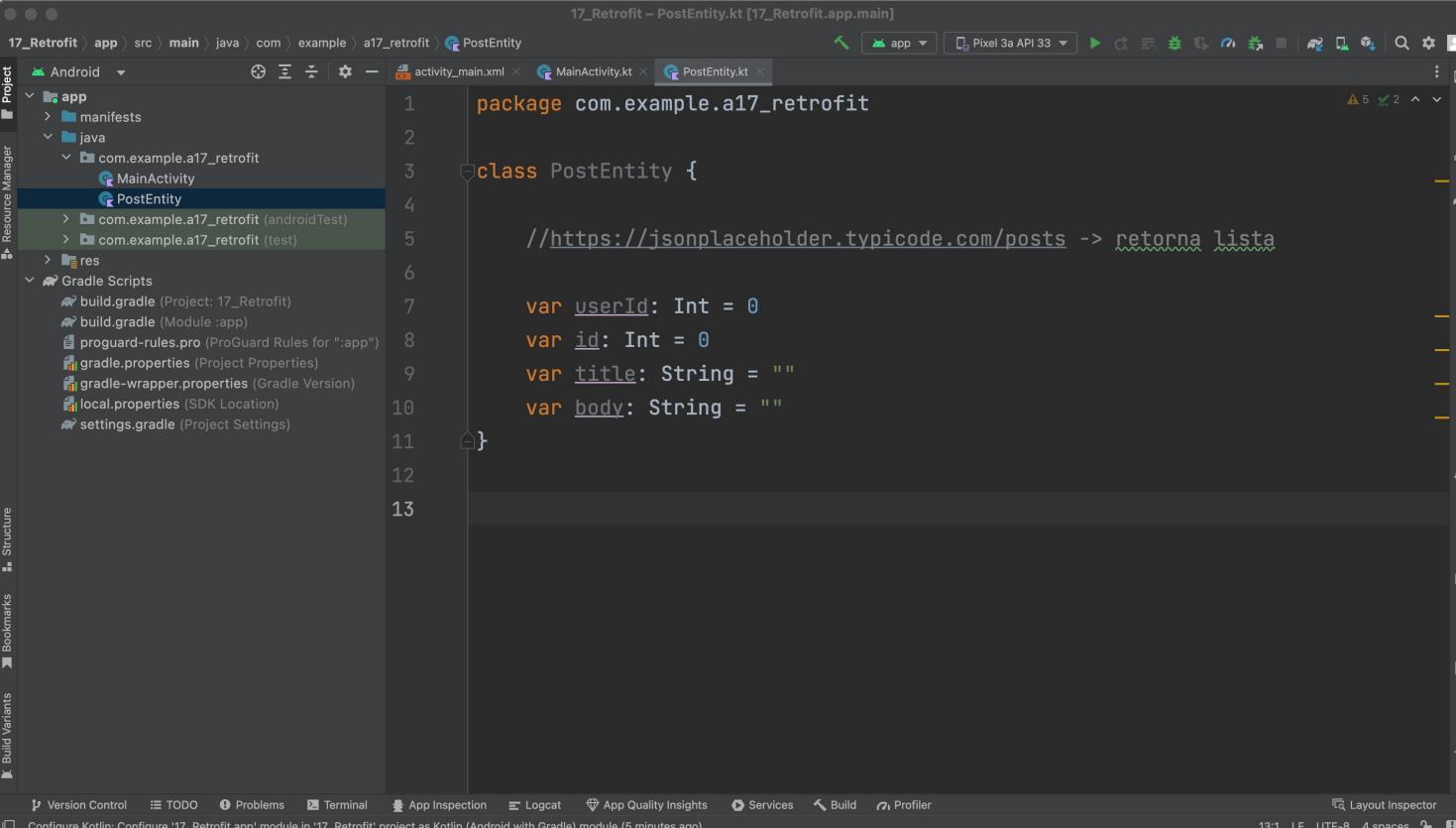
- A Gson é uma biblioteca que vamos utilizar para converter o JSON resultante das chamadas a API para as nossas classes.
- De notar que as versões anteriores (2.9.0) poderão não ser as atuais aquando da utilização.
- Vamos utilizar o JSONPlaceholder (<https://jsonplaceholder.typicode.com/>) como API de exemplo para obter informações na nossa aplicação.

Retrofit

- Utilizando como exemplo a apresentação de posts vamos mapear os elementos na nossa aplicação, para isso criamos a classe PostEntity

```
[  
  {  
    "userId": 1,  
    "id": 1,  
    "title": "sunt aut facere repellat  
    "body": "quia et suscipit\\nsuscip  
  },  
  {  
    "userId": 1,  
    "id": 2,  
    "title": "qui est esse",  
    "body": "est rerum tempore vitae\\n  
non debitis possimus qui neque nisi n  
  },  
  {  
    "userId": 1,  
    "id": 3,  
    "title": "ea molestias quasi exer  
    "body": "et iusto sed quo iure\\nv  
  },  
  {  
    "userId": 1,  
    "id": 4,  
    "title": "eum et est occaecati",  
    "body": "ullam et saepe reiciendi  
voluptatem rerum illo velit"  
  },  
]
```

Retrofit



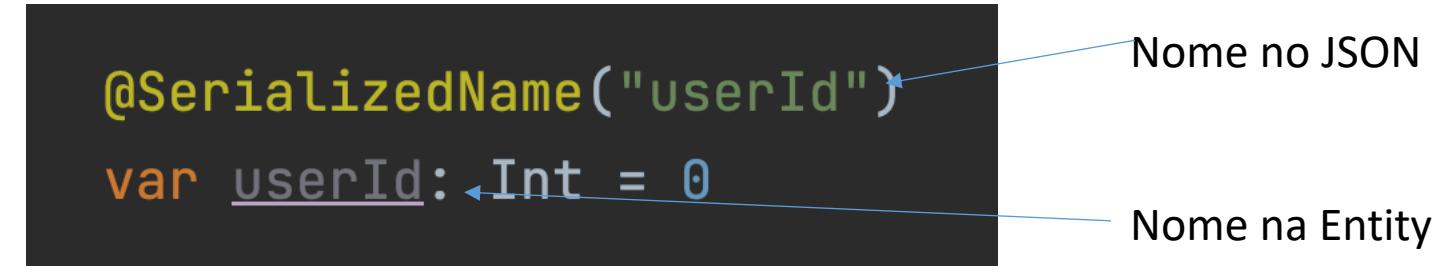
The screenshot shows the Android Studio interface with the code for `PostEntity.kt`. The code defines a class `PostEntity` with variables `userId`, `id`, `title`, and `body`. A comment indicates the API endpoint: `//https://jsonplaceholder.typicode.com/posts -> retorna lista`.

```
1 package com.example.a17_retrofit
2
3 class PostEntity {
4
5     //https://jsonplaceholder.typicode.com/posts -> retorna lista
6
7     var userId: Int = 0
8     var id: Int = 0
9     var title: String = ""
10    var body: String = ""
11 }
12
13
```

Retrofit

- Agora, através do `@SerializedName` informamos qual o nome do atributos que existe no ficheiro JSON e que queremos mapear na classe. De notar que vamos utilizar o mesmo nome, o que é uma boa prática, mas não seria necessário.

```
@SerializedName("userId")
var userId: Int = 0
```

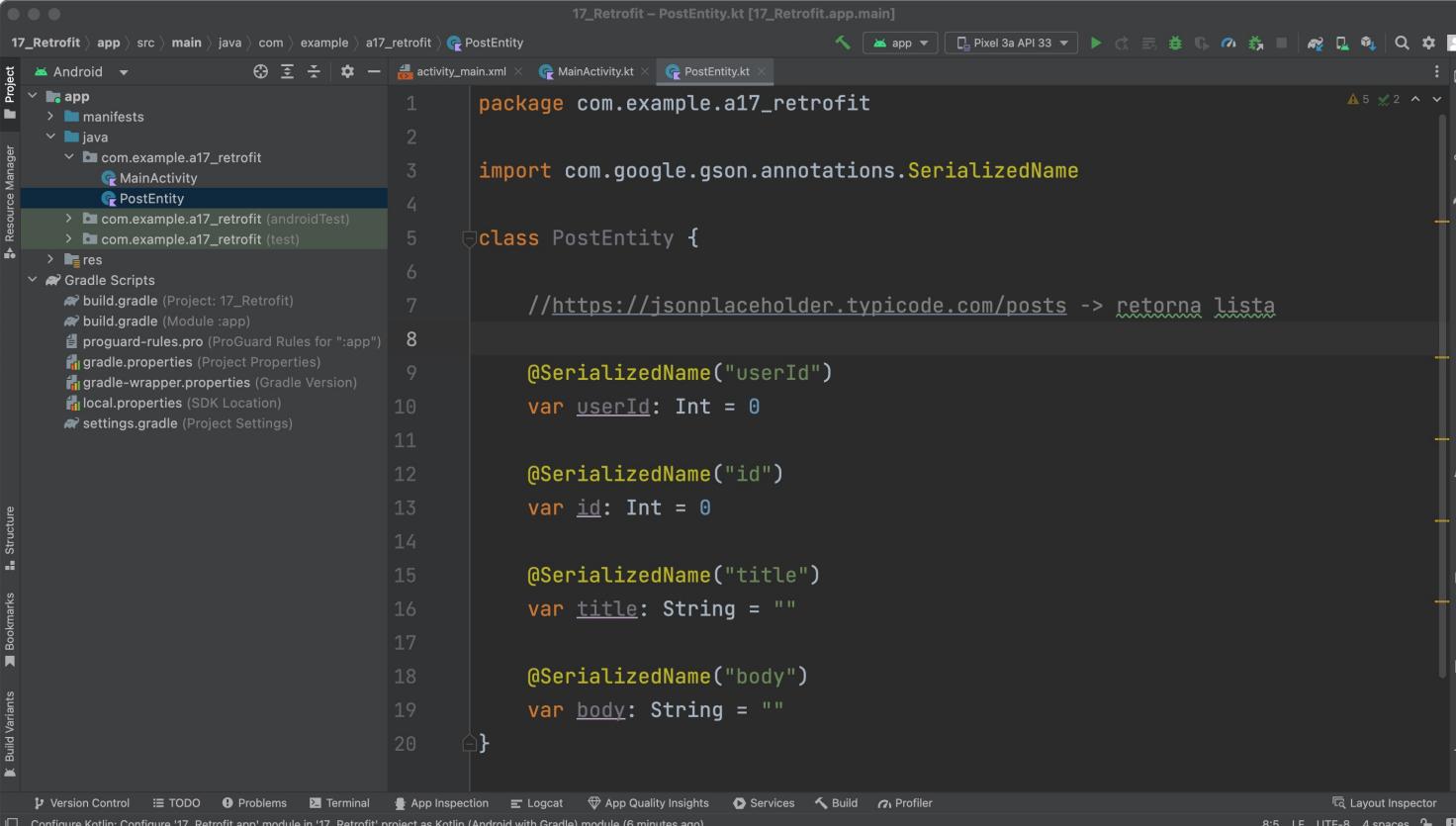


The diagram illustrates the mapping between JSON field names and Java entity field names. It shows two lines pointing from the code to labels: one line points from the `@SerializedName("userId")` annotation to the label "Nome no JSON", and another line points from the `userId` variable name to the label "Nome na Entity".

Nome no JSON

Nome na Entity

Retrofit



The screenshot shows the Android Studio interface with the code editor open to the `PostEntity.kt` file. The code defines a data class for a post entity, annotated with `@SerializedName` for Gson serialization. The class has four properties: `userId`, `id`, `title`, and `body`. The `title` and `body` properties are empty strings. The `id` and `userId` properties are integers set to 0. The `PostEntity` class is annotated with `@SerializedName("PostEntity")`.

```
17_Retrofit - PostEntity.kt [17_Retrofit.app.main]
17_Retrofit > app > src > main > java > com > example > a17_retrofit > PostEntity

package com.example.a17_retrofit

import com.google.gson.annotations.SerializedName

class PostEntity {

    //https://jsonplaceholder.typicode.com/posts -> retorna lista

    @SerializedName("userId")
    var userId: Int = 0

    @SerializedName("id")
    var id: Int = 0

    @SerializedName("title")
    var title: String = ""

    @SerializedName("body")
    var body: String = ""
}
```

Retrofit



Centro para o Desenvolvimento
de Competências Digitais

- De seguida criamos a interface do serviço: PostService
- Nesta classe vamos criar a função list que irá retornar a lista de posts, no entanto vamos utilizar a funcionalidade Call do Retrofit para que o mesmo nos ajude a mapear a resposta.
- Importante: aquando da seleção da Call deve ser selecionada a opção retrofit2

Retrofit

```
interface PostService {  
    fun list():Call|  
}  
  
Call (android.telecom)  
Call<T> (retrofit2) selected  
Call (okhttp3)  
Call<T> (retrofit2)
```

Retrofit

- Temos ainda de informar, através de uma annotation qual o método de chamada, sendo neste caso GET, com a informação da URL .

```
1 package com.example.a17_retrofit
2
3 import retrofit2.Call
4 import retrofit2.http.GET
5
6 interface PostService {
7
8     @GET("posts")
9     fun list():Call<List<PostEntity>>
10 }
11
```

Retrofit



Centro para o Desenvolvimento
de Competências Digitais

- Como passo final da organização, vamos criar a classe que irá interagir com o Service e o Entity, a qual vamos chamar RetrofitClient

Retrofit



Centro para o Desenvolvimento
de Competências Digitais

```
companion object {

    private lateinit var INSTANCE: Retrofit
    private const val BASE_URL = "https://jsonplaceholder.typicode.com/"

    private fun getRetrofitInstance(): Retrofit {
        val http = OkHttpClient.Builder()
        if (!::INSTANCE.isInitialized) {
            INSTANCE = Retrofit.Builder() Retrofit.Builder
                .baseUrl(BASE_URL) Retrofit.Builder
                .client(http.build())
                .addConverterFactory(GsonConverterFactory.create())
                .build()
        }
        return INSTANCE
    }
}
```

Retrofit



Centro para o Desenvolvimento
de Competências Digitais

- Implementamos um companion object e aplicamos a variável INSTANCE de forma a criar um Singleton, impedindo assim a múltipla criação de instâncias.
- Passamos na variável BASE_URL o endereço da API (muito importante tem sempre de terminar em /)
- O parâmetro cliente informa quem orquestra a chamada à Internet para comunicar com a API
- O GsonConverterFactory converte o JSON mapeando na classe que definimos

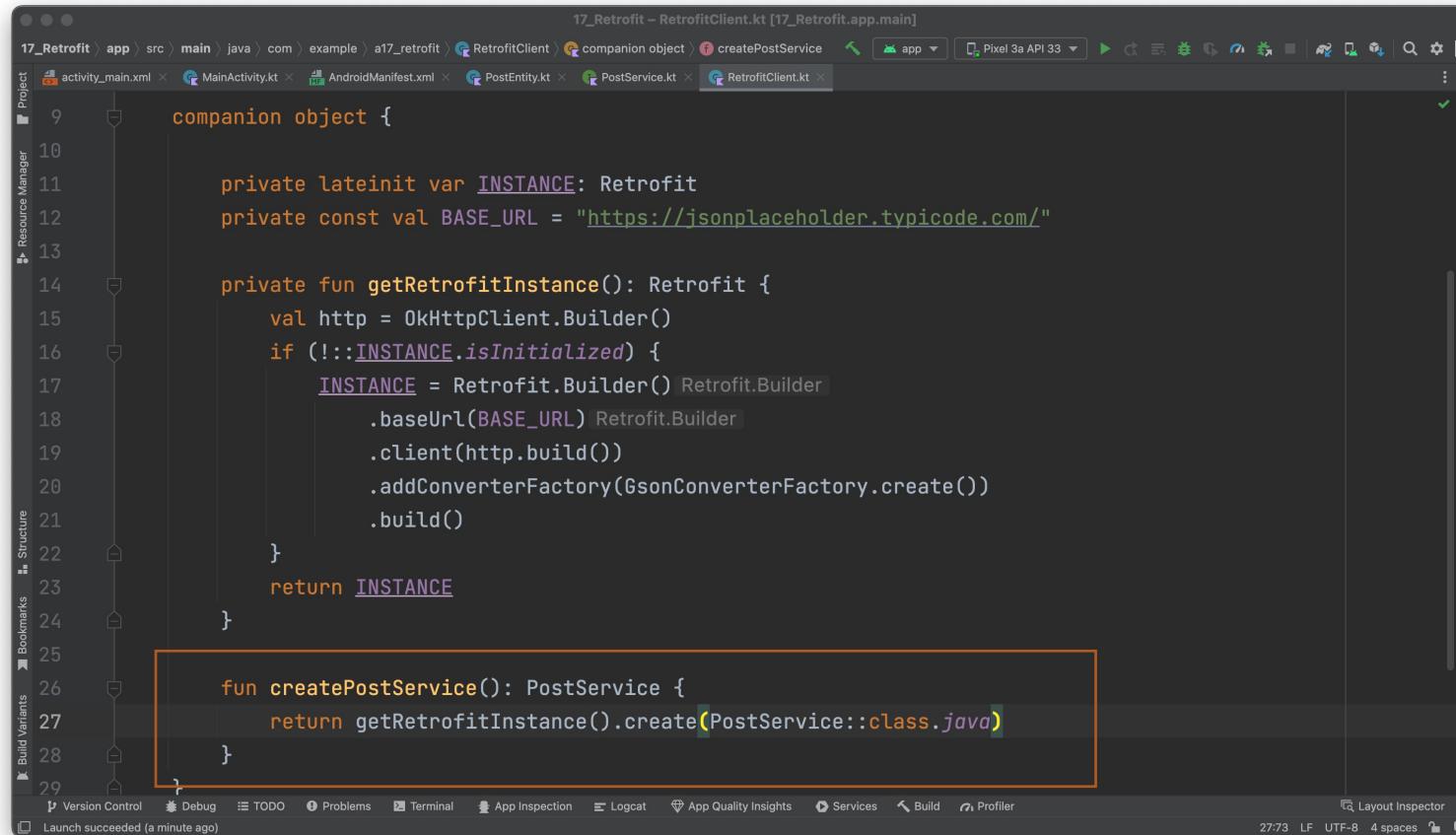
Retrofit



Centro para o Desenvolvimento
de Competências Digitais

- Vamos ainda criar o método que vai permitir a inicialização do serviço

Retrofit



The screenshot shows the Android Studio interface with the file `17_Retrofit/app/src/main/java/com/example/a17_retrofit/RetrofitClient.kt` open. The code implements a companion object with a static factory method `createPostService()` that returns an instance of `PostService`. The `getRetrofitInstance()` method uses the `OkHttpClient.Builder` to configure a `Retrofit` instance with a base URL and Gson converter factories.

```
companion object {

    private lateinit var INSTANCE: Retrofit
    private const val BASE_URL = "https://jsonplaceholder.typicode.com/"

    private fun getRetrofitInstance(): Retrofit {
        val http = OkHttpClient.Builder()
        if (!::INSTANCE.isInitialized) {
            INSTANCE = Retrofit.Builder()
                .baseUrl(BASE_URL)
                .client(http.build())
                .addConverterFactory(GsonConverterFactory.create())
                .build()
        }
        return INSTANCE
    }

    fun createPostService(): PostService {
        return getRetrofitInstance().create(PostService::class.java)
    }
}
```

Retrofit



Centro para o Desenvolvimento
de Competências Digitais

- Agora na classe MainActivity vamos invocar a chamada a API, mas com a chamada é assíncrona temos de criar um mecanismo de a nossa app não ficar pendurada enquanto a chamada é realizada

Retrofit



Centro para o Desenvolvimento
de Competências Digitais

```
super.onCreate(savedInstanceState)
setContentView(R.layout.activity_main)

val service = RetrofitClient.createPostService()
val call: Call<List<PostEntity>> = service.list()

call.enqueue(object : Callback<List<PostEntity>>{
})
```

- Criamos o serviço e chamamos a função list do mesmo, o retorno será armazenado na call
- Uma vez que a call será assíncrona temos de chamar a função enqueue
- Como a classe Callback é uma interface e não pode ser instanciada vamos colocar como classe anónima (object)
- De seguida implementamos os dois métodos da Callback

Retrofit



Centro para o Desenvolvimento
de Competências Digitais

```
call.enqueue(object : Callback<List<PostEntity>> {
    override fun onResponse(call: Call<List<PostEntity>>, response: Response<List<PostEntity>>) {
        }

    override fun onFailure(call: Call<List<PostEntity>>, t: Throwable) {
        }
})
```

Retrofit

- Temos ainda de garantir a possibilidade de a aplicação poder usar a internet, então no AndroidManifest vamos colocar essa permissão

```
<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"
```

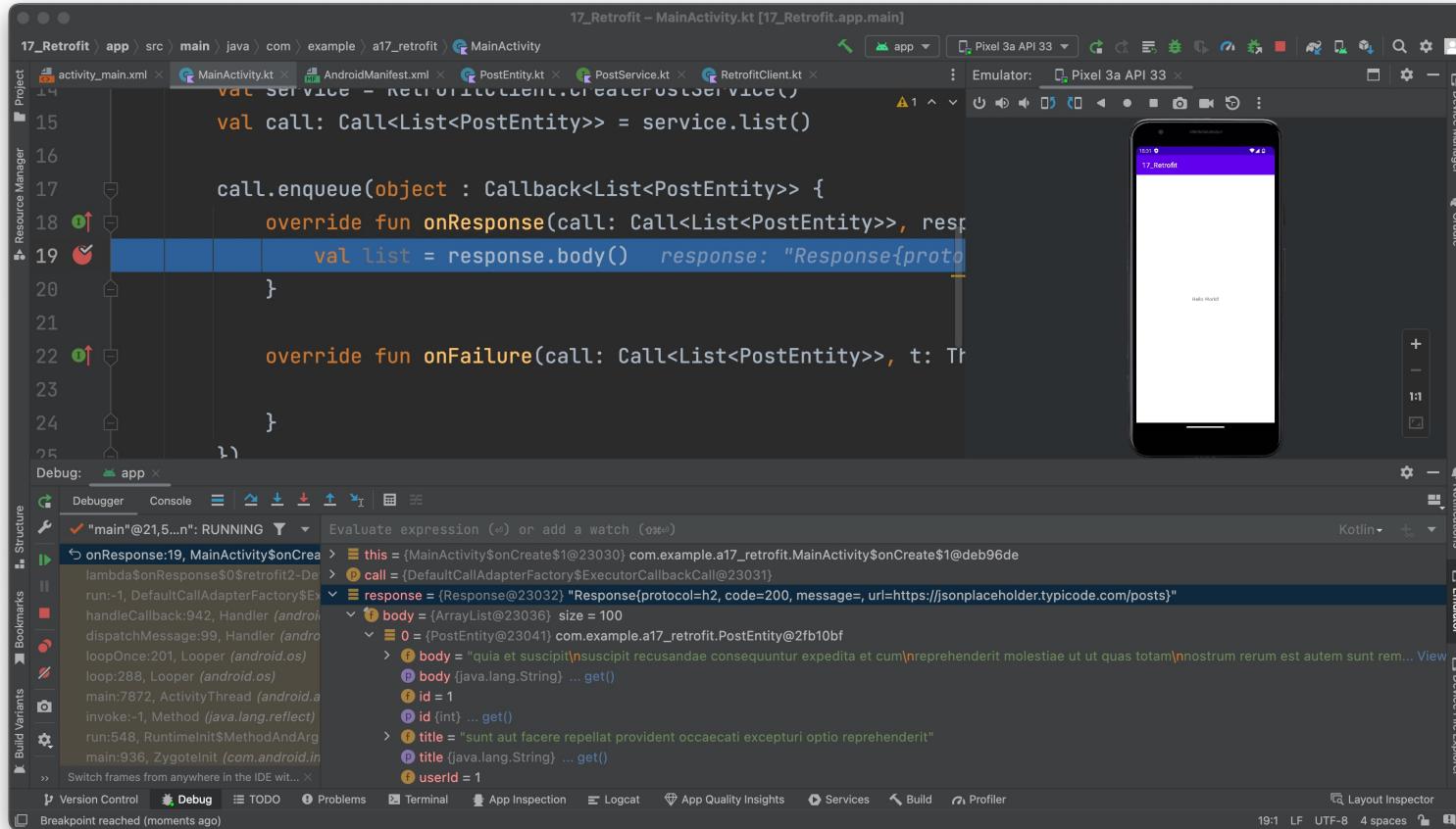
Retrofit



Centro para o Desenvolvimento
de Competências Digitais

- Executando a aplicação com um breakpoint dentro da classe podemos ver quando a API retorna um resultado

Retrofit



The screenshot shows the Android Studio interface during the execution of a Retrofit call. The main window displays the `MainActivity.kt` file with the following code:

```
15     val service = RetrofitClient.createService(PostService::class.java)
16
17     val call: Call<List<PostEntity>> = service.list()
18
19     call.enqueue(object : Callback<List<PostEntity>> {
20         override fun onResponse(call: Call<List<PostEntity>>, response: Response<List<PostEntity>>) {
21             val list = response.body() // Breakpoint reached here
22             ...
23         }
24
25     })

```

The code is currently at line 19, where a breakpoint is set. The `response` variable is highlighted in blue. The bottom right corner of the screen shows a smartphone emulator displaying the app's UI with the text "Hello World". The bottom status bar indicates the time as 19:11, the encoding as UTF-8, and the font size as 4 spaces.

Retrofit



Centro para o Desenvolvimento
de Competências Digitais

- Temos a aplicação a funcionar, no entanto podemos melhorar as chamadas à API no RetrofitClient, utilizando Generix.
- Considerando que criamos uma nova interface (UserService) que tem a mesma classe list que nos apresenta, neste caso os Users. Criamos no RetrofitClient o método createUserService()

Retrofit



Centro para o Desenvolvimento
de Competências Digitais

```
fun <S> createService(service: Class<S>): S {  
    return getRetrofitInstance().create(service)  
}  
  
/*fun createPostService(): PostService {  
    return getRetrofitInstance().create(PostService::class.java)  
}  
  
fun createUserService(): UserService {  
    return getRetrofitInstance().create(UserService::class.java)  
}*/
```

Retrofit



Centro para o Desenvolvimento
de Competências Digitais

```
//val service = RetrofitClient.createPostService()  
val service = RetrofitClient.createService(PostService::class.java)  
val call: Call<List<PostEntity>> = service.list()
```

Retrofit



Centro para o Desenvolvimento
de Competências Digitais

- Desta forma conseguimos organizar o nosso código para ser muito mais genérico e organizado

Exercício 1



Centro para o Desenvolvimento
de Competências Digitais

- Implemente o processo semelhante para retornar os users (<https://jsonplaceholder.typicode.com/users>)
- Considere que aquando da existência de nested objects (objetos dentro de outros objetos) é necessária criar uma classe dentro da classe principal (ver exemplo seguinte)

Exercício 1

```
@SerializedName("id")
var id: Int = 0

@SerializedName("name")
var name: String = ""

@SerializedName("username")
var username: String = ""

@SerializedName("email")
var email: String = ""

@SerializedName("address")
var address: AddressEntity? = null

@SerializedName("phone")
var phone: String = ""
```

```
class UserEntity {
    class AddressEntity {
        @SerializedName("street")
        var street: String = ""

        @SerializedName("suite")
        var suite: String = ""

        @SerializedName("city")
        var city: String = ""

        @SerializedName("zipcode")
        var zipcode: String = ""

        @SerializedName("geo")
        var geo: GeoEntity? = null
    }
}
```

Exercício 2



Centro para o Desenvolvimento
de Competências Digitais

- Apresente numa listagem (ListView ou RecyclerView) os nomes de todos os utilizadores (users)