



# Reskilling 4Employment Software Developer

Acesso móvel a sistemas de informação

Bruno Santos

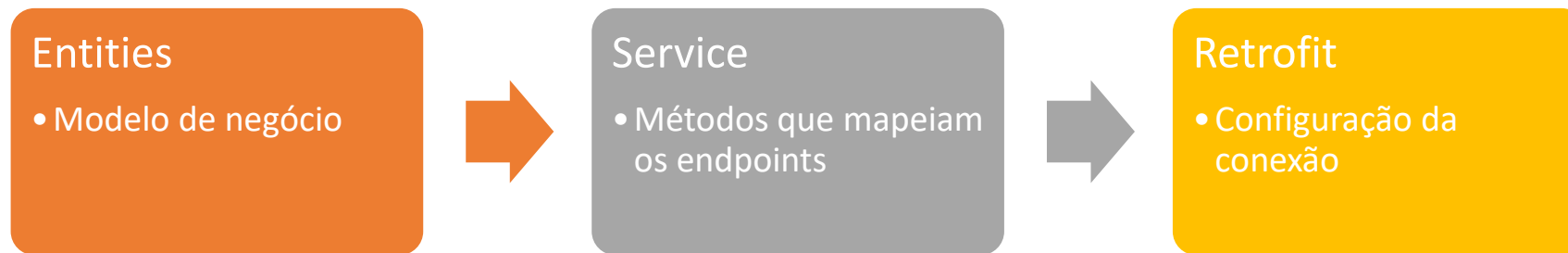
[bruno.santos.mcv@msft.cesae.pt](mailto:bruno.santos.mcv@msft.cesae.pt)

# Tópicos

- Retrofit

# Retrofit

- Camada de abstração para chamdas a API



# Retrofit

- Para adicionar o Retrofit a um projeto precisamos 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

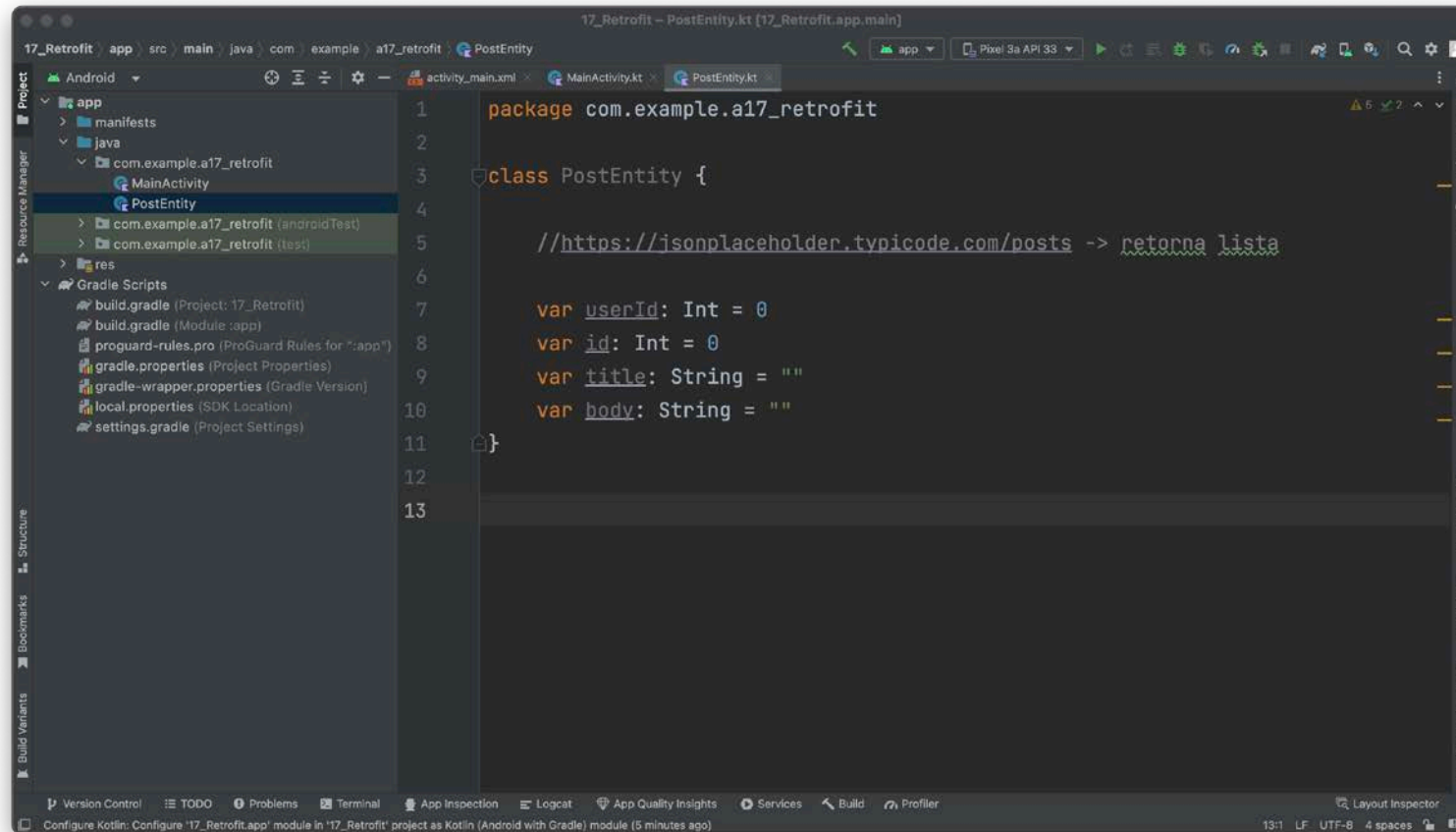
- A Gson é uma biblioteca que vamos utilizar para converter o JSON resultante das chamadas a API para as nossas classes.
- De notar que as versões anteriores (2.9.0) poderão não ser as atuais aquando da utilização.
- Vamos utilizar o JSONPlaceholder (<https://jsonplaceholder.typicode.com/>) como API 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\nsuscipit"
  },
  {
    "userId": 1,
    "id": 2,
    "title": "qui est esse",
    "body": "est rerum tempore vitae\
non debitis possimus qui neque nisi r"
  },
  {
    "userId": 1,
    "id": 3,
    "title": "ea molestias quasi exercitation",
    "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"
  },
  {
    "userId": 1,
    "id": 5,
    "title": "tempora accusamus",
    "body": "voluptatem blanditiis molestiae no"
  },
  {
    "userId": 1,
    "id": 6,
    "title": "sunt et consectetur expedit",
    "body": "voluptatem blanditiis molestiae no"
  },
  {
    "userId": 1,
    "id": 7,
    "title": "facere et autem",
    "body": "voluptatem blanditiis molestiae no"
  },
  {
    "userId": 1,
    "id": 8,
    "title": "facere et autem",
    "body": "voluptatem blanditiis molestiae no"
  },
  {
    "userId": 1,
    "id": 9,
    "title": "facere et autem",
    "body": "voluptatem blanditiis molestiae no"
  },
  {
    "userId": 1,
    "id": 10,
    "title": "facere et autem",
    "body": "voluptatem blanditiis molestiae no"
  }
]
```

# Retrofit





# Retrofit

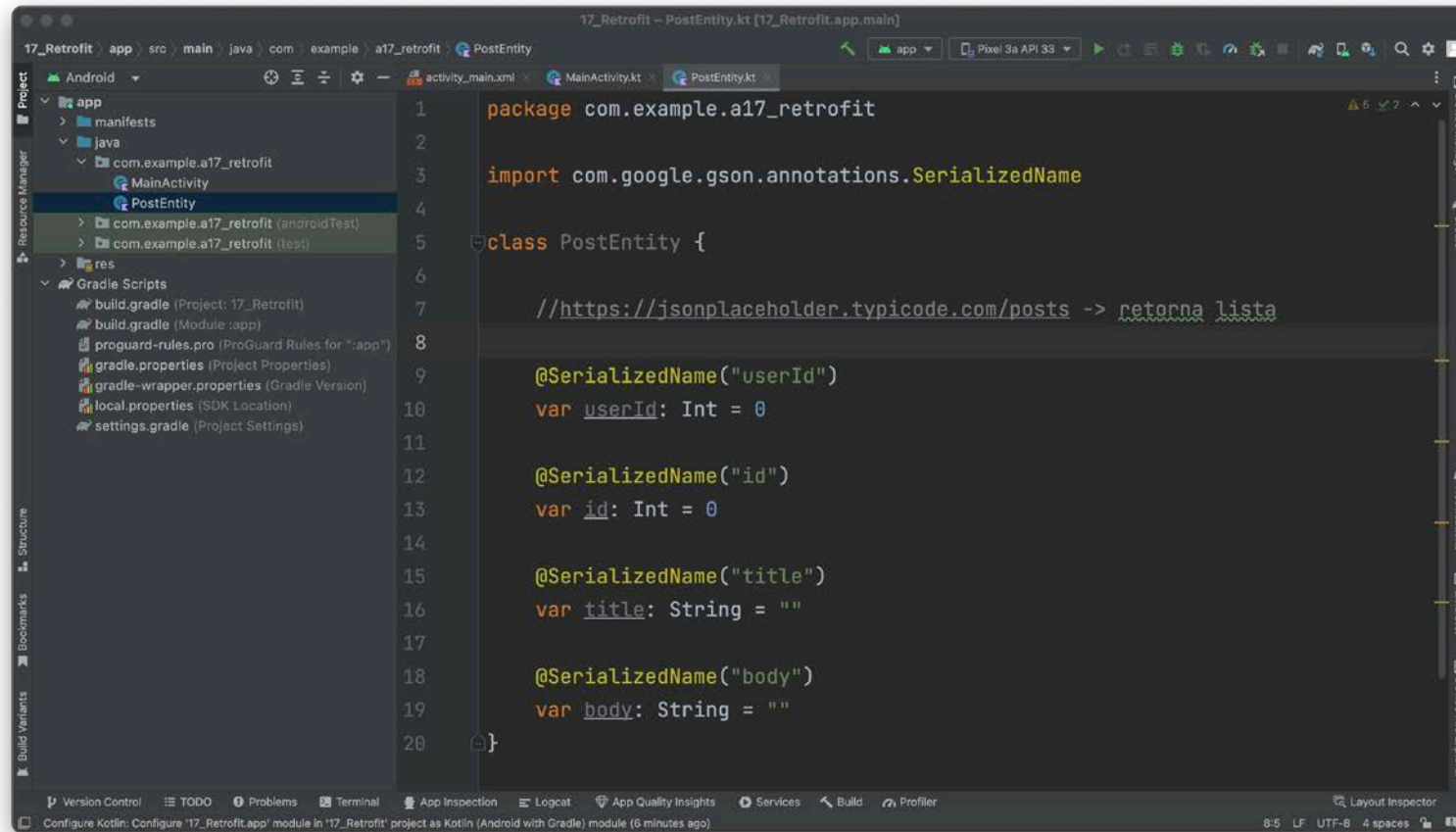
- Agora, através do `@SerializedName` informamos qual o nome do atributos que existe no ficheiro JSON e que queremos mapear na classe. De notar que vamos utilizar o mesmo nome, o que é uma boa prática, mas não seria necessário.

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

Nome no JSON

Nome na Entity

# Retrofit

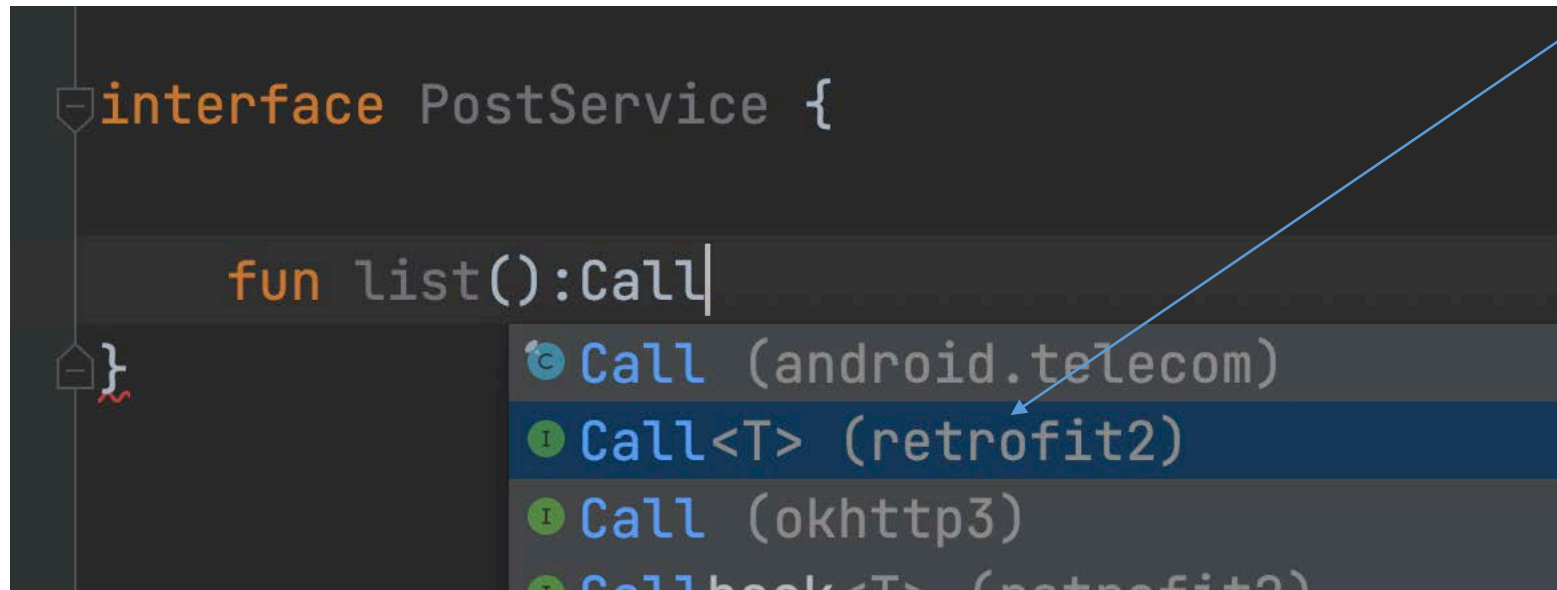


# Retrofit

- 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)  
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     }
```

# Retrofit

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

# Retrofit

```
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

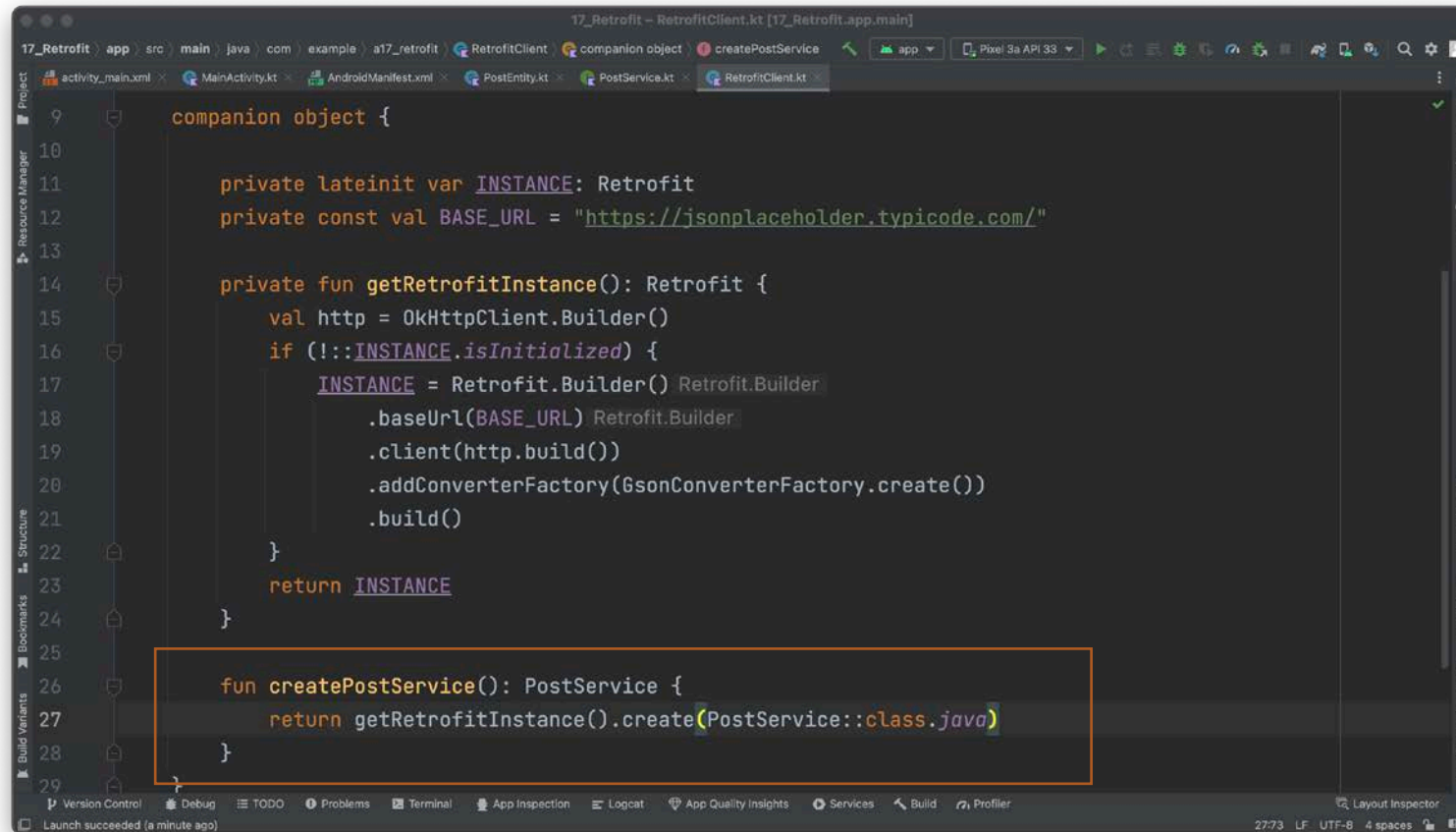
- Implementamos um companion object e aplicamos a variável INSTANCE de forma a criar um Singleton, impedindo assim a múltipla criação de instâncias.
- Passamos na variável BASE\_URL o endereço da API (muito importante tem sempre de terminar em /)
- O parâmetro cliente informa quem orquestra a chamada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



# Retrofit

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

# Retrofit



# Retrofit

- 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

```
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

```
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">

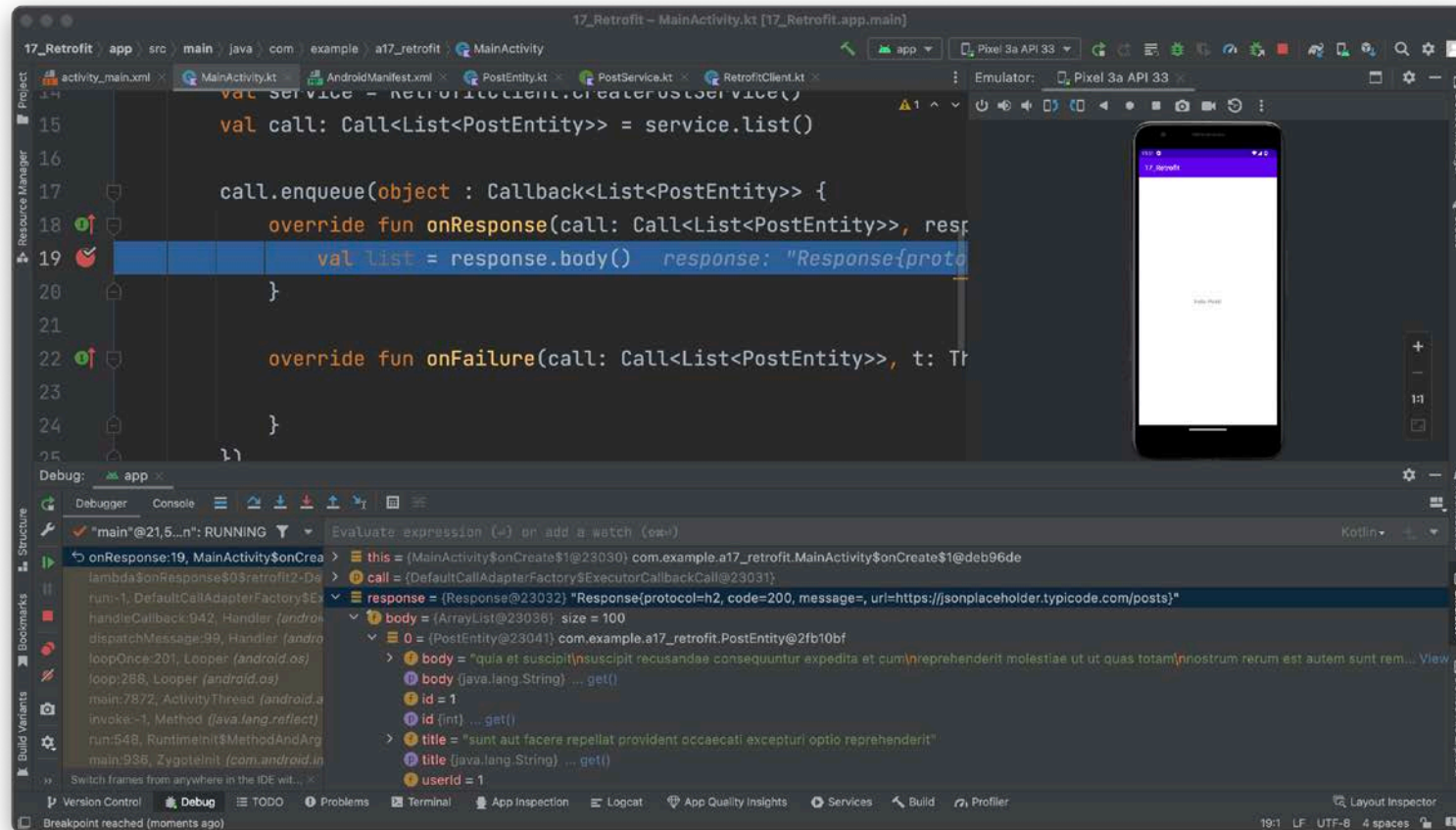
    <uses-permission android:name="android.permission.INTERNET"/>

    <application
        android:allowBackup="true"
```

# Retrofit

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

# Retrofit





# Retrofit

- 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

```
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

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

# Retrofit

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

# Exercício 1

- 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

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