

Retrofit

Álvaro Castilla Loaiza

2º DAM

Retrofit es una librería de Android destinada a hacer solicitudes de red de manera más sencilla y eficiente.

Para utilizarla debemos seguir una serie de pasos:

Crear el proyecto en Android Studio

Accederemos a Android Studio y crearemos un nuevo proyecto de Jetpack Compose que esté vacío.

New Project

Empty Activity

Create a new empty activity with Jetpack Compose

Name: Retrofit

Package name: com.acasloa946.retrofit

Save location: C:\Users\AlvaroPC\AndroidStudioProjects\Retrofit

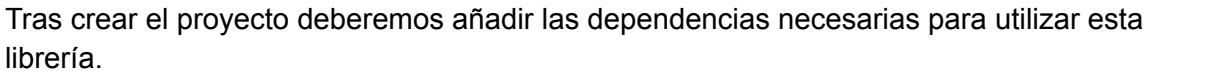
Minimum SDK: API 24 ("Nougat"; Android 7.0)

i Your app will run on approximately 96,3% of devices.
[Help me choose](#)

Build configuration language ? Kotlin DSL (build.gradle.kts) [Recommended]

w 'Retrofit' already exists at the specified project location and it is not empty.

Previous Next Cancel Finish



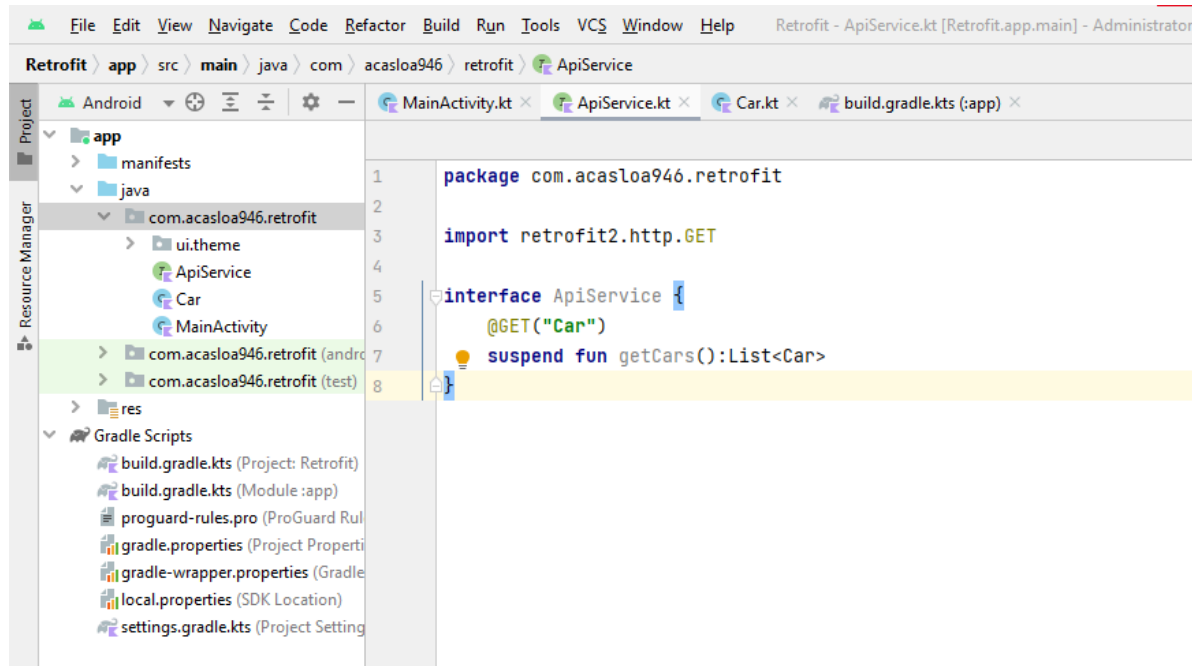
```
42 composeOptions { this.ComposeOptions:
43     kotlinCompilerExtensionVersion = "1.4.3"
44 }
45 packaging { this.Packaging:
46     resources { this.ResourcesPackaging:
47         excludes += "/META-INF/(AL2.0|LOPL2.1)"
48     }
49 }
50
51 dependencies { this.DependencyHandlerScope:
52
53     implementation("androidx.core:core-ktx:1.9.0")
54     implementation("androidx.lifecycle:lifecycle-runtime-ktx:2.7.0")
55     implementation("androidx.activity:activity-compose:1.8.2")
56     implementation(platform("androidx.compose:compose-bom:2023.03.00"))
57     implementation("androidx.compose:compose:ui:ui")
58     implementation("androidx.compose:compose:ui:ui-graphics")
59     implementation("androidx.compose:compose:ui:ui-tooling-preview")
60     implementation("androidx.compose.material3:material3")
61     testImplementation("junit:junit4:13.2")
62     androidTestImplementation("androidx.test.ext:junit:1.1.5")
63     androidTestImplementation("androidx.test.espresso:espresso-core:3.5.1")
64     androidTestImplementation(platform("androidx.compose:compose-bom:2023.03.00"))
65     androidTestImplementation("androidx.compose.ui:ui-test-junit4")
66     debugImplementation("androidx.compose.ui:ui-tooling")
67     debugImplementation("androidx.compose.ui:ui-test-manifest")
68
69
70 // Implementaciones de Retrofit
71 implementation("com.squareup.retrofit2:retrofit:2.9.0")
72 implementation("com.squareup.retrofit2:converter-gson:2.9.0")
73
74 // ViewModel y LiveData para MVVM
75 implementation("androidx.lifecycle:lifecycle-viewmodel-compose:1.0.0-alpha07")
76 implementation("androidx.lifecycle:lifecycle-livedata:2.3.1")
77 }
```

Incluiremos las dependencias tanto de Retrofit como de LiveData y Navigation ya que son necesarias para la arquitectura MVVM.

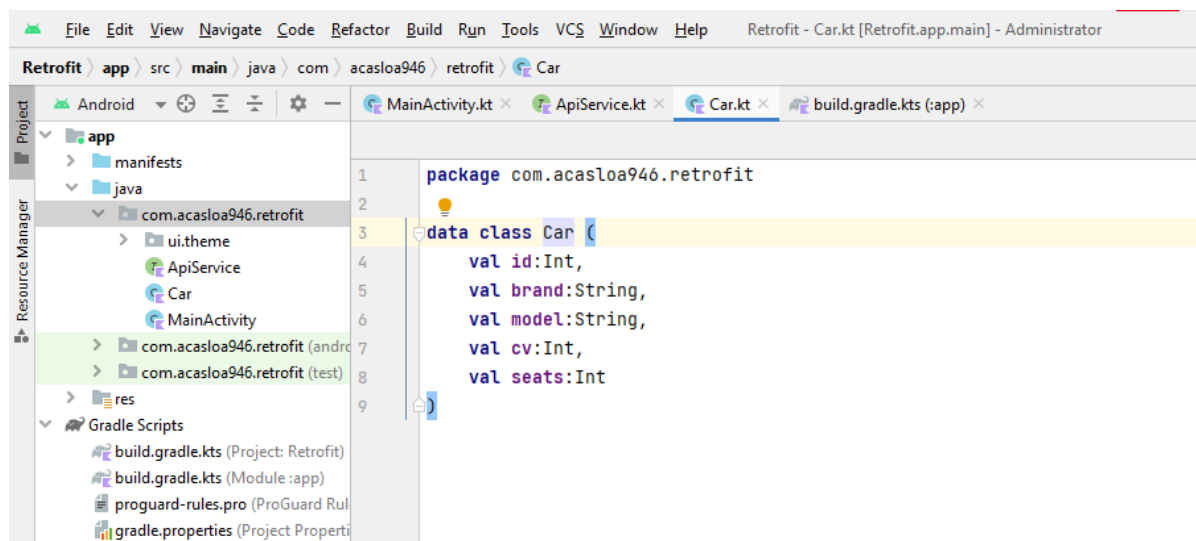
Empezando con Retrofit

Crear interface de Retrofit

Para trabajar con Retrofit necesitaremos crear una serie de clases. Primeramente, la interfaz que define los endpoints de la API.



Crearemos también la “data class” de Car que sería la siguiente:



Crear instancia de Retrofit

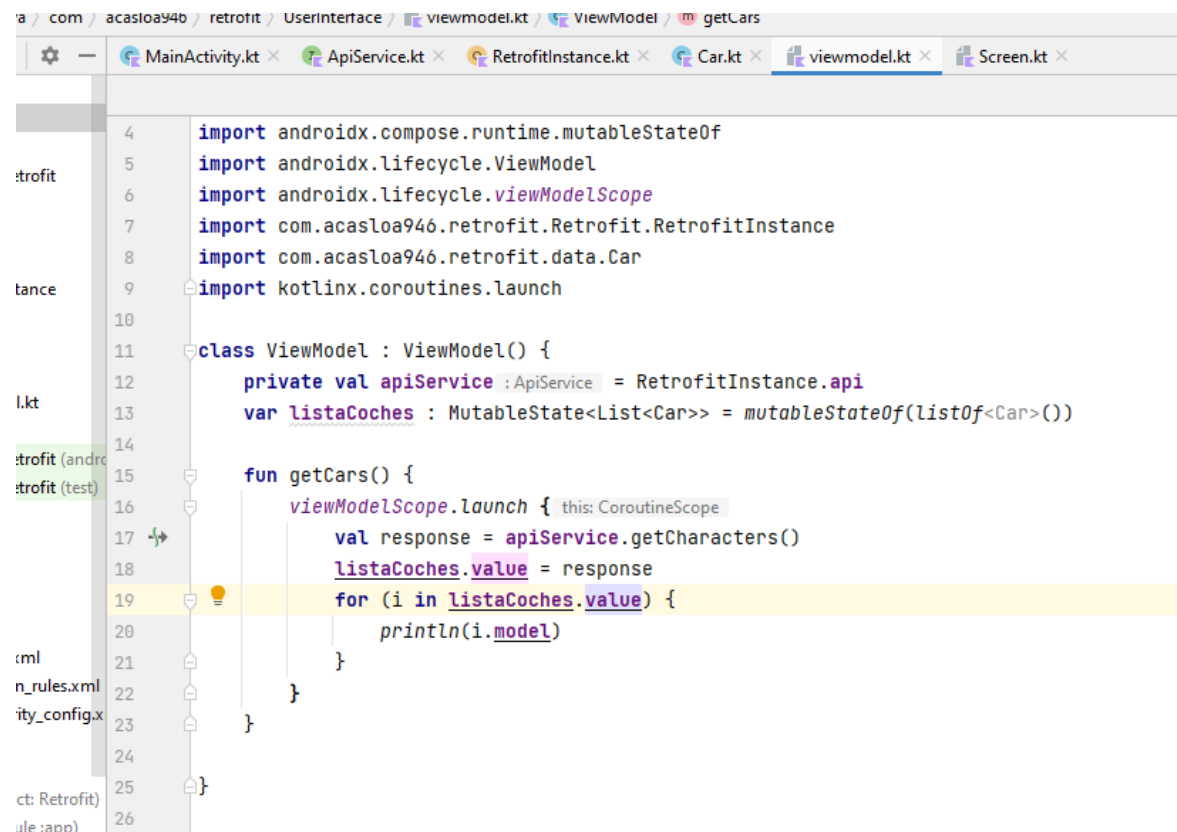
Aquí realmente es donde Retrofit accede a la API y la pasa al ApiService, sacando los datos de ésta. Debemos pasar la URL de la API, que en mi caso es <http://10.0.2.2:8083/CarAPI/> ya que utilizo una API propia.

```
import ...

object RetrofitInstance {
    private
    const val URL_API = "http://10.0.2.2:8083/CarAPI/"
    val api: ApiService by lazy {
        val retrofit = Retrofit.Builder()
            .baseUrl(URL_API)
            .addConverterFactory(GsonConverterFactory.create())
            .build()
        retrofit.create(ApiService::class.java)
    }
}
```

Crear ViewModel e UI

Para gestionar Retrofit y la UI crearemos un ViewModel.



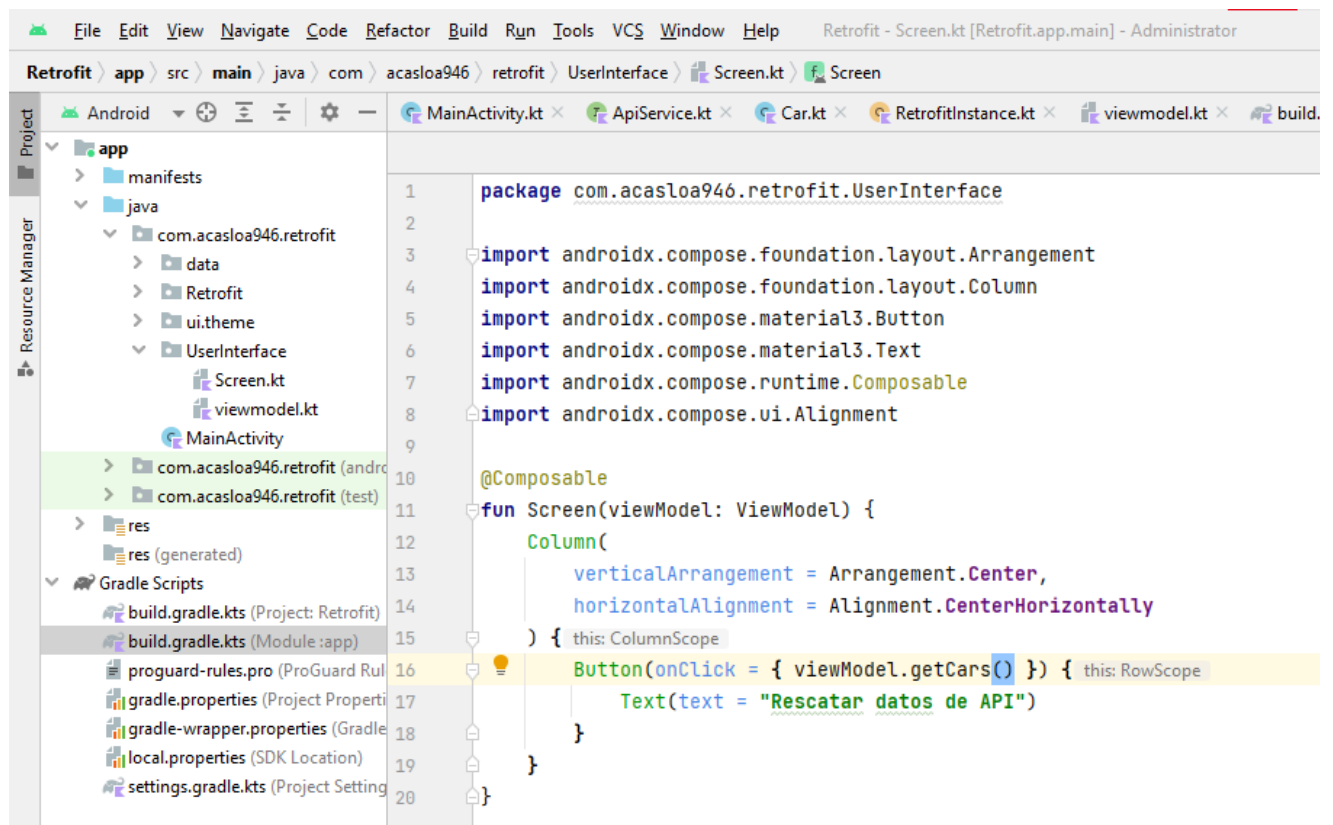
```
com / acasloa946 / retrofit / UserInterface / viewmodel.kt / ViewModel / getCars
MainActivity.kt x ApiService.kt x RetrofitInstance.kt x Car.kt x viewmodel.kt x Screen.kt x

4 import androidx.compose.runtime.mutableStateOf
5 import androidx.lifecycle.ViewModel
6 import androidx.lifecycle.viewModelScope
7 import com.acasloa946.retrofit.Retrofit.RetrofitInstance
8 import com.acasloa946.retrofit.data.Car
9 import kotlinx.coroutines.launch
10
11 class ViewModel : ViewModel() {
12     private val apiService : ApiService = RetrofitInstance.api
13     var listaCoches : MutableState<List<Car>> = mutableStateOf(listOf<Car>())
14
15     fun getCars() {
16         viewModelScope.launch { this: CoroutineScope
17             val response = apiService.getCharacters()
18             listaCoches.value = response
19             for (i in listaCoches.value) {
20                 println(i.model)
21             }
22         }
23     }
24
25 }
26
```

En este ViewModel crearemos una instancia del ApiService, una lista de coches vacía donde se almacenarán los coches que obtengamos de la API y una función que rescata estos datos.

Tras esto solo nos quedaría desarrollar una Screen para mostrar los resultados obtenidos de esta API.

Yo crearé una Screen con un botón para saber que la API nos devuelve los resultados correctamente y los imprimiré por consola.



Posteriormente cambiaremos esta Screen para mostrar los resultados de manera más vistosa, pero actualmente solo queremos comprobar que funciona.

Tras iniciar la aplicación y pulsar el botón nos sale lo siguiente por consola:

```
!024-01-31 00:10:25.917 8083-8083 Choreographer com.acasloa946 retrofit I Skipped 60 frame
!024-01-31 00:10:25.937 8083-8108 OpenGLRenderer com.acasloa946 retrofit I Davey! duration:
NewestInputEvent=0, HandleInputStart=3992658703800, AnimationStart=3992658730000, PerformTraversalsStart=3992667910
IssueDrawCommandsStart=3992669830200, SwapBuffers=3992670992100, FrameCompleted=3992678623700, DequeueBufferDurati
!024-01-31 00:10:26.071 8083-8098 lo946.retrofi com.acasloa946 retrofit I Background young
paused 8.710ms total 82.056ms
!024-01-31 00:10:26.082 8083-8100 System com.acasloa946 retrofit W A resource fail
!024-01-31 00:10:27.444 8083-8083 System.out com.acasloa946 retrofit I M4
!024-01-31 00:10:27.444 8083-8083 System.out com.acasloa946 retrofit I M4
!024-01-31 00:10:29.798 8083-8115 ProfileInstaller com.acasloa946 retrofit D Installing profi
```

Con esto comprobamos que la API nos devuelve datos, por lo que lo siguiente será mostrar los resultados por pantalla.

```

1 package com.acasloa946.retrofit.UserInterface
2
3 import ...
4
5 @Composable
6 fun Screen(viewModel: ViewModel) {
7     val listaCoches = viewModel.listaCoches.value
8
9     Column(
10         verticalArrangement = Arrangement.Top,
11         horizontalAlignment = Alignment.CenterHorizontally
12     ) { this: ColumnScope
13         CarList(cars = listaCoches)
14         Button(onClick = { viewModel.getCars() }) { this: RowScope
15             Text(text = "Rescatar datos de API")
16         }
17     }
18 }
19
20 @Composable
21 fun CarList(cars: List<Car>) {
22     LazyColumn { this: LazyListScope
23         items(cars) { this: LazyItemScope it: Car
24             CarCard(it)
25         }
26     }
27 }
28
29 @Composable
30 fun CarCard(
31     coche: Car
32 ) {
33     Card(
34         Modifier
35             .width(200.dp)
36             .padding(10.dp),
37         elevation = CardDefaults.cardElevation(defaultElevation = 8.dp),
38         colors = CardDefaults.cardColors(
39             containerColor = MaterialTheme.colorScheme.onPrimaryContainer,
40         )
41     )
42 }

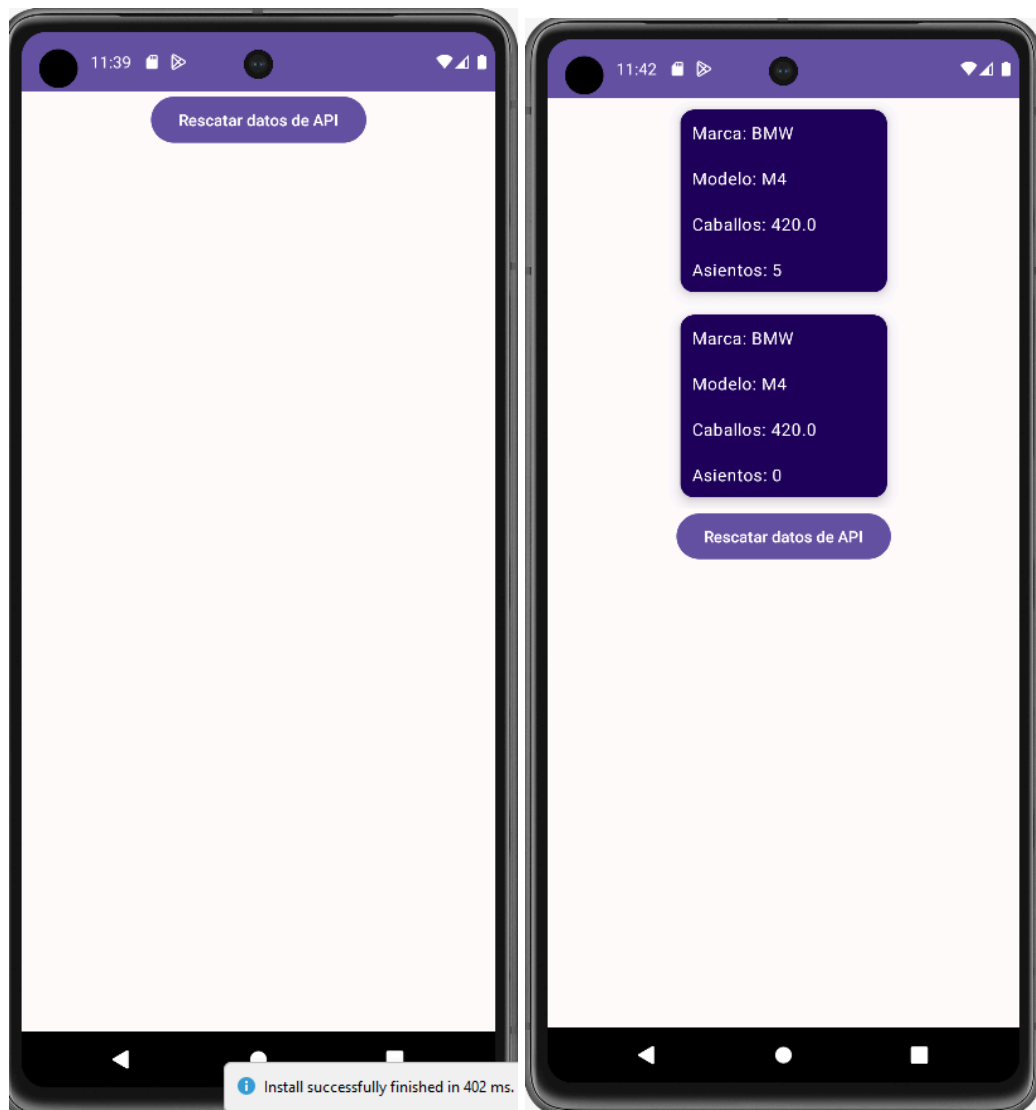
```

```

@Composable
fun CarCard(
    coche: Car
) {
    Card(
        Modifier
            .width(200.dp)
            .padding(10.dp),
        elevation = CardDefaults.cardElevation(defaultElevation = 8.dp),
        colors = CardDefaults.cardColors(
            containerColor = MaterialTheme.colorScheme.onPrimaryContainer,
        )
    ) { this: ColumnScope
        Text(text = "Marca: ${coche.brand}", color = Color.White, fontSize = 15.sp, modifier = Modifier.padding(10.dp))
        Text(text = "Modelo: " + coche.model, color = Color.White, fontSize = 15.sp, modifier = Modifier.padding(10.dp))
        Text(text = "Caballos: " + coche.cv, color = Color.White, fontSize = 15.sp, modifier = Modifier.padding(10.dp))
        Text(text = "Asientos: " + coche.seats, color = Color.White, fontSize = 15.sp, modifier = Modifier.padding(10.dp))
    }
}

```

Tras cambiar la Screen abriremos la aplicación y veremos lo siguiente:

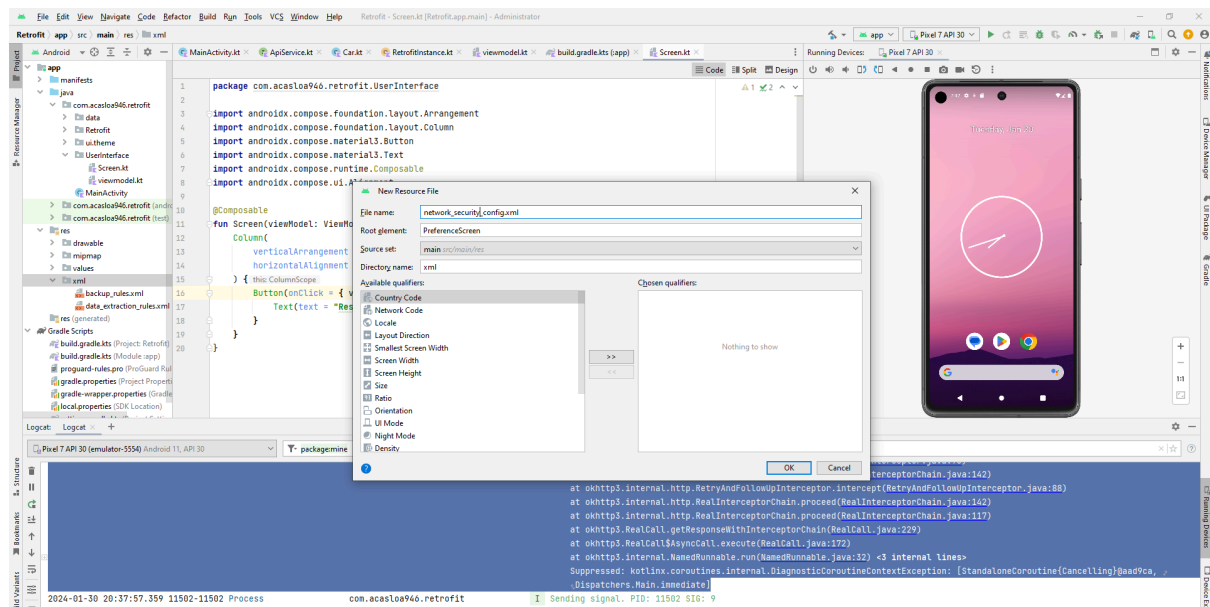


Si pulsamos el botón veremos que salen las fichas de 2 coches que son los que están en mi API (en mi caso en una BDD MySQL). Ya habríamos terminado nuestra aplicación con Retrofit. En el caso de que queramos utilizar este código para otra API solo tendremos que cambiar la clase de datos, el link de la API y la interfaz.

Adicional

En mi caso estoy accediendo a una API propia que se ejecuta en mi ordenador, por lo que tengo que utilizar el protocolo HTTP, no HTTPS. Android de manera predeterminada no habilita esta opción por problemas de seguridad pero al ser una prueba habilitaremos esta característica.

Haremos lo siguiente:



Crearemos un archivo en la carpeta /res/xml llamado `network_security_config`. La extensión de este archivo será XML.



Insertaremos el anterior código y nos iremos a `AndroidManifest.xml` para vincular las opciones de seguridad de redes a este archivo.


```
<application
    android:allowBackup="true"
    android:dataExtractionRules="@xml/data_extraction_rules"
    android:fullBackupContent="@xml/backup_rules"
    android:icon="@mipmap/ic_launcher"
    android:label="Retrofit"
    android:roundIcon="@mipmap/ic_launcher_round"
    android:supportRtl="true"
    android:theme="@style/Theme.Retrofit"
    android:networkSecurityConfig="@xml/network_security_config"
    tools:targetApi="31">
    <activity
        android:name=".MainActivity"
        android:exported="true"
        android:label="Retrofit"
        android:theme="@style/Theme.Retrofit">
```