

# MODULO 4 | Desarrollo de Aplicaciones Móviles

## Android Kotlin | Ignacio Cavallo

---

[https://github.com/cavigna/modulo\\_desarrollo\\_de\\_aplicaciones\\_moviles\\_android\\_kotlin](https://github.com/cavigna/modulo_desarrollo_de_aplicaciones_moviles_android_kotlin)

### Clase 98 | 17-11

La clase de hoy nos dividimos por grupos, para implementar el testing de retrofit. La verdad, que ha sido muy dificultoso, por temas de librerías y el uso de corrutinas. Después de horas, logré implementar lo siguiente:

<https://github.com/cavigna/DogsPics>

A continuación muestro unas capturas de pantalla

Structure

Favorites

Build Variants

Tests	Duration	Pixel_4_API_30
✓ Test Results	638 ms	1/1
✓ RemoteTest	638 ms	1/1
✓ listadolimagenesRaza	638 ms	✓

TODO

Problems

Git

Terminal

Logcat

Build

Profiler

Tests Passed: 1 passed (4 minutes ago)

Run: 

app ×

RemoteTest ×

▶ Status

1 passed

1 tests, 2 s 568 ms

⚙ Filter tests:

✓

⊖

↶ ↷

↑

↓

🔄

📄

📄

Tests	Duration	Pixel_4_API_30
✓ Test Results	670 ms	1/1
✓ RemoteTest	670 ms	1/1
✓ perroTestRandom	670 ms	✓

Structure

Favorites

Build Variants

# CODIGO

---

## RemoteTest.kt

```
package com.example.dogspics

import android.content.Context
import androidx.arch.core.executor.testing.InstantTaskExecutorRule
import androidx.room.Room
import androidx.test.core.app.ApplicationProvider
import androidx.test.ext.junit.runners.AndroidJUnit4
import androidx.test.filters.MediumTest
import com.example.dogspics.dao.PerroDao
import com.example.dogspics.db.BaseDeDatos
import com.example.dogspics.model.PerroRandom
import com.example.dogspics.network.DogApiService
import com.example.dogspics.repository.Repositorio
import com.google.common.truth.Truth.assertThat
import kotlinx.coroutines.*
import kotlinx.coroutines.Dispatchers.IO
import kotlinx.coroutines.Dispatchers.Main
import okhttp3.mockwebserver.MockResponse
import okhttp3.mockwebserver.MockWebServer
import org.junit.*
import org.junit.runner.RunWith
import retrofit2.Retrofit
import retrofit2.converter.gson.GsonConverterFactory

@ExperimentalCoroutinesApi
@RunWith(AndroidJUnit4::class)
@MediumTest
class RemoteTest {

    @get:Rule
    var instantTaskExecutorRule = InstantTaskExecutorRule()

    private val mockWebServer = MockWebServer()
    private val MOCK_WEBSERVER_PORT = 8000
    private lateinit var repositorioTest: Repositorio
    private lateinit var repositorio: Repositorio

    private lateinit var database: BaseDeDatos
    private lateinit var dao: PerroDao

    @Before
    fun setup() {
        mockWebServer.start(MOCK_WEBSERVER_PORT)
```

```

    val context = ApplicationProvider.getApplicationContext<Context>()
    database = Room.inMemoryDatabaseBuilder(context, BaseDeDatos::class.java)
        .allowMainThreadQueries()
        .build()

    dao = database.dao()

    val retrofitClientTest by lazy {
        Retrofit.Builder()
            .baseUrl(mockWebServer.url("/"))
            .addConverterFactory(GsonConverterFactory.create())
            .build()
            .create(DogApiService::class.java)
    }

    repositorioTest = Repositorio(retroClientTest, dao)
    repositorio = Repositorio(retroClientTest, dao)

}

@After
fun teardown() {
    mockWebServer.shutdown()
    database.close()
}

@Test
fun perroTestRandom() {

    val perroOriginal =
        PerroRandom("https://images.dog.ceo/breeds/keeshond/n02112350_9580.jpg",
            "success")

    mockWebServer.apply {
        enqueue(
            MockResponse()
                .setResponseCode(200)
                .setBody(
                    FileReader.readStringFromFile("respuesta_falsa_perro_random.json")
                )
        )

        runBlocking {
            val perro = repositorioTest.perroRandomTest().body()

            assertThat(perro).isEqualTo(perroOriginal)
        }
    }
}

```

```

    }

    }

}

@Test
fun listadoRazasTest(){

    mockWebServer.apply{
        enqueue(
            MockResponse()
                .setBody(
                    FileReader.readStringFromFile("respuesta_falsa_perro_random.json")
                )
        )

        runBlocking {
            val listadoApi = repositorio.listadoRazaAPI()
            val listadoMock = repositorioTest.listadoRazaAPI()

            assertThat(listadoApi.size).isEqualTo(listadoMock.size)
        }
    }
}

@Test
fun listadoImagenesRaza(){
    mockWebServer.apply {
        MockResponse()
            .setBody(
                FileReader.readStringFromFile("mock_raza.json")
            )

        GlobalScope.launch {
            val listadoApi = repositorio.imagenesPorRaza("labrador").imagenes
            val listadoMock =
                repositorioTest.imagenesPorRaza("hound").imagenes

            assertThat(listadoApi.size).isEqualTo(listadoMock.size)
        }
    }
}

}

```