

Autor: Pablo Palanques Gil

Pruebas UI

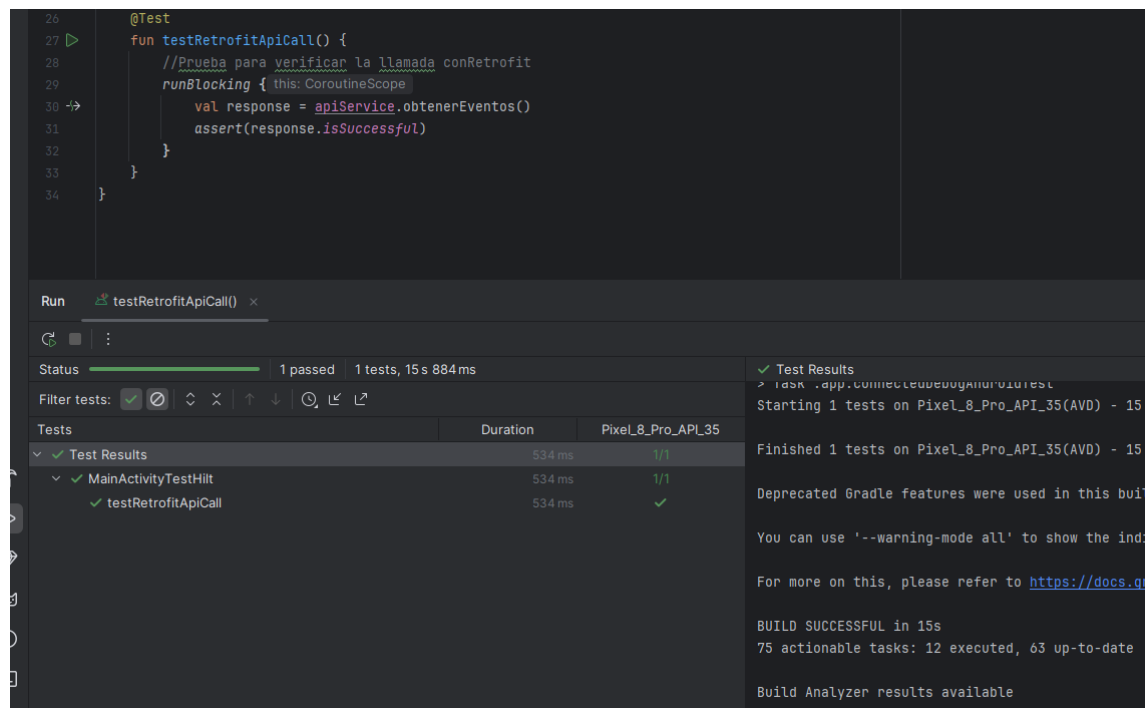
En este documento se recogen las pruebas realizadas sobre la aplicación Eventos-Retrofit y sus resultados. Se ha implementado Hilt con objeto de poder inyectar la api de Retrofit en las pruebas. Dichas pruebas están codificadas en la clase MainActivityTestHilt, en el directorio androidTest.

Test Retrofit

Test de la Api Retrofit: primero que nada se ha verificado que hilt funciona inyectando la clase EventoApiService y comprobando que la respuesta de la función obtenerEventos es exitosa.

```
@Test
fun testRetrofitApiCall() {
    //Prueba para verificar la llamada conRetrofit
    runBlocking {
        val response = apiService.obtenerEventos()
        assert(response.isSuccessful)
    }
}
```

Resultado: positivo.



The screenshot displays an IDE interface with a code editor at the top and a 'Run' panel at the bottom. The code editor shows the same Kotlin test function as in the previous block. The 'Run' panel indicates that the test 'testRetrofitApiCall' passed successfully. The 'Test Results' table shows the test duration as 534 ms and a status of 1/1. The 'Build Results' section on the right confirms that the build was successful in 15s, with 75 actionable tasks: 12 executed and 63 up-to-date.

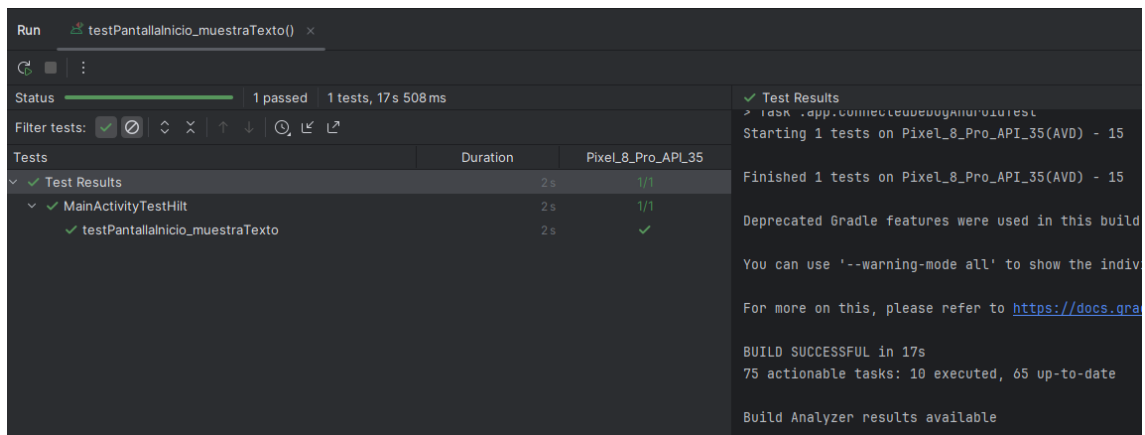
Test Results	Duration	Pixel_8_Pro_API_35
Test Results	534 ms	1/1
MainActivityTestHilt	534 ms	1/1
testRetrofitApiCall	534 ms	✓

Build Results: BUILD SUCCESSFUL in 15s, 75 actionable tasks: 12 executed, 63 up-to-date.

TESTS UI

Test UI 1: comprobamos que del MainActivity se carga la PantallaInicio y se muestra el texto “Toca para comenzar”.

```
//Primer test UI: Main Activity llama a AppNavigation y ésta carga
PantallaInicio, donde deberá
//aparecer un texto: "Toca para empezar". Comprobamos que el texto se
muestra:
@Test
fun pantallaInicio_muestraTocaParaComenzar() {
    composeTestRule.setContent {
        AppNavigation()
    }
    //Verificamos que el texto "Toca para comenzar" aparece en la UI
    composeTestRule.onNodeWithText("Toca para
comenzar").assertIsDisplayed()
}
```



Test UI 2: se pretende hacer clic en la pantalla de inicio, navegar a la segunda pantalla “MostrarEventos” y verificar que se muestran los eventos que deben cargarse del servidor.

```
//Segundo test UI: Carga de Eventos del servidor. Si pulsamos sobre la
pantalla de inicio
// navegamos hasta la pantalla MostrarEventos donde deberán cargar los
eventos del backend.
// Comprobamos que se muestran:
@Test
fun mostrarEventosTest() {
    //Prueba para verificar la llamada con Retrofit
    runBlocking {
        val response = apiService.obtenerEventos()
        assert(response.isSuccessful)
    }
    //Le damos a Compose la información para la navegación
    composeTestRule.setContent {
        AppNavigation()
    }
    composeTestRule.onNodeWithText("Toca para
comenzar").performClick()
    composeTestRule.waitForIdle() //Esperamos a que termine
    //Comprobamos que se muestran los eventos
}
```

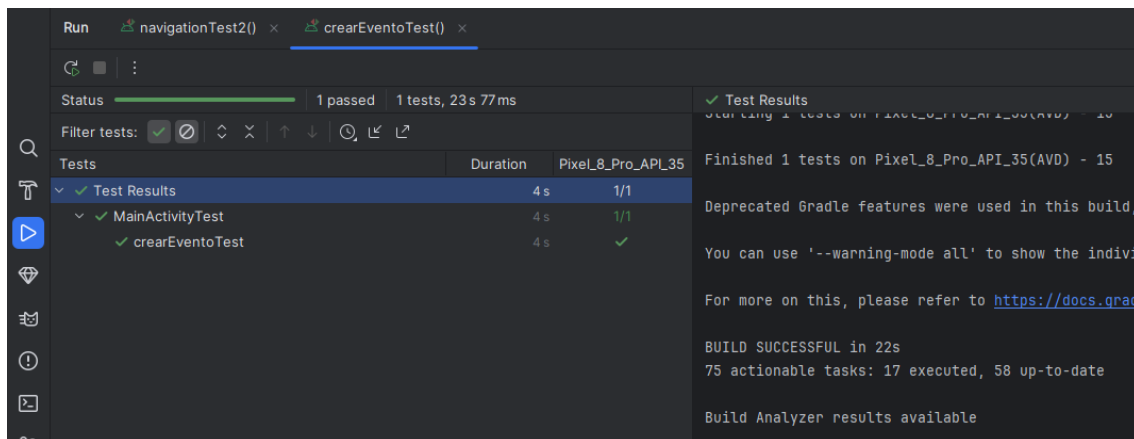
```
        composeTestRule.onNodeWithTag("Evento").assertIsDisplayed()
    }
}
```

Test UI 3: Se pretende verificar que saltan los mensajes de error al intentar introducir un formulario no válido.

Para comprobarlo, se debe navegar hasta la pantalla *MostrarEventos*, localizar el FAB para crear un nuevo evento, hacer clic y verificar que se navega hasta la pantalla *EditarEvento*. Una vez aquí, en principio no se muestran los mensajes de error por contenido de campos no válidos (por ejemplo, el *título* y la *categoría* de un evento son obligatorios). Comprobamos que efectivamente al llegar no se muestra el texto de error del campo *título*, seguidamente localizaremos y clicaremos en el botón de *Guardar* y finalmente comprobaremos que sí se muestra el texto de error por campo *título* vacío.

```
//Tercer test UI: crear Eventos. Si pulsamos sobre el FAB nos llevará
a la pantalla de
// EditarEvento. Por defecto los campos estarán vacíos, si intentamos
guardar el evento
// aparecerán los Text de error que indican los campos obligatorios.
// Comprobamos que aparece el aviso del campo Título:
@Test
fun crearEventoTest() {
    composeTestRule.setContent {
        //Prueba para verificar la llamada conRetrofit
        runBlocking {
            val response = apiService.obtenerEventos()
            assert(response.isSuccessful)
        }
        //Le damos a Compose la información para la navegación
        composeTestRule.setContent {
            AppNavigation()
        }

        composeTestRule.onNodeWithText("Toca para
comenzar").performClick()
        composeTestRule.waitForIdle() //Esperamos a que termine
        composeTestRule.onNodeWithTag("CrearEventoFAB").performClick()
        composeTestRule.waitForIdle() //Esperamos a que termine
        //Una vez en la pantalla EditarEvento, comprobamos que por
defecto no aparecen
        //los mensajes de error por campos no válidos. Comprobamos el
de título:
        composeTestRule.onNodeWithTag("Titulo es
obligatorio").assertIsNotDisplayed()
        //Ahora localizamos el botón de Crear Evento y clicamos
        composeTestRule.onNodeWithTag("Guardar evento").performClick()
        composeTestRule.waitForIdle() //Esperamos a que termine
        //Comprobamos que aparece el Text de error por campo Título
vacío:
        composeTestRule.onNodeWithTag("Titulo es
obligatorio").assertIsDisplayed()
    }
}
```



Test UI 4: Se pretende comprobar que podemos editar un evento, actualizarlo y que se reflejan los cambios en la interfaz.

Para ello, hacemos clic en la pantalla de inicio, vamos a la pantalla MostrarEventos.kt, localizamos un Evento cualquiera y clicamos en él para acceder a su pantalla de detalle (DetalleEvento.kt). Desde ahí, almacenamos el valor de su estado de *favorito* (Boolean) que se muestra por pantalla. Seguidamente, hacemos clic en el FAB de edición, cambiamos su estado de favorito, guardamos y una vez de vuelta en su pantalla de detalle, verificamos que el estado actual es distinto al almacenado previamente.

```
//Cuarto test UI: comprobamos que podemos acceder a la información de
//un evento y modificarlo
//y los cambios se reflejan en la UI:
@Test
fun editarEventoTest() {
    //Prueba para verificar la llamada conRetrofit
    runBlocking {
        val response = apiService.obtenerEventos()
        assert(response.isSuccessful)
    }
    //Le damos a Compose la información para la navegación
    composeTestRule.setContent {
        AppNavigation()
    }

    //Comprobamos que automáticamente carga la pantalla de inicio
    composeTestRule.onNodeWithText("Toca para
comenzar").performClick()
    composeTestRule.waitForIdle() //Esperamos a que termine
    //Navegamos a la pantalla de MostrarEventos
    composeTestRule.onNodeWithTag("Evento").performClick()
    composeTestRule.waitForIdle() //Esperamos a que termine
    //Navegamos a la pantalla de DetalleEvento y localizamos el icono
    //del corazón
    //que indica si es no favorito y almacenamos su estado:
    composeTestRule.waitForIdle() //Esperamos a que termine
    val fav = composeTestRule.onNodeWithContentDescription("Evento
favorito").isDisplayed()
    //fav será true si este evento es favorito y false en caso
    contrario.
    //Ahora localizamos el FAB de edición de evento y hacemos clic:
    composeTestRule.waitForIdle() //Esperamos a que termine
```

```

        composeTestRule.onNodeWithTag("Editar evento FAB").performClick()
        composeTestRule.waitForIdle() //Esperamos a que termine
        //En la pantalla de edición de evento, clicamos en el icono de
        favorito:
        composeTestRule.onNodeWithTag("Alternar favorito").performClick()
        composeTestRule.waitForIdle() //Esperamos a que termine
        //Localizamos el botón de actualizar evento y clicamos para volver
        a la pantalla anterior:
        composeTestRule.onNodeWithTag("Guardar evento").performClick()
        composeTestRule.waitForIdle() //Esperamos a que termine
        //Almacenamos de nuevo el estado del icono de favorito:
        val newFav =
            composeTestRule.onNodeWithContentDescription("Evento
favorito").isDisplayed()
        composeTestRule.waitForIdle() //Esperamos a que termine
        //Y comprobamos que su estado ha cambiado:
        assertFalse(fav == newFav)
        //Si tiene éxito, ejecutar de nuevo para no alterar los datos de
        la BD
    }

```

The screenshot shows the Android Studio Run interface. At the top, there are tabs for 'navigationTest2()' and 'editarEventoTest()'. Below the tabs, the 'Run' button is visible. The 'Status' bar indicates '1 passed' and '1 tests, 27s 956ms'. The 'Filter tests' section shows a search icon and a list of tests. The 'Tests' table shows the following results:

Tests	Duration	Pixel_8_Pro_API_35
✓ Test Results	3 s	1/1
✓ MainActivityTest	3 s	1/1
✓ editarEventoTest	3 s	✓

On the right side, the 'Test Results' panel shows the following output:

```

Finished 1 tests on Pixel_8_Pro_API_35(AVD) - 15
Deprecated Gradle features were used in this build
You can use '--warning-mode all' to show the indi
For more on this, please refer to https://docs.gr
BUILD SUCCESSFUL in 27s
75 actionable tasks: 21 executed, 54 up-to-date
Build Analyzer results available

```

Actualización 04/02/2025: al reintentar los tests vemos que salvo el de hilt con retrofit y el primero, el resto fallan. Parece que la navegación en los tests no está funcionando. Queda pendiente la revisión. La aplicación funciona normalmente, sólo fallan los tests.