Android Integration Test Activity Scenario Kullanımı

Activity'ler uygulamamız içinde her kullanıcı etkileşiminin gerçekleştirildiği konteynerdır. Bundan dolayı aşağıda belirtilen olaylar gerçekleştiğinde activity'inin gerçekleştireceği işlemleri test etmemiz çok önemlidir:

- Telefon uygulaması gibi başka bir uygulamanın bizim activity sınıfının işlemini böldüğünde,

- İşletim sisteminin activity destroy ve recreate ettiğimi durumlarda,

- Kullanıcı activity'i çoklu pencereye taşıdığında

Özetle bir activity'nin hayat döngülerinde sağlıklı çalışıp çalışmadığını test etmemiz gerekmektedir.

ActivityScenario Sınıfı

Bu sınıfı kullanarak yukarıda belirtmiş olduğumuz durumları test edebiliriz. Bu sınıfı hem unit testing'de hem de integration testing'de kullanabiliriz.

Öncelikle app/build.gradle dosyasına `androidTestImplementation 'androidx.test:core-ktx:1.3.0'` kütüphanesi eklenmelidir.

```
@RunWith(AndroidJUnit4::class)
class MyTestSuite {
     @Test fun testEvent() {
          val scenario = launchActivity<MyActivity>()
     }
}
```

Activity create edildiğinde, **ActivityScenario** tarafından RESUMED durumuna geçirilir. Bu aşama activity'nin ekranda göründüğü aşamadır. **Activity'nin** kullandığı View elementlerine (Button gibi) Espresso UI testlerini kullanarak erişebiliriz.

Yukarıdaki kod kullanımına alternatif olarak, her testten önce launch fonksiyonunu çağırmak için **ActivityScenarioRule** sınıfını kullanabiliriz. Bu sınıf test bittiğinde **ActivityScenario.close** metodunu çağırır. Yani otomatik olarak testten önce launch, testten sonra close çağrılır. Aşağıdaki örnekte **testEvent** fonksiyonu çalışmadan önce launch fonksiyonu otomatik olarak çağrılır. Çağrıldığında **Activity** başlatılır, close fonksiyonu ile birlikte activity otomatik olarak kapatılır.

```
@RunWith(AndroidJUnit4::class)
class MyTestSuite {
     @get:Rule var activityScenarioRule = activityScenarioRule<MyActivity>()

     @Test fun testEvent() {
          val scenario = activityScenarioRule.scenario
     }
}
```

Activity Yeni State Durumuna Getirmek

ActivityScenario sınıfının **moveToState()** metodunu kullanarak **Activity'i** CREATED, STARTED gibi durumlara getirebiliriz.

```
@RunWith(AndroidJUnit4::class)
class MyTestSuite {
     @Test fun testEvent() {
          val scenario = launchActivity<MyActivity>()
          scenario.moveToState(State.CREATED)
     }
}
```

Activity'nin Durumunu Test Etmek

ActivityScenario nesnesinin public olarak tanımlanmış state field'i ile activity'nin o anki durumuna erişebiliriz. Bu field ile activity'nin başka bir activity'i başlatıp başlatmadığını, destroy durumunu vs kontrol edebiliriz. Örneğin aşağıdaki test kodu ile **MyActivity'den MyOtherActivity'nin** başlatılması işlemi yapılmaktadır.

```
  @RunWith(AndroidJUnit4::class)
class MyTestSuite {
    @Test fun testEvent() {
        val scenario = launchActivity<MyActivity>()
        scenario.onActivity { activity ->
          startActivity(Intent(activity, MyOtherActivity::class.java))
        }

        val originalActivityState = scenario.state
    }
}
```

Activity ReCreate Etmek

Android işletim sistemi RAM, CPU fazla kullanıldığında çalışan uygulamaları kapatabilmektedir. Kullanıcı bizim uygulamamıza geri döndüğünde otomatik olarak sistem **Activity** recreate eder. Bu işlemi aşağıdaki gibi simulate edebiliriz:

```
@RunWith(AndroidJUnit4::class)
class MyTestSuite {
    @Test fun testEvent() {
        val scenario = launchActivity<MyActivity>()
        scenario.recreate()
    }
}
```

Activity Sonucunu Almak

Activity tamamlandığında dönüş kodunu veya activity ile ilişkili veriyi almak için **ActivityScenario** nesnesinin result field kullanılmaktadır.

```
@RunWith(AndroidJUnit4::class)
class MyTestSuite {
    @Test fun testResult() {
        val scenario = launchActivity<MyActivity>()
        onView(withId(R.id.finish_button)).perform(click())

        // Activity under test is now finished.

        val resultCode = scenario.result.resultCode
        val resultData = scenario.result.resultData
    }
}
```

Uyarı: **ActivityScenario** otomatik olarak **Activity'nin finish()** metodunu çağırmaz. Test edilen **Activity finishing** veya **finished** olmazsa, **testResult** metodu timeout olur ve exception meydana gelir.

Activity sınıfındaki View'lerle Etkileşime Geçmek

Espresso view matcher'larını kullanarak aşağıdaki gibi bir butona tıklama event'ini simulate edebiliriz:

```
@RunWith(AndroidJUnit4::class)
class MyTestSuite {
     @Test fun testEvent() {
          val scenario = launchActivity<MyActivity>()
          onView(withId(R.id.refresh)).perform(click())
     }
}
```

Activity sınıfında tanımlanmış bir fonksiyonu yukarıdaki gibi bir event ile çağırmak yerine direk şu şekilde çağırabiliriz:

```
@RunWith(AndroidJUnit4::class)
class MyTestSuite {
     @Test fun testEvent() {
          val scenario = launchActivity<MyActivity>()
          scenario.onActivity { activity ->
            activity.handleSwipeToRefresh()
          }
     }
}
```