# C.4 - Integration test

Finora abbiamo testato il codice in Unit test che, sebbene abbiano molte qualità, non consentono però di testare funzionalità che possono fallire anche se ogni componente, formalmente, si comporta in maniera corretta.

Gli Integration Tests che vedremo in questo capitolo, insieme ai End-to-End tests, sono considerati **Instrumented Tests,** perché gireranno non sulla nostra macchina dev, ma su di un device o emulatore Android.

Questi test sono inoltre salvati all’interno del progetto nel package *androidTest* per differenziarli dagli Unit Test.

***Graphical user interface, text, website

Description automatically generated***

Gli Instrumented test solitamente scambiano la velocità di esecuzione degli Unit test per una fedeltà maggiore all’esperienza dell’utente. In una immagine:

Diagram

Description automatically generated

Mentre nella parte bassa della piramide avremo tanti test a fedeltà bassa, piu’ saliamo meno sarà il numero di test, che avranno anche maggiore fedeltà e tempo di esecuzione.

## Environment setup

Come gli Unit test, importiamo delle librerie con le quali lavoreremo.

In build.gradle :app sostituisci gli import *androidTestImplementation* con:



Questi import copriranno la maggior parte degli Instrumented Test del corso.

Bisogna anche modificare alcune impostazioni del nostro device\emulatore.   
Andando su **Settings > Developer options** disattiva le seguenti animazioni:

* Window animation scale
* Transition animation scale
* Animator duration scale

## Introduzione a IT su Android

Gli Integration Tests verificano l'interazione tra diverse classi, per assicurarsi che si comportino come previsto quando vengono utilizzate insieme.

Un modo di sviluppare Integration Tests è dunque quello di testare funzionalità, come ad esempio **l’abilità di rigenerare la lista**.

Creiamo un test per MainActivity, assicurandoci di salvarlo nella directory *androidTest*

*Graphical user interface, text, application, chat or text message

Description automatically generated* *A screenshot of a computer

Description automatically generated with medium confidence*

Aggiungiamo le seguenti annotazioni prima della dichiarazione della classe test:

*Text

Description automatically generated*

*MediumTest* identifica lo scopo del test, e sarà utile al sistema per assegnargli risorse, e a noi per differenziare tra i vari test.

Il nostro test ora dovrà testare le funzionalità di Main Activity, ma ancora una volta non ci è possibile farlo direttamente, altrimenti ci ritroveremmo a testare tutto il framework.

La libreria di test ci offre degli *scenari* che possiamo creare per simulare delle situazioni.

Text

Description automatically generated

Abbiamo creato uno scenario che simula la Main Activity.

Ora siamo pronti ad effettuare i nostri test!

## Espresso

<https://developer.android.com/training/testing/espresso/basics>

Useremo la libreria di test **Espresso** che similmente a Truth ha una struttura dichiarativa, ma oltre alle assertions ci permette di fare anche delle azioni, come cliccare su un pulsante o inserire del testo in un campo editabile.

Espresso lavora su 3 categorie:

**ViewMatchers – targettano la view**

**ViewActions – effettuano azioni**

**ViewAssertions – condizionali per la riuscita del test**

Text

Description automatically generated

Lanciando il test, vedremo che la nostra app sarà eseguirà velocemente le interazioni che abbiamo scritto – cerca la view, click, controlla – il cui risultato sarà poi disponibile nell’IDE come per gli Unit Test.

Controlliamo che la pressione del FAB effettivamente cambi il dataset.

* In Main Activity, rendi *dataset* pubblico.
* In Main Activity test, salva un riferimento a *dataset*

Text

Description automatically generated

Scenario.*onActivity* ci da un riferimento all’activity appena è pronta, permettendoci di interagire con i suoi componenti.

Poi in un nuovo test:

Text

Description automatically generated

* Crea una Deep Copy di dataset in *oldDataset*. Questo è necessario perché altrimenti otterremmo solo un riferimento all’oggetto *dataset*, e alla fine ci ritroveremmo a compararlo con se stesso!
  + Approfondimento: [Deep Copy vs Shallow Copy](https://www.baeldung.com/cs/deep-vs-shallow-copy) (baeldung.com)
* Click sul FAB
* Check che il nuovo dataset sia diverso dal vecchio

## Recyclerview Utilities

Vogliamo anche assicurarci che il nuovo dataset sia correttamente mostrato nella lista. Potremmo controllare l’id del primo elemento, ma facciamo un passo in più decidendo di controllare l’id dell’ultimo elemento.

Per fare questo pero’, bisogna scrollare la lista fino alla fine.

Le **RecyclerView utilities** permettono di effettuare azioni sulle liste, come scroll e azioni targettate a una specifica riga

Text

Description automatically generated

Sono tutte actions, quindi devono essere inserite come ViewActions.

Continuando il test di prima, scriviamo un’azione che scorre la lista fino a un membro che abbia come id l’ultimo del dataset:

Text

Description automatically generated

* hasDescendant permette di andare a fondo nella gerarchia del layout:   
  RecyclerView > ViewHolder > ConstraintLayout > TextView

Infine, controlliamo con una assertion Espresso se è visibile. Ecco il test completo:

Text

Description automatically generated

## Esercizio: Test Card Collection

Tramite Integration Test, testa le funzionalità della Card Collection. Non dimenticarti degli Unit Test!