# C.1 – Input Utente

## EditText, TextInputLayout

Un’operazione molto comune in un’app è l’inserimento di testo o numeri in un campo di testo. Lo strumento classico è il widget EditText, mentre uno più moderno è il TextInputLayout. Entrambi sono validi e possono essere applicati in base alle casistiche.

Nell’editor *Design*, sotto la categoria Text è possibile utilizzare dei campi precompilati. Questi campi limitano l’input dell’utente a certe categorie – ad esempio, solo numeri.

Graphical user interface, text, application

Description automatically generated

Trascina *Plain Text* nella parte bassa del layout, regolando i constraint delle due view per fare in modo che siano una sopra l’altra:

Text

Description automatically generated

Lanciando l’app vediamo che è possibile scrivere all’interno del campo di testo.

EditText è un widget che permette di inserire testo

Decidiamo che l’utente potrà spingere un pulsante per aggiungere il testo ad Hello world, per formare un racconto.

Ora il campo di testo è già prepopolato, e bisogna cancellare a mano il testo “Name”. Però questo testo è necessario, altrimenti l’utente non capirebbe cosa bisogna scrivere nel campo di testo. Inoltre, quando apriamo la tastiera, l’impostazione *inputType=textPersonName* fa sì che la tastiera venga modificata per non includere un “a capo”, cosa che non è adatta a scrivere una storia.

Andiamo a cambiare il *text* in *hint*, inoltre modifichiamo anche l’inputType in *text:*

Text

Description automatically generated

Ora l’*hint* sarà visibile in grigio e potremo scriverci sopra. La tastiera non avrà impostazioni particolari.

Application

Description automatically generated with low confidence

Fin qui tutto bene, ma c’è una limitazione: se dovessimo inserire più campi di testo, come ad esempio in un form dove si richiedono le informazioni dell’utente, le hint dei vari EditText diventerebbero meno intuitive, perché una volta scritto nel campo di testo non sono piu visibili. Ecco un esempio:

Table

Description automatically generated Table

Description automatically generated with medium confidence

Una volta scritto all’interno dei campi non abbiamo più riferimenti per capire chi sia l’eroe e chi il villain.

Potremmo cominciare ad inserire delle label sopra i campi di testo, ma essendo questa una casistica molto comune, Android ci fornisce un widget che fa proprio questo: il *TextInputLayout*

TextInputLayout è un widget che permette di inserire testo, facilitando lo stile

A picture containing chart

Description automatically generated

Aggiungiamolo e vediamo che è in realtà una coppia di widget:

A picture containing text

Description automatically generated

* TextInputLayout (TIL) si occupa di gestire constraint, dimensioni e animazioni
* TextInputEditText (TIET) gestisce testo, stile e input

Applica, secondo questi paradigmi, le impostazioni della edittext precedente a questo widget;   
Diamo a entrambi i componenti del widget un id;   
Infine, commenta l’edittext:

Text

Description automatically generated

Per il nostro obbiettivo, aggiungiamo anche un pulsante che l’utente andrà a spingere una volta che ha scritto il testo:

Graphical user interface, text, application

Description automatically generated

Layout:



Ora andiamo ad ottenere programmaticamente il testo scritto.

Secondo il nostro caso d’uso, quando l’utente spinge il pulsante dobbiamo aggiungere il testo al messaggio stampato dall’app.

All’interno del *clickListener* di storyAddBtn:

Graphical user interface, text, application, website

Description automatically generated

* È necessario convertire il testo di storyWriteTiet in stringa, perché è un *Editable*, a mostrare il fatto che può essere modificabile.
* Il carattere speciale “\n” in una stringa crea una nuova riga.

Una volta ottenuto il testo, concatenalo al testo di storyTextTv:

Text

Description automatically generated

* Kotlin permette di compattare le funzioni *setText* e *getText* semplicemente in *text*, che verrà considerato getter o setter in base a come viene utilizzato.
* Il Tipo di TextView.text è l’interfaccia [Charsequence](https://developer.android.com/reference/java/lang/CharSequence), perché il contenuto di una TextView puo’ essere qualsiasi tipo di testo (ad esempio, un *Editable*)

Infine, *clean* il campo di testo in maniera tale che l’utente possa continuare a scrivere comodamente:

storyWriteTiet.setText("") //sets the Editable content to a string

Ora lanciando l’app, possiamo vedere che alla pressione di Add il testo viene aggiunto ed il campo di testo resettato.

Graphical user interface, text, application

Description automatically generated

## RadioButtons

I RadioButtons vengono utilizzati quando l’utente deve effettuare una scelta tra più opzioni.

RadioGroup è un layout che include più RadioButton, un widget che permette all’utente di fare una scelta univoca.

Aggiungiamo un RadioGroup con 3 opzioni, inoltre aggiungiamo un testo come label, perché RadioGroup non fornisce una label.

Graphical user interface

Description automatically generated

Layout:



In questo esempio abbiamo impostato il secondo pulsante del radio group a *checked,* in maniera tale che sarà subito selezionato.

Decidiamo che la scelta del radio group influenzerà il background del testo

Similmente a Button, RadioGroup fornisce la possibilità di aggiungere un listener per alcuni eventi, ad esempio quando cambia il membro selezionato tra i suoi child.

storyOptionsRg.setOnCheckedChangeListener **{** radioGroup, checkedId **->**

andando a fare un razionale su *checkedId* possiamo impostare come modificare l’interfaccia in base al pulsante che è stato *checked*

Text

Description automatically generated

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

Description automatically generated

## Compatibilità dei widget

Alcune volte è necessario porre particolare attenzione alla versione del widget che si sta usando. Poniamo ad esempio di voler inserire uno Switch.

Switch è un interruttore, permette all’utente di attivare o disattivare una funzione

Aggiungiamo uno Switch, sopra il pulsante di aggiunta:

<Switch  
 android:id="@+id/story\_caps\_sw"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"   
 android:text="@string/all\_caps"  
 android:layout\_margin="12dp"  
 app:layout\_constraintEnd\_toEndOf="parent"  
 app:layout\_constraintBottom\_toTopOf="@id/story\_write\_til"/>

Graphical user interface, application

Description automatically generated

L’IDE ci menziona che non è ideale usare questa versione del widget e ci suggerisce delle alternative:

Text

Description automatically generated

Questo perché il widget Switch non include delle retrocompatibilità con vecchie versioni di android, implementate invece da SwitchCompat e dal suo successore SwitchMaterial.

È spesso più importante utilizzare classi compatibili con le versioni di android che si decide di supportare, rispetto a classi più aggiornate e funzionalità più avanzate

## ScrollView

Il nostro layout si riordina in base alle dimensioni dello schermo del device, ma nel caso le view contenute nel layout richiedano più spazio di quello disponibile su schermo, potrebbero cominciare a sovrapporsi o ad estendersi al di fuori dello spazio visibile all’utente.

Aggiungi questi campi alla textview:

android:layout\_height="300dp"  
android:background="@android:color/darker\_gray"

Questo imposterà il campo di testo ad un’altezza statica. Vedendo la preview in modalità landscape:

A screenshot of a computer

Description automatically generated with medium confidence

Ora il campo di testo richiede troppo spazio, e si sovrappone ai comandi.

Rendiamo il nostro layout scrollabile

Per farlo, dobbiamo racchiudere il Constraint Layout dentro una scrollview.

ScrollView è in grado di far scrollare verticalmente o orizzontalmente il suo contenuto. Può avere solo un singolo discendente diretto.

Una implementazione comune è impostare la scrollview a match parent, per fare in modo che occupi tutto lo spazio dedicato al layout. Il suo contenuto invece viene impostato a wrap content, cosi’ da potersi allungare quanto necessario al di fuori dello schermo visibile all’utente.

* Aggiungi il viewgroup *ScrollView* prima del constraint layout

Text

Description automatically generated

* Sposta i campi *xmlns* , che effettuano gli import di configurazione, da constraint layout a scrollview. Sposta anche *tools:context*
* Rendi l’altezza di constraint layout *wrap\_content*

Text

Description automatically generated

* Alla fine del file, chiudi il tag ScrollView

Graphical user interface, text

Description automatically generated

Ora il layout scorre verticalmente.

## Esercizio: Hero Generator

Sviluppa un’app in una singola pagina che permette di generare la storia di un eroe.

* Un campo di testo deve definire il nome
* Un radioGroup permette di scegliere un superpotere o qualità
* Uno switch indica se ha un acerrimo nemico, oppure no
* Cliccando un pulsante, è possibile generare un testo discorsivo che include tutte le qualità selezionate.
  + Es: “Batman ha il superpotere di essere ricco. Il suo acerrimo nemico è Mario Rossi.”
* Il testo deve essere posto in fondo al layout. Il layout deve essere scrollabile.
* Estrarre la o le funzioni di generazione, ed effettuare unit test.