# C.2 - Material Design

Il Material Design è un insieme di guidelines per definire non solo come progettare l’interfaccia (UI), ma anche come le view debbano interagire visivamente, con l’utente e tra di loro, in maniera tale da creare una user experience (UX) fluida e istintivamente comprensibile.

Il tema del MD è molto ampio, per ulteriori approfondimenti e strumenti:

[m2.material.io](http://www.m2.material.io)

Per cominciare, lavora sulla versione 2 del material design, in quanto la più consolidata.

*Comincia questo capitolo creando un nuovo progetto con una Navigation Drawer Activity*

## Introduzione a design UI e UX

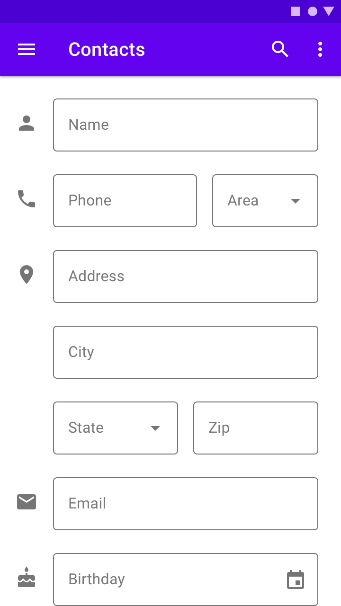
L’obbiettivo del design di un’app è rendere la sua interfaccia naturalmente fruibile da un utente, mantenendo al tempo stesso una sua unicità. Si può dividere il design in due concetti fondamentali:

* User Interface (UI): è come viene rappresentata graficamente l’interfaccia
* User Experience (UX): è come l’utente interagisce con l’app e naviga tra le funzioni

Molti concetti e azioni sono parte del nostro quotidiano utilizzo di telefoni e pc: è necessario allinearsi a queste abitudini per evitare che l’utente debba “studiarsi” l’interfaccia per utilizzare il nostro prodotto.

Da un punto di vista UX, l’ambiente mobile ha inoltre delle particolarità specifiche che rinforzano il concetto di rapida usabilità: ad esempio, è inusuale su un app trovare il pulsante “salva”, in quanto ci si aspetta che delle opzioni modificate siano subito attive, che un messaggio scritto ma non inviato venga salvato come bozza, o che una foto sia subito disponibile nella galleria.

Analizziamo la UI della seguente schermata:



Tutto quello che vediamo è facilmente identificabile come un’azione o un aiuto visivo.

* La Lente di ingrandimento in alto a destra identifica il campo di ricerca, senza che ci sia un testo a spiegarlo.
* Tappando i tre puntini in alto a destra ci aspettiamo che esca un menu contestuale alla schermata, mentre le tre linee in alto a sinistra rappresentano un menu generale dell’app
* Le icone vicino ai campi di testo non sarebbero necessarie, ma istintivamente ci dividono il form in delle sezioni piuttosto chiare. L’icona “location” ci fa capire che quei campi di testo rappresentano una posizione geografica
* Vedendo l’icona in basso a destra, ci si aspetta che tappando quel campo o l’icona stessa si aprirà un picker di calendario

Questa iconografia è spesso necessaria per questioni di spazio. Le app vengono utilizzate su di uno schermo relativamente piccolo per la quantità di informazioni che contengono: quindi dobbiamo veicolare queste informazioni in modo chiaro, ma coinciso, ed anche pratico da utilizzare quando si è per strada.

Sono disponibili guidelines per ogni view Material, così da avere un riferimento durante la progettazione e lo sviluppo, a questo indirizzo

<https://m2.material.io/components>

## Colori

*Comincia questo capitolo creando un nuovo progetto con una Navigation Drawer Activity*

<https://m2.material.io/design/color/the-color-system.html>

Nell’interfaccia di un’app, i colori non sono solo un abbellimento o una brandizzazione, ma rappresentano funzionalità e concetti.

Immagina di dover progettare l’app per IKEA.

A picture containing text, sign, clipart

Description automatically generated

Conosciamo questo brand per il giallo ed il blu, ma notiamo che nonostante il logo sia scritte blu su giallo, nei loro negozi invertono i colori. Questo perché un capannone giallo sarebbe un pugno in un occhio!

Un’app lavora su concetti simili. Avremo due “serie” di colori che identificano come vengono mostrate le nostre view:

* colorPrimary identifica il colore principale del brand, e sarà quello mostrato più frequentemente nelle schermate dell’app
* colorSecondary fa parte o no del brand, e può essere una variante del colorPrimary o un altro colore. Identificherà parti nell’app che vogliamo mettere in risalto, come i pulsanti o delle selezioni.

Il Material Design propone delle palette di colori da utilizzare:

Graphical user interface, application

Description automatically generated

<https://m2.material.io/design/color/the-color-system.html#tools-for-picking-colors>

Queste palette forniscono delle tonalità che sono studiate per avere un buon contrasto sugli schermi dei telefoni, differenziandosi tra le varie sfumature, oltre ad essere funzionali alla vista della maggior parte delle persone.

È quindi possibile differenziarsi da questi colori, ma è consigliato prima cercare di utilizzarli.

Queste palette includono delle varianti con la lettera **A (accent)** che è la dicitura precedentemente usata per i **colorSecondary**

Il file **/res/values/colors.xml** contiene le risorse colore.

Graphical user interface, text

Description automatically generated

I colori vengono definiti con la **hex notation** estesa, dove le coppie di lettere indicano in sequenza: trasparenza, rosso, giallo, blu.

Android studio ci offre un’anteprima del colore nella colonna di sinistra, che se andiamo a cliccare attiverà un color picker

Graphical user interface

Description automatically generated

Nella parte bassa, potremo selezionare i colori della palette Material.

## Theme

Oltre alle opzioni che permettono di stilizzare singolarmente ogni singola view, Android ci fornisce degli strumenti per effettuare una stilizzazione generica della nostra app.

Un Theme è una raccolta di attributi applicata a un'intera app, flusso o gerarchia di views

Apri il file themes.xml

Text

Description automatically generated

Vediamo che una serie di attributi mappa altrettanti colori tramite l’xml tag **<style>**. Questi attributi sono predefiniti, e vengono applicati ad ogni view Material tramite il tema Theme.MaterialComponents. Le view Material includono tutti i widget ed i layout che abbiamo utilizzato finora, e che utilizzeremo.

I nomi degli attributi rappresentano i concetti di primary e secondary espressi nelle guidelines del Material Design. Alla seguente pagina sono presenti delle tabelle che indicano la funzionalità di ogni attributo:

<https://m2.material.io/develop/android/theming/color>

Ecco come appare l’app ora:

Graphical user interface, application, Teams

Description automatically generated

Vediamo come è possibile modificare il tema generale dell’app, per stilizzarla come se fosse l’app IKEA.

Aggiungi questi colori a colors.xml

<color name="indigo\_600">#3949AB</color>  
<color name="indigo\_900">#1A237E</color>  
<color name="yellow\_400">#FFEA00</color>  
<color name="yellow\_700">#FFD600</color>

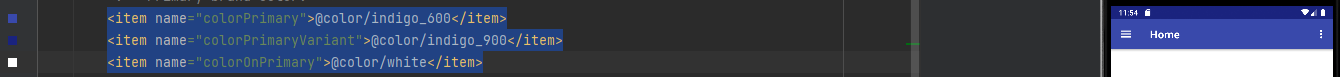
Modifica ora i riferimenti ai colori nel tema, cambiando i primary in indigo e i secondary in yellow

Rilancia l’app:

Graphical user interface, application

Description automatically generated

Vediamo come i colori primari influenzano alcune view principali, come ad esempio la toolbar dell’app (primary) e di sistema (primaryVariant).   
OnPrimary definisce il colore delle scritte o icone che si trovano sopra i colori primari, in maniera che risaltino.



I colori secondari definiscono invece view come pulsanti o dialoghi di notifiche, che veicolano all’utente la possibilità di effettuare azioni, o messaggi.



## Style

Spesso ci ritroviamo a dover standardizzare alcune view: un caso d’uso tipico è che tutte le TextView che fungono da titolo abbiano la stessa grandezza del carattere e siano in bold.

Uno Style è una raccolta di attributi applicata a molteplici istanze di una view.

Nel nostro progetto abbiamo tre pagine con una descrizione:



Ansiamo a cambiare lo *stile* di questo testo, in maniera tale da raggruppare gli attributi che poi applicheremo a 3 textview in 3 layout diversi.

Text

Description automatically generated

In un nuovo file di risorse *styles*, creiamo uno *style* dandogli un nome, estendendo un parent di sistema: quest’ultimo punto è necessario per mantenere la retrocompatibilità dei widget.  
Nel nostro caso, lo stile sarà uguale a TextAppearance.AppCompat, tranne per gli attributi che specifichiamo.

Ora definiamo degli attributi come se li stessimo applicando alla TextView:

<item name="android:textSize">20sp</item>  
<item name="android:textStyle">bold</item>  
<item name="android:fontFamily">sans-serif-medium</item>

Possiamo scrivere qualsiasi attributo nello stile, ma la view lo considererà solo se è uno dei suoi attributi.

Infine, modifichiamo le 3 textview sostituendo   
android:textSize="20sp"  
che non è più necessario perché già incluso nel nostro nuovo stile, con  
style="@style/BoldMessage"

Graphical user interface

Description automatically generated with medium confidence

Lanciando l’app, vediamo che tutte le textview sono ora uguali.



Possiamo anche creare dei sotto-stili del nostro stile:

Text

Description automatically generated

Impostando BoldMessage*.large* avremo lo stesso risultato che con BoldMessage, tranne che il textSize sarà più grande.

## Icone

Le icone nelle app sono un metodo di comunicazione. Mettere un’icona può aiutare l’utente a chiarificare funzionalità, mentre icone superflue o fuori contesto possono confondere l’utente.

Android supporta vari tipi di immagine, ma i più comuni sono: png e jpg (raster), e vettoriali tramite xml.

È possibile scaricare numerose icone dal sito Material dedicato, ed è buona norma cercare di utilizzarlo, in particolare per le icone che identificano un’azione molto comune come “settings” o “share”. Le icone in questo sito sono infatti lo standard per Android, rimuovendo così una serie di problemi di ottimizzazione e mantenendo uno stile coerente non solo nell’app, ma anche con il sistema operativo.  
<https://fonts.google.com/icons?icon.platform=android>

Accade spesso però che non ci sia l’icona che stiamo cercando.   
Aggiungiamo un’icona vettoriale non-Material presa dal sito [Noun Project](https://thenounproject.com/icons/).   
Nel resource manager, importa un Vector Asset:

Graphical user interface, application

Description automatically generated

Scegli un file svg dal tuo computer

A screenshot of a computer

Description automatically generated with medium confidence

L’import convertirà il file da svg a xml, e nel caso di errori cercherà di completare il lavoro con il miglior risultato possibile. Sarà importata nella cartella *res/drawable*

Aprendo il file, vediamo che Android Studio ci notifica dei warning. Il Primo:

Text

Description automatically generated

Le immagini vettoriali vendono ridimensionate in base alla risoluzione – sono una serie di punti tra i quali vengono tracciate delle righe (vettori). Non è necessaria quindi una grandezza specifica, ma solo mantenere le proporzioni correte. Abbassiamo quindi i valori a 100 e 100.

Andiamo a vedere il secondo errore:

A screenshot of a computer

Description automatically generated

Il nostro disegno è troppo complesso con 1172 caratteri: la lunghezza consigliata è <800 caratteri.

Valutiamo subito se è possibile cambiare l’icona con un'altra, in quanto i passaggi di ottimizzazione sono spesso lunghi e complessi.

Se l’icona è necessaria, sono disponibili numerosi tool per ridurre la complessità dei file vettoriali. Nel caso anche i tool non siano sufficienti, è possibile modificarli a mano con editor vettoriali come Adobe Illustrator.

Infine, andiamo ad utilizzare l’icona come risorsa *drawable.*



## Launcher Icon

L’icona launcher rappresenta la nostra app nei menu di sistema, nello store e in altri contesti generici.

Viene implementata tramite la risorsa *mipmap*: una risorsa speciale esclusiva per le icone *launcher,* che devono essere particolarmente chiare non solo nelle varie risoluzioni, ma anche nelle customizzazioni del sistema operativo effettuate dai vari produttori di device.

Modifichiamo l’icona della nostra app (launcher icon).

Nel resource manager, importa un Image Asset:

Graphical user interface, application

Description automatically generated

ic\_launcher è il nome di default dell’icona dell’app. Per evitare di sovrascrivere, chiamiamola ic\_launcher\_2.

Graphical user interface, application

Description automatically generated

Andiamo a impostare i seguenti valori:

* **Foreground layer**: selezioniamo una clip art (nell’esempio, *business*) e un colore, dove utilizzeremo uno dei colori del nostro tema, primary o secondary (FFEA00)
* **Background layer**: utilizzeremo un colore del nostro tema (3949AB)

Assicurandosi che l’immagine sia all’interno del cerchio che rappresenta la *safe zone* dove l’immagine non potrà essere tagliata.

Confermando, Android Studio ci propone un’anteprima delle immagini che verranno create, incluso il launcher, varie risoluzioni, e per il play store.

Graphical user interface, application

Description automatically generated

Dopo aver creato le immagini, modifichiamo il *manifest* dove è specificata l’icona dell’app nel campo *icon*

Text

Description automatically generated

Lanciando la nostra app vediamo la nuova icona nell’elenco delle app:

Graphical user interface, application, Teams

Description automatically generated

## Esercizio: Stilizziamo Hero Generator

Seguendo i principi del Material Design fai queste modifiche a Hero Generator:

* Cambia il tema dell’app con dei colori personalizzati
* Crea ed applica uno Style per la\le textview che hanno lo stesso scopo
* Riposiziona le view e cambiane margin e padding, per trovare un bilanciamento tra UI e UX
* Aggiungi delle icone dove pensi sia utile o necessario
* Modifica campi di testo in maniera tale che siano Outlined