

Univeristà degli studi di Padova
Programmazione di sistemi embedded

Report - Building UIs

Marco Pavanetto, Lorenzo Croce, Alberto Bottari

14 giugno 2024

Indice

1	Introduzione	3
2	XML-Based Layouts	5
2.1	Approccio	5
2.2	Introduzione alle View e ViewGroup	5
2.3	Tipi di ViewGroup	5
2.3.1	LinearLayout	5
2.3.2	RelativeLayout	6
2.3.3	ConstraintLayout	7
2.3.4	GridLayout	7
2.3.5	RecyclerView	8
2.3.6	CoordinatorLayout	8
2.3.7	NestedScrollView	8
2.4	Tipi di View	9
2.5	Attributi delle View	10
3	Jetpack Compose	11
3.1	Introduzione a Jetpack Compose	11
3.2	Concetti Chiave di Jetpack Compose	11
3.2.1	Composable Functions	11
3.2.2	State	11
3.2.3	Recomposition	11
3.2.4	Modifiers	12
3.2.5	Layouts	12
3.3	Best Practices per Jetpack Compose	13
4	Hybrid Approaches	14
4.1	Flutter	14
4.1.1	Caratteristiche principali	14
4.1.2	Vantaggi	14
4.1.3	Svantaggi	14
4.1.4	Casi d'uso ideali	15
4.2	React Native	15
4.2.1	Caratteristiche principali	15
4.2.2	Vantaggi	15
4.2.3	Svantaggi	15
4.2.4	Casi d'uso ideali	16
4.3	Unity	16

4.3.1	Caratteristiche principali	16
4.3.2	Vantaggi	16
4.3.3	Svantaggi	16
4.3.4	Casi d'uso ideali	16
4.4	Conclusione	17
5	Custom Views	18
5.1	Introduzione alle Custom View	18
5.2	Vantaggi e Svantaggi delle Custom View	18
5.3	Best Practices per lo sviluppo di Custom View	19
6	Conclusione	20
6.1	Considerazioni Finali	20
7	Economy tracker	21
7.1	Struttura dell'applicazione	21
7.1.1	La pagina Home	21
7.1.2	La pagina Registro	24
7.1.3	La pagina Categorie	28
7.2	Differenze tra XML e Jetpack Compose	33
8	Bibliografia	34
8.1	XML	34
8.2	Jetpack Compose	35
8.3	Hybrid	35
8.4	Custom View	36

Capitolo 1

Introduzione

L'interfaccia utente (UI) è uno degli elementi cruciali nel design e nello sviluppo delle applicazioni mobili. Non solo rappresenta il punto di contatto tra l'utente e il software, ma influenza anche significativamente sull'esperienza complessiva dell'utente, sulla percezione dell'applicazione e sulla sua usabilità. Nel contesto del sistema operativo Android, la definizione e l'implementazione della UI possono essere affrontate attraverso diverse metodologie e strumenti, ciascuno con i propri vantaggi, svantaggi e casi d'uso ideali.

Android, sviluppato da Google, è il sistema operativo mobile più diffuso al mondo, alimentando miliardi di dispositivi. Questa vasta adozione richiede un ecosistema di sviluppo robusto e flessibile, capace di soddisfare le esigenze di una vasta gamma di applicazioni, da quelle più semplici a quelle più complesse e avanzate. Per rispondere a questa necessità, Android offre diversi approcci per la creazione della UI, ognuno progettato per ottimizzare specifici aspetti del processo di sviluppo.

Il primo e più tradizionale approccio è l'uso di XML (Extensible Markup Language). Questo metodo, consolidato e ampiamente supportato, consente una chiara separazione tra la definizione del layout dell'interfaccia e la logica dell'applicazione. Utilizzare XML per descrivere l'aspetto e la disposizione dei componenti dell'interfaccia offre numerosi vantaggi, inclusa la facilità di manutenzione del codice e la possibilità di riutilizzare i layout in diversi contesti.

Recentemente, Google ha introdotto Jetpack Compose, un toolkit moderno che adotta un approccio dichiarativo per la creazione della UI. In Jetpack Compose, gli sviluppatori descrivono l'interfaccia utente in termini di "cosa" dovrebbe essere visualizzato piuttosto che "come" dovrebbe essere costruito. Questo permette una maggiore concisione/sintesi e flessibilità del codice, facilitando la creazione di layout complessi e animazioni fluide, integrandosi perfettamente con il linguaggio Kotlin.

Oltre agli strumenti nativi di Android, esiste anche la possibilità di sviluppare delle web app usando, per esempio, Flutter, un framework open-source di Google per la creazione di interfacce utente multipiattaforma. Flutter utilizza il linguaggio Dart e permette di sviluppare applicazioni che funzionano su Android, iOS, web e desktop da un'unica base di codice. L'approccio dichiarativo di Flutter, simile a Jetpack Compose, insieme alla vasta gamma di widget personalizzabili, rende questo framework una scelta potente per i progetti multipiattaforma.

Esiste, inoltre, la possibilità di utilizzare le Custom View per lo sviluppo di app Android, le quali rappresentano una potente tecnica per creare interfacce utente uniche e personalizzate. Mentre le librerie UI standard come Material Design offrono componenti predefiniti e linee guida per costruire interfacce coerenti e funzionali, le Custom View consentono agli sviluppatori di andare oltre questi limiti, progettando componenti che rispondano esattamente alle esigenze specifiche dell'applicazione e del brand. Le Custom View vengono spesso utilizzate da aziende che desiderano distinguere visivamente le loro applicazioni dalla concorrenza, offrendo un'esperienza utente esclusiva. Ad esempio, le app bancarie e di servizi finanziari frequentemente adottano Custom View per creare interfacce sicure, affidabili e in linea con la loro identità di marca. Questi componenti personalizzati permettono di implementare funzionalità avanzate e design sofisticati che non sono facilmente realizzabili con i widget standard.

In un panorama tecnologico in continua evoluzione, la scelta della strategia giusta per la definizione della UI può avere un impatto significativo sul successo di un'applicazione, questo report si propone quindi di esplorare e categorizzare queste principali strategie per la definizione della UI in Android. Ogni approccio verrà analizzato in dettaglio, esaminando i linguaggi e gli strumenti necessari, i componenti disponibili, nonché i vantaggi e gli svantaggi associati. Inoltre, verrà fornito un confronto tra i diversi metodi per evidenziare i contesti di utilizzo ideali per ciascuno di essi.

Capitolo 2

XML-Based Layouts

2.1 Approccio

L'utilizzo di XML (Extensible Markup Language) per definire le interfacce utente è uno tra i metodi tradizionali e consolidati in Android. I layout, in XML, descrivono la struttura visiva dell'interfaccia utilizzando una rappresentazione dichiarativa, nella quale gli elementi sono elencati esternamente al codice. Questo approccio consente una chiara separazione tra la definizione del layout e la logica dell'applicazione, la quale fa uso del linguaggio Kotlin, facilitando la manutenzione e l'aggiornamento del codice. In questa sezione, esploreremo nel dettaglio i vari componenti utilizzabili con XML, inclusi i diversi tipi di View, ViewGroup, gli attributi associati e come gestire gli ID delle View.

2.2 Introduzione alle View e ViewGroup

In Android, un layout definisce la struttura di un'interfaccia utente nell'app, in cui, tutti gli elementi sono costruiti usando una particolare gerarchia di View e ViewGroup (un container "invisibile" che contiene altre View e/o ViewGroup, il cui scopo è quello di definire come dovrebbero essere disposte sullo schermo). Per indicare le View ci si riferisce di solito anche come *widget*, come ad esempio **Button** o **TextView**, e rappresentano i componenti visivi di base. I ViewGroup, invece, sono generalmente chiamati *layouts*, i quali possono fornire differenti strutture, come, ad esempio, **LinearLayout** e **RelativeLayout**.

2.3 Tipi di ViewGroup

I ViewGroup sono utilizzati per organizzare e gestire la disposizione delle View figlie. Di seguito sono descritti alcuni dei ViewGroup più comuni utilizzati in Android.

2.3.1 LinearLayout

LinearLayout è un ViewGroup che posiziona le sottoclassi (View posizionate al suo interno) in una singola direzione, verticale o orizzontale (in base alla

direzione specificata nell'attributo *android:orientation*). Supporta anche l'assegnazione di un peso tramite l'utilizzo dell'attributo *android:layout_weight*, il quale assegna ad una View (figlia di LinearLayout) "un'importanza" che si traduce in spazio occupato sullo schermo (più il peso è grande, maggiore è lo spazio occupato); il peso di default è zero. Per avere una distribuzione equa (ogni View figlia di LinearLayout occupa lo stesso spazio) è necessario settare *android:layout_weight* allo stesso valore per tutte le View.

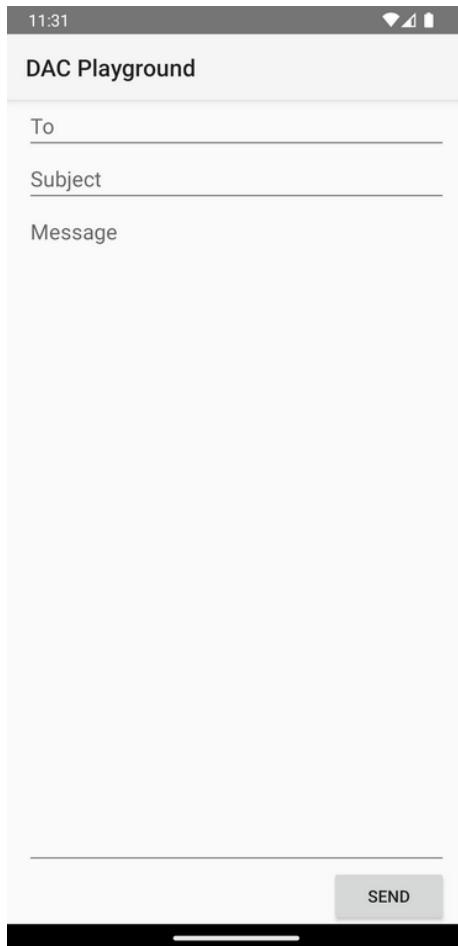


Figura 2.1: Tre EditText e un Button orientati verticalmente in un LinearLayout

2.3.2 RelativeLayout

RelativeLayout permette di eliminare i ViewGroup nidificati, mantenendo così la gerarchia del layout piatto e migliorando le performance dell'app. Per esempio se nel codice XML sono presenti più *LinearLayout* potrebbe essere conveniente passare ad un *RelativeLayout* per migliorare sia le performance che la leggibilità del codice. I figli di questo layout possono specificare la loro posizione relativamente al "parent" (*RelativeLayout*) o tra di loro attraverso l'ID. Di default tutte le View dei "figli" del *RelativeLayout* vengono posizionate nella zona in

alto a sinistra del layout; è quindi necessario definire la posizione di ogni View usando le proprietà del layout.

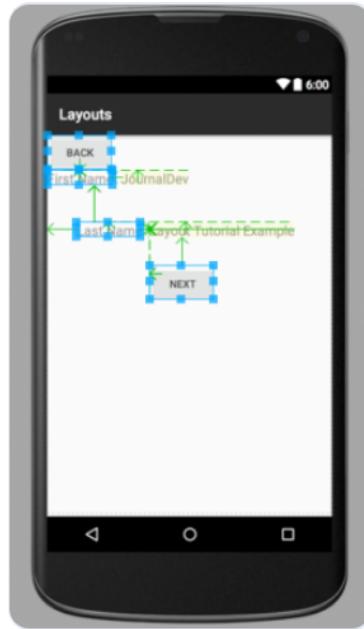


Figura 2.2: Le frecce nell'immagine mostrano come gli elementi sono posizionati l'uno rispetto all'altro e rispetto al contenitore

2.3.3 ConstraintLayout

ConstraintLayout è più avanzato e offre un controllo flessibile e potente sulla disposizione delle View permettendo di eliminare i ViewGroup nidificati e mantenere la gerarchia del layout piatto, in modo simile, ma più flessibile, rispetto al RelativeLayout. Utilizza i vincoli per definire le relazioni tra le View figlie, consentendo layout complessi con prestazioni migliori.

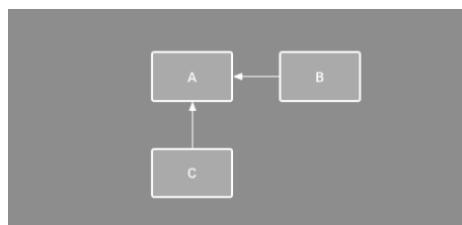


Figura 2.3: Tre elementi posizionati con dei vincoli di distanza

2.3.4 GridLayout

GridLayout è un ViewGroup che organizza le View figlie in una griglia con righe e colonne definite dall'utente. Offre un controllo fine, consentendo di specificare la larghezza e l'altezza delle celle, nonché la possibilità di unire due o più celle tra di loro.

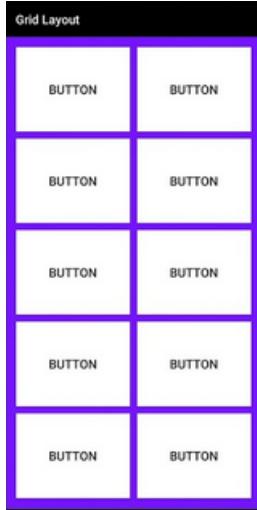


Figura 2.4: Dei Button in una GridLayout

2.3.5 RecyclerView

RecyclerView è un contenitore "scrollable" che semplifica la visualizzazione efficiente di grandi set di dati e crea dinamicamente gli elementi quando sono necessari. Come suggerisce il nome, ricicla gli elementi, cioè, quando uno scorre fuori dallo schermo, non ne viene distrutta la View, ma invece viene riutilizzata per i nuovi elementi che sono comparsi sullo schermo. Permette così di migliorare le prestazioni e la reattività dell'app e ridurre il consumo energetico.

2.3.6 CoordinatorLayout

CoordinatorLayout viene utilizzato per coordinare le interazioni tra i vari componenti all'interno di un'interfaccia utente. Questo layout è particolarmente utile per creare animazioni complesse e comportamenti di scorrimento sofisticati, come quelli visti nelle app moderne. Una delle caratteristiche principali del CoordinatorLayout è la sua capacità di gestire le interazioni tra i suoi figli attraverso i "Behaviors" (comportamenti), delle classi che definiscono come un particolare componente reagisce alle azioni degli altri componenti o dell'utente. Ad esempio, un "FloatingActionButton" potrebbe scomparire o ridimensionarsi quando si scorre su una pagina dell'app.

2.3.7 NestedScrollView

NestedScrollView è un componente che estende le funzionalità di uno ScrollView tradizionale, permettendo lo scorrimento verticale di una vista gerarchica annidata. È particolarmente utile quando si ha a che fare con contenuti che richiedono uno scorrimento all'interno di un layout complesso, come un insieme di View incorporate in altre View scorrevoli.

2.4 Tipi di View

Di seguito sono descritti alcuni dei tipi di View più comuni per costruire le interfacce utente in Android:

- **TextView:** È uno dei componenti più semplici e utilizzati, progettato per visualizzare testo sullo schermo. TextView supporta vari attributi come textSize, textColor, textStyle (che può essere normale, grassetto, corsivo), e gravity (che definisce l'allineamento del testo all'interno della View)



Figura 2.5: Un TextView con scritto "hello_world"

- **Button:** è una sottoclasse di TextView che rappresenta un pulsante cliccabile. Oltre agli attributi ereditati da TextView, supporta attributi specifici come onClick, che definisce l'azione da eseguire quando il pulsante viene premuto

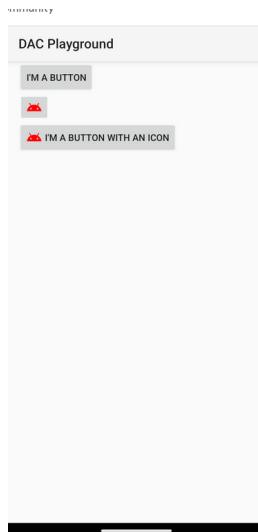


Figura 2.6: Tre Button posizionati in verticale

- ImageView: è progettato per visualizzare immagini. Supporta attributi come src (per specificare la risorsa dell'immagine), scaleType (per definire come l'immagine dovrebbe essere ridimensionata o scalata), e contentDescription (per fornire una breve descrizione dell'immagine)
- EditText: è una sottoclasse di TextView che consente l'input di testo da parte dell'utente. Supporta attributi aggiuntivi come hint (per visualizzare un suggerimento quando il campo è vuoto) e inputType (per specificare il tipo di input, come testo, numero, password, ecc.)

2.5 Attributi delle View

Gli attributi delle View definiscono le proprietà visive e comportamentali dei componenti dell'interfaccia utente. Di seguito sono descritti alcuni degli attributi più comuni utilizzati in XML per configurare le View.

Alcuni attributi più importanti:

- id: Sono identificatori univoci assegnati ai componenti dell'interfaccia utente (come le View) per poterli riferire e manipolare nel codice dell'applicazione; sono definiti utilizzando l'attributo id nel formato "@+id/-nome_id", dove "@+id" indica che un nuovo ID viene creato se non esiste già, e "nome_id" è il nome univoco dell'ID
- layout_width e layout_height: Specificano la larghezza e l'altezza della View. I valori possono essere "match_parent" (per occupare tutto lo spazio disponibile) o "wrap_content" (per adattarsi al contenuto della View)
- padding: Spazio interno tra il contenuto della View e i suoi bordi. Può essere specificato per tutti i lati o individualmente per "paddingLeft", "paddingTop", "paddingRight" e "paddingBottom"
- margin: Spazio esterno tra la View e le View circostanti. Può essere specificato per tutti i lati o individualmente per "layout_marginLeft", "layout_marginTop", "layout_marginRight" e "layout_marginBottom"
- background: Colore di sfondo o risorsa/immagine di sfondo della View
- visibility: Stato di visibilità della View. I valori possono essere "visible", "invisible" (la View non è visibile ma occupa lo stesso lo spazio) o "gone" (la View non è visibile e non occupa spazio)

Capitolo 3

Jetpack Compose

3.1 Introduzione a Jetpack Compose

Jetpack Compose rappresenta un punto di svolta nella definizione dell'interfaccia utente per le applicazioni Android. Introdotto da Google come parte della suite Jetpack, Compose adotta un approccio dichiarativo per la costruzione delle UI, sostituendo il tradizionale sistema basato su XML con una sintassi più concisa e moderna che fa utilizzo del linguaggio Kotlin.

3.2 Concetti Chiave di Jetpack Compose

Per utilizzare Jetpack Compose in modo efficace, è fondamentale comprendere alcuni concetti chiave.

3.2.1 Composable Functions

Le "composable functions" sono il cuore di Jetpack Compose. Qualsiasi elemento UI in Compose è creato tramite una funzione annotata con "@Composable". Queste funzioni possono essere combinate e riutilizzate per costruire componenti UI più complessi, inoltre l'ordine in cui sono state scritte non indica l'ordine in cui verranno eseguite.

3.2.2 State

Lo stato è un altro aspetto cruciale in Jetpack Compose. Il framework ne facilita la gestione in modo reattivo, permettendo alla UI di rispondere automaticamente ai cambiamenti di stato, garantendo così che la visualizzazione sia sempre sincronizzata con i dati. Compose offre poi anche la possibilità di memorizzare il valore di una variable locale usando la funzione "remember" che permette di monitorare e aggiornarne il valore.

3.2.3 Recomposition

La recomposition è un concetto fondamentale in Jetpack Compose, rappresenta il processo attraverso il quale il framework aggiorna l'interfaccia utente in risposta ai cambiamenti di stato. Quando il dato su cui si basa una composable

function cambia, Compose riesegue solo quella funzione per aggiornare la UI in modo reattivo.

3.2.4 Modifiers

Il parametro Modifier è uno dei concetti chiave di Jetpack Compose, essenziale per personalizzare e controllare il comportamento e l'aspetto dei composable. Funziona come una catena di decoratori che permette agli sviluppatori di applicare vari effetti e trasformazioni ai componenti dell'interfaccia utente. Ogni funzione che accetta un Modifier lo utilizza per applicare una serie di modifiche all'elemento UI a cui è collegato. Queste modifiche sono applicate in ordine, permettendo una grande flessibilità e controllo sulla configurazione della UI.

Layout e dimensionamento

Uno degli usi principali è il controllo del layout e del dimensionamento. Può essere utilizzato, infatti, per specificare le dimensioni, il padding, il margin, l'allineamento e la disposizione degli elementi all'interno di un layout. Questa capacità di influenzare il layout consente agli sviluppatori di creare interfacce utente complesse e ben strutturate.

Aspetto e Stile

Oltre al layout, Modifier permette di personalizzare l'aspetto visivo degli elementi. È possibile applicare colori di sfondo, bordi, ombreggiature e trasformazioni visive come la rotazione e la scala. Questi strumenti aiutano a definire uno stile coerente e accattivante per l'interfaccia utente dell'app.

Gestione degli Eventi

Modifier è anche utilizzato per gestire gli eventi dell'utente, come clic, scroll e drag. Permette di dichiarare in modo chiaro come i composable devono rispondere a diverse interazioni, semplificando l'implementazione di UI interattive e reattive.

Composizione di Modificatori

Una caratteristica distintiva di Modifier è la sua capacità di essere composto. Gli sviluppatori possono concatenare diversi modificatori per ottenere l'effetto desiderato. Questa composizione facilita la creazione di modifiche complesse senza sacrificare la leggibilità del codice.

3.2.5 Layouts

Jetpack Compose fornisce una vasta gamma di componenti predefiniti che possono essere utilizzati per costruire l'interfaccia utente. Di seguito sono descritti alcuni dei componenti più utilizzati:

- **Text:** è utilizzato per visualizzare stringhe di testo. Supporta attributi come fontStyle, color e fontSize

- Button: è un componente cliccabile che esegue un'azione quando viene premuto. Può essere personalizzato con attributi come onClick, shape, e border
- Image: è utilizzato per visualizzare immagini. Supporta attributi come painter, contentDescription, e alignment
- TextField: è utilizzato per l'input di testo da parte dell'utente. Può essere personalizzato con attributi come value, onValueChange, label, e placeholder
- Row e Column: vengono utilizzati rispettivamente per visualizzare i componenti al loro interno in righe o colonne

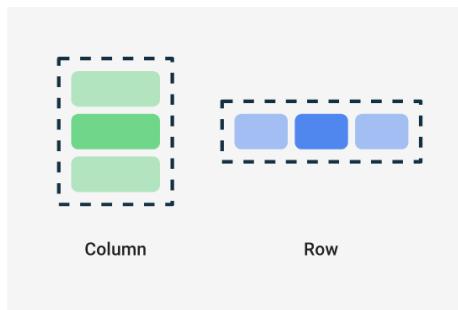


Figura 3.1: Rappresentazione di una Column e una Row

- LazyColumn e LazyRow: nel caso in cui si debbano mostrare a schermo molti elementi, usare Row o Column può risultare inefficiente poiché tutti gli elementi verranno renderizzati e disposti indipendentemente dal fatto che siano visibili o meno. Una soluzione è quella di usare una LazyColumn o LazyRow, se si vogliono gli elementi disposti rispettivamente in verticale o orizzontale, che permettono di fare il rendering solo degli elementi visibili e di renderli "scrollabili"

3.3 Best Practices per Jetpack Compose

Per sfruttare al meglio Jetpack Compose, è importante seguire alcune best practices:

- Utilizzare state hoisting: Passare lo stato come parametro alle composables per garantire una migliore separazione del codice.
- Evitare le "recomposition" costose: Utilizzare "remember" per conservare il valore tra le ricomposizioni e ridurre il carico computazionale.
- Organizzare il codice in moduli riutilizzabili: Creare composables piccole e riutilizzabili per migliorare la leggibilità e la manutenibilità del codice.
- Utilizzare i "modifiers" per personalizzare le funzioni composables e aggiungere comportamenti.

Capitolo 4

Hybrid Approaches

Nell'ecosistema mobile moderno, lo sviluppo di applicazioni per più piattaforme è una necessità crescente. Gli approcci ibridi permettono di scrivere il codice una sola volta e di distribuirlo su diverse piattaforme, tra cui Android e iOS. Questi metodi utilizzano diverse tecnologie e framework per raggiungere questo obiettivo, consentendo di ridurre i tempi di sviluppo e i costi associati alla manutenzione di codici separati. In questa sezione esploreremo i principali approcci ibridi per lo sviluppo multipiattaforma, tra cui Flutter, React Native e Unity.

4.1 Flutter

Flutter è un framework open-source sviluppato da Google per la creazione di interfacce utente nativamente compilate, utilizza il linguaggio Dart e un approccio dichiarativo simile a Jetpack Compose.

4.1.1 Caratteristiche principali

- Prestazioni elevate: Flutter è nativamente compilato, garantendo performance elevate e un'esperienza utente fluida
- Widget personalizzabili: Offre un'ampia gamma di widget che possono essere facilmente personalizzati e combinati per creare interfacce utente complesse

4.1.2 Vantaggi

- Sviluppo rapido: La possibilità di utilizzare un'unica base di codice per più piattaforme (Android, iOS, web, Windows, Mac, Linux) riduce i tempi di sviluppo
- Supporto nativo: Le applicazioni Flutter sono nativamente compilate, offrendo prestazioni comparabili a quelle delle applicazioni native

4.1.3 Svantaggi

- Dimensioni dell'applicazione: Le applicazioni Flutter tendono ad avere dimensioni maggiori rispetto alle controparti native

- Linguaggio Dart: Dart non è tra i linguaggi più studiati o conosciuti dagli sviluppatori, il che può essere una barriera di ingresso rispetto ad altri framework che si basano su linguaggi di programmazione molto più popolari

4.1.4 Casi d'uso ideali

- Applicazioni che richiedono una forte coerenza visiva tra le diverse piattaforme
- Progetti con budget limitati che richiedono una distribuzione su più piattaforme
- Applicazioni che necessitano di un'esperienza utente fluida e performante

4.2 React Native

React Native è un framework open-source sviluppato da Facebook per la creazione di applicazioni mobili utilizzando JavaScript e React. React Native consente di costruire applicazioni che funzionano su entrambe le piattaforme Android e iOS utilizzando componenti nativi.

4.2.1 Caratteristiche principali

- Componenti nativi: Le UI sono renderizzate utilizzando componenti nativi, garantendo prestazioni elevate e un'esperienza utente fluida
- Ampio ecosistema: Grazie alla sua popolarità, React Native gode di un ampio ecosistema di librerie e strumenti di terze parti

4.2.2 Vantaggi

- JavaScript e React: Utilizza JavaScript, uno dei linguaggi di programmazione più diffusi, e il framework React per costruire interfacce utente
- Community e supporto: Un'ampia comunità di sviluppatori e una vasta documentazione sono disponibili, rendendo facile trovare risorse e assistenza

4.2.3 Svantaggi

- Dipendenza da componenti nativi: Alcune funzionalità dell'ultima versione di Android potrebbero non essere ancora state inserite in React Native richiedendo quindi l'implementazione di moduli nativi, aumentando la complessità
- Prestazioni: Anche se le prestazioni sono generalmente buone, potrebbero non essere al livello delle applicazioni completamente native per scenari altamente intensivi dal punto di vista grafico.

4.2.4 Casi d'uso ideali

- Progetti che beneficiano dell'ampio ecosistema JavaScript e React
- Applicazioni che non richiedono un'ottimizzazione grafica estrema.

4.3 Unity

Unity è un potente motore di gioco multipiattaforma che può essere utilizzato anche per sviluppare applicazioni mobili. Come linguaggio di scripting utilizza C# e offre strumenti avanzati per la grafica 3D e 2D, rendendolo una scelta eccellente per applicazioni che richiedono animazioni complesse e grafica avanzata.

4.3.1 Caratteristiche principali

- Motore grafico avanzato: Supporta grafica 2D e 3D di alta qualità, con strumenti per la gestione di fisica, animazioni, e rendering
- Ampio ecosistema: Unity ha un vasto ecosistema di asset, plugin e strumenti disponibili attraverso l'Asset Store

4.3.2 Vantaggi

- Potente per giochi: Ideale per lo sviluppo di giochi e applicazioni con grafica avanzata
- Community e supporto: Un'ampia comunità di sviluppatori e una vasta documentazione sono disponibili, rendendo facile trovare risorse e assistenza
- Strumenti avanzati: Offre strumenti avanzati per il debugging, il profiling e l'ottimizzazione delle prestazioni

4.3.3 Svantaggi

- Curva di apprendimento: L'apprendimento di Unity e delle sue funzionalità avanzate può richiedere tempo, specialmente per i nuovi sviluppatori
- Peso dell'applicazione: Le applicazioni sviluppate con Unity possono essere più pesanti rispetto a quelle sviluppate con altri framework

4.3.4 Casi d'uso ideali

- Giochi 2D e 3D con grafica avanzata
- Applicazioni che richiedono animazioni complesse e simulazioni fisiche
- Progetti multipiattaforma che necessitano di strumenti avanzati di rendering e sviluppo

4.4 Conclusione

Gli approcci ibridi per la definizione della UI offrono numerosi vantaggi, tra cui la riduzione dei tempi di sviluppo e dei costi di manutenzione, consentendo di distribuire applicazioni su più piattaforme da un'unica base di codice. La scelta del framework più adatto dipende dalle specifiche esigenze del progetto, dalle competenze del team di sviluppo e dagli obiettivi a lungo termine dell'applicazione.

Flutter è ideale per applicazioni che richiedono coerenza visiva e prestazioni elevate su più piattaforme. React Native offre un'esperienza di sviluppo rapida e un'ampia community di supporto, mentre Unity è la scelta migliore per giochi e applicazioni con grafica avanzata.

Capitolo 5

Custom Views

La creazione di Custom View in Android consente ai programmati, utilizzando il linguaggio Kotlin, di definire componenti UI unici ed altamente personalizzati per le loro applicazioni. Le Custom View permettono di implementare funzionalità specifiche e stili visivi non disponibili nelle librerie UI standard, come Material Design.

5.1 Introduzione alle Custom View

Le Custom View sono componenti UI definiti dagli sviluppatori per soddisfare esigenze specifiche che non possono essere soddisfatte con i widget standard di Android.

Motivazioni per l'uso di Custom View:

- Controllo completo: Gestire ogni aspetto del comportamento e dell'aspetto dei componenti UI
- Requisiti specifici: Implementare funzionalità avanzate o interazioni complesse non supportate dai widget standard

5.2 Vantaggi e Svantaggi delle Custom View

Vantaggi:

- Flessibilità e Controllo: Le Custom View offrono il massimo controllo su ogni aspetto del comportamento e della vista, consentendo di creare interfacce utente uniche e personalizzate
- Prestazioni Ottimizzate: Le viste personalizzate possono essere ottimizzate per specifiche esigenze di prestazioni, riducendo il carico computazionale e migliorando l'efficienza

Svantaggi:

- Maggiore Complessità: La creazione di Custom View richiede una comprensione approfondita del rendering grafico e della gestione degli eventi di input in Android

- Manutenzione: Le viste personalizzate possono aumentare la complessità del codice e richiedere più tempo e risorse per la manutenzione
- Compatibilità: Garantire la compatibilità tra diverse versioni di Android e dispositivi può essere più complesso con Custom View rispetto all'uso di librerie standard

5.3 Best Practices per lo sviluppo di Custom View

Per massimizzare i benefici delle Custom View e minimizzare i potenziali problemi, è importante seguire alcune best practices:

- Documentazione: Documentare accuratamente il comportamento e l'utilizzo delle Custom View per facilitare la manutenzione e la comprensione del codice
- Testing: Eseguire test approfonditi su diverse versioni di Android e dispositivi per garantire la compatibilità e le prestazioni
- Modularità: Organizzare il codice in moduli riutilizzabili per migliorare la manutenibilità e facilitare l'aggiornamento delle viste personalizzate
- Ottimizzazione: Ottimizzare le operazioni di rendering e gestione degli eventi per ridurre il carico computazionale e migliorare le prestazioni

Capitolo 6

Conclusione

La progettazione e lo sviluppo di interfacce utente (UI) per applicazioni Android rappresenta un aspetto molto importante per garantire un'esperienza utente ottimale e distintiva. In questo report, abbiamo esplorato diverse strategie per la definizione della UI in Android, comprendendo i metodi tradizionali basati su XML, i moderni approcci dichiarativi con Jetpack Compose, le soluzioni ibride per lo sviluppo multipiattaforma come Flutter, e l'uso di Custom View per creare interfacce altamente personalizzate. Ciascuna di queste metodologie presenta vantaggi, svantaggi e scenari d'uso ideali che gli sviluppatori devono considerare attentamente per selezionare l'approccio più adatto alle proprie esigenze.

6.1 Considerazioni Finali

Vediamo ora una breve comparazione tra le varie tecnologie:

- XML rimane una scelta valida per progetti tradizionali o legacy che richiedono una chiara separazione tra layout e logica
- Jetpack Compose rappresenta il futuro della definizione delle UI in Android, offrendo maggiore flessibilità e una sintassi moderna che semplifica la gestione dello stato e delle animazioni
- Gli approcci ibridi come Flutter, React Native e Unity offrono soluzioni efficienti per lo sviluppo multipiattaforma, riducendo i tempi di sviluppo e i costi di manutenzione
- Le Custom View permettono di creare componenti UI unici e altamente personalizzati, ideali per applicazioni di alto profilo che richiedono un'esperienza utente distintiva

In conclusione, la scelta della strategia più adatta per la definizione della UI deve considerare le specifiche esigenze del progetto, le competenze del team e gli obiettivi a lungo termine dell'applicazione. Con una comprensione approfondita delle diverse opzioni disponibili, gli sviluppatori possono creare interfacce utente di alta qualità che offra un'esperienza utente eccezionale e soddisfi le esigenze del mercato moderno.

Capitolo 7

Economy tracker

Avendo scelto il progetto "Building UIs" abbiamo sviluppato due app, molto simili a livello di interfaccia grafica, la quale è stata implementata usando XML per il primo approccio e Jetpack Compose per il secondo. L'applicazione "Economy tracker" permette, quindi, di tracciare le uscite e le entrate di denaro, dando la possibilità all'utente di categorizzare, modificare ed eliminare le entrate e le uscite.

7.1 Struttura dell'applicazione

7.1.1 La pagina Home



Figura 7.1: Home - Jetpack Compose



Figura 7.2: Home - XML

Entrando nell'app si viene accolti dalla schermata Home nella quale è presente una lista verticale scrollabile di "SummaryCard", dei componenti custom di cui ne è presente una copia per ogni mese in cui sono stati inseriti dei movimenti; al loro interno contengono una sintesi dell'andamento mensile comprendente di saldo complessivo, guadagni e spese.

Impostazioni

Per accedere alle impostazioni è necessario cliccare la rotellina in alto a destra, dopodiché si aprirà un "Dialog" nel quale è possibile scegliere in che valuta visualizzare i movimenti e il tema, ovvero se passare dalla modalità chiara alla scura o viceversa. Entrambi questi cambiamenti causano il reload dell'activity, ovvero per osservare i cambiamenti non sarà necessario uscire e poi rientrare nell'app, ma basterà premere il tasto conferma.

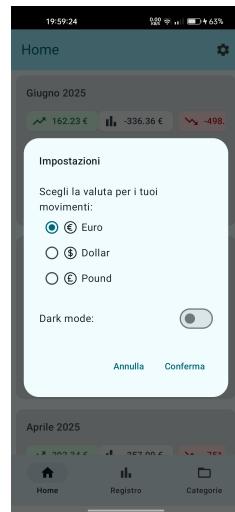


Figura 7.3: Pagina delle impostazioni - Jetpack Compose



Figura 7.4: Pagina delle impostazioni - XML

7.1.2 La pagina Registro

Qui è possibile vedere la "SummaryCard" del mese ed anno corrente ed una lista verticale scorrevole nella quale sono visibili i movimenti con tutte le loro caratteristiche: ammontare, data, ora e categoria. Selezionando "Entrate", "Uscite" o "Tutto" si possono visualizzare rispettivamente solo le entrate, solo le uscite, oppure entrambe grazie ad un selezionatore.

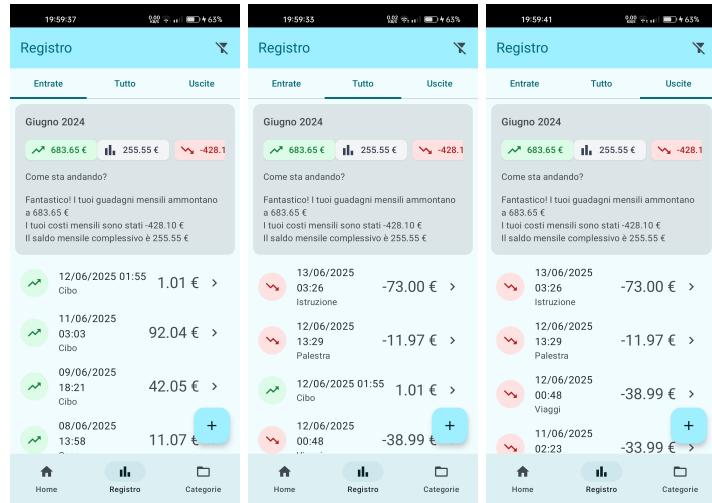


Figura 7.5: Registro - Jetpack Compose

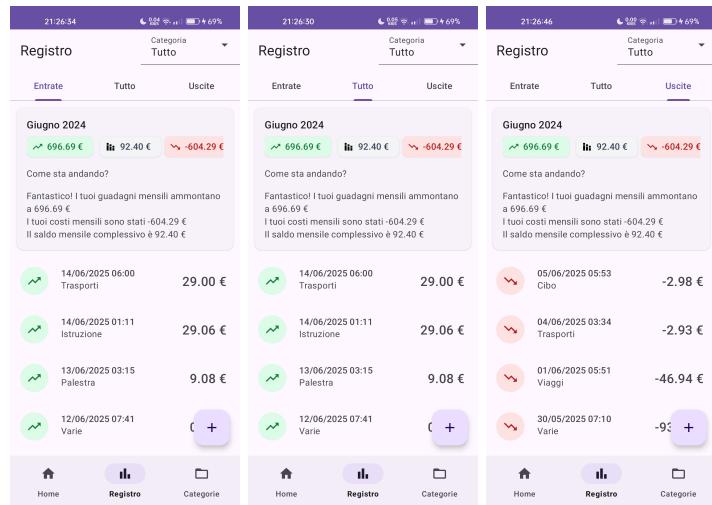


Figura 7.6: Registro - XML

Cambio della categoria

In alto a destra è presente un selezionatore di categorie il quale, nel momento in cui viene premuto, apre una piccola finestra dove è possibile selezionare la

categoria di movimento che si vuole visualizzare. Una volta selezionata la vista dei movimenti verrà aggiornata istantaneamente.

Aggiunta di un movimento

Per aggiungere un movimento è necessario cliccare sul bottone presente in basso a destra avente un "+" al centro. Una volta premuto si aprirà un "Dialog" (finestra) che, senza riempire l'intero schermo, permetterà di inserire la quantità di denaro entrata/uscita, la categoria di cui fa parte (cibo, viaggi, ...) e la data in cui è avvenuto il movimento. Una volta completati i campi si può annullare o salvare e, se l'operazione è confermata il movimento verrà aggiunto nel database; sarà quindi possibile vedere una nuova aggiunta nella lista dei movimenti.

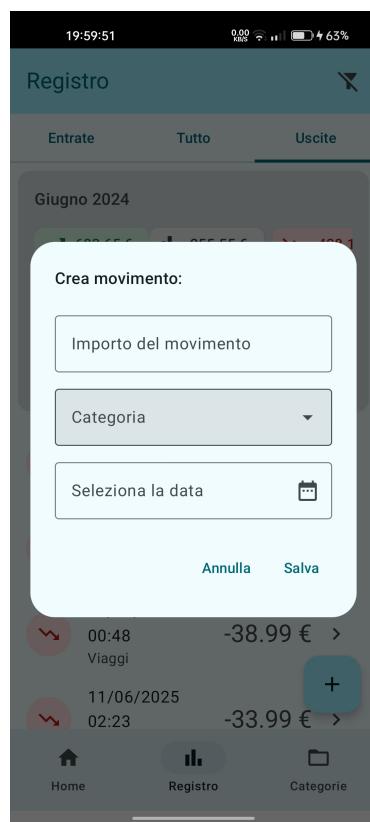


Figura 7.7: Crea un nuovo movimento - Jetpack Compose

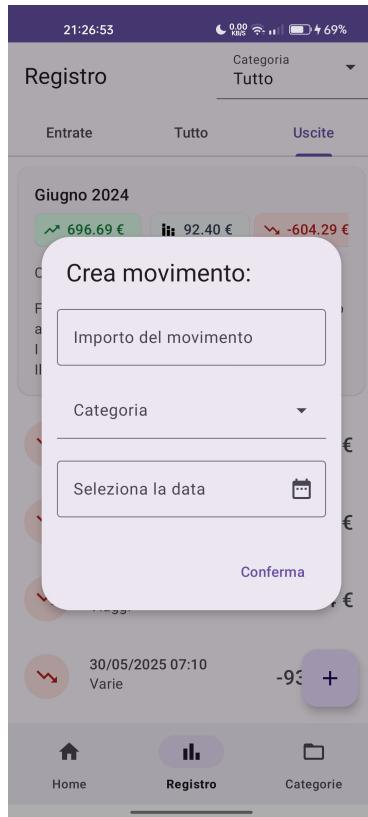


Figura 7.8: Crea un nuovo movimento - XML

Modifica ed eliminazione di un movimento

Mentre su XML è necessario fare un "long click" sul movimento, in Jetpack Compose bisogna cliccare sulla freccia presente alla destra di un movimento per aprire un pannello nella parte inferiore dello schermo, il quale offre la possibilità di eliminare o modificare il movimento selezionato. Se si sceglie di cambiarlo si aprirà un altro pannello che permetterà di modificare la data e/o la categoria e/o il quantitativo di denaro. Confermando questa azione verrà aggiornato il database e sarà possibile vedere il cambiamento nella lista dei movimenti. Se invece si vuole eliminare verrà chiesto di confermare l'operazione, dopodiché il movimento selezionato verrà rimosso dal database.



Figura 7.9: Pannello con opzioni di modifica ed eliminazione di un movimento
- Jetpack Compose



Figura 7.10: Pannello con opzioni di modifica ed eliminazione di un movimento
- XML

7.1.3 La pagina Categorie

In questa pagina è possibile vedere una lista di tutte le diverse categorie che sono state create in passato ed usate per i movimenti.

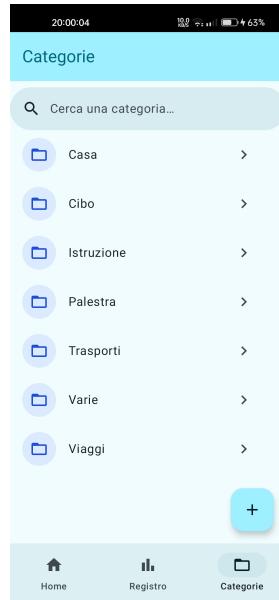


Figura 7.11: Vista della pagina "Categorie" - Jetpack Compose

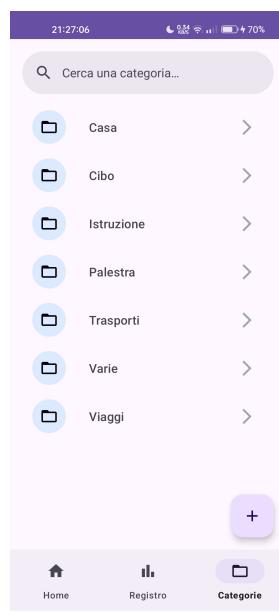


Figura 7.12: Vista della pagina "Categorie" - XML

Ricerca di una categoria

È possibile cercare una categoria in particolare tra la lista di tutte quelle create semplicemente cliccando sulla zona di ricerca.

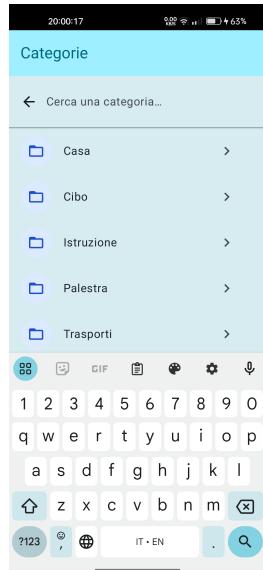


Figura 7.13: Ricerca di una categoria - Jetpack Compose

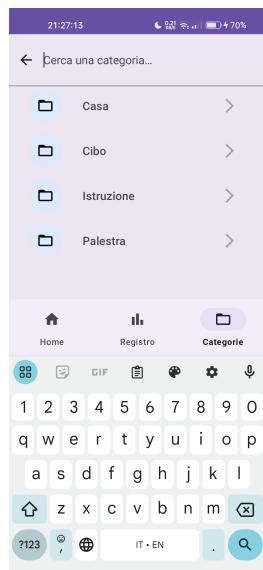


Figura 7.14: Ricerca di una categoria - XML

Aggiunta di una nuova categoria

Nel caso in cui ci fosse bisogno di una nuova categoria per un movimento è necessario, nella pagina "Categorie", cliccare sul bottone posizionato in basso a destra avente al centro un "+". Questo farà comparire a schermo un Dialog nel quale potrà essere inserito il nome della nuova categoria, la quale non può essere uguale ad una già esistente.



Figura 7.15: Aggiungere una nuova categoria - Jetpack Compose



Figura 7.16: Aggiungere una nuova categoria - XML

Modifica ed eliminazione di una categoria

È possibile modificare il nome o eliminare una categoria: per farlo è necessario cliccare sulla freccia posizionata alla destra della relativa categoria; si aprirà così un pannello nella parte inferiore dello schermo che permetterà di eseguire l'operazione desiderata. L'unica accortezza è che l'eliminazione di una categoria causa in automatico l'eliminazione dei movimenti associati.

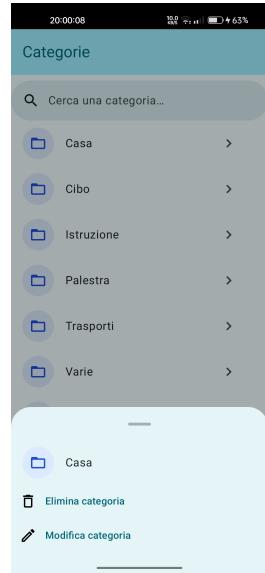


Figura 7.17: Pannello con opzioni di modifica ed eliminazione di una categoria
- Jetpack Compose

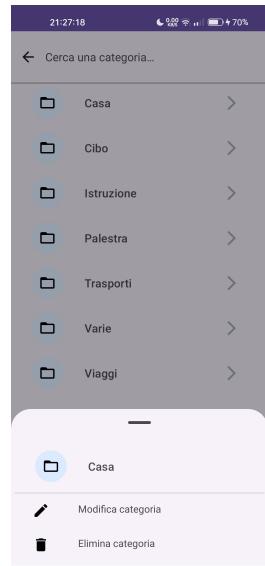


Figura 7.18: Pannello con opzioni di modifica ed eliminazione di una categoria
- XML

7.2 Differenze tra XML e Jetpack Compose

Una delle differenze presenti, tra le due app, si può notare nella visualizzazione delle impostazioni. Infatti su XML è stato usato il componente "MaterialButtonToggleGroup", mentre nella versione con Jetpack Compose, non essendo presente una alternativa che abbia le stesse caratteristiche (in material design 3), è stato usato un "RadioGroup". Alcune differenze minori derivano dagli approcci utilizzati: ad esempio, la visualizzazione del pannello per modificare o eliminare movimenti viene gestita in modo diverso. Questo è dovuto al fatto che i diversi approcci offrono metodi differenti, con varie difficoltà di implementazione, per ottenere lo stesso risultato.

Capitolo 8

Bibliografia

8.1 XML

- View/ViewGroup
- LinearLayout
- Linear e Relative Layout
- RelativeLayout
- ConstraintLayout
- GridLayout
- RecyclerView
- RecyclerView
- CoordinatorLayout
- CoordinatorLayout
- NestedScrollView
- NestedScrollView
- TextView
- Button
- ImageView
- EditText
- Attributi delle View

8.2 Jetpack Compose

- Concepts in Compose
- Layouts
- Concepts in Compose
- Modifiers
- Modifiers
- Modifiers
- Text
- Button
- Image
- TextField
- Row e Column
- LazyRow e LazyColumn
- Thinking in Compose

8.3 Hybrid

- Flutter vs React Native
- React Native
- Flutter
- Unity
- Unity
- Flutter
- Flutter
- React Native
- Unity
- Unity
- Languages popularity

8.4 Custom View

- Custom view
- Custom view
- Custom view
- Custom view
- Custom view