

App Widgets

Balzan Alessandro
2000114

Di Bella Riccardo
2000163

Zhu Yihui
2009684

Indice

Introduzione	3
Caratteristiche	3
Limiti	3
Tipi di widget	4
Design guidelines	6
Contenuto	6
Navigazione	6
Ridimensionamento	7
Suggerimenti sul layout	8
Configurazioni utente	9
Novità di Android 12	10
Architettura	10
Struttura di base	10
Widget Provider	11
Collection Widgets	12
AppWidgetManager	13
Limitazioni di AppWidgetProvider	14
Integrazione con Google Assistant	14
Invocazione esplicita	15
Invocazione tramite Built-in Intent	15
Requisiti minimi	16
Layout e stile	17
Layout responsivi	17
Colori dinamici	18
Preview nel widget picker	19
Pomodoro Timer	21
Lista sessioni	22

Dettagli sessione	22
Timer	23
Statistiche	24
Impostazioni	24
Widget statistiche	25
Widget sessioni	26
Widget di controllo	27
Colori dinamici nei widget	28
Integrazione con Google Assistant	29
Bibliografia	30

Introduzione

Caratteristiche

I widget in Android sono delle finestre o viste di app che si possono disporre sulla griglia della home page o su una qualsiasi altra app che implementa un app Widget Host come i launcher alternativi e Google Assistant.

Generalmente più grandi di un'icona, occupano più celle della griglia della home screen e servono a mostrare informazioni essenziali senza la necessità che l'utente apra l'applicazione a cui appartengono.

I widget sono interattivi: possono essere spostati e ridimensionati, variando la quantità di informazione mostrata ed il layout. Spesso hanno pulsanti di controllo e possono essere premuti aprendo l'applicazione a schermo intero di conseguenza.

Limiti

Siccome i widget si trovano nella home page non possono rispondere alla gestione di scorrimento orizzontale, perché questa serve alla navigazione tra schermate. Gli unici gesti utilizzabili per interagire con un widget sono quindi il tocco e lo scorrimento verticale.

Dato che i widget sono contenuti in altre applicazioni (launcher), essi possono essere composti solo dal sottoinsieme di layout e componenti supportati da RemoteViews, la View che può essere mostrata in altri processi, e usare i suoi metodi per modificarne il contenuto. Non è possibile utilizzare sottoclassi di questi componenti per la realizzazione di widget.

In particolare, i layout supportati sono:

- AdapterViewFlipper
- FrameLayout
- GridLayout
- LinearLayout
- ListView
- RelativeLayout
- StackView
- ViewFlipper
- AnalogClock

I componenti supportati sono:

- Button
- Chronometer
- ImageButton
- ImageView
- ProgressBar
- TextClock
- TextView

Ad esempio, per mostrare un grafico su un widget bisogna prima ottenere l'immagine bitmap dalla custom view su cui normalmente si disegna, poi passare questa bitmap alla componente ImageView del widget.

Per conservare batteria, la frequenza di aggiornamento dovrebbe essere il più bassa possibile. I widget si possono aggiornare automaticamente ad un intervallo di tempo specificabile nei metadati, ma questo deve essere superiore o uguale a 30 minuti, con la condizione che istanze diverse dello stesso widget si aggiorneranno contemporaneamente. Per aggiornamenti più frequenti si devono usare altri componenti per mandare un intent broadcast con l'azione AppWidgetManager.ACTION_APPWIDGET_UPDATE:

- WorkManager con intervalli di almeno 15 minuti. Questo approccio è necessario in caso il widget sia l'unica componente dell'app
- Foreground Service
- Activity
- Alarm Manager, con tempi di aggiornamenti inesatti da API level 19 per motivi di risparmio batteria

Per impostare alcuni parametri degli elementi contenuti nei layout a cui la RemoteView è associata ci sono due tipi di metodi di accesso, a seconda della proprietà da modificare: per attributi comuni si possono utilizzare metodi specifici per quel campo di quel tipo di elemento, come setTextviewText per impostare il campo text di una TextView, oppure metodi generici come setInt e setBoolean. I

metodi specifici accettano come parametri l'id dell'elemento nel layout e il valore da impostare, mentre i metodi generici ricevono l'id dell'elemento, il nome del metodo da invocare e il valore da passare come parametro a quel metodo.

Il meccanismo dei “metodi generici” per le impostazioni delle RemoteViews impone una limitazione che non è presente quando si possono invocare direttamente i metodi delle View: costringe ad usare solo metodi che accettano i tipi di dati previsti a priori. Un esempio di situazione in cui questa limitazione può essere un problema è quando si vuole applicare programmaticamente (all'interno di onUpdate) un filtro di colore ad un drawable (metodo setColorFilter). Se si vuole specificare un tipo di filtro (ad esempio moltiplicativo o additivo) normalmente si possono usare un metodo che accetta due interi (un colore e una modalità) o un metodo che accetta un oggetto di tipo ColorFilter. Con RemoteViews, però, l'unica opzione a disposizione è passare il colore come intero al metodo setInt, senza la possibilità di passare la modalità di applicazione del filtro ed essendo costretto ad usare quella di default.

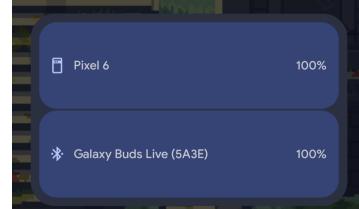
Tipi di widget

La più recente linea guida sui widget elenca 4 categorie principali, già previste nelle linee guida di Material Design 1:

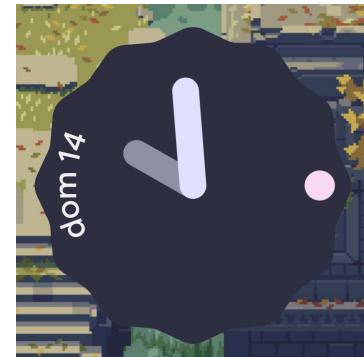
- Information widgets: mostrano informazioni importanti che l'utente vuole vedere a colpo d'occhio, perché queste sono di frequente consultazione o per evitare di cercare la applicazione ed interagire con essa ogni volta che si cerca quell'informazione. Alcuni esempi possono essere l'ora, il meteo, la batteria e il tempo di utilizzo dello schermo. Premendo questi widget viene aperta l'app associata che mostrerà una vista più dettagliata delle informazioni



Meteo



Batteria



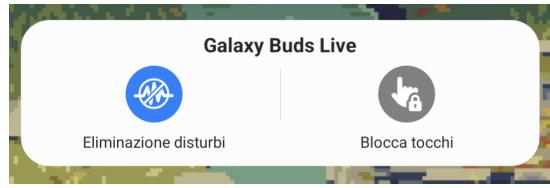
Orologio

- Collection widgets: mostrano un insieme di foto, una lista di notizie, un elenco di chat o email etc. La lista degli elementi può essere scrollata verticalmente, un singolo elemento può essere premuto per aprire una vista dettaglio delle sue informazioni nell'app a schermo intero per avere più controlli, si può interagire con gli elementi attraverso i compound buttons (Android 12 e successivi), per esempio le checkbox e gli switch. I Collection Widgets si concentrano su due funzionalità chiave: navigare all'interno della collezione ed aprire una finestra di dettaglio riguardo ad un elemento specifico.



Google keep. Una lista di to-do interagibili tramite checkbox

- Control widgets: rendono disponibili funzioni di uso frequente che l'utente può attivare direttamente dalla schermata home, come se il widget fosse un telecomando per la app

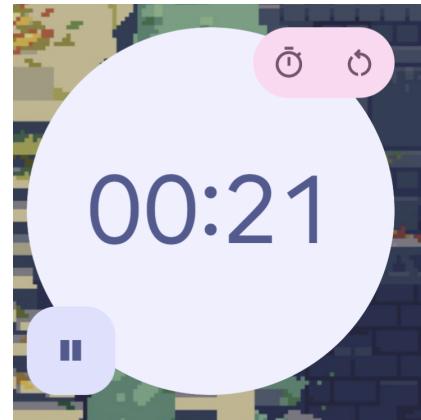


Galaxy wearable. Controlli per le impostazioni delle cuffiette

- Hybrid widgets: combinazione dei tipi di widget precedenti



YouTube music. Controlli e informazioni sul brano corrente



Cronometro. Controlli sul tempo e visualizzazione del tempo trascorso

Design guidelines

Contenuto

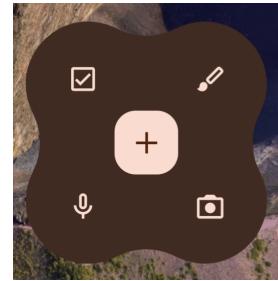
I widget dovrebbero mostrare informazioni succinte ma nuove e interessanti che sono presenti all'interno dell'app, in modo da sollecitare l'interesse dell'utente. Il widget agisce quindi da preview per un contenuto più dettagliato.

Navigazione

Il widget potrebbe contenere pulsanti che ridirezionano a parti di uso frequente della app, per avere una navigazione più rapida all'interno della stessa grazie a queste shortcut. Inoltre, le funzioni sono rese raggiungibili direttamente dalla home page.

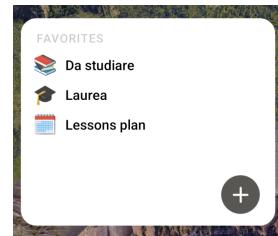
Esempi di funzioni per cui fornire navigation link:

- Generative functions: creare un nuovo documento, scrivere una nuova email o un messaggio, registrare una nuova nota vocale, scattare un selfie



Google keep. Shortcut per creazione di note rapide

- Aprire la app in pagine specifiche: fornisce un modo comodo per raggiungere pagine altrimenti raggiungibili solo dopo diversi livelli di navigazione, oppure raggiungere i preferiti senza dover trovare il percorso



Notion. Shortcut per le aprire le pagine dei preferiti

Ridimensionamento

Lo sviluppatore del widget può decidere se consentire all'utente di ridimensionarlo o fornirne una versione con grandezza fissa. Il ridimensionamento permette all'utente di regolare la superficie occupata dalla finestra sulla home screen per comporre a suo piacimento l'interfaccia e di decidere quanta informazione vedere a colpo d'occhio senza dover scrollare.

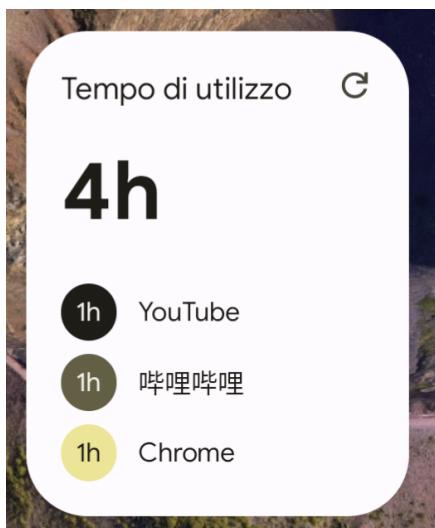
I widget che non prevedono lo scorrimento dovrebbero progettare il layout e il contenuto in modo che si adatti dinamicamente alle variazioni di dimensione e forma.



Benessere digitale 2x1



Meteo 2x2



Benessere digitale 2x2



Meteo 4x2



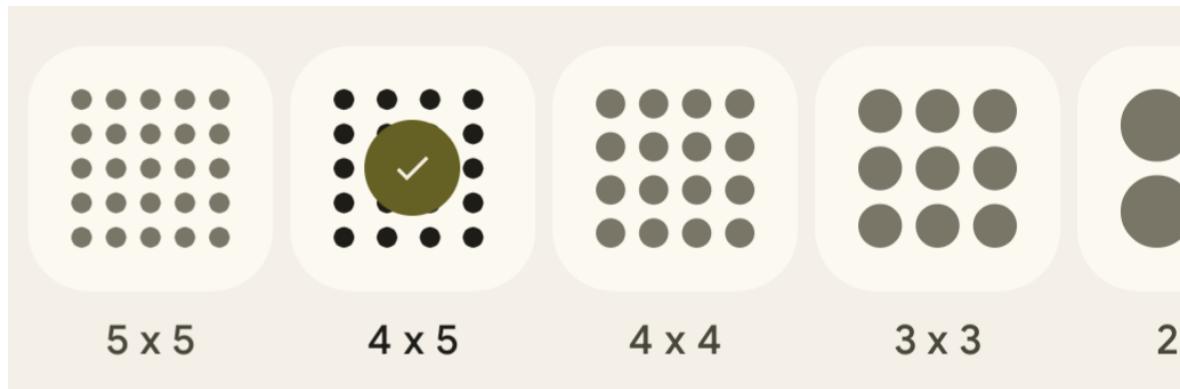
Benessere digitale 3x2



Meteo 4x3

Suggerimenti sul layout

I dispositivi Android di diversi produttori e con diverse versioni dei launcher presentano home page con griglie di icone diverse, anche perché spesso l'utente stesso può variarne le impostazioni. Il numero di celle, le loro dimensioni e i margini che le separano sono variabili tra dispositivi e launcher diversi.

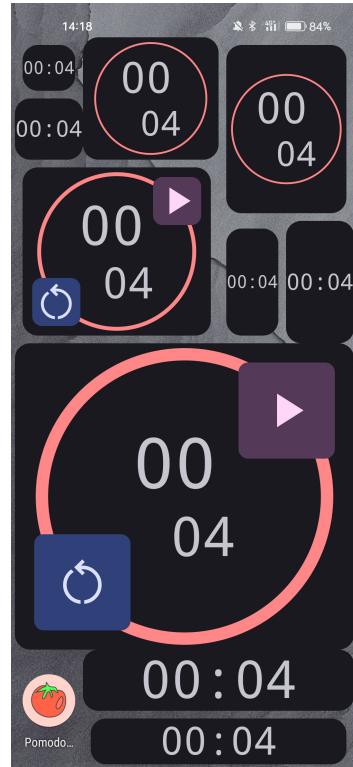


Per adattarsi alle configurazioni di una vasta utenza, non sarebbe sufficiente limitarsi a progettare il widget basandosi sulla dimensione della griglia di un solo dispositivo, magari quello con cui si sta sviluppando.

Per avere risultati consistenti è consigliato progettare i ridimensionamenti sulla base di “size buckets”, intervalli di dimensioni, invece che su numero di celle da occupare, in quanto il widget deve essere flessibile nell'adattarsi a spazi leggermente diversi da quelli anticipati.

Quando il widget viene ridimensionato, il sistema fornisce uno spazio espresso in dp in cui esso può disegnarsi.

Va comunque considerato che launcher di terze parti possono ignorare completamente le impostazioni sui limiti alle dimensioni del widget e fornire invece un controllo sul ridimensionamento molto più alto di quello previsto dallo sviluppatore. Un esempio di launcher con questo comportamento è Nova, che permette di avere qualche decina di livelli di dimensionamento per entrambe le dimensioni dello schermo a prescindere dalle impostazioni del widget.



Granularità nel controllo delle dimensioni con Nova Launcher

Configurazioni utente

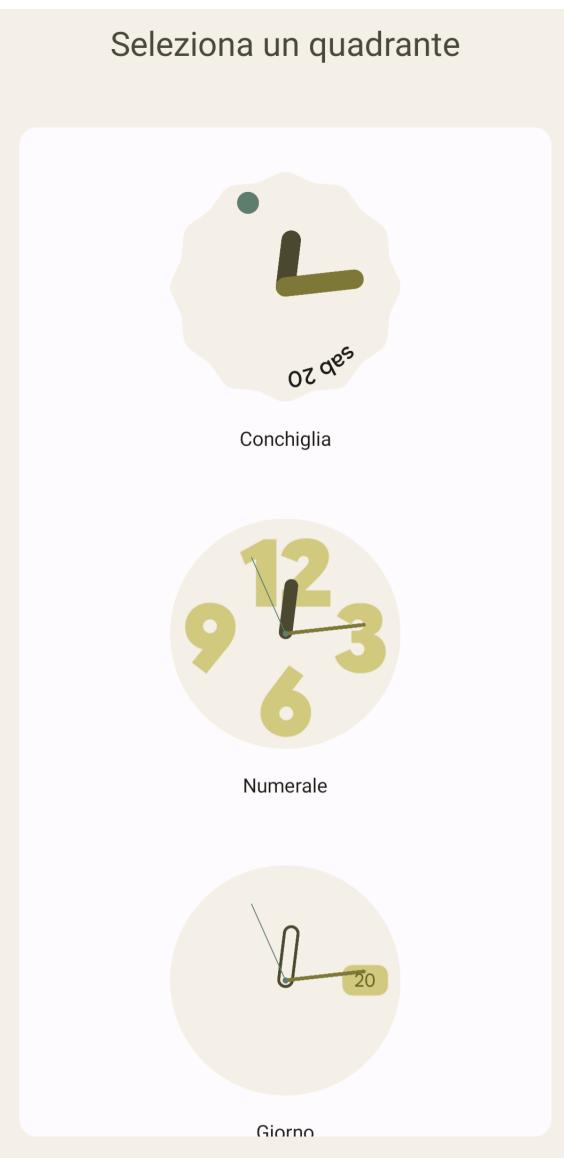
Spesso il widget richiede una iniziale configurazione utente prima di essere utilizzabile.

Per esempio, un widget che fornisce scorciatoie per telefonare/inviare messaggi rapidamente a un contatto predefinito prima di essere utilizzabile richiede che si selezioni la persona dalla lista dei contatti, un widget per aggiungere delle note alla homepage chiederà quale delle note si vuole mostrare e un widget per l'orologio può chiedere il tipo di quadrante.

Le scelte di configurazione appaiono quando l'utente rilascia il widget che si vuole aggiungere sulla schermata home, tramite un'Activity di configurazione specificata nelle impostazioni del widget.

Ci sono alcuni requisiti da rispettare per consentire una corretta configurazione del widget:

- l'Activity deve essere inserita nel manifest con l'intent-filter APPWIDGET_CONFIGURE
- il fully qualified name dell'Activity deve essere inserito nell'attributo android:configure nel file di informazioni sul widget
- fino al momento in cui avviene la scelta delle impostazioni, l'activity deve trovarsi in uno stato tale per cui se viene chiusa viene segnalato che le impostazioni non sono state selezionate (viene impostato un risultato di tipo RESULT_CANCELED)



L'activity legge l'id del widget da configurare in un extra e, quando le impostazioni da applicare sono decise, ci sono due modi per applicare queste configurazioni al widget:

- costruire l'oggetto RemoteViews direttamente all'interno della Activity e aggiornare manualmente il widget con l'id specificato tramite AppWidgetManager
- inviare il broadcast di tipo ACTION_APPWIDGET_UPDATE dall'activity (questo broadcast non viene inviato automaticamente nel caso di widget configurabili quando vengono aggiunti alla schermata home)

In entrambi i casi può essere utile salvare per ogni widget aggiunto in questo modo le impostazioni appena configurate (ad esempio nelle Shared Preferences) per poterle recuperare nelle successive chiamate a onUpdate e costruire il widget di conseguenza.

Entrambi gli approcci descritti sopra hanno sia vantaggi che svantaggi: il primo costringe a duplicare il codice di creazione delle RemoteViews o almeno a racchiuderlo in un metodo del companion object per poterlo usare come procedura di costruzione dall'activity di configurazione, il secondo impone di salvare le shared preferences in modo sincrono (usando commit invece di apply) per avere quei dati disponibili immediatamente all'interno di onUpdate.

Novità di Android 12

- Bordi arrotondati. I launcher arrotondano automaticamente gli angoli del widget spuntandone lo sfondo. Bisogna assicurarsi di non mostrare contenuti proprio sugli angoli, e non usare sfondi del widget difficili da riconoscere dallo sfondo di sistema. Si possono definire i raggi di curvatura degli angoli, sia dello sfondo che delle viste all'interno.
 - Tema di sistema. I widget possono colorare i componenti con i colori del tema del sistema, i "dynamic colors". Ciò permette al widget di integrarsi ed avere un aspetto più consistente con gli altri elementi della UI, per esempio quanto varia il tema da chiaro a scuro o viene cambiato completamente lo schema colori, per esempio in seguito al cambiamento di immagine di sfondo.
 - Configurazione utente. Si può permettere all'utente di saltare la fase iniziale di configurazione, fornendo una configurazione di default, e cambiare le impostazioni di widget già presenti sulla home screen.
 - Supporto a CompoundButton. Sono stati aggiunti i seguenti componenti a quelli già supportati da RemoteView:
 - CheckBox
 - RadioButton
 - RadioGroup
 - Switch
- I widget rimangono comunque stateless, il salvataggio degli stati e delle variazioni è a cura della app.
- Layout più flessibili
 - si può esprimere la dimensione iniziale del widget in termini di numero di celle orizzontali e verticali che si vuole occupare, al posto di indicare le dimensioni minime in dp. Si possono specificare le dimensioni massime consentite
 - si possono fornire layout responsivi, ovvero un insieme di layout ciascuno valido per un intervallo di dimensione e che verrà selezionato dinamicamente a runtime dopo il resizing.
 - si possono definire layout exact-size, nel caso in cui l'interfaccia varia drasticamente o i layout responsivi non sono sufficienti/pratici, per esempio versioni portrait e landscape e widget per i foldables
 - nuove formule e istruzioni per calcolare le dimensioni del widget
 - Esperienza migliorata nel widget picker. La preview del widget mostrato nel widget picker può avere dimensione scalabile, mentre in precedenza era un'immagine statica, fornerendo un layout di preview. Per supportare dispositivi con versione di Android precedenti, fornire tutte e due le risorse.
 - Animazioni più fluide. Introdotte animazioni più fluide quando viene aperta la app dal widget, se si fornisce uno sfondo al layout top level.
 - RemoteCollectionItems. Quando si deve passare una collection di data per popolare un collection view del widget, si può invocare il metodo setRemoteAdapter invece di implementare RemoveViewsFactory.
 - Modifiche a runtime di RemoteView. Nuovi metodi per modificare attributi di una RemoteView a runtime.

Architettura

Struttura di base

Gli elementi minimi necessari per aggiungere un widget ad un'applicazione sono:

- Un file xml con le informazioni sul widget(che chiameremo AppWidgetProviderInfo)
 - Regole di dimensionamento, categoria del widget (schermata home/schermata di blocco), file di layout, intervallo di aggiornamento, immagine e layout di anteprima, descrizione, regole di configurazione.
- Uno o più file di layout per il widget
 - Gli elementi e i layout consentiti si limitano a quelli supportati da RemoteViews
- Un file per l'AppWidgetProvider (approfondito di seguito)

È inoltre necessario aggiungere il widget provider come BroadcastReceiver nel manifest, specificando l'intent-filter (APPWIDGET_UPDATE) e un meta-data che contiene un riferimento al file AppWidgetProviderInfo.

Se l'intervallo di aggiornamento è 0 l'unica volta che onUpdate viene chiamato automaticamente per un widget è quando viene aggiunto alla schermata home: da quel momento in poi tutti gli aggiornamenti dovranno essere effettuati manualmente.

```
<appwidget-provider xmlns:android=
    " http://schemas.android.com/apk/res/android"
    android:minHeight="50dp"
    android:minWidth="60dp"
    android:minResizeHeight="50dp"
    android:minResizeWidth="60dp"
    android:description="@string/control_widget_description"
    android:targetCellHeight="2"
    android:targetCellWidth="2"
    android:previewLayout="@layout/control_widget_preview"
    android:previewImage="@drawable/control_widget_preview_img"
    android:widgetCategory="home_screen"
    android:resizeMode="horizontal|vertical"
    android:updatePeriodMillis="0"
    android:configure=
        "it.unipd.dei.esp2023.control_widget.ControlWidgetConfiguration" />
```

Un esempio di AppWidgetProviderInfo

minHeight e minWidth specificano le dimensioni iniziali nei dispositivi che non supportano i nuovi attributi targetCellHeight e targetCellWidth. previewImage specifica una risorsa drawable per l'anteprima, mentre previewLayout(Android 12 e successivi), riferisce un file di layout xml, che può quindi per esempio adattarsi al tema di sistema, oltre ad avere dimensioni realistiche.

```
<receiver
    android:name=".control_widget.ControlWidgetProvider"
    android:exported="true"
    android:label="@string/control_widget_label">
    <intent-filter>
        <action android:name=
            "android.appwidget.action.APPWIDGET_UPDATE" />
        <action android:name=
            "com.google.assistant.appactions.widgets.PIN_APP_WIDGET" />
    </intent-filter>
    <meta-data
        android:name="android.appwidget.provider"
        android:resource="@xml/control_widget_info" />
</receiver>
```

Un esempio di come dichiarare il widget nel manifest

Widget Provider

Per la realizzazione di un widget è necessario implementare una sottoclasse di AppWidgetProvider, che a sua volta eredita da BroadcastReceiver.

Il metodo principale della classe widget provider da realizzare è onUpdate: questo metodo viene invocato dall'implementazione di default di onReceive ogni volta che l'aspetto grafico del widget va aggiornato (sia nell'istante in cui il widget viene aggiunto alla homescreen che ogni volta che trascorre l'intervallo di aggiornamento specificato nel file di info).

Altri metodi di cui è possibile fare l'override per reagire ad eventi a cui i widget sono soggetti sono:

- `onAppWidgetOptionsChanged`: chiamato sia quando il widget viene aggiunto alla schermata sia ogni volta che viene ridimensionato
- `onDelete`: chiamato ogni volta in cui un widget associato viene eliminato
- `onEnabled`: chiamato la prima volta in cui un widget viene aggiunto a una superficie
- `onDisabled`: chiamato quando l'ultimo widget viene eliminato
- `onRestored`: legato al recupero delle istanze del widget da un backup, segnala anche se il recupero dell'istanza ha avuto successo o meno

Tutti i metodi precedenti sono invocati correttamente dall'implementazione di default di AppWidgetProvider del metodo `onReceive`, rispettivamente in risposta ai seguenti tipi di broadcast (definiti in `AppWidgetManager`) ricevuti:

- `ACTION_APPWIDGET_UPDATE` → `onUpdate`
- `ACTION_APPWIDGET_OPTIONS_CHANGED` → `onAppWidgetOptionsChanged`
- `ACTION_APPWIDGET_DELETED` → `onDelete`
- `ACTION_APPWIDGET_ENABLED` → `onEnabled`
- `ACTION_APPWIDGET_DISABLED` → `onDisabled`
- `ACTION_APPWIDGET_RESTORED` → `onRestored` → `onUpdate`

Se si vogliono gestire anche altri tipi di broadcast, come quelli inviati con la pressione su elementi interattivi del layout o altri broadcast di sistema, è possibile fare override dell'implementazione di default di `onReceive` e al suo interno gestire questi broadcast aggiuntivi, mantenendo come fallback l'implementazione di default di `onReceive` per tutti gli altri casi.

```
override fun onReceive(context: Context?, intent: Intent) {
    if (intent.action == Intent.ACTION_DATE_CHANGED ||
        intent.action == Intent.ACTION_TIME_CHANGED) {
        /*
        aggiornare il widget anche se la azione dell'intent non e'
        ACTION_APPWIDGET_UPDATE, ma altri eventi a cui si e' registrati,
        nell'esempio il cambiamento della data dovuto allo scoccare
        della mezzanotte e il settaggio manuale da parte dell'utente
        */
        val ids = AppWidgetManager.getInstance(context)
            .getAppWidgetIds(ComponentName(context!!, 
                StatisticsWidgetProvider::class.java))
        onUpdate(context, AppWidgetManager.getInstance(context), ids)
    }
    super.onReceive(context, intent)
}
```

Implementazione di `onReceive` per gestire broadcast del cambiamento di data

Collection Widgets

Per la realizzazione di un Collection Widget, Android mette a disposizione due tipi di approcci: uno basato su una Factory per costruire gli elementi della lista, l'altro incentrato sul passare direttamente la collezione di elementi tramite un oggetto di tipo `RemoteCollectionItems`.

Per costruire un collection widget, ad esempio che implementi tramite `ListView` una lista scrollabile verticalmente, con il primo dei due approcci descritti sopra è necessario definire una classe che erediti da `RemoteViews.RemoteViewsFactory` che si occupa di caricare i dati da mostrare nel widget e costruire l'oggetto `RemoteViews` contenente i dati da mostrare in una certa posizione.

`RemoteViewsFactory`, similmente ad un Adapter di `RecyclerView` utilizzato all'interno di Activities e Fragments, agisce da collante tra la lista finale ed i dati che si vogliono mostrare.

I metodi principali di cui fare override sono `onCreate` e `getViewAt`.

- In `onCreate` è necessario ottenere la lista degli elementi, per esempio richiedendo un Cursor al Content Provider

- Il metodo getViewAt invece si riferisce all'elemento nella specifica posizione della lista, restituendo l'istanza di Remoteviews. E' importante specificare il layout dell'item ed eventualmente l'onClick Intent.

Poichè sarebbe molto dispendioso in termini di risorse creare un nuovo Pending Intent ogni volta che deve essere popolato un oggetto RemoteViews, si specifica un template per il Pending Intent comune, dopodichè tramite la funzione setOnClickFillInIntent si differenziano gli specifici Intent, "riempendo" i campi lasciati vuoti nel template.

Altre funzioni di cui può essere utile fare override sono onDataSetChanged, tramite la quale è possibile ottenere una lista o un cursore aggiornati, e getCount che permette al Widget di sapere quanti elementi andranno inseriti nella lista (ad esempio una ListView).

Inoltre, tramite un service che eredita da RemoteViewsService, il widget provider può richiedere gli oggetti di tipo RemoteViews. E' necessario specificare tale service nel Manifest, con permesso BIND_REMOTEVIEWS in modo che i widget possano collegarsi ed al contempo impedire che altre app riescano ad accedere ai dati.

```
<service
    android:name=".sessions_widget.SessionsWidgetService"
    android:enabled="true"
    android:permission="android.permission.BIND_REMOTEVIEWS" />
```

Un esempio di come dichiarare il RemoteViewsService nel manifest

L'approccio per creare un Collection Widget tramite RemoteCollectionItems è stato introdotto con Android 12 e, se usato per dati con numerosità limitata (liste con pochi elementi) e che non coinvolgono oggetti con all'interno Bitmap da mostrare nell'elemento della lista, può portare a prestazioni migliori azzerando i tempi di caricamento degli elementi quando viene percorsa la lista contenuta nel widget. L'oggetto RemoteCollectionItems va passato come parametro al metodo setRemoteAdapter di RemoteViews.

Per casi semplici questo strumento semplifica la creazione di un Collection Widget, non necessitando di creare una classe figlia di RemoteViewsFactory e non dovendo segnalare manualmente l'aggiornamento dei dati, sebbene nel complesso venga offerta meno versatilità rispetto al primo approccio.

La limitazione sull'ingombro di memoria eccessivo per oggetti che contengono Bitmap può essere superata se le immagini vengono invece conservate come URI.

AppWidgetManager

AppWidgetManager è un oggetto che permette all'app di gestire programmaticamente le istanze degli App Widget, che provengono da diversi providers. È creato e gestito dal sistema Android e si ottiene un suo riferimento chiamando AppWidgetManager.getInstance.

Con il metodo getAppWidgetIds si ottengono gli id dei widget associati all'AppWidgetProvider fornito come parametro. Questi potranno essere passati come argomenti extra in un intent mandato al Provider per l'aggiornamento.

```
val ids = AppWidgetManager.getInstance(this)
    .getAppWidgetIds(ComponentName(this, WidgetProvider::class.java))
val updateIntent = Intent(this, WidgetProvider::class.java).apply {
    action = AppWidgetManager.ACTION_APPWIDGET_UPDATE
    putExtra(AppWidgetManager.EXTRA_APPWIDGET_IDS, ids)
}
sendBroadcast(updateIntent)
```

esempio di uso di getAppWidgetIds per richiedere l'aggiornamento dei widget

Per i collection widget gli id devono essere passati come parametro al metodo notifyAppWidgetViewDataChanged per invalidare i dati e innescare la chiamata a onDataSetChanged della Factory.

Il metodo updateAppWidget imposta la RemoteViews per il Widget con l'ID specificato. Viene richiamata solitamente nella funzione di aggiornamento del Widget all'interno della mia classe figlia di AppWidgetProvider. Può essere richiamata sia nell'handler del Broadcast receiver che fuori, ed aggiorna completamente la vista, per cui l'oggetto di RemoteViews deve contenere tutte le informazioni per essere rappresentato correttamente.

```
override fun onUpdate(
    context: Context?,
    appWidgetManager: AppWidgetManager?,
    appWidgetIds: IntArray?
) {
    super.onUpdate(context, appWidgetManager, appWidgetIds)
    if (context == null
        || appWidgetManager == null
        || appWidgetIds == null){
        return
    }
    for (id in appWidgetIds) {
        val views = RemoteViews(context.packageName, R.layout.nome_layout)
        /*
         * Logica di costruzione del widget
         */
        appWidgetManager.updateAppWidget(widgetId, views)
    }
}
```

Esempio di utilizzo updateAppWidget

Altri metodi disponibili sono getAppWidgetOptions per ottenere un Bundle con i size bucket di ridimensionamento e getAppWidgetInfo, per ottenere informazioni sul tipo di widget come layout e periodo di aggiornamento.

Limitazioni di AppWidgetProvider

Come per tutti i BroadcastReceiver, anche a un AppWidgetProvider si applicano le stesse restrizioni:

- La durata di esecuzione di onReceive deve essere inferiore a 10 secondi
- Non è possibile fare il binding a un servizio, anche se questo mette a disposizione un'interfaccia di comunicazione tra processi come Messenger. Questo binding non viene consentito perché la connessione ad un servizio, se questo non è già avvitato, può impiegare più di 10 secondi. È possibile però utilizzare il metodo peekService per inviare dei messaggi ad un servizio che è già in esecuzione (il metodo ritorna null se il servizio non è attivo in quel momento)
- Non è possibile osservare oggetti LiveData
- Un BroadcastReceiver è stateless: non deve venire usato per salvare nessun dato persistente al suo interno, neanche nel companion object (che può non essere mantenuto tra le diverse esecuzioni)

Integrazione con Google Assistant

Come già spiegato, i widget possono essere pensati come viste ridotte di un'applicazione che possono essere integrate all'interno di altre applicazioni come ad esempio la schermata home, launcher di terze parti o Google Assistant.

A partire da Android 12, infatti, è stata introdotta la possibilità di utilizzare i widget come risposta ai comando di Assistant, con diversi benefici:

- Facilitare la scoperta dei widget supportati dall'applicazione attraverso interazioni in linguaggio naturale con l'utente

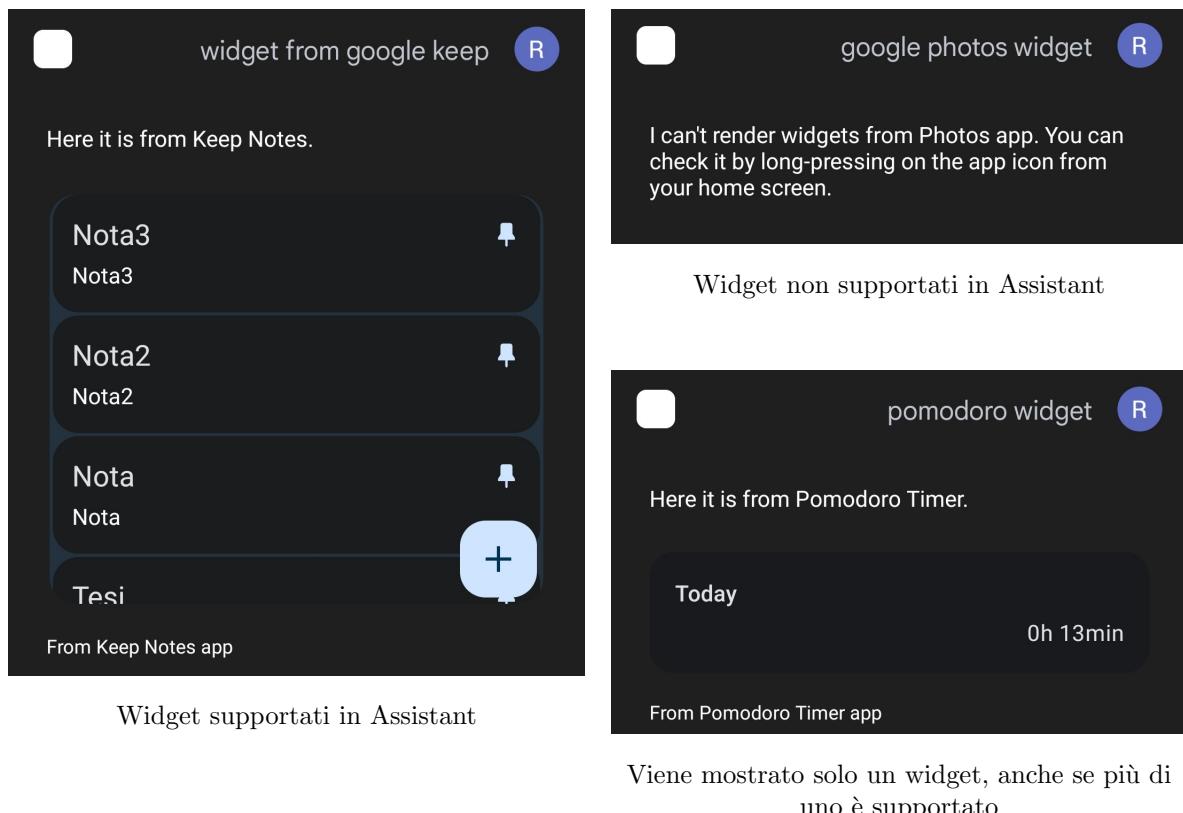
- Possibilità di interagire con l'applicazione anche in contesti in cui non è possibile interagire con tutte le funzionalità dello smartphone, mostrando i widget sulla schermata di blocco (attraverso la funzione "risultati personali" di Google Assistant) o all'interno di Android Auto
- Dare l'opzione di aggiungere il widget alla schermata home direttamente all'interno dell'assistente per facilitare le interazioni future

L'interazione con i widget in Assistant può avvenire sia attraverso una richiesta esplicita che in risposta a richieste di tipi predefiniti (Built-in Intents).

Invocazione esplicita

È possibile visualizzare alcuni Widget all'interno della schermata di Assistant con comandi come "ExampleApp widget".

Si osserva che, anche per applicazioni stock di Google, il supporto per questo metodo di invocazione non è automatico per tutti i widget. Con questo meccanismo, inoltre, qualora ci siano più widget associati a un'applicazione solo un widget viene mostrato in risposta.



Invocazione tramite Built-in Intent

Le richieste in linguaggio naturale più complesse vengono gestite da assistant mappandole su una lista di Intent predefinita, che copre sia necessità comuni a app di qualsiasi categoria (ricerca in-app, apertura di feature specifiche, gestione dell'account) sia interazioni specifiche per applicazioni di alcuni domini (ristorazione, salute, comunicazione eccetera).

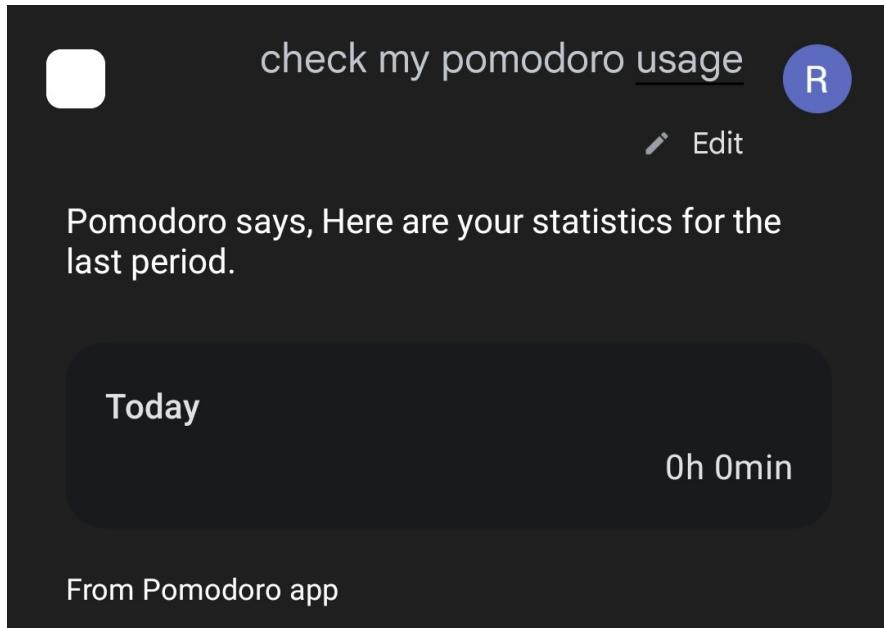
Per alcuni di questi Intent sono supportati anche i widget come risposte, offrendo diversi vantaggi rispetto all'invocazione esplicita:

- Possibilità di ricevere parametri dalla richiesta per modificare il contenuto del widget
- Supporto per text-to-speech personalizzato
- Opzione per aggiungere il widget alla schermata home direttamente in Assistant

Per abilitare questa opzione è necessario aggiungere al progetto il file shortcuts.xml con il relativo riferimento nel manifest e popolare questo file con le informazioni relative all'integrazione di ciascun widget con Assistant (ogni tipo di richiesta gestita corrisponde ad un tag <capability>, che contiene un elemento <app-widget> con le informazioni sul widget provider e i parametri accettati e un intent di fallback da usare in caso non venga specificato nessun parametro).

In aggiunta, è necessario modificare nel manifest i receiver associati ai widget da integrare con Assistant per aggiungere agli intent filter l'azione relativa all'aggiunta del widget nell'assistente (PIN_APP_WIDGET).

Per fornire risposte scritte e vocali personalizzate contestuali al widget si utilizza la Widget Extension Library, che attraverso il builder della classe AppActionsWidgetExtension permette di impostare i messaggi desiderati.



Esempio di invocazione tramite Built-in Intent con TTS personalizzato

Requisiti minimi

L'inglese è l'unica lingua in cui sono supportati tutti i Built-in Intents. Per alcuni di essi sono presenti anche altre lingue supportate, come lo spagnolo e il portoghese, ma in nessun caso l'italiano. Per avere la certezza di poter usufruire di tutte queste funzionalità, quindi, è necessario avere sia la lingua del telefono che quella di Assistant impostate su "en-US".

Per gli sviluppatori di applicazioni il supporto completo per le App Actions è garantito solo se si dispone di un'account Google Developer con accesso alla Play Console per poter registrare l'app che si vuole integrare con Assistant.

Layout e stile

Layout responsivi

Creando un insieme di file di layout, ciascuno per un range di dimensione, si permette di avere un widget che si ridimensiona e cambia quantità di informazione e struttura mostrata in base allo spazio disponibile.

Per permettere il ridimensionamento del widget, specificare i seguenti attributi nel file AppWidgetProviderInfo:

- minResizeWidth e minResizeHeight indicare le dimensioni minime, se inferiori a minWidth/minHeight/targetCellWidth/targetCellHeight
- maxResizeWidth e maxResizeHeight per indicare le dimensioni massime
- resizeMode per le direzioni di espansione, per esempio horizontal, vertical o entrambe (horizontal—vertical)

Questo snippet esemplifica come creare le view e fornire la mappatura dei diversi layout alle dimensioni supportate:

```
override fun onUpdate() {  
    val smallView = RemoteViews(...)  
    val mediumView = ...  
    ...  
  
    val viewMapping: Map<SizeF, RemoteViews> = mapOf(  
        SizeF(80f, 100f) to smallView,  
        SizeF(80f, 220f) to mediumView,  
        SizeF(80f, 340f) to largeView,  
        SizeF(80f, 460f) to extraLargeView  
    )  
    val remoteViews = RemoteViews(viewMapping)  
  
    appWidgetManager.updateAppWidget(appWidgetId, remoteViews)  
}
```

Le dimensioni di SizeF(width, height) si riferiscono alle dimensioni a partire dalle quali verrà mostrato il layout corrispondente.

Nell'esempio smallView verrà mostrato fino ad una dimensione 80dp x 219dp, mediumView a partire da 80dp x 220dp fino a 80dp x 339dp, etc.

L'alternativa ai layout responsivi sono i layout exact-size: implementare il metodo onAppWidgetOptionsChanged() e invocare AppWidgetManager.getAppWidgetOptions() per ottenere un bundle da cui leggere la lista delle dimensioni esatte occupabili a runtime con la chiave OPTION_APPWIDGET_SIZES, oppure una specifica dimensione con OPTION_APPWIDGET_MIN_WIDTH, per esempio. Dopodiché sarà possibile creare il layout specifico sapendo la dimensione del widget.

Colori dinamici

Per usare i colori dinamici creare un tema che erediti da @android:style/Theme.DeviceDefault.DayNight o Theme.Material3.DynamicColors.DayNight, che permetterà di assegnare agli attributi i colori ?android:attr/... oppure @android:colors/system_accent...

I colori di accento varieranno in base allo sfondo impostato e alla posizione relativa sullo schermo. Derivano dalla palette di uno o tre colori generata dal sistema in base ai colori dominanti dell'immagine di sfondo(che l'utente può comunque modificare)



Palette di colori generati a partire dall'immagine di sfondo



Widget meteo con tema di sistema scuro



Widget meteo posizionato sopra una parte prevalentemente celeste dello sfondo



Widget meteo posizionato sopra una parte prevalentemente arancione dello sfondo

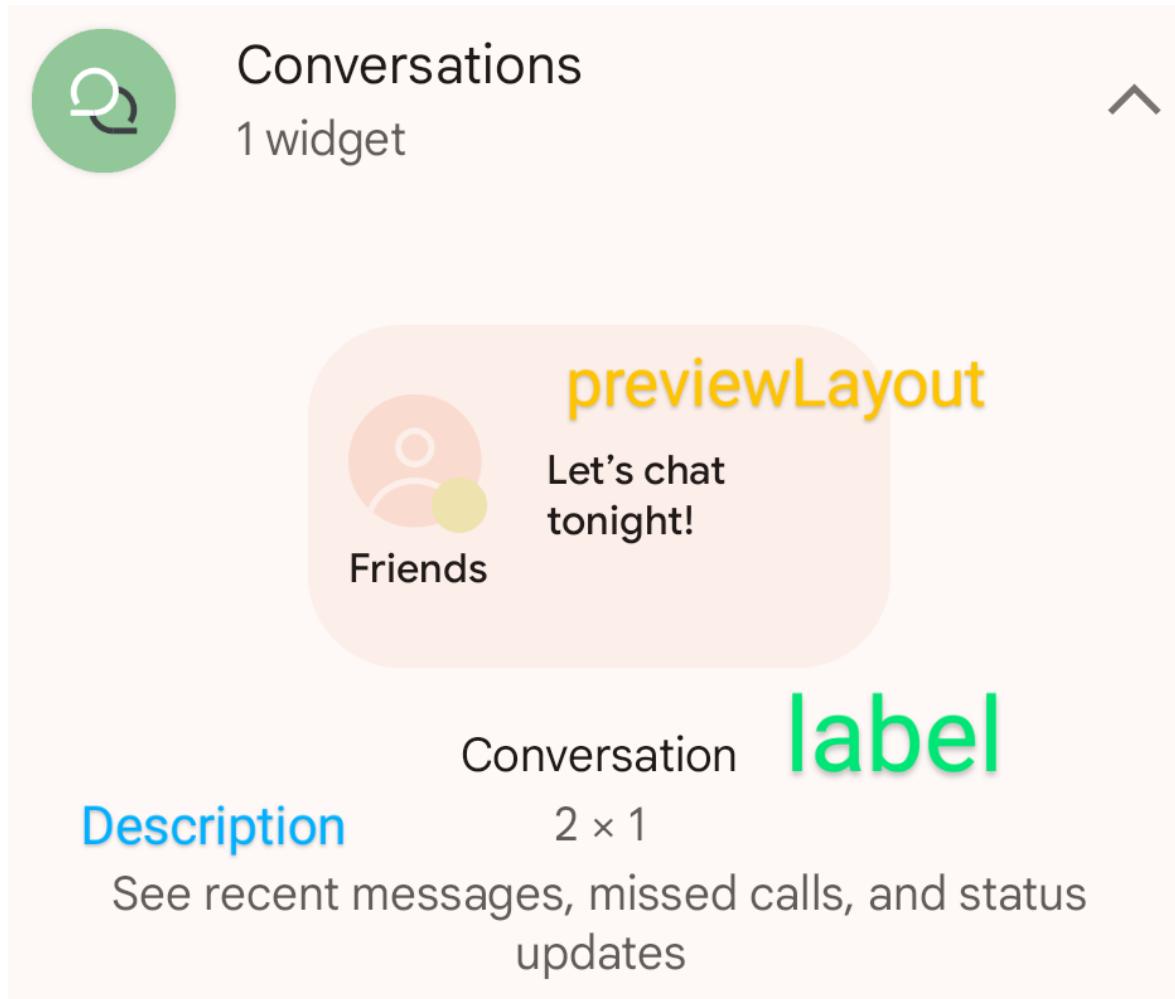
Preview nel widget picker

Nell'AppWidgetProviderInfo è consigliato specificare per l'attributo previewLayout lo stesso layout di initialLayout, se possibile, cosicché la preview del widget nel widget picker avrà lo stesso aspetto del widget quando verrà selezionato e posizionato sulla home screen, a meno dei valori placeholder finti.

Nel caso di launcher che non supportano l'attributo previewLayout bisogna assegnare al previewImage una immagine statica, per esempio uno screenshot del widget.

All'attributo description si assegna una stringa descrittiva.

Per fornire un titolo al widget bisogna assegnarlo all'attributo label nel componente dichiarato dentro AndroidManifest.xml.



Esempio di widget che fornisce un layout di preview, una label e una descrizione

Nel caso di widget che implementano collection view, quali ListView, GridView o StackView, se si usasse come previewLayout lo stesso layout effettivo, l'anteprima risulterebbe quasi vuota in quanto gli elementi vengono caricati dinamicamente.

Viene consigliato di creare un layout apposta per l'anteprima, inserendo manualmente degli elementi finti che popolano la vista.

Nel caso la collection view contenga elementi dal layout complesso, è più conveniente includere più volte tale layout con il tag <include> per evitare duplicazioni di codice. Nel seguente esempio si vuole creare la preview di una ListView in cui ogni elemento della lista è un layout costituito da una TextView.

- in attrs.xml aggiungere un attributo che rappresenta il contenuto finto del list item, in questo caso una stringa

```
<resources>
    <attr name="ListItemText" format="string" />
</resources>
```

- assegnare gli attributi appena dichiarati agli attributi del list item

```
<LinearLayout ...>
    ...
    <TextView
        android:id="@+id/widget_list_item_text"
        style="@style/TextAppearance.Material3.HeadlineMedium"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        ...
        android:text="?ListItemText" />
</LinearLayout>
```

- definire stili per ogni elemento dell'anteprima, per esempio in themes.xml

```
<style name="AppWidgetTheme.ListItem">
    <item name="ListItemText" />
</style>

<style name="AppWidgetTheme.ListItem.Preview1">
    <item name="ListItemText">
        @string/sessions_widget_fake_item_1
    </item>
</style>

<style name="AppWidgetTheme.ListItem.Preview2">
    <item name="ListItemText">
        @string/sessions_widget_fake_item_2
    </item>
</style>
```

- applicare gli stili agli elementi finti del layout

```
<LinearLayout ...>
    ...
    <include
        layout="@layout/sessions_widget_list_item"
        android:theme="@style/AppWidgetTheme.ListItem.Preview1" />

    <include
        layout="@layout/sessions_widget_list_item"
        android:theme="@style/AppWidgetTheme.ListItem.Preview2" />
</LinearLayout>
```

Pomodoro Timer

Per esemplificare tutte le funzionalità messe a disposizione dai widget abbiamo realizzato la app Pomodoro Timer, che può essere utilizzata in molte delle sue funzioni sia normalmente, con la tradizionale interazione a tutto schermo con l'applicazione in primo piano, che tramite widget.

La nostra applicazione è basata sulla [Tecnica del Pomodoro](#) per la gestione del tempo, utile per aumentare la produttività e mantenere la concentrazione.

Il tempo viene scandito secondo tre fasi che si alternano secondo un ciclo predefinito: fase Pomodoro (periodo di lavoro), pausa breve e pausa lunga. Dopo il completamento di una fase Pomodoro segue sempre una pausa (ogni tre pause brevi viene fatta una pausa lunga). Ciascun tipo di fase ha una durata impostabile dall'utente.

All'utente viene data la possibilità di organizzare il lavoro da fare in Task, ciascuno con un certo numero di fasi Pomodoro assegnate. I task vengono, a loro volta, organizzati in Sessioni, utili per dividere logicamente task tra di loro non collegati.

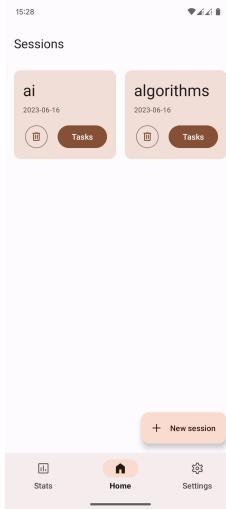
Per tutti i contenuti dell'applicazione è stata scelta la lingua inglese, ad eccezione della dicitura "pomodoro" che non è stata tradotta.

Alcuni aspetti nel funzionamento di questa applicazione sono realizzati prendendo ispirazione da [Forest](#), ad esempio la gestione dei timer e delle statistiche, e dall'applicazione [Orologio](#) di Google.

In tutta l'applicazione e nei widget sono stati usati i colori dinamici, che se supportati dal dispositivo aggiornano automatico il tema dell'applicazione in base ai colori dello sfondo della schermata home e alle impostazioni sul tema (chiaro o scuro).

Il codice del progetto può essere consultato a questo indirizzo: <https://github.com/Aehronburn/Pomodoro-widget>

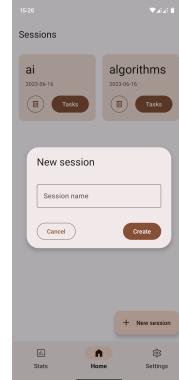
Lista sessioni



Schermata iniziale: lista delle sessioni

All'apertura dell'applicazione, la prima schermata presentata all'utente è quella della lista delle sue sessioni, ordinate dalla più recente alla più vecchia.

Da questa schermata si può navigare verso quelle di statistiche e di impostazioni, aprire i dettagli della sessione o crearne una nuova.



Creazione di una nuova sessione

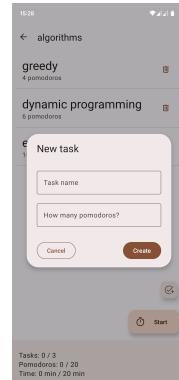
Dettagli sessione



Dettagli sessione

Nella schermata di dettagli, si possono visualizzare tutti i dati correlati a una certa sessione: nome della sessione, lista dei task associati con relativo numero di pomodoros per ciasuno di essi, statistiche sullo stato di completamento dei task.

Da questa schermata è possibile tornare alla lista delle sessioni, creare un nuovo task o aprire il timer.



Creazione di un nuovo task

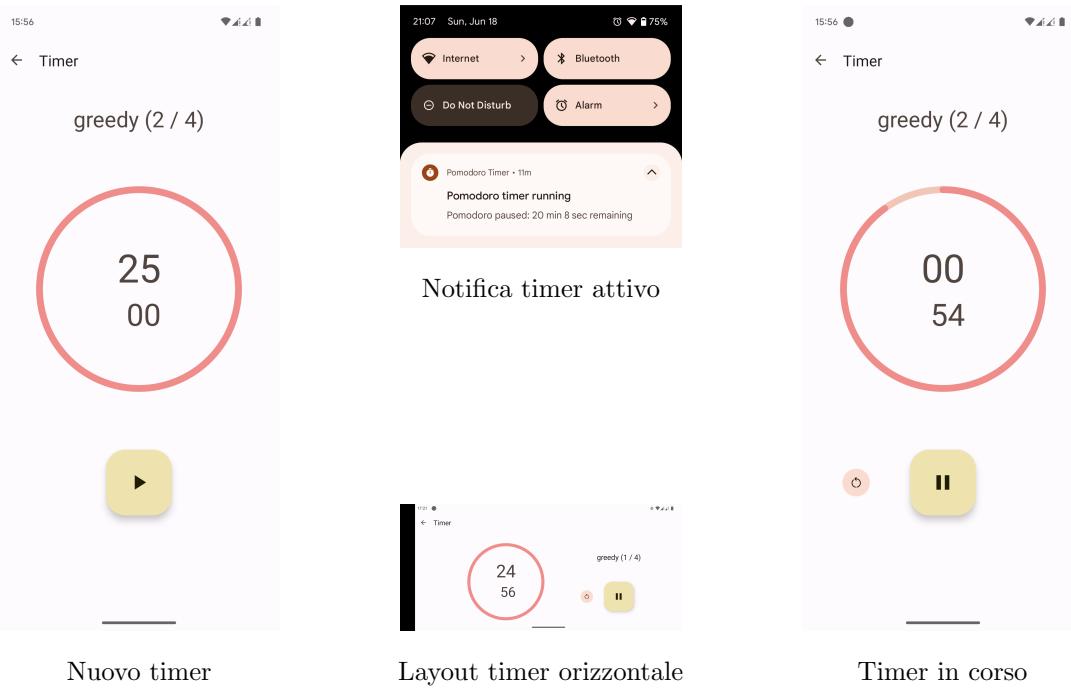
Timer

In questa schermata si può controllare il timer del Pomodoro, la funzionalità principale dell'app.

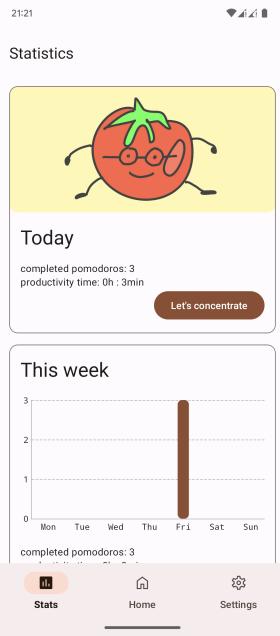
Per ogni fase, la durata iniziale a cui è impostato il timer dipende dal valore scelto dall'utente nella parte di impostazioni. Sono presenti due bottoni di controllo: uno per far partire/mettere in pausa il timer, l'altro per fermarlo e resettarlo al valore iniziale. Quando un timer viene completato, viene aggiornato automatico con la prossima fase da iniziare e si rimane in attesa che l'utente dia il comando di iniziare il timer successivo.

Il timer rimane in esecuzione solo fino a quando questa finestra è aperta: l'applicazione può essere messa in background senza che il timer venga arrestato, ma se si cambia schermata tornando ad esempio in quella dei dettagli della sessione o si termina manualmente la app il timer corrente viene perso. Come spiegato nella parte introduttiva, questo comportamento è fortemente ispirato al funzionamento di Forest, un'applicazione mirata alla limitazione dell'uso dello smartphone con molti meccanismi simili alla nostra.

Mentre il timer è attivo viene mostrata una notifica che contiene informazioni sulla fase corrente e sul tempo rimanente.



Statistiche



Schermata statistiche

La pagina statistiche fornisce informazioni sull'andamento delle proprie performance di produttività riguardo all'ultimo giorno, ultima settimana e ultimo mese.

Per quanto riguarda il giorno corrente, vengono mostrati il numero di pomodori completati e di minuti di concentrazione effettuati. I minuti di concentrazione non vengono calcolati partendo dal numero di pomodori completati e dalle impostazioni correnti della durata delle fasi, ma fanno riferimento alla somma delle durate effettive dei pomodori completati nel periodo.

Per quanto riguarda le statistiche settimanali e mensili, vengono mostrati due grafici che raffigurano l'andamento del tempo giornaliero di concentrazione dall'inizio della settimana corrente e dall'inizio del mese corrente rispettivamente. Il tempo di concentrazione viene ottenuto in modo analogo a quanto descritto per le statistiche del giorno.

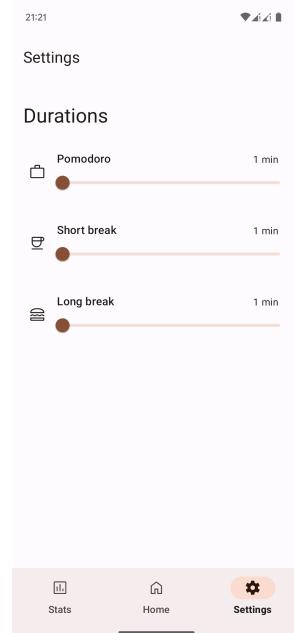
Per la realizzazione dei grafici è stata utilizzata la libreria [Vico](#).

Da questa schermata è possibile la navigazione verso la lista delle sessioni e le impostazioni.

Impostazioni

In questa schermata si possono impostare la durata di ciascuno dei tre tipi di fasi (pomodoro, pausa breve, pausa lunga) mediante degli slider. Queste impostazioni vengono salvate in modo persistente nel dispositivo mediante Shared Preferences.

Da questa schermata è possibile la navigazione verso la lista delle sessioni e le statistiche.



Schermata impostazioni

Widget statistiche

Questo widget dà la possibilità di controllare le proprie statistiche di concentrazione a colpo d'occhio, senza aprire l'app. Man mano che le dimensioni del widget aumentano si incrementa anche la quantità di informazioni disponibili: nella versione più piccola(layout small) sono visibili solo il numero di pomodori completati e il tempo di concentrazione nel giorno corrente, ingrandendo progressivamente il widget vengono mostrati anche i dati sulla settimana(layout medium) e poi sul mese corrente(layout large). Nel layout più grande, infine, viene anche mostrata un'immagine per indicare la produttività odierna(layout extralarge).



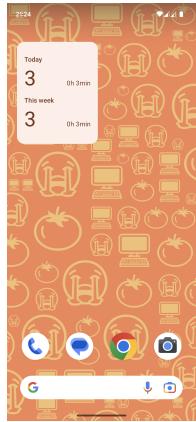
Widget statistiche 2x1 Layout
small



Widget statistiche 2x3 Layout
large



Widget statistiche 4x5 Layout
extralarge



Widget statistiche 2x2 Layout
medium



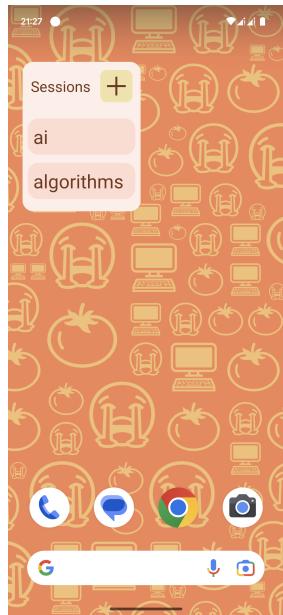
Widget statistiche 3x4 Layout
extralarge

Widget sessioni

Con questo widget viene data la possibilità di visualizzare la lista delle sessioni nella schermata home.

Questo widget ha due componenti principali: una lista di sessioni, scrollabile verticalmente, in cui è indicato il nome di ciascuna sessione (l'ordinamento delle sessioni è uguale a quello della schermata Lista sessioni all'interno dell'app), e un bottone per inserire una nuova sessione. Vi è un solo layout ridimensionabile da 2x2 a tutto lo schermo.

Premendo su un elemento della lista viene aperta la corrispondente schermata di dettagli della sessione all'interno dell'app, mentre con il bottone di inserimento sessione si apre l'applicazione con la lista delle sessioni e viene mostrato il dialog corrispondente.



Widget sessioni 2x2



Widget sessioni 2x5



Widget sessioni 4x2



Widget sessioni 4x5

Widget di controllo

Quando un timer è in corso, questo può essere controllato dalla schermata home tramite un apposito widget.

Il layout è stato strutturato in modo da non imporre limiti di dimensioni che comunque in alcuni contesti possono essere ignorati (come in launcher di terze parti), in modo da funzionare bene anche in quei casi. Sono stati predisposti quattro layout (denominati dal più piccolo al più grande "tiny", "small", "medium" e "large"), applicati a seconda della dimensione minore tra altezza e larghezza. I primi due layout fornisono solo informazioni riguardo al progresso del timer in corso, e per il layout small si mostra anche l'informazione del tipo di fase corrente tramite il colore del cerchio colorato sullo sfondo (in modo analogo a quanto avviene nella schermata Timer). Per i due layout più grandi si mostrano anche i bottoni di controllo con funzionalità identiche a quelle messe a disposizione all'interno dell'applicazione. In tutti i layout, premendo sul widget si apre la schermata con il timer attivo.

Quando non c'è nessun timer in corso, il widget entra in uno stato "idle" e mostra l'icona della app invece degli elementi dell'interfaccia Timer. Il layout idle è unico, a prescindere dalle dimensioni del widget.



Widget controllo 1x1
Layout tiny



Widget controllo 2x1
Layout small



Widget controllo 2x2
Layout medium



Widget controllo 4x4
Layout large

Quando il widget viene aggiunto alla schermata home viene fornita l'opzione di scegliere se il suo sfondo deve avere lo stesso sfondo usato all'interno dell'applicazione o essere trasparente. Questo sfondo viene applicato in tutti i layout per tutte le dimensioni, sia per il timer in corso che quando è idle.

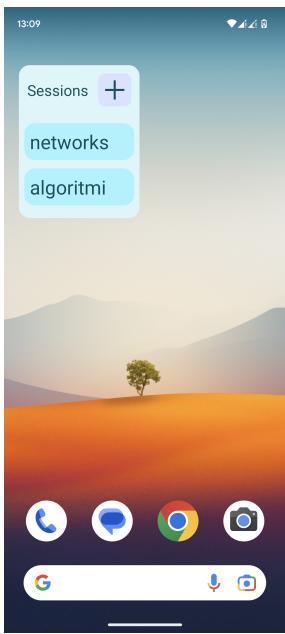


Activity di configurazione

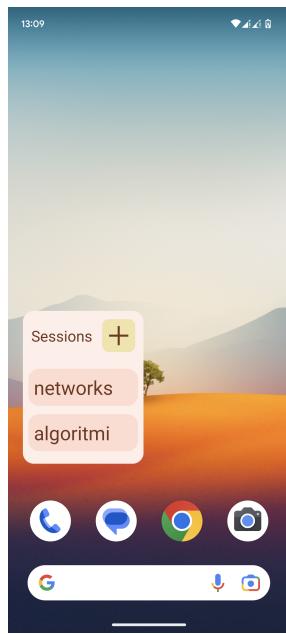


Widget con sfondo trasparente

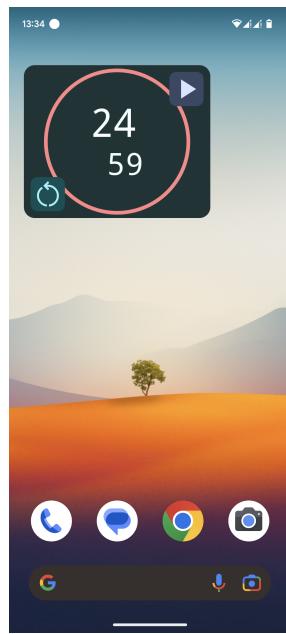
Colori dinamici nei widget



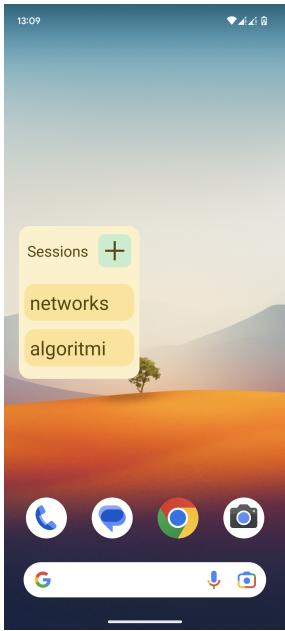
Widget sessioni su sfondo celeste



Widget sessioni su sfondo rossastro



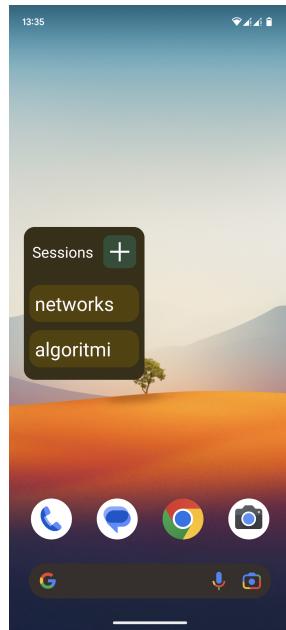
Widget controlli tema scuro



Widget sessioni su sfondo giallastro



Widget statistiche tema scuro



Widget sessioni tema scuro

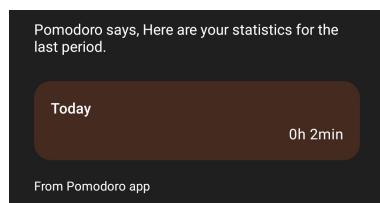
Integrazione con Google Assistant

Questa app include la possibilità di un'integrazione con Google Assistant per alcuni dei suoi widget con lo scopo di mostrare le possibilità offerte da questo strumento e i suoi limiti.

In particolare, dei tre widget (statistiche, controllo, lista delle sessioni) per i primi due è stata predisposta qualche forma di integrazione con l'assistente di Google.

Come spiegato nella sezione del report riguardo ad Assistant, ogni widget da integrare deve essere associato a un Built-in Intent per essere accessibile tramite questo strumento. I BII che sono stati scelti sono "Get service observation" per il widget delle statistiche e "Get news article" per quello di controllo. Mentre per il widget delle statistiche il BII scelto può essere appropriato (in quanto si può invocare tramite comandi come "check my Pomodoro usage"), per il compito di mostrare il controllo del timer non è presente nessun BII veramente calzante nella lista, quindi è stato scelto quello più concettualmente simile. Si noti che non tutti i BII supportano widget come risposta: "Open app feature" e "Get thing" possono essere usati solo per aprire funzioni interne all'app. Pur non essendo disponibile un BII adatto, abbiamo comunque voluto integrare anche il widget di controllo in Assistant per evidenziare come i widget possano essere aggiornati ogni secondo e interagire con i servizi anche in questo contesto, in modo identico a come avviene per i widget nella homepage.

Per entrambi i widget sono previste delle risposte testuali e sotto forma di Text to Speech alle interazioni in linguaggio naturale.



Widget statistiche (Test App Actions)



Widget statistiche (linguaggio naturale)



Widget di controllo (Test App Actions)

Come spiegato nella sezione del report a riguardo di questa integrazione, ci sono dei requisiti minimi da rispettare per testare questa funzionalità nel proprio dispositivo. Alcuni di questi requisiti riguardano la lingua da impostare sul dispositivo e in Assistant (en-US per entrambi), l'accesso con lo stesso account in Android Studio e nell'assistente, la creazione di una Preview per il test delle App Actions tramite il plugin Google Assistant. Maggiori informazioni a riguardo possono essere trovate nel file shortcuts.xml del progetto.

Bibliografia

- Panoramica e tipi di widget
<https://developer.android.com/develop/ui/views/appwidgets/overview>
- Spiegazione funzionamento di base Widget
<https://developer.android.com/develop/ui/views/appwidgets>
- Documentazione RemoteViews
<https://developer.android.com/reference/android/widget/RemoteViews>
- Documentazione AlarmManager
<https://developer.android.com/reference/android/app/AlarmManager>
- Extras update widget
https://developer.android.com/reference/android/appwidget/AppWidgetManager#ACTION_APPWIDGET_UPDATE
- Dimensionamento layout
<https://developer.android.com/develop/ui/views/appwidgets/layouts>
- Novità Android 12
<https://developer.android.com/about/versions/12/features/widgets>
- Documentazione AppWidgetProvider
<https://developer.android.com/reference/android/appwidget/AppWidgetProvider>
- Creazione Collection Widgets
<https://developer.android.com/develop/ui/views/appwidgets/collections>
- Limitazioni binding BroadcastReceiver
<https://stackoverflow.com/questions/18931670/binding-service-by-broadcastreceiver>
- Introduzione integrazione Assistant
<https://developer.android.com/guide/app-actions/widgets>
- Lista BII
<https://developer.android.com/reference/app-actions/built-in-intents/bii-index>
- Codelab Assistant widget
<https://codelabs.developers.google.com/codelabs/appactions-widgets>
- Tipi di BII e requisiti
<https://developer.android.com/guide/app-actions/get-started>
- Annuncio supporto widget in Assistant
<https://www.androidpolice.com/2021/05/18/widgets-are-coming-to-google-assistant>
- Assistant schermata di blocco
<https://support.google.com/assistant/answer/7684543>
- Creazione shortcuts.xml
<https://developer.android.com/guide/app-actions/action-schema>
- Colori dinamici
<https://developer.android.com/develop/ui/views/appwidgets/enhance#dynamic-colors>