Programmazione di Dispositivi Mobili

Lezione 1 - Introduzione al Corso

5 Ottobre 2022

Lezione 2 - Introduzione ai Dispositivi Mobili

6 Ottobre 2022

Lezione 3 - La complessità dello sviluppo, testing, costo e guadagni.

7 Ottobre 2022

La complessità dello sviluppo e il testing

Un'applicazione deve essere semplice, economica e creare dipendenza, ovvero "addicting". Per farlo ci poniamo 11 sfide:

1. Prima sfida - Idea: trovare un'idea di un app coinvolgente e che gli utenti useranno.
2. Seconda sfida - Costo: costi di sviluppo, trovare dei finanziatori.
3. Terza sfida - Diversi Dispositivi: diversi sistemi operativi con diversi schermi.
4. Quarta sfida - Interattività: non limitarsi a click e swipe, utilizzare i sensori. Usare UI e UX (User Interface e User Experience). La User Interface è come sono disposti gli elementi all'interno dell'interfaccia grafica. La User Experience è la modalità di interazione che il cliente deve sostenere per ottenere ciò che vuole.
5. Quinta sfida - Gestione dei Contenuti: immagini, video, animazioni devono essere sempre aggiornate! Problema di aggiornamento legato alla larghezza di banda, dunque utilizzare i M-CMS (Mobile Content Management System), ovvero dei sistemi in grado di mantenere contenuti e di pusharli su piattaforme diverse in modo che ci sia un'unica fonte di contenuti per ogni app. Firebase è un esempio.
6. Sesta sfida - Navigabilità: l'interfaccia deve essere chiara.
7. Settima sfida - Architettura: Deve essere chiara per facilitare il debug. Usare MVVM (Model-View-View-Model) e MVP (Model-View-Presenter).
8. Ottava sfida - Batteria: Rendere l'app Lazy First, ovvero uso intelligente della batteria.
9. Nona sfida - Performance: le prestazioni devono essere garantite per far sì che l'app venga usata.
10. Decima sfida - Promozione: promuovere l'applicazione, acquisizione per gli utenti e monetizzazione. Usare ASO (App store optimization)

11. Undicesima sfida - Test: testare e migliorare. Usare Monkey, che genera pseudo random test.

Costi

Costi di sviluppo:

- Piattaforma: più piattaforme supportate e più alto il costo.
- Obiettivi e Modello di Sviluppo: documento di specifiche tecniche, set fisso/dinamico di caratteristiche.
- Design: occorrono persone con esperienza di UI e UX.
- Sviluppo: possiamo usare piattaforma di sviluppo, strumenti cross-platform, sviluppo nativo in base alle nostre esigenze e costi.
- Caratteristiche dell'app: utilizzo di memoria, sensori, batteria.
- Infrastruttura: applicazioni con componente remoto, configurazione server, memorizzazione ecc..
- Developer Account e Componenti server-side e servizi Cloud.
- Aggiornamenti: evitare app zombie, ovvero app che non sono più visibili ai clienti perché nessuno li considera e non sono più mantenute.

Guadagni

Ci sono 4 modelli di business:

1. Purchase-app-once: applicazioni a pagamento da comprare una volta. Sono poche.
2. Freemium app: propongono due versioni, la versione base gratuita e la versione premium a pagamento.
3. Subscription app: applicazioni dove gli utenti pagano un canone periodico per l'utilizzo.
4. In-App Revenue: App che permettono di acquistare bundle o inseriscono advertisement.

Ci sono 4 tipi di advertising:

1. Banner: occupano poco spazio, meno invasivi e l'utente può usare l'app mentre il banner viene mostrato.
2. Interstitial: a schermo intero, visualizzati ogni tot di tempo.
3. Incentivized rewarded video: video pubblicitari per garantire qualcosa all'interno dell'app.
4. Native: si presenta come naturale continuazione dei contenuti, accesso più spontaneo. Esempio: Instagram

Lezione 4 - Tipologie di App Mobile

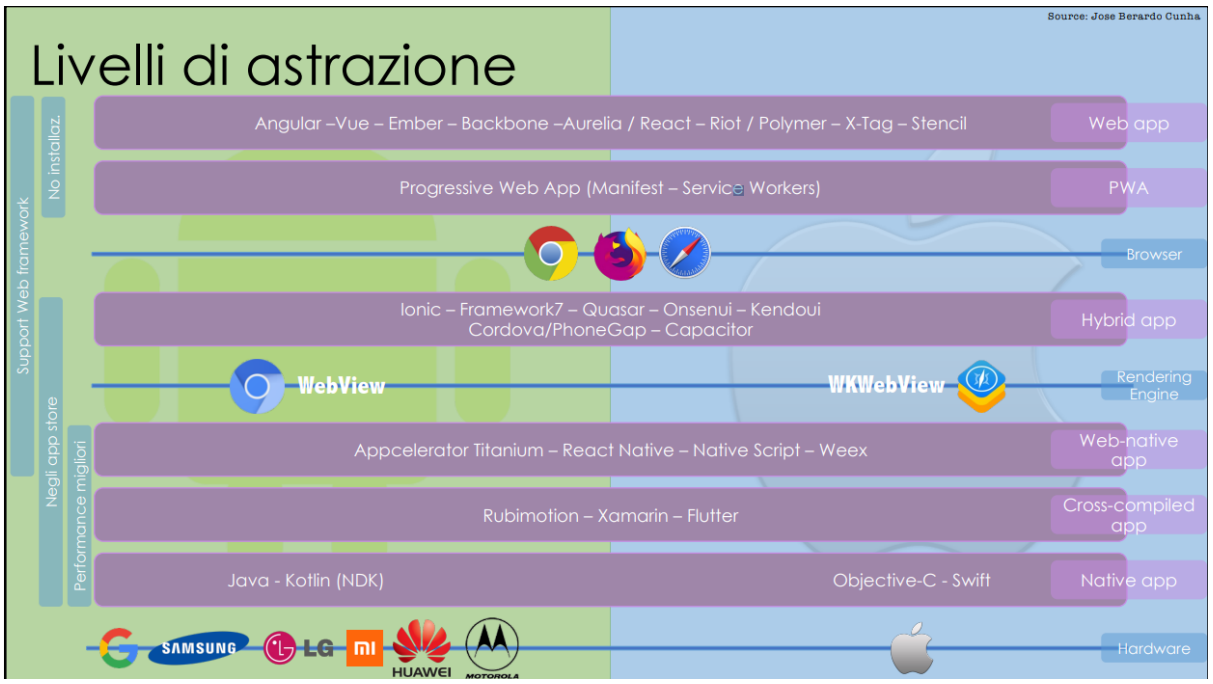
12 Ottobre 2022

Tipi di Architetture

Al momento abbiamo due principali player, ovvero android e ios, ma abbiamo molti linguaggi, molti framework e molte architetture per sviluppare mobile. Ecco le varie architetture presentate da quelle più a basso livello fino a quelle in cui vi è maggiore astrazione.

- Native App: sfrutta al meglio tutte le feature dell'ambiente in cui programmo. Es: Java, Kotlin, Swift.
- Cross-Compiled: App che sono compilate in un unica sorgente codebase che viene tradotta in codice effettivo. Ci schiodiamo da HTML, CSS e JS. Es. RubyMotion, Flutter.
- Web-Native App: non richiedono che ci sia un engine o una view ma si portano dietro dei motori che interpretano. Sono più veloci. Es. React Native.
- Hybrid App: si appoggiano agli engine del browser per fare il rendering, sia per catturare l'interazione dell'utente che interfacciarsi con le risorse tramite le API del OS.
- Progressive Web App (PWA): sono come delle web app ma sono installabili. Es. Twitter, Instagram, Tinder. Capable, Reliable e Installable. Alla base funzionano attraverso Manifest e Service Worker. Manifest è un file JSON che contiene metadati sull'app, come il nome, le icone e altre proprietà. Service Worker sono file JavaScript che vengono eseguiti in background e interagiscono con la funzionalità di memorizzazione nella cache del browser.
- Web App: sono framework completi o a livello di UI che permettono di girare indipendentemente dal browser. Non hanno accesso diretto alle API native ma possono arrivarci tramite le sue API web che interagiscono con l'OS. Utilizzano HTML, CSS e JS. Es. Angular, Vue, React. Le Web app costituiscono il miglior strumento per raggiungere molte persone ma hanno un più basso coinvolgimento dell'utente e viceversa.

Le Instant App o App Clip sono delle applicazioni dove non è necessario installare per usarlo. Le app clip sono di Apple e sono, ad esempio, app utilizzabili tramite QR Code, mentre le instant app sono più un try now.



La mia App ha una necessità intensiva dei processi della CPU? App Native.
Il mio team NON è abbastanza grande per mantenere due codebase? Tutto tranne App Native.
Cosa sa fare meglio il mio team? C# → Xamarin, Java → Native, Javascript → Web to Web Native.

Lezione 5 - Piattaforma Android, Prima App

19 Ottobre 2022

Piattaforma Android

Architettura Android:

- Linux Kernel: si interfaccia con l'hardware, open source.
- Hardware Abstraction Layer (HAL): Fornisce interfacce dei dispositivi alle API.
- Android Runtime: converte il bytecode in macchina.
    - Dalvik: Inizialmente era Dalvik che si basava su Just in Time Compilation.
    - ART: Dopo abbiamo usato ART che si basa su Ahead of Time Compilation.
    - PGC: Dopo abbiamo usato una combinazione delle due con la Profile Guided Compilation, dove si usava AOT per il run code frequente.
    - PITC: Adesso usiamo Profiles in The Cloud dove vengono registrate le modalità dell'utente che vengono subito compilate all'installazione e dopo si prosegue con PGC.
- Native C/C++ Libraries: insieme di librerie runtime.

- Java API Framework: Insieme di API Java che forniscono le funzionalità.
- System Apps: Applicazioni di base e nuove app.

Prima App

L'activity è un tipo di componente che fornisce la UI. SDK definisce la classe e si scrivono sottoclassi per implementare le funzionalità richieste.

Un Layout definisce un insieme di elementi della UI e la loro posizione. Ad ogni Activity è associato un Layout.

Screen size è la dimensione fisica misurata come diagonale dello schermo.
Screen density è il numero di pixel all'interno di un'area fisica dello schermo indicata come dpi.
Screen resolution è il numero totale di pixel sullo schermo.

I dp sono un'unità astratta che si basa su 1px in uno schermo a 160 dpi.
Gli sp sono gli scalable-independent pixel sono come i dp normali ma scalati in base alla preferenze dell'utente.

Struttura progetto Android:
- manifest contiene informazioni di base dell'activity.
- java contiene i file sorgente java.
- resource contiene le risorse:
    - drawable contiene le immagini.
    - layout che contiene il layout della UI dell'activity main.
    - mipmap contiene le icone, in base ai dpi.
    - values contiene i file xml come strings e styles.

Osservazione: resource può contenere altre cartelle.

La classe main Java estende la classe AppCompatActivity è una sottoclasse Activity che fornisce la compatibilità per le versioni precedenti di Android.

Ereditiamo il metodo onCreate() che aggancia xml al layout attraverso setContentView.

Le risorse sono file addizionali e contenuti statici che l'app usa, da mantenere separati al codice. Una risorsa viene acceduta tramite il suo ID all'interno della classe R. A seconda del tipo di risorsa, il suo ID è all'interno di una sottoclasse apposita. Un esempio è R.layout.activity_main.

Dunque, riprendendo l'esempio della stringa e quest'ultimo della risorsa, scopriamo che ci sono due modi per accedere ad una risorsa:
- All'interno del codice: usando R.tipo.risorsa.

- All’interno del file XML: usando @tipo.risorsa.

Le applicazioni Android sono tipicamente event-driven. Si definisce un listener per rispondere ad un evento che implementa un'interfaccia specifica per il tipo di evento.

Lezione 6 - UX-UI, Material Design

26 Ottobre 2022

UX-UI

UX è un processo articolato e complesso che coinvolge più persone con competenze diverse.

Il processo per ottenere una buona UI è:

1. Identificare il problema.
2. Definire le storie utente.
3. Ideare una soluzione.
4. Disegnare uno sketch e wireframe per l’organizzazione sulle schermate.
5. Realizzare mockup e prototipi.

- Sketch: vista meno dettagliata, semplice immagine che aiuta a comprendere rapidamente l’idea, senza dettagli, usa e getta fatta con carta e penna.
- Wireframe: utilizziamo layout a blocchi per visualizzare delle schermate dal punto di vista funzionale. Usiamo Balsamiq come strumento.
- Mockup: riempiamo gli spazi bianchi con colori, immagini. Mostrano solo le schermate e senza interazioni. Usiamo Sketch o Adobe come strumenti.
- Prototype: permettono di interagire con gli elementi dell’app. Simulazione dell’esperienza finale. Come strumenti usiamo UXPin e Marvel.

Material Design

Lo scheumorfismo indica un ornamento grafico apposto su un oggetto per richiamare un altro oggetto. L’ha usato Steve Jobs, con Apple, alle origini delle interfacce delle applicazioni, è stato poi abbandonato per la sua pesantezza visiva e pratica.

Bill Gates, con Microsoft, invece, utilizza il Metro/Flat Design dove elimina le sfumature, texture e ombre, introducendo spazi netti e definiti.

Il Material Design, introdotto da Google nel 2014, è la via di mezzo tra Microsoft ed Apple. Il Material Design si basa su principi del mondo reale, come le ombre, la profondità, i riflessi. Codice Open Source.

Lezione 7 - Fondamenti MD, Componenti

2 Novembre 2022

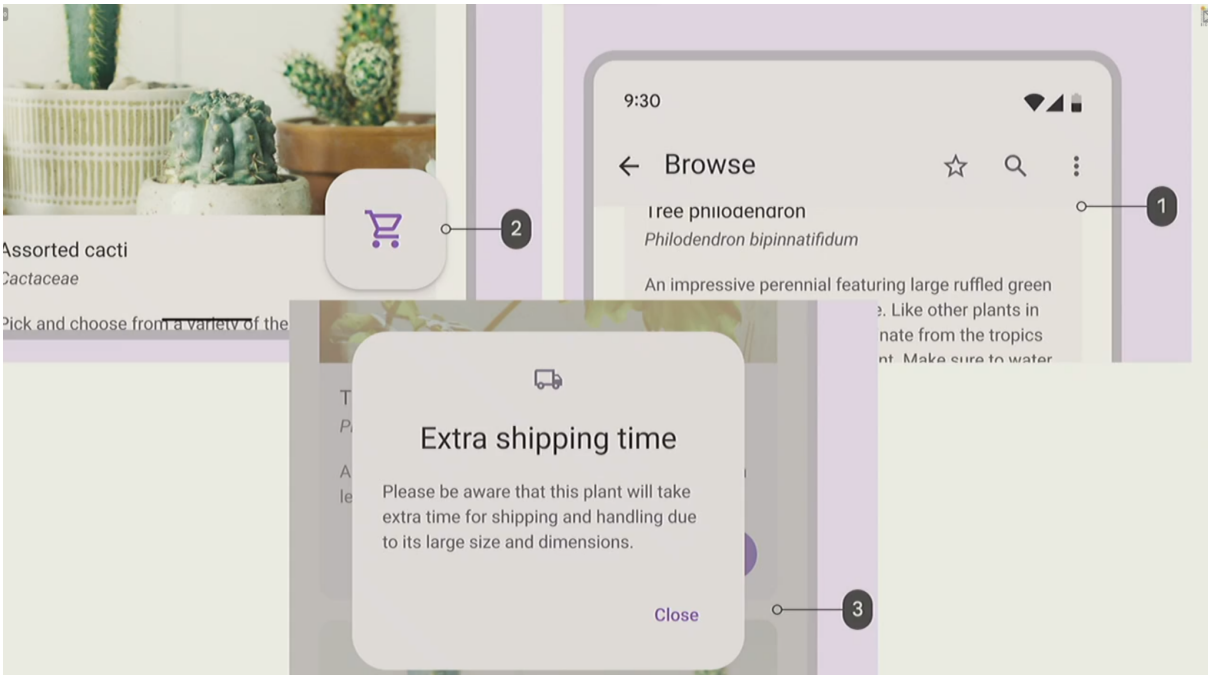
Fondamenti MD

Il materiale è l'elemento base, come un foglio di carta intelligente sull'asse tridimensionale.

Non esiste altezza nei materiali, hanno spessore uniforme di 1dp.

L'elevazione è la distanza relativa tra due superfici lungo l'asse z. Ci sono 3 tipi

1. riempimenti di superficie con una differenza di tonalità.
2. ombre o spunti visivi.
3. schermature.



I componenti hanno valori di default per l'elevazione in resting elevations. I componenti possono cambiare dp in risposta agli input dell'utente. Ci sono 6 livelli, di cui gli ultimi due non sono usati a resting level.

Il layout, che è fatto da 8dp per elementi e 4dp per testo e icone, deve essere responsivo tramite 3 elementi:

- columns, dove vanno gli elementi.
- glutters, spazi tra elementi.
- margins, ai lati degli elementi.

Il numero di colonne visualizzato è chiamato Breakpoint range.

Ci sono diversi tipi di navigazione:

- laterale, spostamento tra schermate di stesso livello gerarchico.
- avanti, spostamento a livelli consecutivi di gerarchia.
- inversa, spostamento all'indietro tra le schermate sia gerarchicamente che cronologicamente.

Componenti

Android assegna un unique user ID ad ogni app (UID) per impostare una application sandbox a livello kernel in modo da implementare il principio del minor privilegio.

Ci sono però attacchi dall'ingegneria sociale, ovvero app che convincono l'utente a fornire permessi per scopi diversi da quelli indicati.

Un app Android è composta da componenti, di 4 tipi diversi. Ogni tipo ha uno scopo specifico ed un ciclo di vita proprio:

1. Activity.
2. Service.
3. Broadcast.
4. Content Provider.

Un'activity è il componente che ha una UI. Facilita le seguenti interazioni tra sistema e app:

1. Resumed Activity: traccia le activity visualizzate.
2. Stopped Activity: traccia le activity interagite in precedenza.
3. Destroyed Activity: facilita il ripristino di activity interrotte.

I service sono componente senza UI che girano in background, assumono due forme:

- Started Service: avviato da componente e rimane in vita fino al completamento della sua task, possono essere conosciuti o sconosciuti dall'utente.
- Bound Service: qualcuno vuole utilizzare un servizio e lui li vive fino a quando sono necessari.

Un broadcast receiver consente al sistema di far pervenire all'app eventi al di fuori del suo flusso regolare di esecuzione.

I content provider gestiscono dati di un app che possono essere memorizzati in un file system.

Lezione 8 - Ciclo di Vita Activity, Attivazione Componenti, Fragment

9 Novembre 2022

Ciclo di Vita di Activity

Ogni activity può avviare un'altra activity per eseguire azioni diverse, come ad esempio Facebook che lancia Messenger.

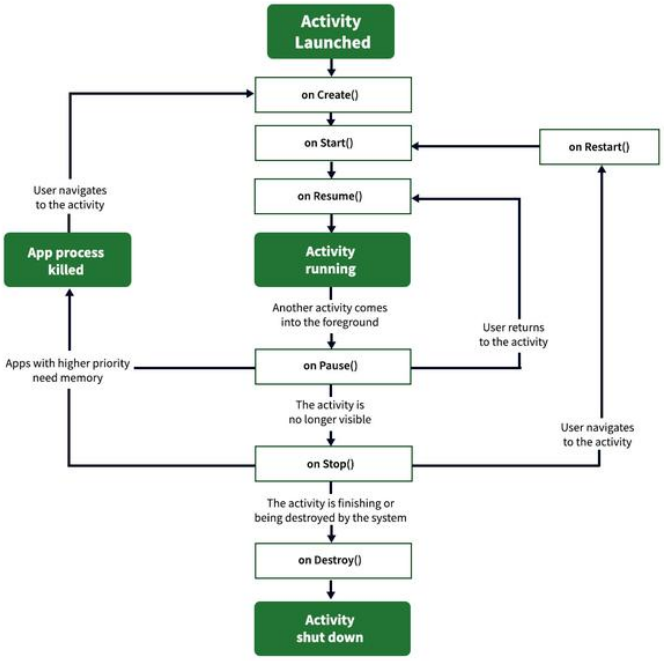
Anche se le activity formano un'esperienza coesa in un'app, ogni activity deve essere debolmente legata alle altre.

Affinchè vengono utilizzate, devono essere registrate nel file manifest e bisogna gestire il loro ciclo di vita.

Un'activity può trovarsi in diversi stati: created, started, resumed, paused.

Le callback permettono alla activity di sapere che lo stato è cambiato.

Lo sviluppatore deve implementare all'interno delle callback il giusto comportamento atteso della activity.



onCreate() crea l'activity:

- inizializza attributi.
- setContentView() imposta layout.
- savedInstanceState() ha lo stato precedentemente salvato della activity.
- sempre override.

onStart() avviene dopo onCreate() o dopo onStop():

- operazioni affinché sia visible e in foreground.
- quasi mai override.

onResume() avviene dopo onStart() o onPause()

- interazione con la activity.
- acquisizione risorse del sistema.
- quasi sempre override.

onPause() è l'evento che porta l'activity a non essere in foreground, come una telefonata o telefono che si spegne:

- arresto funzionalità, rilascio risorse del sistema.
- non si eseguono le operazioni di salvataggio perché il metodo dura poco.

onStop() rende l'activity non visibile all'utente:

- arresto funzionalità, rilascio risorse del sistema.
- si eseguono le operazioni di salvataggio perché il metodo potrebbe durare.

onRestart() l'utente ritorna alla activity dopo onStop():

- ripristino oggetti creati e poi disabilitati.
- quasi mai override.

onDestroy() quando l’attività sta terminando o il sistema la distrugge perché l'utente ha chiesto una modifica di configurazione:

- rilascio tutte le risorse non rilasciate nelle callback.

Quando l’utente si aspetta di ritrovare una activity nello stato in cui l’ha lasciata, occorre salvare lo stato volatile della activity affinché possa essere ripristinato attraverso la Classe ViewModel, il metodo onSaveInstanceState o la memorizzazione locale.

Se i dati della UI sono semplici e leggeri, come un tipo di dato primitivo o un oggetto semplice, si può usare solo onSaveInstanceState() per persistere lo stato della activity.

In casi più complessi, si dovrebbe usare sia ViewModel che onSaveInstanceState() poiché onSaveInstanceState() comporta costi di serializzazione/deserializzazione.

I dati salvati che il sistema utilizza per ripristinare lo stato precedente sono chiamati instance state e sono una raccolta di coppie di chiave-valore memorizzate in un oggetto di tipo Bundle.

Il Bundle va usato per dati di dimensioni ragionevoli poiché fa serializzazione, occupando il thread principale e consuma quindi risorse.

Una task è un insieme di activity con cui l’utente interagisce per svolgere un compito. Sono organizzate in un back stack nell'ordine con cui sono state aperte:

- la nuova activity viene pushata sopra alle precedenti.
- quando si preme bruck viene distrutta con una pop con politica FILO.

Si hanno tanti Back Stack quanti i task avviati dall’utente. Quando si preme il tasto Home, lo stack viene messo in background. Se si preme la icon launcher il back task torna in foreground.

Attivazione Componenti

Un’app realizza le funzionalità per cui è stata implementata avvalendosi di componenti sia propri sia di altre app.

Activity, Service e Broadcast receiver vengono attivati da messaggi asincroni chiamati intent. Esistono due tipi di intent:

- esplicito, si chiede il nome del componente. Utilizzato con i Service.
- implicito, si chiede un tipo di componente attraverso una action. Il sistema sceglie il componente che soddisfa la richiesta, confrontando con gli intent filter nel file Manifest.

Oltre ad Action contiene altre informazioni utili, tra i quali:

- Component name.
- Data per specificare i dati su cui operare.
- Type, per specificare il tipo di dati.
- Category, per specificare il tipo di componente che dovrebbe gestire l'intent.
- Extras.

I Content provider sono attivati quando sono i destinatari di una richiesta chiamata content resolver, che gestisce le transazioni al componente che le necessita, fa da intermediario tra i due.

Fragment

I Fragment permettono di creare UI:
- dinamiche: si può aggiungere o rimuovere mentre la activity è in esecuzione senza distruggere l'activity.
- riutilizzabile: si può riutilizzare in contesti differenti.

Di solito una attività contiene diversi Fragment. Hanno un ciclo di vita distinto da quelle delle activity ma integrato con esso quando l'attività nasce, va in pausa o muore.

Ci sono dei metodi aggiuntivi rispetto ad una activity:
- onAttach() associa il Fragment alla Activity.
- onCreateView() crea la view hierarchy associata al Fragment.
- onViewCreated() quando la View è creata.
- onDestroyView() quando la view hierarchy è stata rimossa.
- onDetach() dissocia il Fragment alla Activity.

Per creare un Fragment sono necessarie almeno 4 fasi:
1. Definire il Layout del Fragment.
2. Definire la classe che realizza il Fragment.
3. Definire la host activity se non già definita.
4. Aggiungere il fragment alla activity.

FragmentManager è la classe responsabile CRUD dei Fragment ospitati.

A differenza dell'activity se la transazione non viene esplicitamente aggiunta al back stack, non è possibile ripercorrere la navigazione a ritroso.

Le *transazioni* sono necessarie in quanto garantiscono che le operazioni vengano fatte in modo atomico. Le transazioni sono effettuate da un Fragment Manager che utilizza i FragmentTransaction.

Le *transizioni* sono opzionali ma piacevoli in quanto si occupano di rendere esteticamente gradevole tali operazioni.

Il fragment può accedere all'istanza della Activity madre usando getActivity(). L'activity può accedere alle sue Fragment figlie mediante il FragmentManager.

Lezione 9 - Principi di Navigazione, Architecture Component

16 Novembre 2022

Principi di Navigazione

Il pulsante Back o lo swipe di sistema viene utilizzato per navigare in ordine cronologico inverso.

Il pulsante Up, che invece inseriamo nella nostra applicazione, ha lo stesso comportamento di Back, se non diversamente specificato.

La differenza principale consiste nel fatto che possiamo, con Up, costruire una navigazione gerarchica che mi permette di risalire la gerarchia logica.

Le parti del Navigation Component si dividono in 3:

1. Navigation graph è una risorsa XML che contiene le informazioni relative alla navigazione.
2. Nav Host contenitore vuoto che visualizza le destinazioni del navigation graph.
3. Nav Controller gestisce lo scambio di contenuti nel Nav Host.

Il vantaggio principale è che gestisce le azioni Up e Back secondo i principi spiegati prima.

Architecture Components

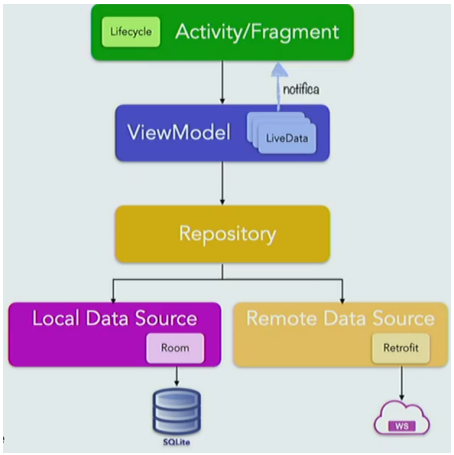
Jetpack è un insieme di librerie che facilitano il lavoro del programmatore, come Navigation Component. Gli elementi sono chiamati Android Architecture Components aiutano a creare app robuste, testabili e mantenibili nel tempo.

Vogliamo ridurre al minimo la dipendenza da queste UI class. Activity e Fragment devono contenere solo la logica che gestisce le UI.

Overview:

- Activity e Fragment si occupano della gestione della UI, condividendo il loro stato tramite un Lifecycle.
- ViewModel memorizza i dati con LiveData che osservano il LifeCycle per permettere l'aggiornamento quando l'Activity o il Fragment sono active. Inoltre ViewModel si interfaccia con il Repository per l'accesso ai dati
- Repository si occupa di recuperare i dati forniti dal ViewModel.

- Room è una mapping library SQLite che facilita il processo di gestione di un database.
- Retrofit è una libreria che facilita l'accesso ai dati remoti.



Bisogna definire una dipendenza uno a molti tra gli oggetti in modo che quando un oggetto (Observable) cambia stato, tutti quelli dipendenti (Observer) siano notificati e aggiornati.

Lifecycle è un observable che ha associato una Activity o un Fragment che è un LifecycleOwner, possiamo allora estenderla con un Observer che ne gestisce i metodi:

- Model: Lifecycle.
- View: LifecycleOwner (Activity/Fragment).
- Control: DefaultLifecycleObserver.

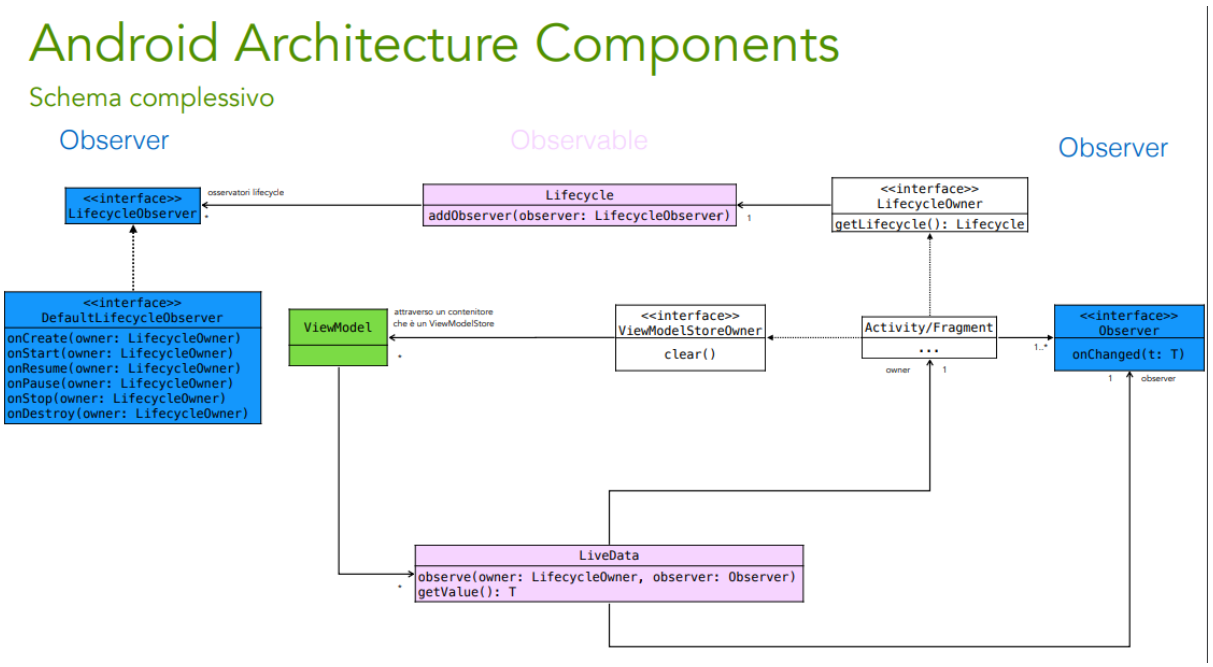
LiveData mantiene i dati della UI e notifica il cambiamento ad un observer attraverso il metodo onChanged. La notifica avviene solo se la UI owner è active.

ViewModel gestisce in maniera semplificata le informazioni a fronte di cambi di configurazioni, associandosi ai LiveData.

ViewModel non sostituisce onSaveInstanceState():

- ViewModel funziona solo per activity distrutte e ricreate a seguito di cambi di configurazione.
- onSaveInstanceState() può essere usato anche per activity che vengono terminate dal sistema per liberare memoria.

Usiamo ViewModel per memorizzare tutti i dati della UI, onSaveInstanceState() per memorizzare i dati necessari a ricostruire la UI.
ViewModel può anche essere usata come mezzo di comunicazione tra i fragment di una activity.

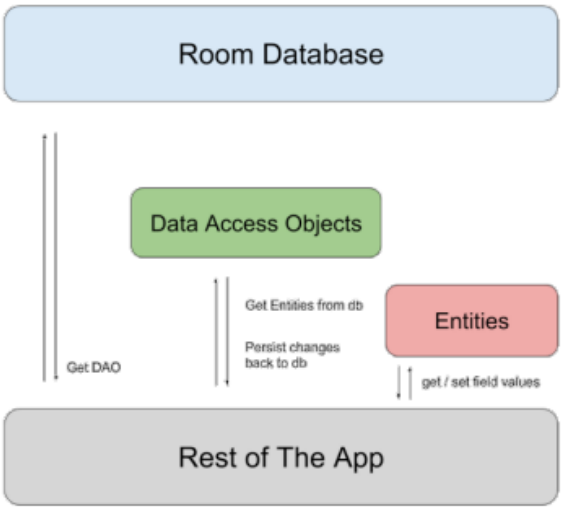


I ViewModel delegano il data fetching a un Repository, in questo modo l'app (ovvero il ViewModel stesso) non conosce la fonte dei dati. La room consente un accesso semplificato al database. Per crearla serve:

- Entity: la classe che rappresenta l'entità
- DAO (Data Access Object): interfaccia per query
- Database: classe astratta che funge da punto di accesso.

L'app quindi utilizza:

1. La Room per ottenere i DAO.
2. I DAO per ottenere le Entity.
3. Le Entity per ottenere i valori per il database.



Lezione 10 - View Binding, Data Binding, Service

23 Novembre 2022

View Binding

View Binding permette di scrivere codice più semplice per interagire con le view: genera una classe java di binding per ogni file di layout XML. I vantaggi, oltre la semplicità del codice, è la null safety e la type safety.

Data Binding

Data Binding lega i componenti della UI alle loro fonti dati: invece di accedere ai componenti a livello programmatico, assegniamo il testo direttamente nel file XML.

Services

Un service è un componente che una app definisce quando:

1. deve eseguire operazioni di lunga durata in background.
2. vuole fornire funzionalità ad altre app.

Sono state inserite limitazioni affinché si evitano spreco di risorse. Infatti, per quanto riguarda le operazioni in background, sono sempre meno in uso a favore dei WorkManager che lavorano inground. Restano molto utili per fornire funzionalità ad altre app.

Un service non crea un proprio thread ma gira in quello del suo processo hosting. Allora si appoggia a un processing asincrono per evitare errori Application Not Responding.

Un service quindi non è un processo separato e non è un thread.

Ci sono 3 tipi di service:

- Foreground: operazioni che sono evidenti all'utente. (WorkManager preferibile).
- Background: operazioni che non sono notate direttamente dall'utente. (WorkManager preferibile).
- Bound: offre interfaccia client-server che permette ai componenti di interagire con il servizio.

Anche i Service hanno il loro lifecycle, uno attraverso il startService() per il ground, e il bindService() per il bound. Un servizio può essere sia started che bound.

L’intent consegnato con startService() è l’unico mezzo di comunicazione tra componente e service. Se la comunicazione deve essere bidirezionale, occorre usare un PendingIntent.

Quando uso un foreground service devo avere una notifica nella status bar.
Per il passaggio di parametri si usa il pattern Builder.

Lezione 11 - WorkManager, Testing, Firebase

30 Novembre 2022

WorkManager

In generale è opportuno eliminare qualsiasi attività bloccante dal thread principale della UI.

Il task in background rientra in una delle tre categorie:
- immediato, deve essere eseguito e completato subito.
- di lunga durata, richiede tempo per completare.
- differibile, non deve essere eseguito subito.

Inoltre possono essere persistenti e non persistenti, anche se di lunga durata e differibili non è consigliata la seconda modalità.

Per tutti i persistenti si usa il WorkManager, persistenti immediati si usa il threading.

Un task è definito utilizzando la classe Worker, definendo cosa fare. Il WorkRequest definisce quando e come il task deve essere eseguito. Il quando scegliendo tra OneTimeWorRequest e PeriodicWorkRequest. Il come includendo informazioni e vincoli riguardo al task.

Si può personalizzare la WorkRequest sotto diversi aspetti:
1. Si può specificare che un task parti solo se rispetta delle date condizioni.
2. Si può specificare che un task parta dopo un tempo prestabilito.
3. Si può specificare che un task riprova ad eseguire.
4. Si può specificare che un task gestisca input ed output, gestendoli come coppie chiave-valore di tipo Data.
5. Si può specificare che un task sia raggruppabile in modo logico assegnando un tag.
6. Si può specificare una work chain che specifica una relazione ordinata tra task in quanto alcune task possono dipendere dal buon esito di altre.

Testing

Il testing dell'app è parte integrante del processo di sviluppo in cui si verifica la correttezza, il comportamento funzionale e l'usabilità dell'applicazione. Ci sono diversi tipi di test:

- Funzionali: la mia app fa quello che deve fare?
- Prestazionali: lo fa in modo rapido e veloce?
- Accessibilità: funziona bene con i servizi?
- Compatibilità: funziona bene su ogni dispositivo e livello di API?

I test variano in base alla loro dimensione:

1. I test di unità verificano solo una porzione molto piccola di codice.
2. I test di integrazione verificano più di un componente.
3. I test end-to-end verificano il funzionamento del sistema in diversi scenari di utilizzo.

I test UI verificano il funzionamento dell'interfaccia, eseguibili all'interno dei tre citati prima in base alle esigenze.

Possono essere testati in 4 contesti diversi:

1. dispositivo fisico: massima fedeltà ma più tempo per eseguirli
2. virtuale: equilibrio tra fedeltà e velocità
3. simulato: più veloci ma meno fedeli
4. macchina di sviluppo: ideali quando non occorrono features del framework Android.

Inoltre possono essere locali o instrumented.

		Localizzazione		
		local		instrumented
		workstation (JVM)	Simulatore	Device/Emulatore
Dimensione	unità (small)	funzionalità di base che **non** richiedono interazione con il framework Android	funzionalità che **richiedono** interazione con il framework Android (sufficiente in modalità simulata)	funzionalità che **richiedono** interazione con il framework Android
	integrazione (medium)	componenti indipendenti dal framework Android	componenti che **dipendono** dal framework Android (sufficiente in modalità simulata)	componenti che **dipendono** dal framework Android
	end-to-end (big)	x	flussi che **dipendono** dal framework Android (sufficiente in modalità simulata)	flussi che **dipendono** dal framework Android

Firebase

Firebase è un backend as a service che permette di:

- far girare il proprio backend senza gestire i server.

- risolvere facilmente i problemi comuni.
- scalare senza difficoltà per supportare molti utenti.

Ci sono due modalità di database:

- Realtime Database è il database originale di Firebase: è una soluzione efficiente e a bassa latenza per le app mobili che richiedono stati sincronizzati tra i client in tempo reale.
- Cloud Firestore è il più recente database di Firebase: fornisce nuovo modello di dati più intuitivo ed è dotato di query più ricche e veloci ed è più scalabile di Realtime Database.

Altre caratteristiche di Firebase:

- Semplifica la creazione di sistemi di autenticazione sicuro migliorando al contempo l'esperienza di accesso e di registrazione per gli utenti finali.
- Migliora la qualità delle app in meno tempo e con meno sforzo.
- Ottimizza la user experience e rende felici gli utenti.