Programmazione Dispositivi Mobili

Sara Angeretti

2025/2026

Indice

Capitolo 1

Introduzione al corso

1.1 Organizzazione

In più rispetto all'ultimo semestre:

- "gettoni presenza" per punti bonus
- 15 ottobre: conferenza azienda (BendingSpoons) su iOs

1.2 Obiettivi del corso

Il corso ha come obiettivo acquisire:

- **Conoscenze** (principi di buona programmazione) relative al mondo dello sviluppo mobile
- **Competenze** sullo sviluppo Android

Ma perché è stato scelto proprio Android? Comporta diversi vantaggi rispetto ad altri sistemi operativi come iOs:

- più open source
- essendo più open source conosciuto meglio dai docenti che sono quindi più in grado di insegnare e correggere
- Android, nel caso uno voglia poi accedere allo Store e pubblicare un'app, prevede una tassa di iscrizione di ~25$ **una tantum**, mentre per iOs è di 100$ ma penso sia *annuale*.

Alla fine del corso dovremo essere in grado di:

- Sviluppare un'applicazione "from scratch" che segua l'**architettura** di riferimento Android
  - alla fine se abbiamo rispettato o no l'architettura presentata a lezione è quello che guardano di più della nostra app, se non è bellissima o funzionante al 101% importa meno
- Comprendere il funzionamento di applicazioni Android

1.3 Il corso in pillole

1. **Introduzione** alla progettazione e allo sviluppo di applicazioni mobili
2. **Linee guida** sull'architettura dell'app
3. **Sviluppo** di un'app in Java
   Per il nostro progetto possiamo usare Java o Kotlin, la teoria rimane la stessa, ma a lezione useremo solo Java. Questo perché è già stato presentato ed usato in altri due corsi e quindi conosciuto meglio da docenti e studenti. Inoltre, è previsto (penso in entrambi i linguaggi) l'uso di lambda functions, più elegante e funzionale in Kotlin, però buono anche in Java. Infine, Java risulta più conveniente per l'uso di librerie esterne di Kotlin, che è più giovane e meno conosciuto e quindi ha meno librerie disponibili.
   Eventualmente, sul sito Google ci sono disponibili diversi tutorial gratuiti (video) per imparare Kotlin e per la migrazione del mio progetto da Kotlin a Java.
   Se si vuole utilizzare Kotlin, c'è da scrivere alla professoressa spiegando le ragioni della scelta.
   L'app deve essere robusta. La robustezza si basa sull'autonomia dalla connessione di rete. Il concetto "offline-first" è molto importante, prima di tutto l'app deve funzionare senza connessione. Inoltre, deve essere anche evolvibile. Oltre ai suoi componenti funzionali (ovvero le sue funzionalità) di base, ci sono determinate funzionalità che devono essere mantenute ed evolute/ampliate nel tempo.
   La nostra app deve essere:

   3.1 Compliant con l'architettura di riferimento
   La cosa importante della nostra app non è l'estetica o se funziona bene, ma **come** l'architettura presentata a lezione viene sviluppata.

   3.2 Con UI

   3.3 Che accede alla rete per i dati (API esterne)

   3.4 Che fa persistenza (locale + remoto)
   Ovvero deve funzionare *localmente* e salvare localmente i dati (importante per il concetto di "*offline-first*"). Però deve anche salvare *in remoto* i dati, ma deve rispettare la *cross-device synchronization.* Importante per quanto riguarda la cross-device synchronization, ovvero deve funzionare su diversi dispositivi con stato sincronizzato.

   3.5 Che usa Firebase
   Firebase è un framework di Google che offre una serie di servizi per lo sviluppo di applicazioni mobili.

1.4 Il progetto

- Durante il corso si porta avanti un progetto che sarà oggetto dell'esame finale
- Il progetto è svolto in maniera paritaria da un gruppo di studenti

  - Composto da almeno 3 persone e fino ad un massimo di 5
  - Eccezioni verranno valutate singolarmente
- Il progetto è proposto dal gruppo (scadenza 17 ottobre 2025)
  - L'idea può anche essere non nuova, ad esempio un'app che mostra i film e le recensioni...
- Il progetto deve essere sviluppato per Android (in Java)
- Assistenza al progetto (oltre che ai laboratori durante i quali lavoreremo sul **nostro** progetto) verrà fornita *eventualmente* anche durante le esercitazioni (durante le quali vedremo come realizzare un'applicazione da zero tramite un *progetto* scelto dal docente con le proprie funzionalità e sviluppato **esattamente** come vuole la prof che sia realizzata la nostra app)
- Si suggerisce caldamente per chi voglia sfruttare l'appello invernale per sostenere l'esame di seguire i laboratori perché verrà fornita assistenza.
- L'app deve **NECESSARIAMENTE** avere:
  1. architettura
  2. almeno 1 API esterna
  3. autenticazione se necessario
  4. off-line first (deve funzionare fluidamente anche in assenza di connettività)
  5. cross-device synchronization
  6. testing
  7. layout grafico (importante ma meno del resto)
  8. storage locale (opzionale: se necessario, ma ne va giustificata l'assenza in caso)
  9. documentazione di buona qualità (ci sono molti esempi sul drive presente su Moodle)
- Bisogna usare GitLab o GitHub (non per forza ma una settimana prima della consegna va fatto quindi tanto vale)

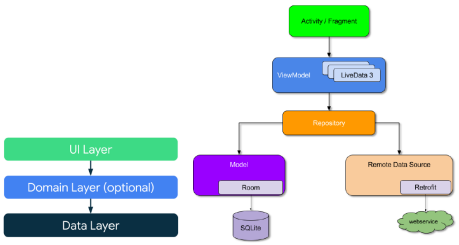
Valutazione

- Codice sorgente e documentazione della nostra app con dentro:
  1. Scelte architetturali
  2. Uso di API esterne (in maniera sensata)
  3. Uso storage locale, per offline-first
  4. Uso del framework Firebase (autenticazione), per cross-device synchronization
  5. Layout grafico che sia (circa) sensato, per es. tramite uso librerie di material design

6. Documentazione

- Punti 1-5 chiaramente via Software, 6 cartaceo
- Uso di GitLab o GitHub (da aggiungere account fornito dai docenti)

1.4.1 1. Scelte architetturali



Queste 5 entità **devono** essere presenti nel nostro progetto.

1.4.2 2. Uso di API esterne

Esempi di API esterne (gratuite):

- Immagini
  - `https://unsplash.com/developers`
  - `https://developers.google.com/maps/documentation/places/web-service/photos`
- Video Giochi
  - `https://www.igdb.com/api`
- Film
  - Film
  - MovieDB
  - Open Moview DB
  - IMDb
  - The Moview DB
- Qualità dell'aria
  - `https://aqicn.org/`
- Cibo
  - TheMealDB.com
  - `https://spoonacular.com/food-api`
- Meteo
  - `https://openweathermap.org/current`

- ...

Tante disponibili free

- `https://github.com/public-apis/public-apis`

Postman è un'applicazione che permette di testare le API, di interrogarle, è molto utile per vedere come funzionano le API e come si comportano prima di integrarle. `https://www.postman.com/`

1.4.3 3. 4. 5. Uso di Firebase

Strumento molto potente, lo vedremo ad esercitazione. Ti aiuta a creare:

- Autenticazione
  - se necessario nell'app
- Database remoto
  - Anche a supporto della cross-device synchronization
- Notifiche push
- ...

1.4.4 6. Testing

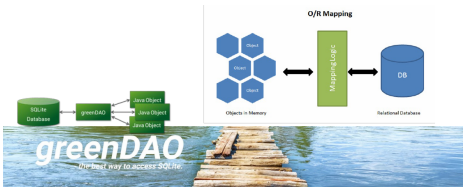
- JUnit 4/5
- Mockito/MockK
- Espresso (per test UI, con un input di testo e comandi per interagire con l'UI) !!!
- Robolectric
- Firebase Test lab
- UI Automator (quella che mi permette di fare il cross-app testing) !!!

Quelli con "!!!" sono quelli evidenziati dalla prof.

1.4.5 7. Layout grafico

Non una delle cose essenziali ma comunque importante, l'app deve comunque essere user-friendly.

Ho delle regole da seguire, il progetto deve seguire i principi di material design (che sono linee guida di Google per lo sviluppo di UI, ma ci sono anche widget grafici, gratuiti ed usabili). Per esempio: color extraction (tutti i widget hanno colorazione uniformata allo sfondo per conferire maggiore fluidità).



1.4.6 8. Uso ORM - Object Relational Mapping

Uso di ORM (Object-relational mapping). Due tool sono:

- Room (integrato in Android), utile per persistenza
- greenDao, più vecchio ma ancora in uso

1.4.7 Documentazione

La documentazione dovrà essere strutturata in 3 sezioni:

1. le **funzionalità offerte**
    - potete scriverle in italiano, oppure usare degli use case
2. l'**architettura complessiva** che specifica i componenti da voi sviluppati e le loro relazioni e modalità di comunicazione con eventuali componenti esterni utilizzati
    - Firebase, API esterne, DB interno, etc.
3. il **design della soluzione** che include i *componenti* che avete sviluppato (inteso come Fragment, Activity, etc.), le loro *responsabilità* (cioè cosa fanno) le loro *modalità di interazione*

Alcuni esempi, sul drive fornito dalla prof su Moodle.

1.4.8 Scadenze

- 10 ottobre 2025: comunicazione gruppo
    - la composizione del gruppo (matricola, cognome, nome), un nominativo per riga;
    - il referente del gruppo.
- 17 ottobre 2025: comunicazione argomento progetto
    - Il titolo del progetto;
    - una descrizione del progetto che si intende svolgere;
    - il referente del gruppo.

Capitolo 2

Introduzione ai dispositivi mobili

2.1 Breve storia

3 aprile 1973, NYC, di fronte a giornalisti: telefonata fra Joel (Motorola) e Marty Cooper (AT&T), per una "guerra" su chi avrebbe progettato il primo cellulare. Venne chiamato DynaTac.

Fu la puntata del 29 dicembre 1967 di Star Trek con l'Enterprise e il capitano Kirk ad ispirare Marty Cooper per lo sviluppo del primo telefono mobile.

2.2 Perché Android?

Un motivo si riscontra nel mercato dei SO disponibili (al 2025): iOS circa 28%, Android circa 71% (KaiOS e altri <1%). Perciò Android è il target su cui concentrarsi (almeno per quanto riguarda il nostro corso).

Cos'è KaiOS? è un sistema operativo (anche lui basato su Linux) per dispositivi specifici, per feature phones, destinati al mercato per fasce di popolazione con meno disponibilità economica. Per questo motivo sono nati i *feature phones*, una via di mezzo fra i *dumb phones* (telefoni a conchiglia e con tastiera fissa esterna tipo Nokia 3310, precedenti agli smartphones) e gli *smart phones* come li conosciamo noi, di cui hanno le principali funzionalità.

2.3 A cosa servono i telefoni cellulari?

Non si parla più di fare solamente telefonate, ora le tecnologie mettono a disposizione sempre più servizi utili.

2.4 Ma quante sono le app disponibili?

Passando da Agosto 2024 a Giugno 2025, si nota un decremento: gli stores fanno un po' di pulizia e eliminano quelle note come *app zombie*, presenti ma che non fanno niente quindi passibili di cancellazione.

Ma il calo nel **Google Store** si deve anche ad altri motivi:

- politiche di rimozione dello store
- sviluppatori più inclini a migrare a iOS (c'è l'impressione che gli utenti spendano di più che sulle app Android)
- duplicazione delle app su Android (tipo quando l'aggiornamento/improving viene rilasciato come una seconda app distinta invece di un aggiornamento)
-
- app web e alternative nel play store (es. Google Store non è l'unico store disponibile per Android, inoltre è possibile scaricare app anche senza passare dallo store)

Comunque quando vado a sviluppare un'app mi conviene vedere dove, in quali ambiti ci sia più **presenza** (non download) che significa più richiesta.

2.5 Mentre quelle scaricate?

2.6 I guadagni

Su Android la distribuzione è: 96.95% free apps, 3.05% paid apps.
Su iOS la distribuzione è: 95.41% free apps, 4.59% paid apps.

2.7 Sintesi

2.8 Challenge nello sviluppo di app mobili

2.8.1 Caratteristiche che un'app deve avere

Un'app deve essere:

- semplice
- economica
- addicting, chi la scarica deve volerla tenere

2.8.2 Sfide per un programmatore

2.8.3 Idea & business

Da dove nasce un'app di successo?

- Un **bisogno** reale da soddisfare (ma che non sia solo una cosa utile a me, ma a più persone)
- Un **problema** quotidiano da semplificare (es. app che esistono già ma non)

- Un'**esperienza utente** migliore rispetto a soluzioni già esistenti

Non basta "avere un'idea": serve capire **perché** qualcuno la userebbe.
Suggerimenti:

- Effettuare **analisi di mercato**: deve essere un'app che l'utente vuole.
- Renderla **attrattiva**: in modo che l'utente decida di tenerla.
- Renderla **semplice** da usare.

Quanto costa sviluppare un'app?
Chiaramente non è solo "scrivere codice", è un investimento continuo.

- **Costi diretti** (fondamentali): sviluppo vero e proprio (Android/iOS), backend, infrastruttura cloud.
  - Quale piattaforme l'app intende supportare?
    - Più piattaforme → costi più alti
    - Fortunatamente non sempre proporzionali, grazie al *riutilizzo di codice*
    - Prima si realizza l'architettura, poi si pensa ad una piattaforma, poi si passa ad altre eventualmente
- **Costi indiretti** (non incidono direttamente sullo sviluppo ma devo tenerne conto comunque): design UI/UX, testing, aggiornamenti, manutenzione.
  - **Account dello sviluppatore** (developer account)
    - *Apple* abbiamo detto costa 99$ all'anno per account individuali mentre 299$ all'anno per gli Enterprise accounts (aziendali)
    - *Google* costa 25$ una tantun per un account sviluppatore
  - Poi abbiamo **componenti server-side e servizi Cloud**
    - Se l'app richiede di archiviare e recuperare dati, allora occorre aggiungere costi per:
      - archiviazione
      - hosting
      - nome di dominio
      - pagine di destinazione
      - backup dei dati
      - etc...
    - Se l'app utilizza anche altri servizi di cloud a pagamento allora occorre aggiungere
  - Altri costi indiretti sono legati alla **manutenzione**
    - Evitare che diventi un'app zombie, perciò l'app è da mantenere:
      - M
- **Range molto variabile**: da poche migliaia a centinaia di migliaia di euro.

Alcuni suggerimenti:

- Trovare

Come guadagnano le app?

Monetizzare un'app significa implementare strategie che permettano di generare

Scelta cruciale da definire ***prima*** di sviluppare: influenza design e architettura.

Processi strategico che deve tenere conto del comportamento degli utenti, delle tendenze di mercato e delle specificità dell'app.

Due modelli principali:

- In-app purchase
  - Paga l'utente
  - Se si diverte, può essere propenso ad **acquistare** al suo interno (tipicamente giochi)
  - Cosa si può acquistare?
    - Sbloccare nuovi livelli o contenuti
    - Scambiare beni e servizi virtuali
    - Ottenere capacità più avanzate
    - Più tempo di gioco o più vite

    Secondo Forbes (una rivista autorevole), le app con acquisti in-app generano il maggior fatturato fra tutte le app.
- Advertising
  - Sono gli inserzionisti a pagare
  - 3 tipi:
    - Advertising - Banner
      - Occupano poco spazio
      - Meno invasivi
      - Gli utenti possono visualizzarli senza interrompere l'attività nell'app
    - Advertising - Annunci interstiziali
      - Occupano tutto lo schermo, a schermo intero
      - Vengono visualizzati in momenti specifici
      - Gli utenti possono chiudere questa pubblicità tramite un pulsante di chiusura (che di solito appare dopo un certo intervallo di tempo o dopo almeno un'interazione) posizionato in un angolo (sx o dx)
      - Devono essere visualizzati al momento giusto per evitare di disturbare troppo l'utente (ad es.: in un gioco al termine di un livello)
    - Advertising - Video rewarded
      - Breve video, di solito 15-30 secondi, che l'utente **sceglie** di vedere in cambio di una ricompensa

        - Usati ad esempio nei giochi per premiare con crediti in-app, vite, . . . , in cambio di un breve video
        - Collocato in un punto in cui l'utente non riesce a progredire nel gioco ed essere più suscettibile ad optare per la visione del video
        - Video pbblicitari sono più efficaci quando
    * Advertising - Native
        - La pubblicità nativa si presenta come una naturale continuazione dei contenuti e non come una rottura, sia da un punto di vista visivo che tematico
        - "Come camaleonti, si adattano alla forma dell'app che li contiene" circa
        - Gli utenti prestano così la propria attenzione online in modo più spontaneo
        - Meno invasivi
        - Profitto maggiore: quando l'utente va ad approfondire il contenuto

Esistono poi le cosiddette ***Freemium app***:

- prevede due o più varianti del prodotto da distribuire a prezzi diversi
- di solito due varianti:
  - *versione base gratuita*
  - *versione premium* a

Esistono poi le cosiddette ***Subscription app***:

- Gli utenti *pagano un canone periodico* per l'uso dell'app
- Gli utenti si aspettano quindi di ricevere più di quanto offra una (once for all) paid app: le app in abbonamento devono fornire continuamente ?
- Es. tipici: Corriere della Sera, servizi streaming tipo Netflix, Disney+, Prime Video...

Esistono poi le cosiddette ***Purchase-app-once*** (anche dette ***paid app***):

- Quasi il 95% delle app che si possono scaricare dagli store sono *gratuite*...
- Gli utenti si aspettano quindi di ricevere più di quanto offra una (once for all) paid app: le app in abbonamento devono fornire continuamente ?
- Es. tipici: Corriere della Sera, servizi streaming tipo Netflix, Disney+, Prime Video...

2.8.4 Sfide tecniche

cose che ho perso

2.8.5 UI/UX cme fattore critico

L'esperienza utente è il primo

Navigabilità

- chiarezza dei percorsi
- accessibilità per utenti con diverse abilità
- ergonomia: ridurre tap superflui, gesti naturali

Testing: le sfide

- Frammentazione OS e dispositivi
  - Esistono differenti S.O. con differenti versioni
  - Senza verifiche cross-platform, si ferdono grossi bacini di utenze
  - Suggerimenti:
    * Occorre verificare che la nuova app sia supportata da diversi S.O. e le due versioni precedenti
    * Infrastrutture
- Rilascio frequente del S.O.
  - I S.O. continuano a evolvere
    * Sia Android che iOS hanno più di 10 versioni dei loro s.o.
    * Continuano a migliorare e aggiornare le loro versioni per migliorare le prestazioni e l'esperienza utente
    * Sen
- ergonomia: ridurre tap superflui, gesti naturali

Capitolo 3

Costo Sviluppo Mobile

Diversi fattori influenzano il costo di sviluppo di un'applicazione mobile, tra cui:

- piattaforma target
- caratteristiche
- design
- eventuale infrastruttura aggiuntiva
- il team di sviluppo (non per forza uno unico per diverse piattaforme, es. uno per Android, uno per iOs)

3.0.1 Fattore #1: Piattaforma Target

La scelta della piattaforma target è uno dei fattori più importanti che influenzano il costo di sviluppo di un'applicazione mobile. Più piattaforme saranno supportate e più alto sarà il costo. Devo aver ben presente a quale mercato voglio indirizzare il mio prodotto.

Fortunatamente, non è sempre proporzionale al numero di piattaforme supportate, grazie al **riutilizzo del codice**.

Prima si realizza su una piattaforma e, una volta creata l'architettura (che vado ad implementare per la mia piattaforma) e validata l'idea, si può replicare su altre piattaforme.

3.0.2 Fattore #2: Obiettivi e modello di sviluppo

Determinare gli **obiettivi di business** e quindi *cosa* dovrà fare l'app, gli obiettivi che deve raggiungere. Es.: tutto ciò che un utente vuole fare senza essere obbligato a stare bloccato davanti ad un computer, un browser. Es.: l'app di una banca.

Se sbaglio, butto via un sacco di soldi. Per questo è legato ai costi.

È necessario creare un **documento di specifiche tecniche** che elenchi le caratteristiche che l'app avrà.

Devo decidere il modello di sviluppo:

- Set fisso di caratteristiche

- Set dinamico di caratteristiche
- Mix: si inizia con una serie di requisiti fissi, ma i clienti hanno una certa flessibilità nel poter decidere di cambiare qualcosa

Chiaramente il meglio è l'ultimo perché si ha una certa flessibilità, se facessi un set più fisso rischierei di non soddisfare le esigenze del cliente e dover buttare via tutto e quindi aumentare i costi. Ci serve un approccio "*agile*", ovvero un approccio che permetta di cambiare le cose in corso d'opera. Vado avanti a pochi obiettivi alla volta (*"user stories"*, per dire l'autenticazione, creazione di un bonifico, vedere i movimenti, sono tutti esempi di user stories), con un ciclo di sviluppo molto breve, e poi passo al prossimo obiettivo. Facendo poco alla volta posso affrontare i miei *debiti tecnologici* (ciò che devo studiarmi man mano perché non conosco) e sentirmi con gli *stackholder* (chi ha interesse nel progetto) per capire se sto andando nella direzione giusta.

3.0.3 Fattore #3: Design

Il design delle app (sia UI (come sono i bottoni, i tasti...) che UX (come l'utente riesce a navigare, a sfruttare le capacità dell'app)) è ciò che separa le buone app da quelle *amazing*.

- **Design classico:** con cui gli utenti hanno più familiarità, meno costoso (es. il design classico di Apple per le applicazioni iOS)
- **Design impressive:** più costoso, richiede più tempo e risorse, ma può fare la differenza

3.0.4 UI & UX

Diversi ruoli, diverse skills e competenze. Ha nominato Figma che aiuta a "disegnare/creare" i prototipi delle app.

3.0.5 Fattore #4: Come Sviluppare?

Che strumenti uso? Anche questa scelta incide sui costi. C'è nelle slide una lista di piattaforme di sviluppo di applicazioni mobili, di strumenti cross-platform e di sviluppo nativo.

Ha nominato **Flutter** come strumento cross-platform, che permette di scrivere una volta sola il codice e poi eseguirlo su diverse piattaforme. È la cosa più vicina a sviluppo nativo però (se ho capito bene).

3.0.6 Fattore #5: Caratteristiche dell'app

Le caratteristiche dell'app sono il fattore determinante per il costo dell'app stessa. Con l'aumentare del numero e della complessità delle funzioni della app, aumenta anche il costo di sviluppo.

Es.: è una to-do list app? Poche semplici funzionalità. Include mail e autenticazione? Richiede l'uso del GPS? Già diverso. Etc.

3.0.7 Fattore #6: Infrastrutture

Un'app che si appoggia ad un componente remoto ha costi di sviluppo più alti di una "off-line".

È necessario prendere in considerazione, tra le altre cose:

- configurazione del server
- requisiti di memorizzazione
- crittografia e sicurezza dei dati
- comunicazione con l'app
- gestione degli utenti
- ...

3.0.8 Fattore #7: Altri costi oltre a quelli di sviluppo

Oltre ai costi di sviluppo, ci sono altri costi da considerare:

- **Account dello sviluppatore (Developer Account):** c'è la slide
- **Componenti server-side e servizi Cloud:** c'è la slide
- **Manutenzione dell'app**:
  - Molte app zombie
  - Se non si vuole - guarda la slide

Grafico di proiezione del guadagno da app: è in crescita, questo anche perché (come ha mostrato uno studio) un utente tende a preferire un'applicazione ad un sito web. Perciò se ho sia un'app che una web app che svolgono gli stessi compiti, l'utente tenderà a preferire l'applicazione.

***: App zombie**

Sono app non gestite. Avevamo parlato di una curva grafico che mostrava quante app sono state eliminate dall'Android App Store, questo era successo anche per la gran quantità di app non gestite o mantenute nel tempo che sono state tirate giù.

3.1 Monetizzazione

Monetizzare un'app significa implementare strategie che permettano di generare, al proprietario dell'app, entrate attraverso il suo utlizzzo. Slide

3.1.1 Purchase-app-once (paid app)

Quasi 95% delle app sono gratuite. Tuttavia ci sono utenti che **pagheranno** per applicazioni di **qualità** che soddisfino un bisogno molto specifico, da pochi centesimi a centinaia di euro.

3.1.2 Freemium app

Prevede due o più varianti del prodotto da distribuire a prezzi diversi. Di solito due varianti:

- **Versione Base:** gratuita
- **Versione Premium:** a pagamento, include funzioni e/o contenuti aggiuntivi (es.: sblocco livelli) o rimuove pubblicità

Filosofia: dare un

3.1.3 Subscription app

Gli utenti **pagano un canone** periodico per l'utilizzo della app. Funziona bene per app che:

- si basano su un *servizio di backend*
- forniscono l'*accesso a contenuti aggiornati costantemente*

Gli utenti, pagando un canone, si aspettano di ricevere più di quanto offra una paid app. Le app in abbonamento devono fornire continuamente nuovi contenuti e/o funzioni.

3.1.4 In-app purchases

Ho perso un paio di slides.
Due tipologie:

- **in-app purchases:** l'utente acquista di sua volontà beni o pacchetti all'interno dell'app.
  Cosa si può acquistare?
  - Sbloccare nuovi livelli o contenuti
  - Scambiare bene o servizi virtuali
  - Ottenere capacità più avanzate
  - Più tempo di gioco o più vite

  Secondo Forbes, le app con acquisti in-app generano il maggior fatturato fra tutte le app.
- **Advertising:**
  - banner
  - interstitial
  - incentivized rewarded video
  - native

Banner

Sono annunci pubblicitari che occupano poco spazio, appaiono nella parte superiore o inferiore dello schermo dell'applicazione senza interromperne l'attività. Sono meno invasivi rispetto ad altri tipi di pubblicità.

Interstitial

Sono annunci pubblicitari a schermo intero che appaiono in momenti chiave dell'esperienza dell'utente, come ad esempio durante il cambio di livello in un gioco o durante la navigazione tra le pagine di un'app.

Posso chiuderlo con un apposito pulsante in alto a destra o a sinistra.

Devono essere visualizzati al momento giusto oer evitare di disturbare troppo l'utente.

Video rewarded

Sono brevi video di circa 15 secondi al massimo che l'utente Slide

Native

La pubblicità nativa si presenta come una naturale contenuazione dei contenuti e non come una rottura, sia da un punto di vista visivo che tematico.

Gli utenti. Slide

3.1.5 Piattaforme Google per la monetizzazione

AdSense è una piattaforma di Google per monetizzare soprattutto i siti web.

AdMob è una piattaforma che consente agli sviluppatori di monetizzare le app mobili attraverso la pubblicità. Funzionamento in breve (vediamo questo ma più o meno funzionano tutte così):

- Si basa sull'integrazione di annunci pubblicitari
- slide

Funziona sia per Android che per iOS.

AdMob: funzionamento

1. Integrazione dell'SDK (Software Development Kit) di AdMob nell'app
    - gli sviluppatori
2. Tipologie di annunci supportati
3. Mediazione degli annunci (**Ad Mediation**)
    - Permette
4. Targeting degli annunci
    - **Targeting contestuale:** gli annunci vengono mostrati in base al tipo del contenuto dell'app
    - **Targeting demografico:** gli annunci vengono ottimizzati in base alle informazioni demografiche degli utenti come età, sesso e posizione geografica
    - **Targeting comportamentale:** AdMob utilizza i dati di navigazione degli utenti

5. Asta automatica (Ad Auction)
    - ad ogni opportunità di visualizzare
6. Analisi e reportistica
    - AdMob fornisce agli sviluèèatori strumenti di analytics per monitorare le prestazioni degli annunci e comprendere come stanno generando entrate.
    -
7. Pagamenti
    - Gli sviluppatori

3.1.6 Modelli di guadagno

Slide

CPC - cost per click

Il publisher
...
Vantaggi

CPM - cost per mille

Il publisher
...
Vantaggi

CPA - cost per acquisition

Il publisher
... azione specifica tipo scaricare un'altra app
Vantaggi

Capitolo 4

Esercitazione 1

4.1 Passi su Android Studio

- New Project
- Empty Views Activity (le altre sono già pronte per fare cose specifiche, non ci interessa)
- package name usato per identificare univocamente l'app, di solito com.nomeorganizzazione.nomeapp (noi com.unimib.nomeprogetto)
- SDK librerie standard da cui partiamo per il progetto; le più recenti hanno le funzionalità più nuove ma rischiamo di lasciare fuori una fetta di pubblico (facciamo tipo Android 30).
- Scegliamo linguaggio Java (l'ultima voce la lascia così com'è)
- una volta avviato nella lista di cartelle e file annidati mi interessano `MainActivity.java` e `activity_main.xml` che è la struttura grafica

4.1.1 Usare Android Studio

Una volta avviato, nel main del java mi interessa tenere il super che richiama il costruttore e la navigazione al corrispondente layout.

4.2 Git

Da terminale (seconda icona dal basso a sinistra) possiamo scrivere:

1. `git init` per inizializzare una repository (fatta dentro perché ci lavoro dentro Android Studio e non da VSCode come faccio con tutte le altre repository)
2. Stato della repository: `git status`
3. `add` e `commit` per aggiungere quanto inizializzato
4. `levels` per vedere i livelli di git

    - da prompt dei comandi `dir` per vedere la lista di directory e file, ma forse è un livello solo, NON LO SO CONTROLLA. `dir` su Windows, `ls -a` su Linux.

5. `git commit` per fare un commit
6. `git branch nomebranch` per creare un nuovo branch
7. `git checkout -b nomebranch` ne crea uno nuovo
8. `git checkout` si aspetta il nome del branch
9. `git checkout nomebranch1` per passare al branch nomebranch1
10. `git checkout nomebranch2` per passare al branch nomebranch2
    Se faccio commit su modifiche non pushate, rischio di andare incontro a conflitti.
    L'asterisco (sul sito `https://learngitbranching.js.org/`) mi indica su quale branch mi trovo.
11. `git merge nomebranch` per fare il merge in nomebranch del branch in cui mi trovo (quello attivo in quello più stabile)
12. `git rebase nomebranch` per fare il rebase in nomebranch del branch in cui mi trovo: è un'alternativa a merge, "riscrive la storia", collassa tutti i branch in uno cancellando la storia di quello in cui mi trovo e mettendola in quello in cui voglio fare il rebase.
13. `git revert nomedelcommit` per fare il revert di un commit, tornare al commit e cancellare le modifiche da quel commit in poi, magari per eliminare branch che ho fatto inutilmente.

GitHub sul suo sito ha una sezione (`https://docs.github.com/en`) con una serie di guide su come usare git.

4.2.1 Sito per simulare funzionamento di git

`https://learngitbranching.js.org/`

4.2.2 Comandi git da terminale

Le commit, partendo dalla cartella del progetto:

- `git pull`
- `git add .`: mette tutto ciò che c'è nella working area nella staging area, il punto mi dice "tutti i file"
- `git commit -m "messaggio in cui dico cosa ho fatto"`: mette tutto ciò che c'è nella staging area nella commit area
- `git push`

4.2.3 I conflitti

In caso di conflitti, per arrivare alla pagina di risoluzione dei conflitti (al centro versione locale, a sinistra un branch e a destra l'altro), quando si verifica un conflitto esce un pulsante "risolvi conflitti" magari in inglese. Tendenzialmente il main è la versione più stabile, quindi si fa il merge del branch in cui si è lavorato in main.

Consiglia di fare branch per funzionalità e non per persona, però è un approccio personale e comunque l'importante è essere consistenti.

Git Ignore è un file che va ad appuntare estensioni di file o file o cartelle da ignorare. Al momento dell'inizializzazione del progetto avremo già una lista di file di base ignorati. Di solito i file compilati (es su Java i `.class`) sono da ignorare.

Il sito `gitignore.io` permette di generare un file `.gitignore` in base al linguaggio di programmazione che si sta utilizzando con i file da ignorare. Metto Android, Windows. Consiglia di farlo all'inizio anche per alleggerire la compilazione.

GitHub e GitLab sono istanze di git, mentre git è il programma generico.

4.2.4 Il progetto

Prima cosa da fare: avviare la repository, il gitignore, il README. Decidere come gestire i branch (non valutato, ma l'approccio deve essere consistente) se per team o funzionalità o cosa, vedi online. Dice che ci dovrebbero essere consigli. Meglio tenere il main branch come quello che funziona di più.

Per creare una nuova repository su GitHub:

- scelgo nome e descrizione
- scelgo se pubblica o privata (per il progetto può anche essere privata perché tanto posso scegliere dopo chi può accedervi)
- posso scegliere se aggiungere un README (che è un file markdown che appare nella home della repository) e un `.gitignore` (che posso scegliere in base al linguaggio di programmazione che sto usando; il punto davanti mi dice che è un file nascosto), ma sono cose che posso aggiungere anche dopo (l'idea è nascondere almeno i file temporanei o le build (es. gradle) lasciando solo il codice sorgente). GitHub mi dà la possibilità di aggiungere un solo gitignore (es. Android).
  C'è un sito () dove puoi aggiungere diversi tag (Android, Windows, etc...) e mi genera il .gitignore con TUTTO quello che ho detto nei tag.
- posso scegliere se installare la repository tramite set up desktop, da linea di comando (sono i comandi che abbiamo visto prima) o pushare da linea di comando una repo già esistente
  - l'unica riga che non abbiamo visto è `git remote add origin url`, che serve per collegare la repository locale a quella remota
- Il `README.md` dovrebbe contenere nome app, composizione gruppo, funzionalità base etc... Ci sono siti che te ne generano di carini.

In `settings` → `collaborators` posso aggiungere collaboratori (per aggiungere membri del gruppo).

Capitolo 5

Tipi di applicazioni

5.1 Situazione ad oggi

Due principali player (iOs e Android). Nonostante questo, gli sviluppatori hanno a disposizione:

- molti linguaggi di programmazione
  - Java
  - Kotlin
  - Swift
  - JavaScript
  - Dart
  - ...
- Molti framework
  - React Native
  - Flutter
  - Xamarin
  - Ionic
  - ...
- E soprattutto, molte architetture
  - app native (oggetto di questo corso) che sviluppo secondo la piattaforma target
  - web app
  - app ibride
  - progressive web app
  - app cross-compiled

  Per scegliere la migliore, non c'è **UNA** scelta migliore, devo considerare l'oggetto del mio progetto (alcuni linguaggi/framework sono più adatti di altri), il tipo di applicazione che voglio realizzare, il tempo a disposizione, il budget, le competenze del team, ...

5.2 Livelli di astrazione

5.2.1 Livello più basso: hardware

L'hardware Apple è proprio dell'azienda Apple, poi magari varia da modello a modello ma è sempre lo stesso costruttore, sia per hardware che software (non c'è frammentazione).

L'hardware Android è invece prodotto da diversi costruttori, ovvero c'è frammentazione di hardware, quindi varia molto da modello a modello.

- es.: Google, Samsung, Huawei, Xiaomi, LG, Motorola...

Su questi hardware andiamo a implementare **app native** (così chiamate perché vanno ad usare il 100% delle API messe a disposizione dal sistema operativo) che sono app target per i dispositivi di destinazione. I linguaggi nativi per implementare queste app sono:

- Android
  - Java (dal 2007 nel panorama Android)
    - compiliamo per linguaggio intermedio (bytecode) e poi viene eseguito da una JVM (Java Virtual Machine); scrivo in un'unica codebase su cui compilo nel momento in cui viene eseguito e quindi sulla piattaforma effettiva
  - Kotlin ($\sim$2017)
    - Rilasciato da JetComeSiChiama nel 2011, abbastanza scagato
    - Poi ripreso nel 2017 da Google e dal 2018 è tornato ad essere usato
  - Problema è che ogni volta che viene rilasciata una nuova API deve essere sviluppata sia in Java che in Kotlin.
  - (NDK) la sigla significa *Native Development Kit*, permette di scrivere codice in C/C++ e di interfacciarsi con il codice Java.
- iOs
  - Objective-C
  - Swift
    - linguaggio di paradigma molto più moderno rispetto a Objective-C che nasce da C e quindi si porta dietro molte complicazioni
    - entrambi a oggetti, versioni semplificate di C

5.2.2 App native

Caratteristiche

Slide con lista

- Sviluppata per uno **specifico s.o. mobile**
- nativa per uno **specifico device o piattaforma**

- distribuita attraverso **codice binario scaricato** e **memorizzato** nel file system
- distribuita attraverso **app store** o **marketplace** (anche Apple ha perso la causa ed è costretta a rendere le sue app disponibili anche su altri store/marketplace)
- eseguita **direttamente** dal **s.o.**
  - viene lanciata attraverso la schermata home del dispositivo con un "tap" (tocco l'icona e parte, non ci sono passaggi intermedi)
  - non contiene un container app in cui girare (ho perso cos'ha detto a voce)
- l'app fa uso diretto delle API del s.o. (GPS, fotocamera, accelerometro, ...) (avevamo visto a prog2, Java mette a disposizione una SDK, una libreria, per facilitare la programmazione). Programmando da d.m. faremo un uso intenso delle API del sistema su cui mi trovo e ci dovremo interfacciare molto con l'hardware. Abbiamo:
  - strato sotto: hardware
  - strato intermedio: sistema operativo che si interfaccia con l'hardware da solo senza che ci debba pensare io
  - strato sopra: API, che il s.o. mette a disposizione per interfacciarsi con l'hardware senza interfacciarsi con l'hardware
  - strato ancora più sopra: SDK librerie che mi permettono di interfacciarmi con le API
  - strato ancora più sopra: open SDK, quello che effettivamente andrò ad usare (es. `open file()`, `append()`, ...)

  Questo è il bello dello sviluppare nativo: lavorare con le API al 100% e poter sfruttare al massimo le potenzialità del dispositivo

Processo di generazione

Abbiamo:

Codice sorgente (Java)	→		→	Risorse (es. img)
Compilatore e **Linker**	→	Eseguibile (binario)	→	**Packager**
				Packager distribuibile (*)
				App store

(*) = (`.apk` ora obsoleto al suo posto c'è `.aab`, `.apks`, `bundle`).
L'`.aab` è tutto un file grosso, poi viene estrapolato da solo l'`.apk` del dispositivo.
A settembre 2025 è stata rilasciata la versione "narvalo" dell'`.apk`.

Store publishing flow

1. Preparazione del pacchetto

   - Android: generazione file `.aab` (Android App Bundle)
   - iOS: generazione file `.ipa`
   - il codice viene **firmato digitalmente** per garantire l'autenticità

   Serve un certificato di sviluppatore (Play Console o Apple Developer Program)

2. Testing e valutazione

   - Android: test interni, chiusi (specifici tester, amici, tester fidati, etc... per raccogliere feedback più ampi), aperti su Play Console
   - iOS: test tramite TestFlight
   - Verifica di funzionalità, compatibilità, performance e privacy

   Fase di qualità tecnica controllata dal team

3. Pubblicazione e revisione

   - Upload su **Google Play Store** o **Apple App Store**.
   - Analisi automatica (Google)
   - Revisione manuale (più rigorosa e lenta su iOS)
   - Controlli su:
     - uso delle API sensibili (es. posizione, GPS)
     - privacy e permessi
     - stabilità e prestazioni

   Fase di qualità tecnica controllata dal team

4. Distribuzione

   - L'app diventa disponibile sugli store pubblici.
   - Possibilità di:
     - release oprogressive
     - aggiornamenti versionati
     - **sospensione o ritiro** in caso di violazioni

   Ricorda: Build → Test → ? → ?

Interazione con il dispositivo

Slide con immagine che mostra quello che abbiamo detto prima: mi appoggio direttamente alle API del s.o. per sfruttare potenzialità e servizi del dispositivo (hardware).

Il vantaggio è un accesso semi-diretto all'hardware. Hai a disposizione **tutto**, non hai restrizioni (grosso pro delle app native) se non quei servizi esplicitamente rifiutati dal cliente (es. non do accesso alla fotocamera).

Lo svantaggio è che devi scrivere due versioni dell'app (una per iOS e una per Android): se vuoi fare un'applicazione Android devi conoscere le API Android, se vuoi fare un'applicazione iOS devi conoscere le API iOS. Perciò usando librerie specifiche, linguaggi diversi ovviamente, e quindi API specifiche, non posso fare un'unica app nativa per entrambi i sistemi operativi.

Vedremo che esistono API di basso livello che si interfacciano ancora più direttamente di altre (quindi sono più di basso livello) con l'hardware. Le altre di livello più alto sono più user-friendly. Non va l'utente direttamente a gestire il touchscreen, la rete, etc.

Mobile apps runtime architecture

Slide con immagine. Ho da una parte la mia app scritta in linguaggio nativo, dall'altra la piattaforma con OEM (Original Equipment Manufacturer) widgets, servizi, ...

Quando faccio app native ho due componenti:

- Da una parte codice nativo (compilato in linguaggio macchina)
- Questo comunica direttamente con componenti della piattaforma (widget grafici, risorse del dispositivo)

Sviluppo nativo: pro e contro

Vantaggi

- ottima esperienza utente
- performance (prestazioni) elevate
- slide con lista
- la presenza negli store raggiunge più rapidamente gli utenti (spesso magari l'utente cerca un'app di interesse direttamente nello store, quindi viene raggiunta prima l'app)

Svantaggi

- due codebase da manterenere (e relativi costi e tempistiche di sviluppo e di manutenzione) e ridotta riusabilità del codice
- convincere gli utenti a scaricare l'applicazione
- dipendenza forte dagli aggiornamenti dei s.o.
- richiede l'installazione
  - ad eccezione di alcune app che non serve installare: es. *Instant Apps* (Google) e a seguire *App Clip* (Apple)
  - Permettono di utilizzare un'applicazione senza installarla, ma *solo per prestazioni limitate*, danno un assaggio dell'applicazione per provarla senza stare lì a installarla; funziona solo se l'app è ben modularizzata

  - A volte possono essere comode, per esempio quando l'utente non ha voglia di installare direttamente un'applicazione per fare una cosa una tantum
  - C'è una documentazione apposita nel sito Android che spiega come creare queste instant app
    slide con immagine tabella verde apple vs google (non c'è scritto ma quelle Apple permettevano di effettuare pagamenti e all'inizio quelle Android no ma poi si sono uniformate)

5.3 Tipi di app

Tornando ai **livelli di astrazione**, abbiamo detto che sugli hardware andiamo a implementare le app native. Questi i due livelli più bassi. Ma partendo dall'alto abbiamo:

- Web App
  - Angular, React, Vue, Ember, Backbone, ... guarda slide
  - non esattamente app mobili
- PWA
- Browser
- Hybrid App (nella slide esempi di framework per sviluppare web app, quindi potrei averli anche al primo livello)
- Rendering engine (webview e wkwebview)
- Web-native app
- Cross-compiled app
- Native App
- Hardware

Ad oggi queste sono le macrocategorie di app mobile che possiamo trovare.

Ma quanti strumenti/framework abbiamo? Slide con lista di **alcuni** esempi, perché è un mondo in costante evoluzione e quindi non si può fare una lista esaustiva. Non bisogna restare ancorati a ciò che si conosce evitando cose nuove, perché rischi di metterci molto più tempo perché magari un nuovo linguaggio o framework ha delle facilities che ti permettono di fare in 10 minuti quello che in un altro linguaggio ti avrebbe richiesto 10 ore. C'è anche il rischio che ciò che usavo tipo 5 anni fa diventi obsoleto e quindi non più supportato o deprecato.

5.3.1 Web App

Una web app è un'applicazione accessibile tramite browser ma progettata per comportarsi come una'app.

Gli strumenti di testing sono molto importanti, mi permettono di vedere se la mia app va o no.

C'è un sito (`https://whatwebcando.today/`) che fa vedere, dato un browser (nella slide Firefox e Safari), cosa quel browser ti permette di fare.

Per esempio, se uso Firefox e voglio fare una web app che usi fotocamera e microfono, posso. Invece per esempio *advanced camera control* non posso con nessuno dei due, mentre *record media* va con Firefox ma non con Safari.

Quindi quando realizzo una web app devo aver presente cosa voglio fare e quali strumenti mi servono e quindi (esattamente come le app mobili devo stare attento al s.o.) devo stare attento al browser che l'utente usa e a ciò che mi permette di fare.

Interazione con il dispositivo

Devo vedere quali API siano effettivamente disponibili, perché non tutte le API sono disponibili su tutti i browser. In verde ciò che posso utilizzare.

Ci sono diversi **rendering engine** (rendering grafico è interpretare ed eseguire il JavaScript) che sono alla base dei browser. Ogni browser ne ha uno suo, ma sono diversi in base a quale dispositivo sto utilizzando. Per esempio, se uso un dispositivo Apple, Safari usa **WebKit**, mentre se uso un dispositivo Android, Chrome usa **Blink**. Lo stesso browser, es. Safari, si comporta in modo diverso su un dispositivo Apple fisso, su un dispositivo Apple mobile, su un dispositivo Android, su un dispositivo fisso che gira su Linux, ...

Tramite W3C calls.

Caratteristiche

3. mi dice che o ho un url o un qr code o non so, diversi punti di accesso ad una web app 4. non serve scaricare e installare, ma basta un browser.

Sviluppo web app: pro e contro

Vantaggi

- accessibilità universale
- unica codebase
- ...

Svantaggi

- performance inferiori
- ...

Perché vale la pena sviluppare più app che web?

Slide tempo medio speso al giorno con uno smartphone e una connessione internet. ~90%!!

Ovviamente dipende dal dispositivo: per dire, se sono da portatile, mi sarà più comoda. Una volta che uso un dispositivo mobile, magari con schermo piccolo, meglio l'app.

Lista di perché. Schermata conclusiva con perché meglio app di web app. Inserisci. Dal pov dell'engagement l'utente mostra di preferire l'uso di app invece di web app. Anche perché un cell ce l'hanno tutti, ma è facile che molti non abbiano un portatile. Magari hanno un tablet, ma è un altro dispositivo mobile.

5.3.2 PWA - Progressive Web App

Sono una via di mezzo fra web app e browser, permettono di avere una web app potenziata che si comporta come un'applicazione mobile, "installata" dal browser senza passare dallo store. Tende a comportarsi come un'app nativa, parte del perché si chiamano così. Si basano su file manifests e service workers. Funzionano anche offline e possono inviare notifiche push.

Tecnologie

- HTML, CSS e JavaScript
- Service Worker
  - **Script** che funzionano in background e consentono l'uso di funzionalità come la cache per uso offline
- Manifest File
  -
- HTTPS

I Service Worker sono un concetto molto importante, perché mi permettono di fare delle cose che non posso fare con una web app. Rendono le PWA:

- **Potenti**
- **Affidabili**
  - Veloci
  - Funzionanti anche in assenza di rete o in presenza ma scarsa (**importante per l'offline-first!**)
- **Installabili**

Dove si collocano?

Caratteristiche di web app (più raggiungibili) e app native (più potenti, più funzionalità). Le PWA si collocano in mezzo, sono più ricche di funzionalità delle web app ma più raggiungibili delle app native.

Nativa o progressiva?
Quando usare PWA?

- lista
- importante tenere a mente quali browser permettono di fare cosa

Quando usare app native?

- occorre
- ...
- le PWA non dovrebbero avere meccanismi di monetizzazione, quindi se li voglio applicare mi serve un'app nativa

Dove trovare le PWA?
Non sono negli store di app tradizionali (su google play store solo se impacchettate dentro un'app nativa)
Non tutti i browser le supportano (guarda la slide)

Sviluppo PWA: pro e contro

Vantaggi

- ...

Svantaggi

- ...

5.3.3 App ibride

Es. Cordova, slide con architettura.

Per dire se non ho API per fare una cosa, ciò che non è direttamente supportato dalle API che hanno a che fare direttamente con il s.o. (es. termoscopio (?)), posso scrivere un plugin che mi permette di fare quella cosa. I **rendering engine** quindi sono "ponti" che fanno da tramite per fare esattamente ciò.

Architettura di Capacitator

Slide con lista di caratteristiche in cui vedo come gli engine fanno un po' da ponte.

Praticamente ho il mio **rendering engine** che comunica direttamente con il s.o., ma se ci sono cose che non riesco a fare da sola mi appoggio ai plugins per ciò che non è supportato dall'engine. Del **HTML rendering engine (WebView)** ho sempre comunque bisogno.

Slide con immagine aggiornata.

Attraverso la webview ora posso osservare e catturare le interazioni dell'utente con ciò che ho realizzato con HTML, CSS, JavaScript.

Se nativo dipendeva dal s.o., ibrido è sempre JavaScript, così come anche il suo bridge è in JavaScript.

Ho i miei vantaggi e svantaggi.

Sviluppo Hybrid App: pro e contro

Vantaggi

- ...
- UNICA CODEBASE! Buono!!
- ...

Svantaggi

- ...
- Potrei non avere tutti i plugin già sviluppati e disponibili

5.3.4 Web-native app

Stiamo sotto alla WebView. Stiamo programmando in JavaScript qua. Tipo iOs ha il motore già compreso, mentre Android no.

Mobile apps runtime architecture

Slide con immagine aggiornata.

Ho sempre JavaScript, ma ora ho un rendering engine unico che è sia per servizi che per piattaforma, quindi sia interfaccia utente che per convertire servizi nativi, gli OEM widgets, ...

è uguale a quelle native, ma c'è il bridge che trasforma il codice in codice nativo (sostanzialmente il bridge è un convertitore just in time o ahead of time).

Uso React Native, Native Script, Appcelerator Titanium...

Sviluppo Web-Native App: pro e contro

Vantaggi

- ...

Svantaggi

- **Bridge complesso**: tradurre JS in codice nativo può creare overhead.
- ...

5.3.5 Cross-compiled app

Slide con immagine.

Sono quelle più vicine di tutte all'hardware: ci stacchiamo completamente da tutte le tecnologie web.

Vuol dire compilate (app compilate, **non più JavaScript, ci dimentichiamo del web**) per più piattaforme. Non usiamo più gli strumenti web, ma strumenti di sviluppo nativo.

Mobile apps runtime architecture

Slide con immagine aggiornata.

Ho un unico codice sorgente, ma ho un compilatore che mi permette di avere un'app per entrambi i s.o. (es. `Flutter`, `.NET MAUI`, `RubyMotion` ...). Tipo RubyMotion compila direttamente per dispositivo target (*native* & *cross-compiled*) e compila ahead-of-time. `.NET MAUI` compila in C# e poi in bytecode, una sorta di *web-native* (ho un bytecode che fa una sorta di bridge). Flutter in Dart e poi in bytecode. Flutter ha un meccanismo diverso, un'architettura tutta sua, due elementi: rendering engine (non si appoggia più su componenti nativi), realizza lui i suoi componenti grafici (widget rendering), poi ho il concetto di platform channels.

Sviluppo Web-Native App: pro e contro

Vantaggi

- ...

Svantaggi

- ...

5.3.6 Quale piattaforma scegliere?

Dipende da quale è la mia base di partenza, cosa voglio ottenere e quali sono le mie competenze.

Tutto ciò che sta sotto WebView è considerato app mobile (quindi anche web-native anche se uso strumenti web oriented).

- **Performance migliori**
  - .
- **Negli app store**
  - .
- **Support web framework**
  - .
- **No installazione**
  - .

Slide con guideline, 3 domande per aiutare a scegliere la piattaforma migliore per il proprio progetto.

Capitolo 6

La piattaforma Android - prime info

6.1 AOSP - Android Open System Platform

L'Android Open Source Project (AOSP) è un'iniziativa open source, slide

6.1.1 Architettura

- basato su un kernel Linux
- architettura a stack
  - un livello sopra all'altro
  - ognuno dotato di proprie funzionalità

Kernel Linux

Fornisce supporto per funzionalità di basso livello (threading, gestione della memoria, l'accesso all'hardware via driver, etc.)

Fornisce modello di sicurezza

Permette ai produttori di dispositivi di sviluppare driver hardware per un *kernel ben noto*

Devo basare il driver per interfacciarmi con le funzionalità di quel modello specifico che sto implementando: ogni hardware ha API specifiche, quando scrivo un driver devo interfacciarmi con quelle API

HAL - Hardware Abstraction Layer

Fornisce un insieme di **interfacce** **standard** che espongono le funzionalità hardware al **Java API Framework**.

Ciascuna interfaccia (ciò che ci si aspetta dal sensore) fornisce un insieme di servizi offerti dall'hardware corrispondente.

Quando una **Java API Framework** effettua una chiamata per accedere all'hardware del dispositivo, il sistema Android carica il modulo che gestisce quel componente hardware specifico.

Quindi quello che devo andare ad individuare è il set di funzioni minime che il sensore deve avere e lo mando poi all'HAL. Io so che devo implementare una interfaccia con queste funzioni specifiche? So quale è l'insieme di funzioni che devo implementare perché l'interfaccia sia usabile.

ART - Android Runtime

È il motore di esecuzione di Android, introdotto dalla versione 5.0 (Lollipop) per sostituire il vecchio Dalvik:

- Dalvik: compilazione JIT (Just In Time)
- ART: compilazione AOT (**Ahead Of Time**) e JIT (compilazione Just In Time)

Dato il bytecode ch emi arriva, serve qualcuno che lo compili **e** lo esegua.
Vantaggi ART:

- miglioramento efficienza complessiva dell'esecuzione
- cose
- slide

Profile guided compilation (introdotto da Android N (Nougat)) praticamente scarica un po' all'installazione un po' quando vedo che l'utente usa particolari funzionalità non inizialmente scaricate all'installazione. Per questo è profile guided, perché devo guardare il profilo dell'utente.

Da Android P (Pie) è stato introdotto Profiles in the Cloud: permette di scaricare il profilo dell'applicazione da un server e non dal dispositivo stesso, in modo da avere un'applicazione più leggera e più veloce.

Librerie Native (C/C++)

Android permette di scrivere librerie native in C/C++ che:

- consentono agli sviluppatori

Java API Framework

Quello che andremo noi ad usare nel progetto.

Insieme di API Java che forniscono un insieme ricco di funzionalità:

- ...

System Apps

Applicazioni preinstallate nel sistema Android, che all'occorrenza posso integrare nello sviluppo della mia applicazione.

Le app di sistema funzionano sia come

Es.: se devo aprire un url, non vado mica a costruire a mano un browser, ma uso quello preinstallato.

Tre concetti importanti:

- minimum SDK
  Es.: la 8 è la minima versione di Android che supporto, perciò se uno ha la 7 non può installare la mia app
- target SDK
  Quella per cui andiamo a sviluppare l'app, però siamo noi che garantiamo che la nostra app (per esempio nel caso di minima 8 e target a 10), andiamo a garantire che se funziona per la 10 funzionerà anche per la 8
- compile SDK

6.2 Componenti minime

6.2.1 Android SDK (Software Development Kit)

Insieme di strumenti per sviluppare app per Android: strumenti e librerie necessari per sviluppare app Android e rendere l'implementazione del codice più gestibile.

L'SDK contiene strumenti principali:

- **Android Studio:** è l'ambiente di sviluppo integrato (IDE) ufficiale per Android
  - basato su IntelliJ IDEA
  - offre strumenti come l'editor di codice, debugger e strumenti di testing
- **Emulatore Android:** permette di simulare diversi dispositivi Android sul computer
  - Utile per testare l'applicazione senza aver bisogno di un dispositivo fisico
- **SDK Tools:** strumenti per compilare, debuggare e testare le app, oltre al profilo delle prestazioni
- **API:**

6.3 Google Play Services

Servizi che possiamo usare, ma serve una *key* da memorizzare sulla nostra applicazione, possibilmente in un posto sicuro quindi non codice sorgente che mando in giro. Senza questa chiave non posso usare i servizi di Google. Di solito si è monitorati perché oltre ad un certo numero di utilizzi si inizia a pagare.

Capitolo 7

Android - la prima applicazione e le risorse

7.1 Strumenti

Essenziale per iniziare a sviluppare un'applicazione.

L'IDE che noi abbiamo usato è Android Studio, che è l'IDE ufficiale per lo sviluppo di app Android. L'ultima versione rilasciata in versione stabile è Ladybug (io avevo scaricato Koala, che va bene uguale, il grosso dei cambiamenti è avvenuto fra Giraffe e Koala).

Gli emulatori sono sempre stati molto lenti o addirittura non funzionavano. Genymotion è un buon escamotage (`https:`) vedi slide.

7.2 Setup

All'installazione, c'è da configurare AVD e .

7.2.1 AVD - Android Virtual Device

Emula un dispositivo fisico Android, configurabile anche come meglio credo. Posso scegliere la versione di Android, la dimensione dello schermo, la memoria, ... c'è la lista sulle slide.

Non basta chiaramente vedere se funziona con l'AVD per andare sullo store: devo sempre ricordarmi della frammentazione di Android, ovvero quante diverse versioni di hardware e produttori ci si affidino, quindi devo testare su più dispositivi possibili.

Set Up di un AVD

Lista sulle slide. Ne ho saltate alcune con screen dei passaggi. Comunque ad usare l'app e smanettarci poi uno ci prende la mano. In ogni caso, è **fondamentale** consultare sempre la documentazione ufficiale di Android che spiega nel dettaglio come fare qualsiasi cosa. Anche perché noi vediamo le best practice, ma è tutto in continua evoluzione quindi la documentazione costantemente aggiornata è la fonte più affidabile.

7.3 Prima di cominciare con la prima app

Sia che usiamo l'`xml` (senti lezione qua)
Es.: la home, la ricerca, la visualizzazione dei risultati della ricerca, ..., sono tutte activities diverse. Mi sono persa cosa sono i fragment, riascolta la lezione.

Ho due componenti:

- *Activity*: componente che gestisce l'interfaccia utente (UI) e che l'utente può usare per interagire con l'applicazione
  - due cose
- *Layout*: elenco degli elementi grafici, definisce un insieme di **elementi** della UI e la loro **posizione** sullo schermo
  -

  Noi avremo Activity () e Layout (xml, quindi xml è "descrittivo") in due file separati.

7.3.1 Es.

Slide con esempio di quello che vogliamo costruire.

7.3.2 Cominciamo

Sezione "Projects"

- New Project
- Ho quattro dispositivi target (phone & tablet, wear OS, tv, automobili), ciascuno con una lista di template
- Es. empty activity non permette di scegliere il linguaggio di programmazione, fisso su Kotlin, **non lo useremo**
- Noi ci basiamo sul concetto di View, quindi andrò a prendere Empty View Activity e faccio Via/Next
- In package name, dovrei mettere l'inverso dell'azienda (es. com.azienda.nomeapp) e nomeapp lo prende dal nome che ho dato al progetto, mentre azienda è l'identificativo dell'azienda
  - il nome del progetto è il nome dell'applicazione e non è modificabile una volta iniziato a sviluppare
  - chiedono se per fare il progetto dobbiamo mettere "it.unimib.nomedelprogetto" e fa "mh sì penso possa valere la pena"
- Language: Java
- Minimum API level: la versione minima che voglio supportare, dalla Android 7 in su **dovrebbe** funzionare, la 6 sa per certo che non funziona e anzi non viene neanche mostrata nello store ed esce scritto che non è compatibile

- Build configuration level: consiglia quello "recommended"
- faccio "Finish"
- Mi trovo davanti due finestre: 1 a sinistra del progetto con la sua struttura, 2 a destra con il file sorgente
- mi escono due file, un `activity_main.xml` (file di Layout in questo caso specifico, anche lui in Java) e un `MainActivity.java` (che ha un nome inverso di xml, è **una convenzione**, compreso l'underscore)
- con il tasto in alto a destra (sulla riga di .xml e .java, ma riguarda solo l'xml perché riguarda il design) "Code" (ALT + Maiusc + Destra) mi cambia visualizzazione xml
- "`<androidx.constraintlayout.widget.ConstraintLayout...`" sono constrain, forzati lì dove sono (c'è una slide "widget" che spiega cosa sono)
  - una widget può mostrare testo o grafica, interagire con l'utente, organizzare altri widget sullo schermo
  - l'SDK Android include molti widget che è possibile configurare
  - e
- il "device manager" (il terzo a destra in verticale) mi fa vedere i dispositivi che ho configurato e mi permette di aggiungerne di nuovi con tutte le cose che posso impostare
- slide che mostrano cosa ConstraintLayout e TextView controllano
- quando vado tipo a toccare "hello world" mi si aprono una serie di proprietà che posso impostare e modificare che sono quelle che trovo nel codice xml
- due a destra di "code" c'è "design" che mi fa vedere come viene visualizzato il layout con una serie di strumenti e comandi
- c'è una slide con dei numeri verdi che spiega un po' tutta la finestra, inserisci
- "component tree" mi mostra la gerarchia dei componenti
- tornando su "code", ho una serie di proprietà con ruoli ben precisi
  - nella slide, quelli evidenziati in verde
  - quelli blu sono i constraint
  - a me interessano quelli che nella slide credo 20 sono sulla sinistra
- Concetti di:
  - **Screen Size:**
    *
  - **Screen Density:**
    *

  - **Screen Resolution:**
    *

- Esempio: "si immagini un'app in cui uno scroll è riconosciuto dopo che l'utente si è mosso sullo schermo per almeno 16 pixel". Il gesto viene riconosciuto dopo:
  - 2,5 mm (25,4mm*)
- Io ragiono in termini di DP e SP, pixel virtuali che sono indipendenti dalla densità:
  - slide
- In questo modo ho una migliore esperienza utente, "guai a voi" se usiamo i pixel invece dei dp. Questa è una delle cose che **va a valutare** nel progetto. Ha mostrato un'immagine di minion, screen size uguale ma diversa densità.
- "ems" è un'altra unità, prende la dimensione della lettera "M" maiuscola e su questa basa la dimensione del testo, mentre "sp" è una unità di misura che tiene conto della dimensione del testo dell'utente
- tornando ad A.S., con la visualizzazione "Project" a sinistra vedo come effettivamente sono organizzate le mie cartelle, ma "Android" è più comoda
  - dentro "Android" c'è una cartella "res" >"values" che contiene xml con diverse risorse (es. aggiunge dentro "strings" "bottone")

```
<resources>
    <string name="app_name">My Application</string>
    <string name="bottone">Saluta</string>
    <string name="campo">Ciao</string>
</resources>
```

  - contiene le mie risorse, stringhe, colori, font, layout, immagini, icone ... In Android c'è una netta distinzione fra codice e risorse, quindi le risorse vanno messe in cartelle apposite (le risorse sono tutto ciò che si distingue dal codice, stringhe non sono codice, colori nemmeno, font nemmeno ...)
  - A questo punto tornando su xml, dentro "Button", invece di

```
Button> android:text="Saluta"
TextView> android:text="Ciao!"
```

    farò

```
Button> android:text="@string/bottone"
TextView> android:text="@string/campo"
```

dove "@" vuol dire "presso", quindi "presso il file delle stringhe prendi quella chiamata bottone".

- Una cosa utile sono le traduzioni:
  - da file "strings.xml" >"Open editor" >"Add locale" >scelgo la lingua >"OK"
  - poi metto le traduzioni delle risorse che ho inserito
- In generale la sintassi è "@ NomeRisorsa / nomeElemento" senza spazi.
- Slide con concetto di risorsa. Nelle slide "raggruppare le risorse" ho una lista delle cartelle che devo avere. Slide "risorsa e id risorsa" interessante ma me la sono persa.

Capitolo 8

Esercitazione 2

Avviamo un nuovo progetto Android Studio, scegliendo il template *Empty Activity*.

Manifest (prima directory nella visualizzazione Android) è un xml che contiene le informazioni dell'applicazione, come il nome, l'icona, le activity, i permessi, ... Dentro, definiamo `MainActivity` come activity principale, con un target ideale. `MainActivity` è il nostro punto di partenza. Estende la classe `AppCompactActivity`, che è un'activity standard che supporta le funzionalità più recenti di Android.

8.0.1 Directory `Java`

Dentro la directory Java e la prima sottodirectory, vado a prendere il `MainActivity.java`. Qui sostituisco l'override che vedo con:

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
}
```

Punto ai layout con il .
Best practice:

- nomi delle classi: camelcase ma con maiuscola
- nomi dei metodi : camelcase ma con minuscola
- nomi dei layout: stesso del Java ma invertito e con underscore
  Es.: `MainActivity.java` e `activity_main.xml`

Le altre due directory dentro Java posso ignorarle per ora.

8.0.2 La cartella `res`

Contiene le risorse layout dell'applicazione, come stringhe, colori, font, layout, immagini, icone, ...

8.0.3 Mipmap

Simili a `drawable` ma di solito usato per le icone. Contiene icone di diverse dimensioni per adattarsi a diverse risoluzioni.

Di solito meglio vettoriali perché png si sgranano.

8.0.4 Values

Riservato a colori e stringhe.

Dentro `colors.xml` posso definire i colori che uso nell'applicazione.

Dentro `strings.xml` posso definire le stringhe che uso nell'applicazione, ovvero tutti i testi che compaiono dentro l'applicazione. Utile perché Android Studio nella sezione "Open Editor" dà uno strumento utile: la traduzione. Posso fare una traduzione automatica delle stringhe in altre lingue, e posso anche fare una traduzione manuale.

8.0.5 Themes

Due file: `themes.xml` e `themes.xml (night)`. Il secondo è per la modalità notturna.

8.0.6 xml

Non andremo a vedere queste cose, ignorare completamente.

8.1 Gradle Scripts

8.1.1 build.gradle.kts (:app)

Ho dentro tipo:

- namespace = "com.example.esercitazione2": identificativo univoco dell'applicazione
- compileSdk = 34: SDK con cui compilo
- `defaultConfig`:
  - `applicationId =`: identificativo univoco dell'applicazione
  - `minSdk =`: SDK minimo con cui funziona
  - `targetSdk =`: SDK con cui è stato testato (min SDK $\leq$ target SDK $\leq$ compile SDK)
  - `versionCode =`:
  - `versionName =`:

Dentro `dependencies` posso aggiungere le dipendenze che voglio. Lui ha inserito `implementation('com.squareup.retrofit2:retrofit:(insert latest version)')`. Ma dà errore, faccio tasto destro e "Show context actions" e "replace with new library".

In breve, a sinistra c'è il tasto "Resource Manager" che mi permette di vedere in sintesi tutto quello che abbiamo visto finora.

Torno su "activity_main.xml", "code" (tastino in alto a destra) e va ad aggiungere

Apriamo l'emulatore. C'è già un emulatore acceso, lo togliamo e mettiamo Pixel 8a (preso a casissimo).

Se vogliamo usare il nostro telefono, dall'emulatore andiamo su impostazioni,"developer options", "about emulated device", "build number" e clicchiamo 7 volte. Torniamo indietro e andiamo su "developer options", "usb debugging" e lo attiviamo. Collego il telefono al computer e mi chiede se voglio usare il telefono per debuggare, accetto. Se non mi chiede, vado su "developer options" e attivo "usb debugging".

Dentro activity_main.xml, andiamo a vedere le variabili

```
android:id="@+id/main"
android:layout_width="match_parent"
android:layout_height="match_parent"
```

Dentro "TextView" vado a mettere

```
android:id="@+id/textView1"
```

Ho due tipi di misure spaziali:

```
android:layout_width="wrap_content"
android:layout_height="match_parent"
```

Il primo sta attorno al contenuto, il secondo si adatta all'altezza dello schermo.

Se faccio linear layout mi affianca gli elementi, da dentro il primo blocco (prima di TextView) vado a mettere `android:orientation="vertical"` li mette uno sotto l'altro.

Capitolo 9

Esercitazione 3

9.1 Il progetto

Cominciamo il progetto `WordNews` con un template *Empty Activity*. Sarà un'app che mostra le notizie.

- `androidx.constraintlayout.widget.ConstraintLayout` diventa `LinearLayout`, mette gli elementi in lista semplicemente, in verticale o in orizzontale.
- dentro ci aggiungo `android:orientation="vertical"`
- `android:gravity` sposta il testo all'interno dell'oggetto
- `android:layout_weight` fa in modo che l'oggetto si adatti alla grandezza dello schermo (no valore di default)
- ora proviamo il Layout `RelativeLayout` (lo useremo per poco o nulla) in cui mettiamo una TextView con `android:layout_below="@id/testo1"` che mette le view in fila verticalmente, comoda per le view ma non tanto per i singoli oggettini
- su moodle ha caricato una risorsa con buoni consigli per la programmazione, su questo sito Material Design Guidelines possiamo anche trovare le palette da inserire nel nostro progetto
- ogni schermata è un'activity: per farne una nuova "New -¿ Activity -¿ Empty View Activity". Diamo il nome "PickCountryActivity" e mettiamo il layout "activity_pick_country" (fa di default veramente)
- dentro "AndroidManifest.xml" andiamo a mettere `<activity android:name=".PickCountryAct />` che sarà il nuovo punto di ingresso dell'app. Di seguito:

```
        <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
```

Questa parte che è dentro MainActivity la mettiamo dentro PickCountryActivity.

- dentro "activity_pick_country.xml" utile sono le "guidelines" che metto verticale
- comunque ha fatto roba tra cui inserire 1-2 cards che mi sono persa, ma quasi tutto mi sembra possa essere fatto dal sito di material design e importato
- comunque poi anche il login mail password e "did you forget your password?" è un bottone anche se sembra testo, sul sito material design si vedono i vari stili di bottoni

Slide ciclo di vita delle activity:

- onStart è quando l'applicazione è in primo piano
- onPause è quando l'applicazione è ?
- onStop è quando l'applicazione è in background (può essere killata qui e tornare a onStart, devo mettere quali risorse voglio ripristinare)

Un'activity va in uno stato di Destroyed per diversi motivi:

- se l'utente vuole eliminare dalla memoria l'activity
- se il sistema deve aggiornare la configurazione (voluto dall'utente, es cambio lingua o orientamento verticale/orizzontale o cambio tema giorno/notte... praticamente qualsiasi cosa mi faccia cambiare la schermata, l'interfaccia che mi trovo davanti)

E l'utente si aspetta di ritrovare l'activity come l'aveva impostata. Per questo devo salvare lo stato dell'activity (cosa che Android di per sé non fa).

Ma un'activity può essere cancellata anche dal sistema proprio e non dall'utente:

- il sistema deve liberare una RAM limitata

9.2 Salvare lo stato volatile di un'activity

Quando un utente si aspetta di ritrovare un'activity come l'aveva lasciata, devo salvare lo stato **volatile** dell'activity (i dati persistenti ci si aspetta vengano sempre salvati).

- classe `ViewModel` (elemento architetturale)
  Usato **insieme a `onSaveInstanceState()`** perché quest'ultimo comporta costi di serializzazione/deserializzazione, nei casi di dati più complessi.
- metodo `onSaveInstanceState`: salva lo stato volatile dell'activity *in quel momento*
  Unico che posso usare per salvare dati della UI **semplici** e **leggeri** (come un tipo di dato primitivo o un oggetto semplice come `String`)
- memorizzazione locale

La prossima lezione IMPORTANTISSIMA vedremo **architetture**.

N.B.: distruggere un processo vuol dire distruggere tutte le activity che ci sono dentro, perdere la memoria.

9.2.1 Salvare lo stato volatile di un'activity con `onSaveInstanceState()`

- Instance state e `Bundle`
  - boh
- Il metodo `onSaveInstanceState()`
  - Invocato dopo onStop()

Capitolo 10

Task e Back Stack

10.1 Cancellazione della memoria: Task e Back Stack

- Task: insieme di activity con cui l'utente interagisce per svolgere un compito
- Le activity vengono messe in uno stack (Back Stack) nell'ordine in cui sono state aperte
  - la nuova activity viene messa (push) sopra a quella vecchia
  - quando viene premuto back, la top task viene istrutta (pop) e si torna alla precedente appena sotto (non visibile) che torna di nuovo visibile
- Seguiamo la politica LIFO (Last In First Out) come ha detto la prof, ovvero FILO (First In Last Out)

10.2 Attivare i componenti

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

Activity, Service e Broadcast Receiver vengono attivati da messaggi asincroni chiamati `Intent`.

L'app demanda il compito al sistema Android inviandogli un messaggio che specifica l'**intent** di avviare un particolare componente.

Gli Intent sono quindi usati per attivare comoponenti e condividere dati.

Un Intent wrappa un Bundle.

I Content Provider (gestiscono insiemi di componenti utili a più applicazioni, es. i contatti della rubrica) non vengono direttamente attivati attraverso Intent. Ma abbiamo **Content Resolver** che fa da tramite/intermediario (non li vedremo, si possono trovare nella documentazione).

10.2.1 Tipi di Intent

- **Espliciti**: specificano l'azione da eseguire e il tipo di dati su cui eseguirla
- **Impliciti**: chi lancia questo Intent non ha idea

Intent esplicito

Messaggio per attivare un componente specifico.

- Da usare quando
-
- Da utilizzare con i Service (ovvero, i Service possono essere attivati **solo** con Intent espliciti)

Intent implicito

Messaggio per attivare uno specifico tipo di componente.

- Da usare quando
- Si specifica il tipo di componente attraverso una **action** che deve essere eseguita e il sistema sceglier
- Android

10.2.2 Intent Filter

Dichiara il file Manifest se conosco il suo nome posso attivarlo altrimenti no

La costruzione di un Intent

Le action hanno semantica ben precisa, se non c'è una action che fa quello che voglio fare, la implemento io. Gli intent hanno una serie di metodi per andare a mettere tutte le informazioni che mi servono per attivare un componente.

L'invio cambia in base al componente: ci sono dei metodi appositi.
registri permettono di registrare intent di ritorno di attività

10.2.3 Intent impliciti: problemi

non è detto che l'app esista ho più app che lo fanno, lascio la scelta all'utente (es condividere un'immagine)

Capitolo 11

Esercitazione 5

C'è un field chiamato "inputText". è bene comune controllare che i campi abbiano sempre il tipo di input corretto (es. mail per le email ...).

"apache validation" è una libreria che permette di fare la validazione dei campi di input.

Abbiamo detto che Gradle è un sistema di build automation, ma è anche un sistema di dependency management. Dentro `build.gradle` c'è una sezione `dependencies` in cui si possono mettere le dipendenze del progetto. Se esce il suggerimento di uprgrade alle nuove linee guida, si può fare con un click e fa tutto in automatico.

Dentro LoginActivity.java c'è un metodo `onCreate` che viene chiamato quando l'activity viene creata. Si può fare l'override di questo metodo e mettere il codice che si vuole eseguire quando l'activity viene creata.

Dentro LoginActivity.java c'è un metodo `isMailOk` che controlla se la mail è corretta. EmailValidator.getInstance().isValid(email) è un metodo che controlla se la mail è corretta. Se la mail è corretta, ritorna true, altrimenti ritorna false.

Le Snackbar sono delle notifiche che appaiono in basso. Su Material3 `"material-components-android/docs/co` c'è la documentazione. Su "activity-login.xml" faccio una nuova Log e metto un id. Dentro LoginActivity.java metodo loginButton.setOnClickListener ¿ "Snackbar.make(findViewBy(android.R.id.content), text: "", Snackbar.LENGTH_SHORT).show(); così fa uscire la Snackbar.

Introduciamo il concetto di intent. Sono oggetti che permettono alle activities e non solo di dialogare fra loro. Possiamo generarlo o senza parametri (intent implicito) o con come parametro l'activity verso cui voglio andare (intent esplicito).

Per fare un intent esplicito, si fa `Intent intent = new Intent(this, PickACountryActivity.class); startActivity(intent);`.

Sono anche contenitori di informazioni. Si possono mettere informazioni dentro l'intent e passarle all'altra activity. Si fa `intent.putExtra("key", "value");` (noi facciamo `(EMAIL_KEY)`). Per recuperare l'informazione si fa `String value = getIntent().getStringExtra("key");`.

Per fare un intent implicito, si fa `Intent intent = new Intent(Intent.ACTION_VIEW, Uri.parse("http://www.google.com")); startActivity(intent);`.

In activity_login.xml per immagine scelta dall'utente, vedi lezione.

Introduciamo il concetto di fragment. Un'activity corrisponde ad una schermata. Fragment funziona un po' come un activity, ma è un pezzo di schermata. Si possono mettere più fragment in una activity.

Un esempio. Google foto. Ha sotto una barra con "Photos" e "Search". Ma se clicchiamo su una delle due non cambia l'activity, ma cambia il contenuto. Ovvero, cambia il fragment. Perché l'activity è sempre la stessa, ovvero la barra sotto (e logo di Google Foto in alto e il burger menu in alto a sinistra).

A differenza dell'activity, il fragment è ideato a runtime. Costruttore vuoto. Dentro LoginFragment.java c'è un metodo LoginFragment newInstance() ....

Dentro onViewCreated() si fa il binding dei campi. Si fa `binding = FragmentLoginBinding.inflat container, false); return binding.getRoot();`.

Dentro activity_login.xml metto "FragmentContainerView" che è un contenitore per i fragment. Si mette un id.

navGraph è un file xml che contiene la navigazione dell'applicazione.

Sulla modalità Design di nav_graph.xml si può fare il drag and drop dei fragment manualmente, collegandoli con delle frecce. La cosa si rifletterà in automatico sul xml.

Ricapitolando, abbiamo un activity con il login. Dentro c'è un fragment FragmentContainerView. Dentro il fragment c'è un bottone. Quando clicco il bottone, voglio che mi porti ad un'altra schermata. Per fare questo, devo creare un altro fragment e collegarlo al primo. Guidelines sono: se sono schermate "piccole" meglio usare fragment, tenere activities per cose più importanti tipo passare da pagina login a pagina principale.

Navigation.findNavController(view).navigate(R.id.action_loginFragment_to_pickACountryFragment è il codice per navigare da un fragment all'altro.

Da un fragment posso passare a due diverse activity. Per fare questo, si fa `navGraph.xml` e si collega il fragment a due activity diverse. A livello di codice gestirò come fare a passare da un'activity all'altra. Per esempio il findNavController di prima era dentro un if.

Per salvare in locale (quindi su file dati persistenti), c'è la funzione `getSharedPreferences`. Guarda la documentazione.

Capitolo 12

Architettura dell'applicazione

L'architettura dell'applicazione è un aspetto molto importante per la nostra applicazione, per garantire che sia robusta, testabile e manutenibile. Un'architettura (pezzi di codice con precise responsabilità) ben fatta permette di scrivere codice più pulito, mantenibile e testabile. Inoltre, permette di dividere il lavoro tra più persone in modo più efficiente.

Android ci facilita nel nostro lavoro perché mette a disposizione un insieme di librerie e componenti per creare un'architettura solida.

12.1 Cos'è un'architettura SW?

Definisce come il sistema (che sarà sviluppato secondo questa architettura) è strutturato, in che modo i suoi componenti e connettori interagiscono tra loro (rendendoli **compatti/coesi**, ovvero che di base si preoccupa di una sua sola responsabilità e solo di quello, e **lascamente connessi**, ovvero che ho poca interazione fra i componenti o che le loro dipendenze siano al minimo per non impattare sugli altri) attraverso *interfacce* e come i dati vengono scambiati.

12.2 Principi di base della programmazione

Parliamo di **S.O.L.I.D.**:

- **S**ingle Responsibility Principle: ogni classe dovrebbe avere una sola responsabilità.
- **O**pen/Closed Principle: le classi dovrebbero essere aperte all'estensione, ma chiuse alla modifica.
- **L**iskov Substitution Principle: gli oggetti di una superclasse devono essere sostituibili con gli oggetti delle sue sottoclassi senza interrompere il funzionamento del programma.
- **I**nterface Segregation Principle: un'interfaccia dovrebbe essere specifica per i suoi clienti.

- **D**ependency Inversion Principle: le classi dovrebbero dipendere da interfacce e non da classi concrete.

12.2.1 Single Responsibility Principle

Ogni classe dovrebbe avere una e una sola ragione per cambiare (una classe deve essere responsabile solo di un unico aspetto o funzionalità del sistema). Se una classe fa troppo, è difficile da mantenere e testare. Se una classe fa troppo poco, è difficile da riutilizzare.

Garantisce:

- **Testing facilitato**: una componente con una sola responsabilità richiederà meno casi di test.
- **Loose coupling**: meno funzionalità in un singolo componente, meno dipendenze.
-

12.2.2 Open/Closed Principle

Le classi, i componenti, dovrebbero essere aperte all'estensione, ma chiuse alla modifica. Questo significa che dovremmo essere in grado di estendere una classe senza modificarla. **Deve essere closed, non posso permettere modifiche**.

Garantisce:

- **Non modificabilità del codice**: il rischio di introdurre bug è limitato.

12.2.3 Liskov Substitution Principle

A parità di **contratto**, un componente dovrebbe poter essere sostituito senza compromettere il sistema.

Ma cos'è il contratto? Quando abbiamo introdotto a Prog2 le interfacce abbiamo detto che si *istituisce un contratto*.

Garantisce:

- **Supporto all'evoluzione** del software.
- **Supporto allo sviluppo incrementale** (sub).

Si basa sui concetti di ereditarietà e polimorfismo visti a Prog2.

Vedremo fra poco il concetto di **dependency injection**.

12.2.4 Interface Segregation Principle

Un'interfaccia troppo ampia dovrebbe essere suddivisa in più interfacce più specifiche e piccole.

C'è l'esempio dell'interfaccia BearKeeper nelle slide: viola il primo principio di separazione delle responsabilità: viene sostituita da una classe BearCarer che implementa BearFeeder, BearCleaner e BearPetter, ciascuna con le proprie responsabilità.

12.2.5 Dependency Inversion Principle

Si riferisce a

Es. Firebase: non posso usare la formulazione a sinistra perché mi lego **troppo** a Firebase. La formulazione a destra è più corretta perché mi lego solo all'interfaccia ProfileSaver, quando voglio sfruttare Firebase mi affido all'istanza più specifica FirebaseProfileSaver.

12.3 Clean Architecture

Clean Architecture è un'architettura software proposta da Robert C. Martin ("Uncle Bob". Non si sa perché questo soprannome, o perché si firmava così nei primi blog o per il suo carattere affabile) nel 2012. È un'architettura che permette di scrivere codice pulito, mantenibile e testabile. È basata sui principi SOLID e su altri principi di progettazione software.

La Clean Architecture è un insieme di linee guida per progettare l'architettura di un software.

Definisce come partizionare in livelli il software definendo in maniera chiara i confini fra questi.

- Al centro: codice di alto livello (logica pura).
- All'esterno, codice di basso livello.

Codice di base che non dipende dalle architetture progettative.

Quando io strutturo quello che è Entities, me ne frego del rosso che è Use Cases. Questo perché dipende dalla dependency ?:

- Il codice di basso livello (più esterno) può dipendere da quello di livello superiore (più interno), ma mai il contrario.
- Ogni cerchio interno può sapere nulla di qualcosa di un cerchio esterno, cioè il cerchio interno non deve dipendere dal cerchio esterno.

12.3.1 Entities

Questo è il livello centrale dell'architettura e contiene le entità principali del sistema.

12.3.2 Use Cases

Questo livello contiene i casi d'uso che rappresentano i comportamenti specifici dell'applicazione.

Siamo ancora nel **cosa**, non nel *come*.

I casi d'uso coordinano le operazioni tra le entità e gestiscono la logica di business.

Si tratta di logica

12.3.3 Interface Adapters

SI occupa di adattare l'applicazione a elementi esterni, come database, UI, servizi web...

Questo livello contiene componenti come **Repositories**, **Presenters**, **Controllers**, ...

- Repositories è il design pattern che si occupa di traduzione da database a
- Presenters: si occupa di tradurre i dati in una forma che può essere visualizzata dall'utente.
- Controllers: si occupa di tradurre le azioni dell'utente in azioni che l'applicazione può eseguire.

12.3.4 Frameworks and Drivers

Questo livello esterno contiene i dettagli tecnici e le librerie esterne che l'applicazione utilizza.

12.4 Architettura moderna delle app Android

Come anticipato a inizio corso, se l'app che andiamo a progettare non rispetta i principi linee huida che stiamo per andare a vedere e che sono universalmente riconosciuti come buone pratiche, l'applicazione sarà valutata non sufficiente.

12.4.1 Principi alla base da seguire

- **Separazione delle responsabilità**.
- **Drive UI from data model**: UI dovrebbe essere generata e aggiornata in base ai dati contenuti nei modelli di dati separati dalla UI.
  Dato che c'è stretta correlazione tra dati e app, per non legare la presentazione (elementi grafici) ai dati, si utilizzano
- **SIngle source of truth (SSOT)**: i dati possono arrivare sia dalla rete che altre fonti. Io devo sempre fare fluire i dati dal mio daatbase che ho istituito come fonte autorevole e affidabile (trustworthy) di dati. Questo garantisce anche di evitare inconsistenze.
- **Unidirectional Data Flow**: il dato fluisce in una sola direzione. Lo stato fluisce in una sola direzione ($\rightarrow$ UI), mentre gli eventi che modificano i dati fluiscono in direzione opposta.

12.4.2 Tre livelli

L'architettura moderna delle app Android si basa su tre livelli:

- **UI layer**
- **Data layer**
- **Domain layer**

UI layer

Overview

- Il ruolo del livello UI (o livello di presentazione) è:
  - la **visualizzazione** dei dati dell'app sullo schermo
  - l'**aggiornamento** dei dati quando cambiano
- Il livello UI è composto da due elementi:
  - **UI elements**:
  - **State Holders**:

Data layer

Overview

- Definisce le regole (business logic) che determinano il modo in cui l'app crea archivia e modifica i dati.
- Il livello dati è composto da due elementi:
  - **Repository** (con cui noi ci interfacciamo):
    - contengono uno o più data source (**un repository per tipo di dato**)
    - espongono i dati al resto dell'app
    - centralizzano la modifica dei dati
    - risolvono i conflitti quando esistono più data source
    - nascondono la data source (parliamo di astrazione)
  - **Data Sources**:
    - fornisce una sola fonte di dati
    - può essere ..., rete, database locale...

Domain layer

Overview

- Incapsula la logica di business complessa oppure quella più semplice ma usata da più State Holders.

12.4.3 Confronto

12.4.4 Gestione delle dipendenze fra componenti

12.4.5 Dependency Injection

Le classi hanno bisogno di riferimenti ad altre classi. Una classe costruisce la dipendenza di cui ha bisogno.

Soluzione Manuale

La classe riceve le istanze da cui dipende dall'esterno. Ho due modi:

- constructor injection:
- field (get/set) injection:

Service Locator

Sono classi che forniscono le dipendenze di cui una classe ha bisogno.

Si chiamano davvero così. Creano e memorizzano le dipendenze e le forniscono quando richiesto.

General Best Practice

Documento importante da guardare: `https://developer.android.com/topic/architecture#best-practices`

12.5 UI Layer

12.5.1 Introduzione

Sintesi

Il ruolo di un'interfaccia utente (UI)

12.5.2 Stato UI

Definizione

Lo stato UI è l'informazione che l'applicazione stabilisce che l'utente dovrebbe vedere.

Gli elementi UI sono un mezzo per mostrare lo stato.

UI elements + UI states = UI

Evoluzione dello stato: gestione attraverso UDF

Lo stato UI può evolvere nel tempo.

L'evoluzione dello stato UI è gestita attraverso unidirectional data flow (UDF).

Gestione dell'evoluzione attraverso UDF: state holders

Gli state holders sono classi responsabili del mantenimento dello stato della UI e della logica necessaria al suo aggiornamento.

Le classi `ViewModel` sono un esempio di state holders.

Esposizione dello stato: Live Data (o StateFlow)

Lo stato (dati) è mantenuto dagli State Holders (ViewModel).

12.5.3 Eventi UI

Sono azioni che

Due tipi di eventi UI:

- business logic: **cosa** occorre fare a fronte di un cambiamento di stato
  Ha inserito una slide di esempio in cui nel caso di business logic, ci interfacciamo direttamente con il `ViewModel`.
- UI behaviour logic: **come** mostrare il cambiamento di stato

12.6 Domain Layer

12.7 Data Layer

A differenza di ViewModel e UseCase, i repository non sono elementi architetturali, non sono classi astratte, ma concrete.

Slide Architettura: naming conventions. molto importante.

Slide Architettura: molteplici livelli di repository. molto importante.

Slide Architettura: source of truth. molto importante ocio.

Capitolo 13

Esercitazione 6

Oggi vedremo una cosa essenziale per la nostra app: la gestione delle viste in maniera dinamica. Quello che dovremo andare ad avere sarà una lista di carte con collegamenti a servizi API.

- Grid Layout contiene una lista di carte.
  Vogliamo andare a renderlo più dinamico.
  - file PickCountryFragment.java
  - creiamo nella cartella Java il package `util` in cui creo il file java `Constants.java`
  - sull'API delle news (https://newsapi.org/) troviamo la lista dei paesi e delle categorie (in codici che sono costanti e che posso prendere e sbatter nel mio codice): dentro il codice ho delle righe con i codici che avrò per gli stati, poi uguale per le categorie, che andrò ad usare. Sotto avrò delle liste (categorie, drawables delle categorie, codici nazioni, nomi nazioni...)
  - dentro "`strings.xml`" andiamo a creare delle stringhe per i paesi e le categorie: `<string name="countries_name">"..."</string>` per ogni nazione
  - Su `developers.android.com` andiamo a cercare "List View". Ne useremo la logica per andare a creare la nostra Grid View (che ha la stessa logica).
  - L'oggetto che stiamo modellando (tipo la categoria) avrà un codice un nome e un'immagine. Fra la lista e l'oggetto ho l'adapter: ho un oggetto, faccio in modo di metterlo dentro una view.
  - definiamo un package `model` (classi container): dentro la dir `java` vado a creare la nuova classe `Country.java` che sarà l'oggetto che voglio andare a modellare. Avrà un costruttore, getter e setter per nome, codice e immagini.
  - La nostra dir `java` avrà:
    * `model` con `Country.java`
    * `ui.welcome` con `fragment` con `PickCountryFragment` e `PickCategoryFragment` e `LoginFragment`.

Dentro `PickCountryFragment` vado a mettere l'adapter: `gridview.setAdapter(new CountryAdapter(getActivity(), countries));`

* `adapter` con `CountryAdapter.java` che importa:
  - android.content.Context
  - android.view.View
  - android.view.ViewGroup
  - android.widget.ArrayAdapter

  e altre tre righe. Avrà poi un costruttore (class CountryAdapter extends ArrayAdapter¡Country¿) con dentro Context context non nullo e un layout (che fa riferimento al card_country in `layout`) in ingresso e un metodo `getView()` che andrà a restituire la view e in ingresso avrà: int position, View convertView, ViewGroup parent.

– La gridview c'è ma va riempita, poi per modificare semplicemente un'immagine vado in CountryAdapter.java e modifico il metodo `getView()` facendo:
`imageView.setImageResource(countriesList.get(Position).getImage);`

– Dentro Constants.java vado a mettere una lista di oggetti Country:
`public static final ArrayList<Country> countriesList = new ArrayList<>();`
Faccio un `for` per la dimensione di COUNTRIES
`countriesList.add(new Country(context.[i]));`

– Dentro PickCountryFragment.java vado a mettere l'adapter: `gridview.setAdapter();`

– MaterialCardView è un componente di Material Design che permette di creare delle card, faccio:

```
MaterialCardView cardView = (MaterialCardView) convertView;\\
cardView.setOnClickListener(new View.OnClickListener()
\{
    @Override
    public void onClick(View view)
    \{
        //qui dentro metto il codice per andare a fare il click
        Navigation findNavController(view).navigate(R.id.action\_pickCount
    \}
\});
```

– In `util` vado a creare una classe `SharedPreferences` che vado a prendere da `developer.android.com` e che mi permette di salvare delle informazioni in maniera persistente.

* Il metodo `writeStringData` va a scrivere una stringa (ci salveremo Country)
* Il metodo `writeStringSetData` va a scrivere un set di stringhe (ci salveremo Category)
* Poi ho i metodi read.

- Dentro `Country.java` metto tre stringhe dove salvo le chiavi di preferenze, nazione e categoria.
- Andiamo a creare dentro `adapter` un nuovo file `CategoryAdapter.java` che sarà uguale a `CountryAdapter.java` ma per le categorie.
  Credo copia incollato sostiutendo Country con Category.
- Dentro `model` creo `Category.java` che sarà uguale a `Country.java` ma per le categorie.
  Credo copia incollato sostiutendo Country con Category.
- Anche in Constants.java vado a creare una lista di oggetti Category. anche in pickCategoryFragment.java vado a mettere l'adapter, copia incollando ma stando attenti a rinominare correttamente. Tanto hanno la stessa identica logica.
- Ora, voglio poter scegliere più categorie assieme e poi poter confermare. Ha aggiunto perciò un bottone. Questo comunque lo vedo in `card_category.xml` e `PickCategoryFragment.java`. La card avrà lo stato selezionato. Dentro `PickCategoryFragment.java` ho il floatingActionButton che mi permette di andare avanti (= view.findViedById(R.id.floatingActio
- Posso andare ad aggiungere un riferimento alla precisa istanza di un fragment. Lo faccio in `PickCategoryFragment.java` con `private PickCategoryFragment fragment;` e aggiungendolo in ingresso al costruttore.
- Solo se la lista di categorie selezionate non è vuota si accende il bottone (introdotto in `fragment_pick_categrories.xml` in `layout` e trovato sul git di material-components-android). Come? Dentro CategoryAdapter.java vado a mettere fragment.tryEnableFloatingActionButton();
  e dentro PickCategoryFragment.java creo il metodo `tryEnableFloatingActionButton()` che abilita il bottone se la lista non è vuota.
- Dentro PickCategoryFragment.java vado a creare il metodo `floatingActionButton.setOnC` in cui vado a salvare le categorie selezionate tramite i loro codici e a navigare verso la prossima schermata.
- aggiungo un intent in PickCategoryFragment per andare alla prossima schermata (HomeActivity).

13.1 L'activity Home

Non la mettiamo in ui.welcome ma in ui direttamente.

Ricorda che viene sempre valutata molto nel progetto la struttura: la ui nella cartella ui, i layout nella cartella layout etc.

IN questa activity avremo una barra sotto con 4 schede: Scelte in base alla selezione, Top Headlines (più recenti), Ricerca (per keyword o argomento) e credo Preferiti. Sono tutti servizi offerti dall'API delle notizie. Useremo di material-components-android il BottomNavigationView.

Ora la nostra struttura sarà:

- `ui`
  - `home` con `HomeActivity.java`

Dentro `res` dentro `menu` (dir che se non esiste creo) creo `home_menu.xml`. Dentro avrò una lista di item con id e titolo e icona (l'ultima non essenziale).

Importa l'icona vettoriale delle notizie.

L'idea è che l'activity abbia sotto la barra con i 4 contenitori e sopra il fragment.

Dentro `navigation` (dir sotto a mipmap, dovrebbe avere già `nav_graph.xml`) creo `home_nav_graph.xml` che è il file di navigazione, dentro al quale col simbolino che ho in alto (rettangolo con il +) aggiungo i 4 fragment con id "preferenceNewsFragment", "popularNewsFragment", "searchNewsFragment" e "favoriteNewsFragment". Avrò i 4 xml dentro layout.

Per fare la toolbar in alto dentro `activity_home.xml` metto un `Toolbar`: `androidx.appcompat.widget.Toolbar`.

Ora posso navigare fra le schede (non si vede perché i fragment sono tutti uguali e comunque ancora vuoti, next time).

Capitolo 14

Android Architecture Components

La volta scorsa abbiamo visto architettura generica, S.O.L.I.D. e Clean Architecture.

Android sfrutta quella detta Modern, che è un'evoluzione di Clean Architecture: tre livelli: UI, Domain (opzionale) e Data. Poi abbiamo visto come si relazionano alla Clean Architecture.

Come abbiamo detto più volte, saper strutturare bene è il cuore dello sviluppo di un'applicazione, e la struttura della Modern Architecture è la base per l'inizio della valutazione della nostra app. Importante quindi che sia strutturata su 3 livelli, con i dati che fluiscono dal Data Layer al UI Layer e gli eventi che fluiscono dal UI Layer al Data Layer.

Android ha introdotto delle librerie per aiutarci a fare questo, che sono le Android Architecture Components. Sono incluse in Jetpack.

SI parla di first class object: si vede a livello programmativo.

La slide "Architectural Component: Overview" contiene tutto ciò che abbiamo bisogno di sapere:

- `Activity/Fragment` si occupano
  - **Condivide** il suo stato sttraverso un **LifeCycle** che è un oggetto osservabile.
    Esistono i pattern Observer e Observable, il cui meccanismo è sfruttato a livello delle classi "basiche" (non quelle di dominio applicativo che devo sporcare il meno possibile) per osservare il ciclo di vita, quindi i cambiamenti di stato, di activity e fragment.
  - **Osserva** i *LiveData* (wrapper di un dato) dal *ViewModel* al fine di aggiornare la UI (activity/fragment sono Observer, attendono notifiche da LiveData).
- `ViewModel` sono contenitori di informazioni di dominio, di cambiamenti di stato. Un ViewModel mantiene i dati (memorizzati in LiveData, elementi osservati) che popolano la UI e ne gestisce la logica. Si interfaccia con il Repository per l'accesso ai dati (**non sapendone la sorgente**: è lo strato intermedio, fornisce delle primitive di Model che vengono poi usate ma non sanno da dove arrivano):

  - **Sopravvive** ai cambi di configurazione (verticale/orizzontale, etc. . .)
  - **Comunica** con il *Repository* per ottenere/aggiornare i dati
  - Altro **mezzo di comunicazione** tra fragment
    N.B.: ad ogni activity è associato un ViewModel (per tipologia): questo permette che .
  - I LiveData **osservano** il *LifeCycle* della Activity/Fragment così da aggiornare solo quando Activity/Fragment sono *active*

- `Repository` è un layer che si occupa di recuperare i dati da tutte le fonti disponibili (i dati vengono resi disponibili al *ViewModel* in termini di *LiveData*).

- `Room` mi sono persa risenti.

- `Retrofit` è una libreria API messa a disposizione da Android che facilita l'accesso a dati remoti.

14.1 Lifecycle-Aware Components

Motivazioni

A seguito di un cambiamento di **stato** dei componenti Android occorre effettuare delle **azioni di risposta**. Di norma sono codificate nei metodi del ciclo di vita del componente.

Ci sono i Lifecycle Owner e i Lifecycle Observer. Devo mantenere però la separazione fra UI e logica applicativa (Separation of Concerns). Saranno i Lifecycle-Aware Components a fare da tramite notificando i cambiamenti di stato ai componenti interessati.

Introduzione ai 3 componenti principali

Activity/Fragment è il Lifecycle Owner.

Interfaccia LifecycleObserver e classe DefaultLifecycleObserver che la estende.

Lifecycle

Stati di Lifecycle

LifecycleOwner

`LifecycleOwner` è un'interfaccia a metodo singolo che indica che la classe che la implementa ha un Lifecycle.

La classe dovrà

LifecycleObserver

Funzionamento

14.2 LiveData

Cosa sono

Rientrano in quanto presentato perché si basano sul meccanismo del LifeCycle e delle notifiche.

La classe LiveData è un "contenitore" di dati osservabile (**observable**).

Notifica gli observer quando i dati cambiano di valore.

A diffferenza di un normale observable, LiveData è lifecycle-aware:

- sa quando un observer è attivo o meno, rispetta il ciclo di vita di un *LifecycleOwner* a lui associato.

- ciò garantisce che LiveData notifichino gli **observer** di un cambiamento del valore del dato solo quando il *LifecycleOwner* associato si trova in uno stato attivo del ciclo di vita (`started` o `resumed`).

onChanged() metodo che invoco quando effettivamente c'è stato un cambiamento di stato.

Lavorare con i LiveData

Funzionamento

Esempio di implementazione

Esempio di aggiornamento

Vantaggi

14.3 ViewModel

Cosa sono

Gestione de

Persistenza dello stato della UI

Esempio di implementazione

ViewModel onSaveInstanceState()

Mezzo di comunicazione tra fragment

14.4 Android Architecture Components: schema complessivo

LiveData

ViewModel

14.5 Fetch Data

14.5.1 Data Layer

Dobbiamo strutturarlo in modo che i livelli superiori non sappiano da dove arrivano i dati. Parliamo di Repository.

14.5.2 Room

Esempio

Capitolo 15

Services

Un Service è un componente (uno dei 4 fondamentali di Android, un insieme di API dette "work-manager") che esegue operazioni in background senza fornire un'interfaccia utente. Possono persistere se programmati bene. Altri componenti possono avviare un service e interagire con esso anche se l'applicazione è in background. Un service può essere avviato da un componente (come un'attività) o può essere avviato in risposta a un'evento (come un broadcast receiver).

Un Service fornisce funzionalità ad altre app (ha fatto esempio di backup a fine giornata).

Posso posticiparne l'esecuzione.

15.1 Cos'è un Service

- Un service ***non*** è un **processo** separato.
  - A meno che non sia specificato diversamente, viene eseguito nello stesso processo dell'applicazione di cui fa parte.
- Un service ***non*** è un **thread**.
  - Boh
- Un service fornisce **due** caratteristiche principali:
  - guarda slide

15.1.1 Platform e Custom Services

`Context` fa da tramite con hardware e boh risenti.

15.2 Tipi di services

Foreground

Background

Bound

15.2.1 Avviare un service:

Started

Bound

Started e Bound

15.2.2 La vita del processo

Un servizio viene killato da Android solo quando la memoria è bassa e risenti.

15.2.3 Le basi per la realizzazione

Creazione

onBind() obbligatorio implementarlo anche se è un servizio started.

15.2.4 Il file Manifest

Services da attivare con intent esplicito (cosa voglio che venga fatto, es.: voglio visualizzare una pagina web, chi c'è che me lo può fare?).

15.2.5 Foreground

Un Foreground Service è un servizio di cui l'utente è attivamente consapevole.

Notifiche

Forniscono informazioni brevi e tempestive sugli eventi di un'app mentre non è in uso.

N.B.: è una classe a sé (), non è un'Activity.

Anatomia di una notifica

Azioni

Notification channel: creazione

15.2.6 Started Foreground Service

startForegroundService() mi chiama un servizio foreground, di default avrei un started. startForeground() chiede 2 parametri in ingresso:

- boh

Capitolo 16

Esercitazione 7

L'altra volta adapter e liste da statiche a dinamiche, e navbar.
Oggi: toolbar in alto, dove vogliamo che esca il fragment corrente.

- HomeActivity.java: andiamo a dirgli (tramite `getSupportActionBar(toolbar)`) quale fragment sarà il principale.
- Facciamo collegamento fra navbar sotto e toolbar sopra che farà da contenitore al fragment: `navigationUI.setUpWithNavController(bottomNav, navController)`.
- `AppBarConfiguration` fa da contenitore per i fragment.
- In `home_menu.xml` andiamo a fare il menù laterale creando gli items per ogni sezione dentro a `<menu> ...</menu>`.
- `reqbin.com`: sito per fare richieste HTTP e API simulator and tester.
- Ha creato un po' di classi con le info di un articolo, autore, titolo, etc.
- Ha fatto un JSON (che mette poi nella dir `assets`) "`sample_api_response.json`" che simula la risposta di una API.
- Libreria GSON: data una stringa JSON, la trasforma in un oggetto Java. Ha i suoi metodi (che sono statici, no need di inizializzazione) per fare il parsing.
- In `util` crea una cartella `JSONParserUtils` e dentro una classe `JSONParserUtils.java` in cui mette il codice per fare il parsing del JSON.
- Per visualizzare la lista di articoli creata con il JSON, non usando la ListView dell'altra volta (li visualizza tutti uno sotto l'altro), ma usando il RecyclerView. Nasce apposta per risolvere questo problema, creando liste più dinamiche. All'inizio crea un tot di oggetti, quando vede che uno di questi non è più visibile, lo rimuove e lo mette in fondo e aggiorna i rimanenti visibili nello schermo. Non ne istanzia di nuovi!!
- Hanno chiesto differenze fra ListView e ScrollView: la ListView è una lista di elementi (messa in un adapter) che si scrollano, mentre la ScrollView è un contenitore che si scrolla.

- Comunque, per usare il RecyclerView, bisogna aggiungere la dipendenza in `build.gradle` e creare un adapter che è leggermente diverso da quello già creato per Country per la ListView.
- Invece di creare es. 10 card, crea es. 10 viewHolder che contengono informazioni di un articolo.
- `getItemCount()` conta le view visualizzabili. Sul sito solito per developers con cose utili di Android, c'è il codice per creare un adapter per il RecyclerView. In particolare il pezzo per sostituire una view quando questa finisce fuori dallo schermo.
- Va a creare un layout, poi la lista degli articoli. La materialCardView che va a prendere ha due LinearLayout, uno per l'immagine e uno per il testo in cui avrà titolo, autore e quanto è passato dalla pubblicazione. Ci sono anche due checkbox nel LinearLayout del testo: uno per il like e uno per impostazioni (?). Per il primo va ad implementare un selector per vedere se selezionato o no (due drawable diversi).
- `onBindViewHolder(ViewHolder viewHolder, final int position)`: funzione che lancio quando voglio riempire una card, prendendo l'oggetto in posizione `position` ovvero la categoria e riempiendo i campi della card con le info dell'oggetto.
- Un'altra cosa che si può fare in una RecyclerView è aggiungere un separatore tra le card. Si può fare creando un drawable che rappresenta il separatore e poi andando a settare il separatore nel layout della card.
- In fragment_preference_news.xml ha messo un padding per separare le card.
- Per dire se volessi salvare in locale gli articoli "favorite": fai riferimento a "save data in a local database using Room" su `developers.android.com`, Room è una libreria. Come? Con un DB in cui andiamo a salvare gli articoli preferiti. Va a mettere la prima riga del codice versione 2.6.1.
- Diagramma dell'architettura di Room: next esercitazione.
- In sample implementation c'è il codice per andare ad aggiungere annotations agli articoli.
- Aggiunge import: `import androidx.room.Entity;` e `import androidx.room.PrimaryKey;`. La chiave primaria è l'uid che ho aggiunto all'inizio dell'entity, ovvero della classe.
- Creo package "database" dove vado a mettere l'interfaccia ArticleDao.java con le relative import: `import androidx.room.Dao;`, `import androidx.room.Delete;`, `import androidx.room.Insert;` e `import androidx.room.Query;`.
- In `ArticleDao.java` vado a creare le query per inserire, cancellare e selezionare articoli:
  - `Insert` per inserire un articolo

- `ArticleDatabase.java` è la classe (che vado a creare nel package "database" che ho creato prima) che estende RoomDatabase e contiene le istanze di ArticleDao.

- Sempre su `developers.android.com` sotto "Usage" c'è il codice per creare un database Room: creo una classe che implementa l'interfaccia di prima, con una funzione che esegue la mia query. Room si occupa di definire lui il codice per eseguire la query, nascondendolo all'utente.

- Dentro Usage però vedo che usa il nome del database, perciò in `Constants.java` vado a creare una costante che contenga questo nome.

- La connessione al database sarebbe meglio farla **asincrona**, quindi creare una view per poi aggiornarla quando ritornano gli output delle query. Noi però facciamo "allowMainThreadQueries().build()" che permette di fare query sul main thread, ma non è una buona pratica.

- Nell'adapter della RecyclerView, vado a salvarmi cose utili quali titolo e autore. Quando clicco sulla checkbox del like, però, abilitandola vado a salvare l'articolo nel database Room. Per farlo, devo creare un'istanza del database Room e poi chiamare la funzione per inserire l'articolo (`insert()`). Nell'adapter va anche ad aggiungere un listener per il click sulla checkbox. Quindi dentro a `onBindViewHolder()` va a settare il listener, con parametri un compoundButton e un booleano. Dentro a questo listener, se l'istanza già esiste nel database, la richiama, altrimenti la aggiunge. In ingresso ha messo `viewHolder.textViewAuthor.getContext()` per prendere il contesto e `newsDao().insertAll(articleList.get(position))` per inserire l'articolo.
  Dentro a `Article.java` ho delle stringhe che sono considerate tipi primitivi quindi quando faccio l'SQL per le query del DB non ho capito.

- Invece nell'else del listener, se l'articolo è già nel DB, lo rimuove. Per farlo, va a creare un'altra funzione in ArticleDao.java che prende in ingresso un articolo e lo rimuove. Poi va a chiamare questa funzione nell'else del listener.

- Per visualizzare la pagina con solo gli articoli salvati tra i preferiti, andiamo sempre ad usare una RecyclerView quindi copiamo quanto fatto. Però, non voglio il JSON quindi vado a togliere il parser e il try catch.

- Vado a fare la lista degli articoli (che non gli interessa se arriva dal JSON, dal DB o online) con la funzione dell'interfaccia "`getAll()`". Per farlo, vado a creare un'altra funzione in ArticleDao.java che mi ritorna tutti gli articoli. Poi vado a chiamare questa funzione nel fragment che visualizza gli articoli preferiti.

- Ha definito una query "custom": quando avvio l'app voglio che il DB venga svuotato. Perciò: `@Query("DELETE FROM article")` che cancella tutti gli articoli. Poi va a chiamare questa query nel fragment che visualizza gli articoli preferiti.

- Poi ha rinominato i label in `home_nav_bar.xml` e va a prendere i label da `strings.xml`.

- L'ultima modifica l'ha fatta andando a dire dentro `favorites_fragment.xml` che quando si istanzia un articolo, lo devo istanziare senza il drawable del like (perché è già stato messo nei preferiti). Dentro l'adapter della RecyclerView, va a settare la visibilità del like dicendo "se visibilità è visible, allora tolgo il drawable del like, altrimenti no". Communque è sempre lo stesso oggetto, non c'è bisogno di creare un nuovo layout, semplicemente in uno nascondo il cuore.

La prossima volta: non più file locali ma andiamo a recuperare i dati direttamente dal servizio, con uso sensato dell'API. Ricorda che deve essere **asincrona**, l'UI non si deve bloccare mentre aspetta la risposta. Poi vedremo come impostare il progetto in base all'architettura.

Capitolo 17

Esercitazione 8

L'ultima volta ArticleRoomDatabase, ArticleDAO, JSON degli assets...

Per quanto riguarda oggi, facciamo riferimento alla scheda Guide to app architecture per andare a parolare dell'srchitettura dell'app, seguendo la Modern App Architecture vista a lezione.

- **UI Layer**: tutto ciò che ha a che fare con fragment e activities.
- **Data Layer**: composto da repositories e data sources.
  Quello che mi interessa è creare un "ponte" fra PreferenceNewsFregment e JSONParserUtils, questo verrà fatto tramite una repository come quelle d'esempio presentate al link di prima sotto "Naming conventions in this guide" in "Data Layer".

Andiamo ad adoperare RetroFit: libreria per fare richieste HTTP.

L'idea è avere un `ArticleRepository` che è un'interfaccia. Andremo ad usare un'altra interfaccia (un'istanziazione della prima) detta `ArticleMockRepository` che andremo ad usare per debugging. Poi `ArticleAPIRepository` che andrà a fare le richieste HTTP. Infine, `ArticleDBRepository` che andrà a fare le query al DB. Oggi ci limiteremo alle prime due.

Definiremo due modalità di lavoro in base a quello che vogliamo andare a fare: `ArticleRepository` e `ArticleMockRepository`.

Domanda in aula: ma fra `PreferenceNewsFregment` e `ArticleRepository` non dovremmo mettere un ViewModel? Sì, ma la prossima volta.

`ArticleRepository` andrà ad appoggiarsi su `ServiceLocator` che serve a creare le istanze delle repository. Questo è un pattern di design che permette di creare un'istanza di una classe in base a un'interfaccia. In questo caso, `ServiceLocator` avrà un metodo `getArticleRepository()` che restituirà un'istanza di `ArticleRepository` e un metodo `getArticleDB()` che restituirà un'istanza di `ArticleDBRepository`.

Ricordiamoci che deve essere asincrona, e offline first: vedi `https://developer.android.com/topic/architecture/data-layer/offline-first`.

Al lavoro! Cominciamo creando un'interfaccia `IArticleRepository` con i metodi che ci servono: `fetchArticles(String country, int page, long lastUpdate())` è uno, `getFavoriteArticles()` è un altro. Sono entrambi `void`.

Poi creiamo l'interfaccia `ResponseCallback` che ci servirà per fare il callback quando la richiesta è completata. Questa ha un metodo `onCreate(List<Article> articles)`, un metodo `onSuccess(List<Article> articlesList, long lastUpdate)`.

Per mentre fa la richiesta, ha messo dentro fragment_preference_news.xml un Progress Indicator che sposta poi al centro.

Creiamo la classe `ArticleAPIRepository` che implementa `IArticleRepository`. Questa ha un costruttore che prende in ingresso un `ResponseCallback` e un `Context`. Poi ha un metodo `fetchArticles(String country, int page, long lastUpdate)` che fa la richiesta HTTP. Per farlo, crea un'istanza di RetroFit, poi un'istanza di `ArticleService` che è un'interfaccia che definisce i metodi per fare le richieste. Poi fa la richiesta HTTP e, se va a buon fine, chiama il metodo `onSuccess()` del `ResponseCallback`.

Alla seconda lezione avevamo visto un file `local.properties` che non viene caricato su git ed è quindi individuale e permette a tutti di avere la propria chiave senza creare conflitti. Dentro ci metto:

```
sdk.dir=/Users/sara/Library/Android/sdk

debug_mode=false
newsap_key= ...
```

Poi non ho capito. Ma dice che non è importante capire cosa facciano. Praticamente dice in caso di compilazione Gradle di aggiungere anche quella roba che ha scritto.

Dentro il fragment di PreferenceNewsFregment.java, va a creare un'istanza di `ArticleAPIRepository` e chiama il metodo `fetchArticles()`. Crea il `onCreate` con in ingresso

Creiamo un nuovo package: `service`, che ci servirà per le chiamate API (sarebbe il `ServiceLocator` dello schema iniziale). Dentro creiamo un'interfaccia `ArticleAPIService` che definisce i metodi per fare le richieste. Dentro ci fa le query per le richieste HTTP.

```
@GET(Constants.TOP_HEADLINES_ENDPOINT) {
    Call<NewsResponse> getTopHeadlines(
        @Query("country") String country, //???
        @Query("apiKey") String apiKey
    );
}
```

Praticamente ha detto che GSON fa da tramite tra JSON e RetroFit.

Torniamo in `PreferenceNewsFregment`. Dentro onCreateView, dopo aver creato la view, andiamo a fare `ArticleRepository.fetchArticles(country, page, lastUpdate)`.

Torniamo in `ArticleAPIRepository`. Vogliamo gestire le callbacks quindi nella classe ArticleAPIRepository che implementa l'interfaccia, fa il costruttore con in ingresso `applications` e `callbacks` e poi i metodi void `fetchArticles(String country, int page, long lastUpdate)` e `getFavoriteArticles()`. Questi metodi vanno a fare le richieste HTTP e a gestire le callbacks.

Dentro `fetchArticles` ha messo un `if` che controlla se il currentTime - lastUpdate è maggiore di ?boh?, in caso affermativo fa la richiesta, altrimenti no.

Nel caso ci siano tante richieste dico a cosa dare priorità con `.enqueue`.

N.B.: quando vai a prendere librerie da internet, devo dirlo nel Manifest per "chiedere il permesso" al sistema operativo di poter usare internet.
Per farlo, devo mettere `<uses-permission android:name="android.permission.INTERNET"/>`.

Dentro `ArticleAPIRepository` creiamo void saveDataInDataBase(List¡Article¿ apiArticles) che salva gli articoli nel DB.
Per farlo, crea un'istanza di `ArticleRoomDatabase` e chiama il metodo `articleDao().insertAll(arti`

Fa un for per scorrere gli articoli: vedo se c'è corrispondeza DB e locale, se non c'è li aggiungo.

Quando inserisco articoli nel DB, assegna id ad ogni articolo ricevuto dall'API. Li aggiorno settando il liked in base a quelli che ho nel DB, poi li unisco.

Dopo che lancio fetchArticles e mette in coda, quando riceve risposta scarica lista articoli e li dà a saveDataInDataBase. Poi chiama onSuccess del callback.

Va ad aggiungere una RecyclerView che visualizza gli articoli. Per farlo, crea un adapter e un ViewHolder. Poi prende la lista locale di articoli, la svuota e aggiunge tutti quelli ricevuti. Avendo cambiato la lista di articoli, segue che devo fare una modifica anche alla UI sul thread main: lancia `requireActivity().runOnUiThread(() -> adapter.notifyDataSetChanged())`.
Poi mette la visibilità della ProgressBar a GONE.

Lancia articleRepository.fetchArticles("...", ..., ...) e return view.

L'ultima volta avevamo visto i liked articles. Dentro l'adapter del ArticleRecycler c'è un bool liked che tramite un listener sul cuore cambia il valore andando a modificare il DB.

Perché sia offline first dentro saveDataInDataBase vado a salvare i dati che aggiornerò quando possibile. Se non ho internet, vado a prendere i dati dal DB.

Prossima volta avremo anche un tempo dall'ultima richiesta API: se è passato troppo poco tempo, non vado a fare un'altra richiesta ma prendo i dati dal DB. Se è passato troppo tempo, vado a fare la richiesta. L'offline first è anche dato dalla struttura che abbiamo creato con le repositories oggi. Comunque, nel caso di offline first ho nel DB tutti i miei articoli e quando faccio la richiesta API, se non ho internet, vado a prendere i dati dal DB. Se ho internet, vado a fare la richiesta e aggiorno il DB.

Potrebbe capitare un errore tipo "Room couldn't verify the data integrity" che è un errore di Room che dice che i dati che ho nel DB non sono quelli che mi aspetto. Questo può capitare se ho cambiato la struttura del DB e non ho aggiornato il DB. Per risolverlo, posso andare a cancellare il DB e ricrearlo. Per farlo, vado a disinstallare l'app e reinstallarla. Questo è un errore che capita spesso quando si lavora con Room.

Prossima volta vedremo come avere immagini in mod asincrona (offine, senza internet) e come usare l'app in quella modalità facendo apparire la barra in alto che dice "no internet connection".

L'importante è avere capito il concetto "UI –¿ Data Layer –¿ Repository –¿ Data Source –¿ metto i dati presi dalle sources nel repository e poi li passo alla UI".

Vedremo anche come fare il ViewModel e come fare il Repository con il ViewModel.textttdatabaseWriteExecutor.execute() che praticamente quando possibile, quando riesce, va a fare gli aggiornamenti necessari.

Capitolo 18

Rivedi Inizio

Ha parlato di:

- Main Thread
- Background processing
  Parte importante della creazione dell'app Android che sia **reattiva** per gli utenti e che sia un **buon cittadino** della piattaforma Android.

18.1 Principio

- Opportuno eliminare qualsiasi **attività bloccante** dal **thread** dell'UI.
- Attività bloccanti più comuni:
  - decodifica di una bitmap
  - accesso allo storage
  - esecuzione di modelli di ML (machine learning)
  - boh

È importante capire

18.1.1 Come scegliere la soluzione giusta

- **Il tempo di inizio e fine task devono essere precisi?**
- **Il task è interrompibile?**
- **Il task può essere rinviato o deve essere svolto subito?**
- **Il task dipende dalle condizioni del sistema?**
- **Il task comporta la raccolta o l'utilizzo di dati sensibili dell'utente?**
  Esempio tipico, geolocalizzazione. Mi avvalgo di foreground services (quelli che mi avvisano l'utente).
- **Il task deve essere eseguito in un momento specifico?**

18.2 Tipi di task in background

Il task in background rientra in una delle tre categorie principali:

- *Immediate*: il task deve essere eseguito subito e completato in tempi brevi.
- *Di lunga durata*: può richiedere del tempo per essere completato.
- *Differibile*:

18.3 Approcci al background work

Task persistenti e non persistenti vanno gestiti in maniera differente:

- **Persistenti:**
  - Tutti i task persistenti vanno gestiti con il **WorkManager**.
- **Non persistenti immediati** (aka, asynch work):
  - *Kotlin*: si usano le **coroutines Kotlin**.
  - *Java*: si usa il **threading**.
- Mi sono persa

18.3.1 Alarms

Caso d'uso speciale che non fa parte del background work.
Si usano
Doze Mode.

18.3.2 Doze mode

Android 6.0
Android 7
Android 8

Capitolo 19

Asynchronous Work

19.1 Caratteristiche

Si parla di work asincrono e persistente. Hanno luogo in background.

- **Work asincrono**
  - si svolge
- **Work persistente**
  - si

19.2 Java and Kotlin

Il modo in cui gestire il lavoro asincrono dipende da quale linguaggio si usa per sviluppare l'app. Abbiamo detto che Java usa Java threads, mentre Kotlin usa coroutines.

Non vedremo altro su Java threads, in quanto sono stati già affrontati nel corso di Sistemi Distribuiti.

Capitolo 20

WorkManager per

20.1 WorkManager: lavoro persistente

20.1.1 Introduzione

Annunciato a
WorkManager è una libreria che facilita la schedulizzazione

20.1.2 Tipi di work persistenti

Supporta 3 tipi di work (immediate, long running, deferrable). Permette di gestire:

- Work one-shot (o one time)
- Periodici

Buona norma: usare foreground services.

20.1.3 Caratteristiche principali

- **Work constrains**
  - Definizione di condizioni entro le quali il task può essere eseguito.
- **Robust scheduling**
  - Pianificazione dell'esecuzione del task.
    Una volta (`OneTimeWorkRequest`) o ripetutamente (`PeriodicWorkRequest`).
  - Task etichettato per una più facile gestione.
- **Expedited work**
  - è possibile utilizzare
- **Flexible retry policy**

20.1.4 Boh

20.1.5 Relazioni con altre API

20.2 WorkManager

20.2.1 How-to

Vedremo come:

-

20.2.2 Definire cosa il task deve fare

20.2.3 Configurare come e quando eseguire il task

20.2.4 Consegnare il task al sistema

20.2.5 Stati del work

20.2.6 Personalizzare una WorkRequest

Personalizzare le WorkRequest per:

1. Specificare dei vincoli per l'esecuzione dei task.
2. ...
3. ...
4. ...
5. ...

Vincoli

Per default il task parte subito.

Si possono impostare alcune regole che specificano quando il task dovrebbe essere eseguito.

Si aggiunge un `Constrains` per

Boh

Boh

Input e output

Un task può necessitare di dati in ingresso e/o può produrre dati in uscita.

20.2.7 Work Immadiato

20.2.8 WorkService e

Capitolo 21

Testing delle app

21.1 Motivazione

Le app mobili sono molto diverse da quelle desktop o web.

Perciò serve definire casi d'uso e

21.2 Vantaggi

è parte integrante del processo di sviluppo si verifica correttezza, comportamento funzionale e usabilità. Altri vantaggi: individuazione tempestiva dei fallimenti nel ciclo di sviluppo (non fare tutti i test alla fine!!!) refactoring del codice più sicura senza preoccupazioni delle regressioni

21.3 Tipi di test in Android

21.3.1 Boh

- **Test funzionali**
- **Test delle prestazioni**
- **Test**

21.3.2 Portata del testing

I test variano in base alla **dimensione** o al **grado di isolamento**.

I **test di unità** (visti a prog2) o *small test* verificano solo una parte del sistema.

I **test di integrazione** verificano come le parti del sistema interagiscono tra loro.

Es.: si assuma che un bottone di una activity è usato per lanciare una seconda activity:

- Un **test di unità** verifica l'intent
- Un **test di integrazione** verifica che il bottone lanci l'activity

I **test end-to-end** o *big test* verificano il funzionamento del sistema in un possibile scenario di utilizzo.

I **test UI** verificano il funzionamento dell'interfaccia dell'app. Sarebbe del tutto appropriato eseguire questo testing con un **backend simulato**.

Es.: E2E vs UI testing

- Un **test E2E**
- Un **test UI**

21.3.3 Dove testare

I test possono essere eseguiti in 4 contesti diversi:

1. **Dispositivo fisico**
    - Offrono massima fedeltà ma
2. **Dispositivo virtuale** (come l'emulatore di Android Studio, o Genymotion)
    - Offrono
3. **Dispositivo simulato** (come Robolectric)
    - Forniscon
4. **Macchina/Server di sviluppo**
    - Offrono

21.3.4 Instrumented vs local

21.3.5 Cosa testare

Una buona strategia di testing trova un giusto equilibrio tra la **fedeltà** di un test e la sua **velocità** ed **affidabilità**.

21.4 Architettura testabile

21.4.1 Introduzione

21.4.2 Approcci al disaccoppiamento

Fa mi raccomando di dividere l'app in layer di domninio

21.4.3 Come testare

21.4.4 Cosa testare

La scelta di quali test E2E e di integrazione

21.4.5 Unit test

Casi limite

La scelta di quali test E2E e di integrazione

Quelli da evitare

Alcuni test di unità dovrebbero essere evitati a causa del loro scarso valore:

- test che verificano il corretto funzionamento del framework o di una libreria, **non il mio codice**
- test che

Quelli da includere

Alcuni test

21.4.6 UI test

Quelli da includere

Alcuni test

21.4.7 Digressione: i test double

Usando i test double, Robolectric va a simulare (esecuzione dummy dei componenti Android) senza aver bisogno di un dispositivo fisico o emulatore. Le prestazioni sono quelle che sono ma mi va bene.

21.5 Strumenti per il testing

21.5.1 Test Locali

Strumento

Il test locale è un test che viene eseguito sulla JVM del computer di sviluppo.
La creazione di test locali è supportata da JUnit.

21.5.2 UI Test

Strumenti

Espresso è

Altri Strumenti

- Espresso Test Recorder
- App Crawler
- Monkey
- Appium

- Selenium e Calabash
- Con BrowserStack automatizzo i test su diversi dispositivi Android

Capitolo 22

Esercitazione 9

22.1 L'ultima volta

- Linee guida su architettura
- Repository
- Come DB
-

22.2 Oggi

- Ricorda: noi vogliamo fare un'app offline first.
- Single source of truth: due fonti di informazioni diverse non vanno bene perché la chiamata va da DB a UI. Il DB è la fonte di verità. Questo mi garantisce affidabilità e consistenza. Ci saranno momenti in cui UI e DB non sono allineati e questo è da evitare.
- Data Layer abbiamo detto che vuole il dato senza fregarsene di da dove viene. Il data layer è il ponte tra il DB e il repository.
- Come istante di attesa quando apri una sezione di notizie, ha fatto un placeholder che ha un'animazione. L'ha messo nel `LinearLayout`.
- La `articleRepository` avevamo detto la scorsa volta che era il ponte fra DB e UI.
- Le risposte vanno nel metodo `onSuccess` che viene definito nel file `ResponseCallback.java` Quando chiamo questo metodo lancio la chiamata a `lastUpdate`, un long. Vado a vedere la differenza con `currentTime` e se questa è troppo grande vado a fare un refresh.
- Va a fare una chiamata ad un service che ha preso dalla documentazione dell'API delle news.
- Poi ha aggiunto un metodo per accettare l'HTTP del client perché apparently non funziona (dentro `ServiceLocator.java`).

- Tutte le chiamate di accesso a DB e API vengono gestite da repository.
- Oggi introdurremo il concetto finora accantonato di **ViewModel**. Siamo nella pagina "About the UI layer" nel sito GoogleDev solito. Ad azione dell'utente, il Data Layer fa una chiamata al ViewModel che va a interpellare la UI la quale manda una risposta al ViewModel che la passa al Data Layer.
- La nostra struttura sarà:
  1. Popular News Fragment
  2. Article ViewModel (non essenziale ma fa in modo che non sia il fragment che deve gestire le chiamate), raggiunto con ServiceLocator (registro dove le classi possono trovare le proprie dipendenze per non continuare a fare riferimenti etc).
  3. Article Repository (sorta di interfaccia, punto di accesso al data layer)
  4. Due data sources: Base Article Remote e Base Article Local.
     - Il primo è il più lento, va a prendere dati remoti. Si divide in due: Article Remote e Article Mock Data Source, il secondo è quello del testing, il primo è quello che effettivamente andrà a contenere le chiamate API oltre che le chiamate ai DB remoti (interpretazioni diverse della stessa interfaccia).
     - Il secondo più veloce, va a prendere dati locali.
  5. Quindi il flusso va dall'alto verso il basso: ma ora è il momento di rispondere. Comincio a tornare indietro tramite le callbacks.
     N.B.: le callbacks (che mi dicono se c'è stato un errore o no) esistono solamente nel Data Layer. Nell'UI layer non ci sono callbacks, ma LiveData.
  6. I LiveData hanno un meccanismo simile alle callbacks. L'idea è che quello che voglio ottenere (e che deve tornare al Fragment) non è quello che viene tornato, ma viene tornato un *contenitore* con dentro quello che abbiamo chiesto. Es.: voglio una stringa? Mi torna il *LiveData di una stringa*. Perciò questi LiveData invece di definire una stringa mi permettono di definire il ViewModel con dentro il LiveData di quella stringa: posso referenziarla ogni volta che la stringa viene riutilizzata e cambiata di valore.
  7. "Observe the LiveData, passing in this activity as the LyfeCycleOwner and the observer." (dal sito di Google Dev sotto "LiveData"). Come faccio ciò?

     ```
     model.viewCurrent
     usa "this" che è istanza della classe
     ```

- Tornando al nostro progetto, abbiamo una repository che ha due data sources: una remota e una locale. N.B.: le ha **entrambe**.
- La repository fa le chiamate ed è in grado di gestire le risposte (tutte le risposte, ha i metodi onSuccess e onFailure, quindi è in grado di gestire le risposte sia positive che negative).

- Il ViewModelFactory è il livello più esterno, praticamente va a gestire il ViewModel nel caso volessi passargli argomenti in ingresso.
- Noi abbiamo detto però che vogliamo una sola fonte di verità, source of truth. Questa è la Data Source Locale (il DB), quindi in breve: se ho già i dati in locale, non vado a fare la chiamata remota. Se non ho i dati in locale, vado a fare la chiamata remota e salvo i dati in locale. Ovvero:
  - Fragment chiama ViewModel che chiama Repository che chiama Data Source Remoto.
  - Remoto fa callback tornando a Repository che va a salvare l'articolo in Locale.
  - Locale fa callback tornando a Repository che torna a ViewModel che torna a Fragment.
- Per vedere sotto il database, è il simbolino sotto al debugger (se non c'è, View ¿ Tool Windows ¿ App Inspection). (?)
- Last thing: abbiamo detto che un Bundle è un contenitore per passare roba fra fragment e activity etc.
- Cerca cos'è Glide, l'ha detto ma mi sono persa. Si occupa (anche in mod. aereo) di gestire le immagini.
- Prossima volta: Firebase.

Capitolo 23

Esercitazione 10

Oggi vedremo Firebase. Torniamo alla schermata del LogIn che implementeremo con Firebase.

Ricorda che "`intent filter`" mi dice quale activity deve essere lanciata per prima.

Firebase fornisce un server su cui caricare un'autenticazione (login) che un database, più altre funzionalità che al momento non ci servono.

All'avvio dobbiamo registrare la nostra app: chiede nome pacchetto android (lo troviamo nel file `build.gradle.kts` in `gradle`, dentro `android`: è tipo "com.unimib.worldnews, quello definito all'avvio del progetto). Poi cerco il simbolino "gradle" a destra (dovrebbe essere sotto la campanella) e cerco "SHA1". I prossimi passi comunque sono sufficientemente guidati: passo per passo ti dice che righe di codice vuole che tu copi. L'ultimo passo sarà aggiungere l'implementazione di Firebase nel `build.gradle.kts` del progetto, che Android Studio ti dovrebbe chiedere di poter rinominare come vuole lui.

Poi in "Autenticazione" dovrebbe esserci una lista dei metodi di login disponibili: noi oggi vedremo accesso tramite mail e password e poi tramite Google.

Poi dovrò andare ad includere il pacchetto di autenticazione di Firebase nel mio progetto (sempre in `build.gradle.kts`).

Ora scrivendo codice nel resto del progetto dovrebbe suggerirti già metodi di Firebase iniziando a richiamarlo.

Il prossimo passo è verificare se sono già autenticato o no. Se non lo sono, mi chiede di fare il login. Se lo sono, potrò rimandare l'utente alla home. Come?

- Costruisco un oggetto `FirebaseAuth` e chiamo il metodo `getInstance()` che mi crea in automatico una istanza nuova. Se vado a chiamare `.getCurrentUser()` estraggo l'utente. Se non ero autenticato, giustamente mi ritorna null. Così posso dare la possibilità all'utente di fare il login.

- Con mail e password, vado a creare un nuovo utente e aggiungo un Listener. Così posso prendere le informazioni che mi interesserà tenere dell'utente. Chiaramente per motivi di sicurezza, la password non viene salvata.

- Su Firebase, c'è una sezione "Utenti" che mi dà una lista degli utenti registrati. Lo fa in automatico: ha provato a fare il login inserendo una

mail a caso, quella mail me la sono ritrovata nella lista degli utenti su Firebase.

- Ad ogni nuovo utente verrà associato un ID univoco.

E se l'utente fosse troppo pigro e volesse autenticarsi con Google? Not to worry, torno sulla documentazione di FIrebase e aggiungo la dipendenza anche per Google. Ocio che ci sia l'account Google installato sul telefono dell'emulatore di Android Studio prima di lanciare l'app.

Il vecchio modo (che vedremo ora) è deprecato da qualche settimana perché ora Google sta cercando di implementare le Android Credential Manager API.

I passaggi del metodo vecchio li ho un pochino persi perché ha fatto un po' un pasticcio col codice (aveva già salvato il SHA1 e quindi non poteva più usarlo per la lezione, è una cosa una tantum **FAI ATTENZIONE**). Però nel `LoginFragment` va a costruire degli oggetti nell'`onCreate`. Servono a chiedere con un intent delle informazioni che poi andrò a prendere e salvare. Lancio un intent, mi esce una lista di account, prendo il risultato che è quello che mi interessa, le credenziali dell'utente scelto e controllo che non siano null (se finisco su `idToken` sta andando tutto secondo i piani).

Quindi:

- Listener: on click faccio la richiesta e va a `signIn`.
- `signIn` ha un altro listener perché asincrono.
- onSuccess prendo il risultato dell'intent e vado a fare il login.

Un po' antipatica è la cosa che se l'utente è già loggato, viene salvato e quindi se mi serve riprovare il login devo andare a cancellare l'utente da Firebase. O disinstallare e reinstallare l'app nell'emulatore. O metto una variabile `debugMode` che mi dice se sono in debug mode o no ed eventualmente non fa il login come farei normalmente.

Come avevamo fatto per gli articoli, anche qua per gli utenti c'è un'architettura: avrò fragment, viewmodel, repository, data source. Il fragment chiama il viewmodel che chiama il repository che chiama il data source. Il data source è diviso in due: uno remoto e uno locale. Il remoto va a fare il login, il locale va a salvare l'utente. Il repository fa le chiamate e gestisce le risposte e le callbacks.

L'altra volta vevamo parlato di "singola fonte di verità" che valeva per gli articoli ma devono esserci anche per gli utenti.

Chiedo a Firebase se ci sono già informazioni sul mio utente (es. se nell'app delle notizie ha già scelto il paese delle notizie nel database). Sulla console di Firebase vado a creare un database (un enooorme file `.json`) nella sezione "realtime database".

Un'altra cosa comoda di Firebase è che posso andare a prendere il current `loggedUser`: se c'è qualcuno loggato, questo mi darà qualcosa, altrimenti `null`.

In questo file .json posso andare a creare un nodo con il nome dell'utente e metterci dentro le informazioni che mi interessano. Ovvero, ho un nodo principale che **nessun** utente può toccare, questo nodo avrà tantin nodi figli quanti utenti ci sono. Poi posso andare a leggere queste informazioni nel mio progetto.

Siccome c'è una singola sorgente di verità, quando vado ad aggiungere un'informazione al database, devo aggiungere un listener. Ovvero, vado ad aggiungere un'informazione al db, lo aggiungo alle shared preferences in modo da poter

prima fare le operazioni da remoto. Avrò un oggetto "dataSnapshot" che sono spesso usati da Firebase e rappresentano una query del db. Questo oggetto è lo stato dle db su quel preciso nodo su cui ho fatto una "push".

Quando poi ho fatto il login, scelto il country questo non è ancora salvato nel db, è salvato solamente nelle shared preferences. Quando vado a fare il logout, vado a cancellare le shared preferences. Scelte le categorie, con il tasto conferma vado a fare un insert nel db delle mie scelte. Riprendendo la struttura ad albero di prima, ogni nodo avrò le preferenze salvate nel db di ogni utente.

Poi su Firebase avrò una sezione "Dati" in "realtime database" che mi mostra il db in tempo reale. Se vado a fare un insert, vedrò che il db si aggiorna in tempo reale. Qui avrò le shared preferences dell'utente (nell'app delle notizie fatte in aula avrò country e categories scelte, oltre ad una voce "stability" che dovrebbe essere recuperata in automatico dalla UI e che noi andiamo ad ignorare).

Capitolo 24

Note per il mio progetto

24.1 Documentazione

Non male nella cartella del Drive 24/25 "CineMatch" e "TripTales".

24.2 Nome

24.2.1 Proposte

- Bookworm (esiste)
- BookKeeper (bella ma ce n'è già una praticamente uguale)
- PageKeeper
- Marginalia (sembra non esistere)
- Paperheart
- Inkwell (bella ma ce ne sono già due)

24.3 Idea

- libreria virtuale/collector di libri (scelta fra copertina rigida, morbida, tascabile o audiolibro?)
- organizzazione per scaffali e/o sezioni
- API per ricerca in database di libri –¿ Google Book API
- sezione note per pensieri vari su libri e non
- sezione citazione del giorno se possibile metterla random (come co-star) –¿ 2 o 3 API di quotes fra cui scegliere
- traguardo letture