Programmazione Dispositivi Mobili

Sara Angeretti

2024/2025

Indice

Capitolo 1

Introduzione al corso

1.1 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

1.2 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.
   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.

1.3 Il progetto

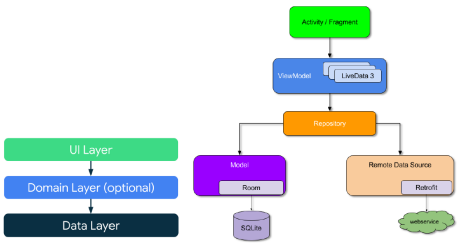
- Durante il corso si porta avanti un progetto che sarà oggetto dell'esame finale
- Il progetto è svolto in maniera paritaria da un gruppo di studenti
  - Composto da almeno 3 persone e fino ad un massimo di 5
  - Eccezioni verranno valutate singolarmente
- Il progetto è proposto dal gruppo (scadenza 18 ottobre 2024)
  - 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 coe 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)

1.3.1 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
- 1-5 per Software, 6 cartaceo
- Uso di GitLab o GitHub

Scelte architetturali



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

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. `https://www.postman.com/`

Uso ORM - Object Relational Mapping

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

- Room (integrato in Android), utile per persistenza
- greenDao

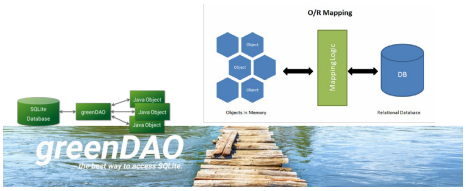
Uso di Firebase

Se ho capito bene, lo vedremo ad esercitazione. Ti aiuta a creare:

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

Layout grafico

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).



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 ... (?)

1.3.2 Discussione progetto

- 24 gennaio, ~09:00
- 21 febbraio, ~09:00

1.3.3 Scadenze

- 11 ottobre 2024: comunicazione gruppo
  - la composizione del gruppo (matricola, cognome, nome), un nominativo per riga;
  - il referente del gruppo.
- 18 ottobre 2024: comunicazione argomento progetto
  - Il titolo del progetto;
  - una descrizone del progetto che si intende svolgere;
  - il referente del gruppo.

Capitolo 2

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)

2.0.1 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.

2.0.2 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.

2.0.3 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

2.0.4 UI & UX

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

2.0.5 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).

2.0.6 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.

2.0.7 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
- ...

2.0.8 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.

2.1 Monetizzazione

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

2.1.1 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.

2.1.2 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

2.1.3 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.

2.1.4 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

2.1.5 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

2.1.6 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