

ANALISI DI PROGETTO

Pokemon by GTPLAY

SOMMARIO

[SOMMARIO 2](#_Toc76754300)

[1 Premessa 3](#_Toc76754301)

[2 Linguaggio 3](#_Toc76754302)

[3 Database 3](#_Toc76754303)

[3.1 Caricamento database non completato 4](#_Toc76754304)

[4 Introduzione e creazione del personaggio 4](#_Toc76754305)

[5 Pattern architetturale : MVVM 4](#_Toc76754306)

[5.1 Filtri 5](#_Toc76754307)

[5.2 ActionMode – Selezione multipla 5](#_Toc76754308)

[6 Dettaglio Pokemon - ViewPager 5](#_Toc76754309)

[6.1 Fragment Stats 5](#_Toc76754310)

[6.2 Fragment Evolution 6](#_Toc76754311)

[6.3 Fragment Move 7](#_Toc76754312)

1. Premessa

L’analisi verrà effettuata evidenziando le scelte progettuali relative agli elementi principali dell’applicazione

1. Linguaggio

Il linguaggio di programmazione usato per lo sviluppo dell’applicazione è Kotlin. La scelta di usare il linguaggio creato da JetBrains, per esempio rispetto a Java, è stata presa per diversi motivi:

* ampio utilizzo di Kotlin per lo sviluppo di applicazioni Android
* più conciso e rapido rispetto a Java
* in grado di interagire e funzionare con Java
* integrato in Android Studio e IntelliJ IDEA
* nessuna conoscenza pregressa di Java da parte dei membri del gruppo

1. Database

La prima scelta progettuale da fare è stata quella della gestione dei dati.

La scelta dell’utilizzo di un database per mantenere le informazioni dei pokemon è stata presa dal gruppo senza troppi ripensamenti. In questo modo abbiamo potuto usufruire delle potenzialità di un database SQLite. Per la comunicazione con il DB è stata utilizzata la libreria Room.

Per quanto riguarda l’acquisizione dei dati da inserire nel DB, le decisioni da poter prendere erano principalmente due:

* Caricare l’intero db subito, richiedendo all’utente una piccola attesa prima di iniziare a usufruire dell’app
* Caricare subito nel db solo delle informazioni fondamentali per l’utilizzo dell’app (come, per esempio, la lista dei pokemon), per poi scaricare il resto delle informazioni quando necessario

La scelta adottata dal team è stata la prima. Il motivo di questa decisione è che abbiamo preferito non dover richiedere all’utente tempi di attesa durante l’utilizzo dell’app stessa, rendendo inoltre l’assenza di internet un elemento non bloccante per gli utilizzi successivi al primo caricamento.

Per questo motivo, all’avvio dell’applicazione nella schermata iniziale di *SplashScreen* viene effettuato un controllo sulla connessione ad internet.

Se il controllo va a buon fine, la *SplashScreen* rimanda alla *MainActivity* dove viene effettuato l’effettivo download del database, accompagnato da una breve procedura di inizializzazione di un personaggio che sarà poi modificabile all’interno dell’app.

Le informazioni sui pokemon sono state acquisite tramite la Libreria *Volley*, con la quale è stato possibile caricare una coda di richieste internet per prendere degli oggetti json direttamente dalle API Pokemon (<https://pokeapi.co>). Le richieste sono generate in successione tramite dei loop che terminano quando sono finiti i dati da richiedere.

Una volta che le richieste vengono generate, la *MainActivity* fa partire un thread che rimane in attesa dell’elaborazione di tutte le risposte internet ricevute tramite Volley.

L’elaborazione dei json ricevuti avviene per mezzo della libreria *Gson* e attraverso le funzioni dedicate per gli oggetti *JsonObject*. La scelta di utilizzo delle funzioni dedicate della classe *JsonObject* è derivante dalla complessa articolazione di informazioni all’interno dello stesso json acquisito dalle API, che ci ha portato a considerare un eventuale utilizzo di *Gson* più complesso.

Inoltre, l’applicazione prevede una procedura di ripristino dati nel caso in cui il database non sia stato caricato correttamente, in modo da gestire la perdita di connessione internet e la chiusura dell’app durante il caricamento.

* 1. Caricamento database non completato

Nel caso in cui il database non venga correttamente caricato (mancanza di internet durante il caricamento o chiusura dell’applicazione durante il caricamento) al prossimo acceso (dopo aver controllato la presenza di una connessione ad Internet), il database verrà completamente ricaricato.

Per verificare l’integrità del database viene effettuata una richiesta Volley con cui si ottiene il numero effettivo dei pokemon esistenti. Se non coincide con il numero di record presenti nel database allora verrà eseguita un’activity ad hoc per il recupero dei dati. In particolare, verrà svuotato il database per poi rieseguire il caricamento del database.

L’activity che si occupa del recupero dei dati è *Update,* il cui compito svolto procede seguendo la linea guida della MainActivity. Se anche il tentativo dell’activity Update non dovesse andare a buon fine, al prossimo avvio verrà eseguito un ulteriore tentativo di caricamento del database.

1. Introduzione e creazione del personaggio

Questa parte introduttiva di selezione del personaggio e inserimento del nome è stata realizzata attraverso l’utilizzo di Fragment inseriti all'interno di un *ViewPager2.*

I *Fragment* sono percorribili in un unico senso tramite il click di un *Button*, il quale è reso cliccabile solo dopo che specifici eventi siano accaduti (come la fine del dialogo della professoressa e l’inserimento dei dati dell’utente).

In particolare

* il *Button* è un elemento che appartiene alla *MainActivity*
* gli eventi che rendono cliccabile il *Button* riguardano i *Fragment* in cui accadono

Questo ha portato il team a prendere la scelta di utilizzare un listener, tramite interfaccia (*IntroToMain*), con cui viene implementata la comunicazione tra *Fragment* e *Activity*. In questo modo la *MainActivity* rimane in ascolto e, appena l’evento si presenta, setta il bottone come cliccabile.

Tutte le informazioni relative all’utente verranno memorizzate in una sharedPrefereces, in modo da poter essere utilizzate dall’applicazione per personalizzare l’esperienza applicativa.

Nel caso in cui l’utente non abbia completato l’inserimento dei dati relativi al personaggio, al successivo avvio dell’applicazione, la MainActivity procederà a ripresentare all’utente l’introduzione, così che possano essere inseriti tutti i dati necessari.

1. Pattern architetturale : MVVM

Con tale approccio architetturale per lo sviluppo della nostra applicazione riusciamo a evitare che le parti di codice che forniscono la logica di business siano ben distinte dalle parti di codice che gestiscono la UI. Le parti fondamentali che costituiscono il pattern sono: il Model, la View, il *ViewModel*.

La *ViewModel* da noi utilizzata è la classe *PokemonVM*, la quale contiene una *MutableLiveData* con una lista di Pokemon come valore. La *PokemonVM* rappresenta la lista dei Pokemon mostrati nella *HomeActivity*, la quale può essere modificata mediante le interazioni dall’utente con l’app.

Sulla *LiveData* andremo a impostare un observer che, ad ogni cambiamento della stessa, reagisce reimpostando l’adapter della recycler view che mostra il pokedex. In questo modo avremo un aggiornamento immediato della lista mostrata in base alle interazioni dell’utente, che possono essere fatte tramite filtri o la selezione multipla per mettere o togliere i pokemon dai propri preferiti.

* 1. Filtri

Sono state messe a disposizione dell’utente diverse modalità con cui poter filtrare la *RecyclerView* contenete la lista dei Pokemon nella *PokemonVM*. Le varie modalità di filtraggio sono selezionabili utilizzando dei *FloatingButton* e la *searchBar* i quali una volta premuti chiamano un metodo della ViewModel che si occupa di aggiornare la *LiveData* in modo opportuno.

Ogni combinazione di filtraggio esegue una rispettiva estrazione dal database.

Una volta che il *LiveData* viene aggiornato entra in azione l’observer, citato precedentemente, che si occupa di aggiornare l’adapter in modo da riportare la modifica anche nella lista visualizzata nella *HomeActivity.*

* 1. ActionMode – Selezione multipla

Nella *HomeActivity* è stata inserita la funzionalità di effettuare una selezione multipla per dare la possibilità di inserire o rimuovere massivamente i pokemon dai preferiti.

Per aggiornare lo stato di preferito del Pokemon sono state implementate due funzioni nella PokemonVM per il set e l’unset dai preferiti.

Invece l’aggiornamento delle carte dei Pokemon nel pokedex e il mantenimento degli item selezionati è un lavoro assegnato al *PokeAdapter.* L’informazione dei pokemon selezionati è mantenuta tramite uno *SparseBooleanArray*, oggetto che perfettamente si adatta al compito.

Dato che la *ActionMode* è invece gestita dalla *HomeActivity,* è stato necessario creare un’interfaccia che permette la comunicazione tra l’adapter e l’activity, per poi passare come parametro all’adapter il relativo listener.

1. Dettaglio Pokemon - ViewPager

Quando viene aperto il dettaglio del Pokemon e quando si clicca sull’immagine del Pokemon, viene eseguito il verso dello stesso prendendolo da internet. Ciò viene fatto attraverso l’oggetto MediaPlayer, passandogli come dataSource “https://play.pokemonshowdown.com/audio/cries/\*nome\_pokemon\*.mp3”.

Per evitare il blocco dell’applicazione quando la connessione dati non è attiva, viene fatto un controllo della connessione prima di ogni riproduzione.

Nel dettaglio di un pokemon è presente una BottomSheet in cui è stata inserita una ViewPager2 con 3 Fragment che mostrano diverse informazioni del pokemon.

* 1. Fragment Stats

Nel Fragment delle statistiche andiamo ad inserire tutte le informazioni riguardo ad uno specifico Pokemon.

La scelta presa per la visualizzazione delle informazioni è stata quella di inserire nella parte alta della schermata le abilità, il peso e l’altezza del Pokemon. Per quanto riguarda le statistiche del Pokemon (punti salute, attacco, attacco speciale, difesa, difesa speciale e velocità) si è optato per l’utilizzo di un Grafico Radar. La decisione di usare questo particolare tipo di grafico per la visualizzazione delle statistiche è stata presa in quanto è il metodo operato nei videogiochi Pokemon. La particolarità di questo tipo di grafico è che, oltre a permettere un’intuitiva rappresentazione dei valori delle statistiche di un Pokemon, consente anche di comprendere il potenziale di un Pokemon avendo a disposizione una visione complessiva della distribuzione delle statistiche.

* 1. Fragment Evolution

Una funzionalità di particolare rilevanza della nostra applicazione è la possibilità di visionare la catena evolutiva di uno specifico Pokemon. Alla base di questa funzione è presente un algoritmo, da noi implementato, che permette di discernere il tipo di catena evolutiva in base al Pokemon.

La difficoltà nel determinare la specifica catena evolutiva risiede nella possibilità che un Pokemon possa essere uno stadio senza evoluzioni o uno stadio base da cui partirà la catena evolutiva o uno stadio intermedio o finale dell’evoluzione. Inoltre, lo stesso Pokemon può avere linee evolutive differenti a seconda, per esempio, del gender o delle particolari statistiche. Oltre al determinare la specifica catena evolutiva, il nostro algoritmo si occupa anche di determinare il particolare metodo evolutivo. Infatti, i Pokemon possono evolversi in diverse modalità (es: livello, strumento, scambio, felicità).

Prima di procedere con la descrizione dell’algoritmo è necessaria una digressione sulla struttura del database in relazione alle catene evolutive. In particolare, il database è strutturato per righe (ogni riga corrisponde ad una specifica catena evolutiva) e per colonne (che indicano la forma base, la prima evoluzione e la seconda evoluzione e in aggiunta tutte le informazioni utili per descrivere la specifica catena evolutiva).

Il primo passo compiuto dall’algoritmo è determinare se il Pokemon sia una forma base o una prima evoluzione o una seconda.

Questa verifica viene eseguita controllando dal database se il Pokemon sia presente nella colonna della forma base. Se è presente vuol dire che il Pokemon è una forma base.

Nel caso non sia una forma base, ispezionando analogamente il database come nel caso della forma base, l’algoritmo determina se sia una prima o una seconda evoluzione controllando rispettivamente se il Pokemon sia presente nella colonna della prima evoluzione oppure nella colonna della seconda evoluzione.

Nel caso sia una forma base si effettua un ulteriore controllo per verificare se il Pokemon in questione sia presente una e una sola volta nella colonna della forma base. Tale verifica è fondamentale per determinare se il Pokemon possieda più di una linea evolutiva (differenti tra loro).

Se il Pokemon è presente una sola volta vuol dire che possiede una sola linea evolutiva e l’algoritmo procede in questo modo: verifica che la colonna della prima evoluzione, della riga del Pokemon in esame, contenga una stringa vuota. In questo caso vuol dire che il Pokemon è uno stadio base che non ha evoluzioni. Qualora la stringa non sia vuota sia ha la sicurezza che abbia almeno la prima evoluzione. A questo punto per verificare se il Pokemon possieda anche la seconda, cioè ultima, evoluzione si procede analogamente al caso precedente analizzando la stringa relativa alla colonna della seconda evoluzione. Se la stringa è vuota allora il Pokemon possiederà sola la prima evoluzione altrimenti possiederà anche la seconda evoluzione.

Nel caso in cui il Pokemon sia presente due volte nella colonna della forma base, allora possiede due differenti linee evolutive. A questo punto l’algoritmo procede determinando se il Pokemon possieda due prime evoluzioni o due seconde evoluzioni.

Per quanto riguarda il caso in cui il Pokemon sia una prima evoluzione o una seconda evoluzione, l’algoritmo procede analogamente a quanto svolto nel caso della forma base, così da determinare la restante linea evolutiva.

L’efficacia di questo algoritmo ha permesso di coprire la quasi totalità delle possibili linee evolutive. In alcuni casi specifici, però, non è stato possibile ricondurre la specifica evoluzione in uno dei casi presi in esame dall’algoritmo.

In particolare, i casi sono stati due: Eevee e Tyrogue. La difficoltà nel gestire queste linee evolutive è dovuta al fatto che entrambe possiedono più di due prime evoluzioni (in particolare Eevee ne possiede 8 e Tyrogue 3).

Per questi due casi specifici si è deciso di optare per una gestione statica delle evoluzioni andando ad inserire direttamente le informazioni necessarie per descrivere le linee evolutive.

Un’importante problematica riscontrata è dovuta ad una differente gestione, da parte delle API, del nome di alcuni Pokemon. In particolare, per alcuni Pokemon il nome fornito dalle API, nel caso della catena evolutiva non corrisponde con il nome fornito, sempre dalle API, nelle informazioni del Pokemon. Ciò ha provocato la non corrispondenza 1:1 tra le informazioni possedute. Anche in questo caso si è deciso di operare staticamente andando a segnalare questa problematica con il termine “ERROR API”, visualizzabile nella schermata della catena evolutiva.

* 1. Fragment Move

Nel Fragment Move in cui sono descritte le mosse apprese dal pokemon viene mostrata una recycler view che, attraverso un adapter, espone degli oggetti JoinPokeMove. Tali oggetti derivano da un’estrazione dal database effettuata facendo una join delle tabelle PokemonMoves e Move. In questo modo è stato possibile associare ad un pokemon tutte le mosse che impara e le relative informazioni della mossa.