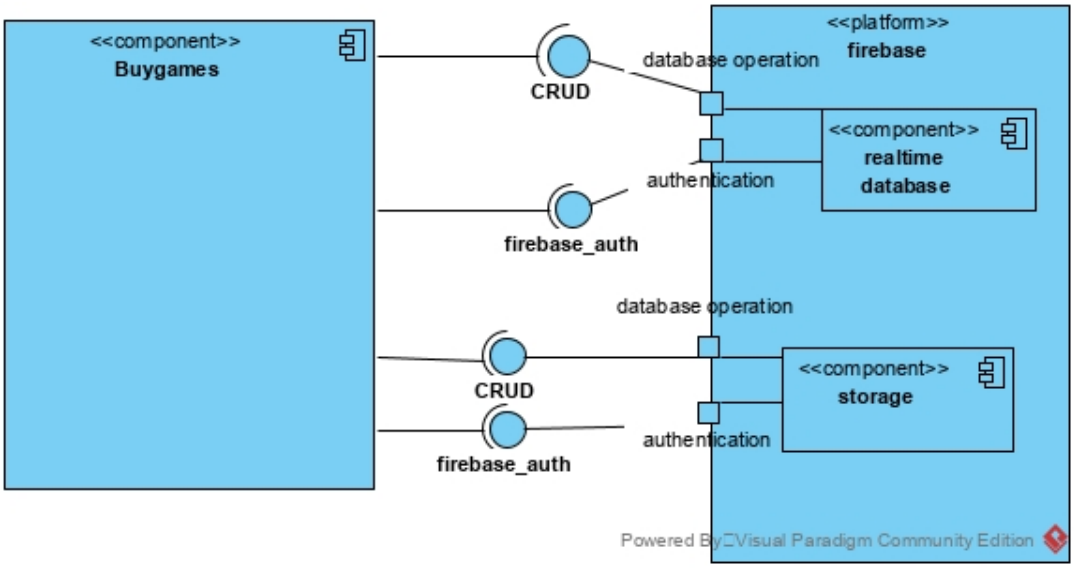
Sommario

Elaborazione dettaglio

Component diagram

Tale diagramma fornisce un catalogo dei componenti del sistema e specifica le interfacce richieste/fornite da ciascuno di essi, ovvero l'insieme di operazioni che ciascun componente può richiedere/fornire a un altro componente con cui interagisce.

Di seguito si riporta il *component diagram* del sistema al fine di mostrare i componenti, le interfacce offerte e quelle richieste.



Stile architetturale

Da un punto di vista architetturale, l'applicazione è stata costruita seguendo il pattern MVVM. Esso prevede la suddivisione delle classi in tre package:

- **View**: contenente le classi che si occupano di operazioni di tipo UI.
- **ViewModel**: contenente le classi che implementano la business logic dell'applicazione.
- **Model**: contenente le data class che definiscono gli oggetti del dominio applicativo.

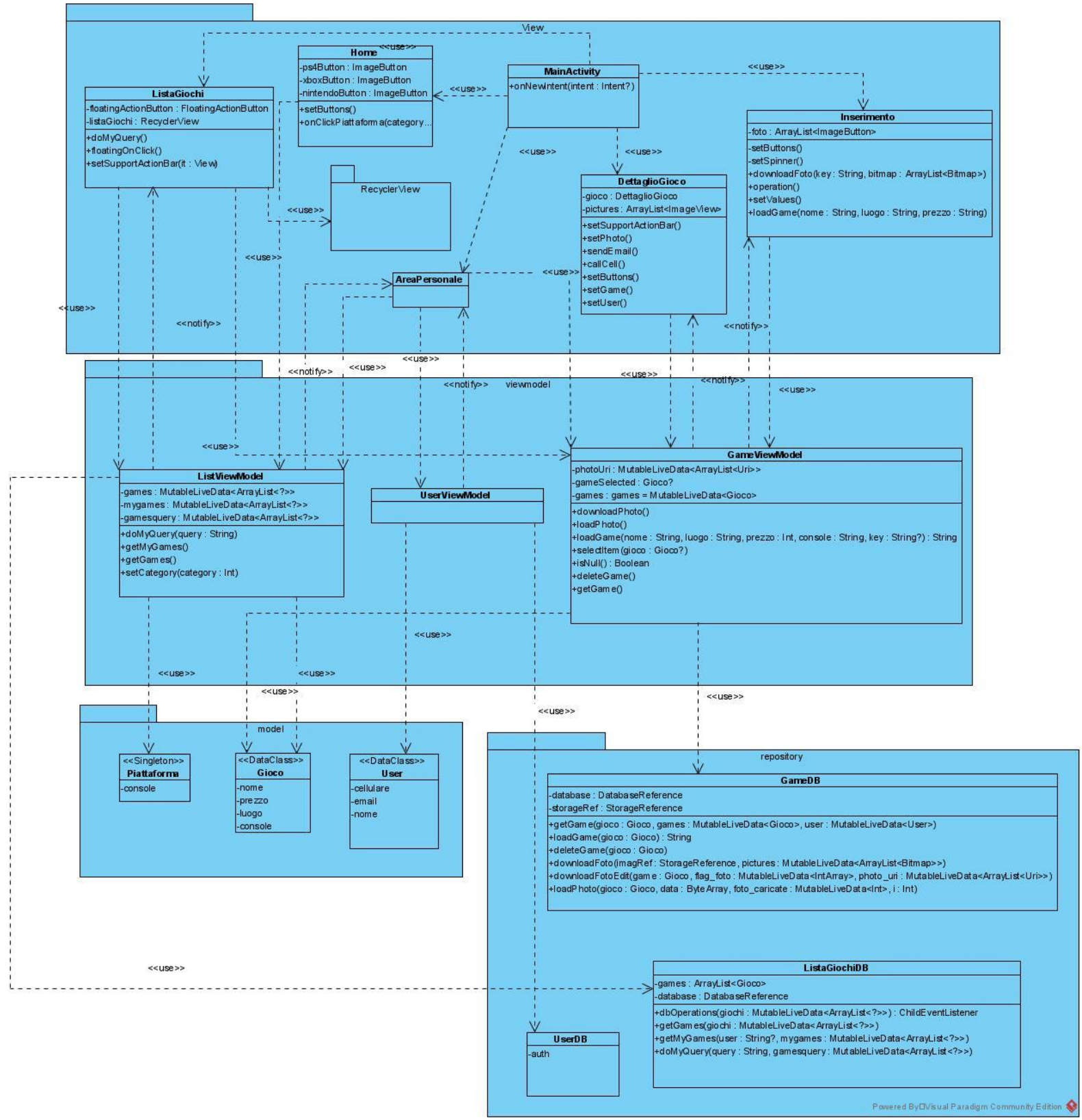
In aggiunta, è stato utilizzato un package di tipo **repository** contenente le classi che fanno da gateway con il mondo esterno, ovvero contengono i metodi che consentono l'accesso ad un database esterno per caricare o scaricare dati e foto.

Nel file n.3 cap. 4 è stato già accennato l'utilizzo di questo pattern ed è stato mostrato ad alto livello il collegamento tra i vari package. Nel seguito viene mostrato lo stesso diagramma nel dettaglio con le classi e i relativi attributi e metodi.

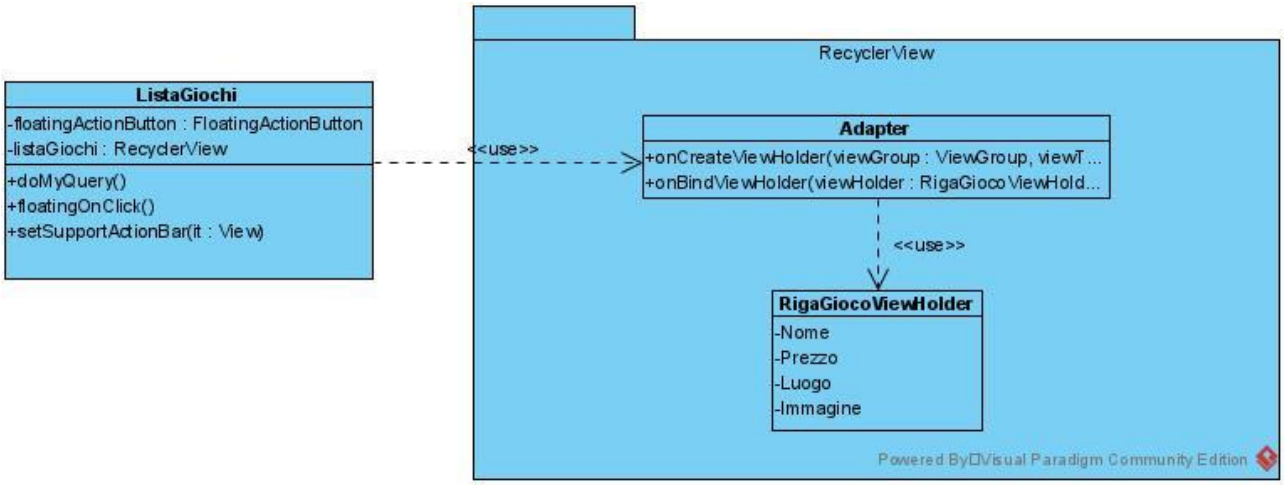
Package diagram

- View: le classi utilizzate per l'implementazione dei requisiti scelti sono MainActivity, Home, ListaGiochi, DettaglioGioco e Inserimento. Si noti come esse dipendono soltanto da classi del livello sottostante, cioè il viewmodel. L'activity presenta una dipendenza rispetto a tutti i fragment poiché è responsabile della loro creazione e distruzione. Inoltre, ListaGiochi presenta una dipendenza dal package RecyclerView necessaria per mostrare correttamente la lista dei giochi. In seguito, verrà illustrato il collegamento nel dettaglio.
- ViewModel: le classi utilizzate sono ListViewModel e GameViewModel. La prima contiene tutte le operazioni necessarie per la gestione della lista dei giochi, mentre la seconda è responsabile delle operazioni relative al singolo gioco nel dettaglio. Si noti che le classi di questo package dipendono dalle classi dei due package sottostanti: model e repository.
- Model: contiene le data class Gioco, User e Piattaforma. Particolare attenzione viene riservata alla classe Piattaforma la quale è stata implementata come **Singleton** poiché rappresenta la categoria di videogiochi che si vuole visualizzare, dunque, una e una sola istanza di tale classe deve essere istanziata.
- Repository: contiene le classi GameDB e ListaGiochiDB. La prima ha al suo interno dei metodi che consentono l'upload di un nuovo gioco o il download di dati e foto relativi ad un singolo gioco. ListaGiochiDB contiene i metodi che

consentono la ricerca di un gioco nel database (per nome o luogo), la visualizzazione della lista dei giochi e la visualizzazione della lista dei propri giochi messi in vendita.



Per quanto riguarda il package RecyclerView, di seguito vengono mostrate le sue classi e la relazione con la classe ListaGiochi.



La classe ListaGiochi per mostrare la lista dei giochi recupera, tramite il ViewModel, il vettore di giochi, il quale viene passato al costruttore della classe Adapter. Quest'ultima tramite il metodo onBindViewHolder associa i valori di ciascun gioco alle variabili della classe RigaGiocoViewHolder. Tale classe si occupa di associare i valori delle sue variabili ai valori mostrati a video all'utente.

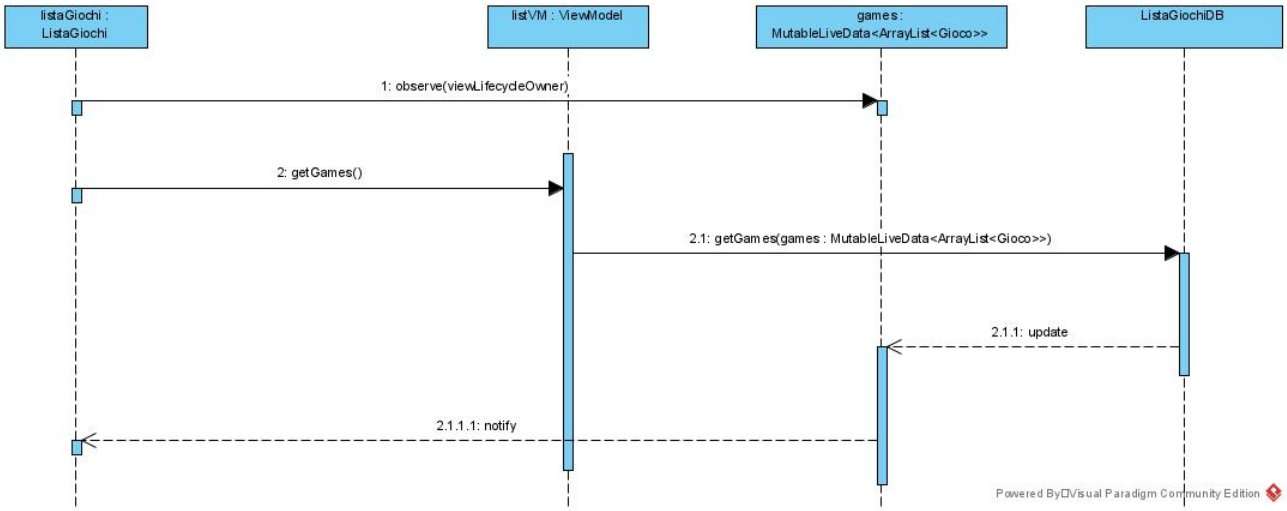
Tale meccanismo verrà illustrato in seguito con l'ausilio dei sequence diagram.

Sequence diagram di dettaglio

Nel seguito verranno mostrati i sequence diagram di dettaglio per mostrare la dinamica dell'applicazione in relazione ai casi d'uso "acquista gioco" e "inserisci annuncio". Inoltre, viene presentato sia il sequence diagram relativo al pattern **observer** utilizzato per la comunicazione tra le classi della view e del ViewModel, sia il sequence diagram relativo all'interazione tra i fragment, l'adapter e la classe RigaGiocoViewHolder.

Pattern observer

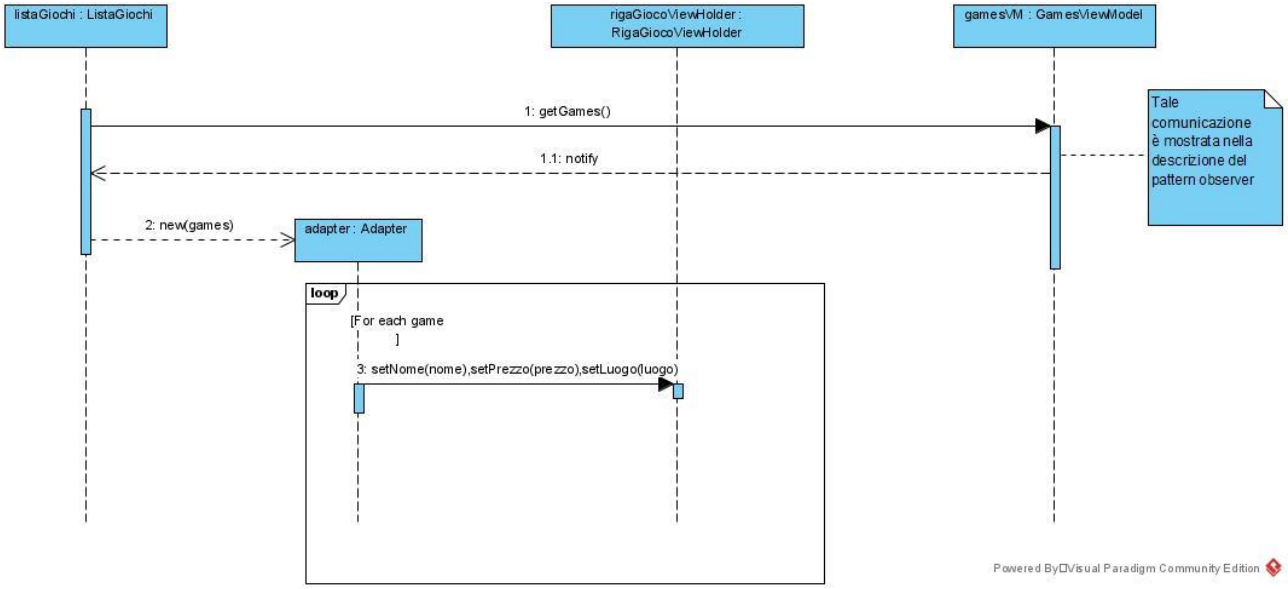
Nel package diagram è stato mostrato come le classi del ViewModel comunicano con le classi della View mediante lo stereotipo <<notify>>. Tale comunicazione è relizzata sfruttando il pattern observer. Poiché questo pattern si ripete più volte, viene mostrato a titolo di esempio il seguente diagramma per mostrare la comunicazione tra ListaGiochi e ListViewModel ma la dinamica è la medesima anche per gli altri casi in cui compare lo stereotipo.



Si noti come il viewModel non invoca alcun metodo della classe listaGiochi. La classe listVM ha tra i propri attributi un mutable live data di tipo ArrayList<Gioco>. Quando la classe listaGiochi invoca il metodo getGames, il viewModel invoca il metodo getGames sulla classe del repository il quale si occupa di aggiornare il mutable live data. Una volta aggiornato, la classe listaGiochi della view viene automaticamente notificata dal MutableLiveData.

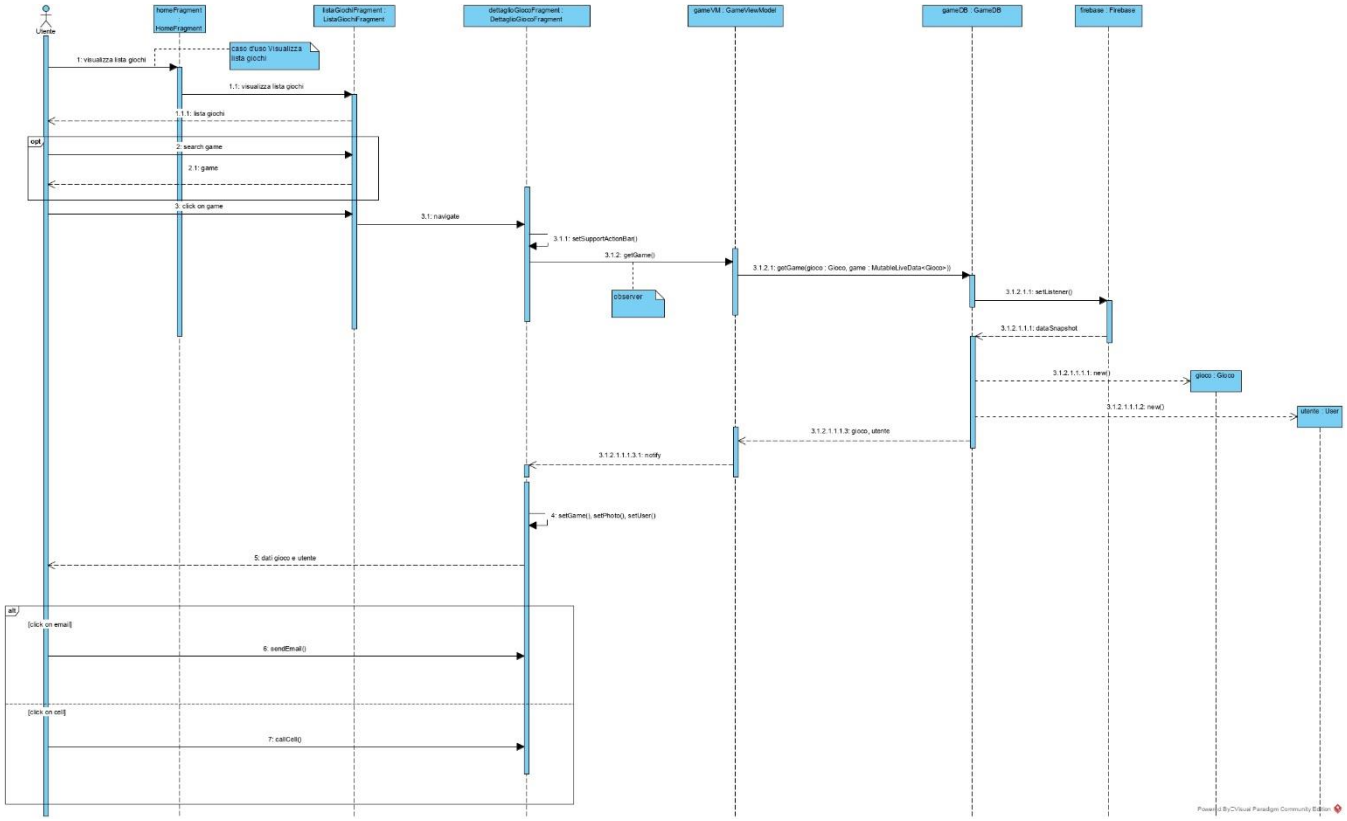
RecyclerView, Adapter e ViewHolder

Di seguito viene mostrata la dinamica per l'aggiornamento della lista dei giochi. Tale operazione viene effettuata nel caso d'uso "Acquista gioco" mostrato di seguito. Per semplicità di lettura viene riportata qui l'operazione di aggiornamento della lista nel dettaglio e nel seguito si rimanderà a questo diagramma.

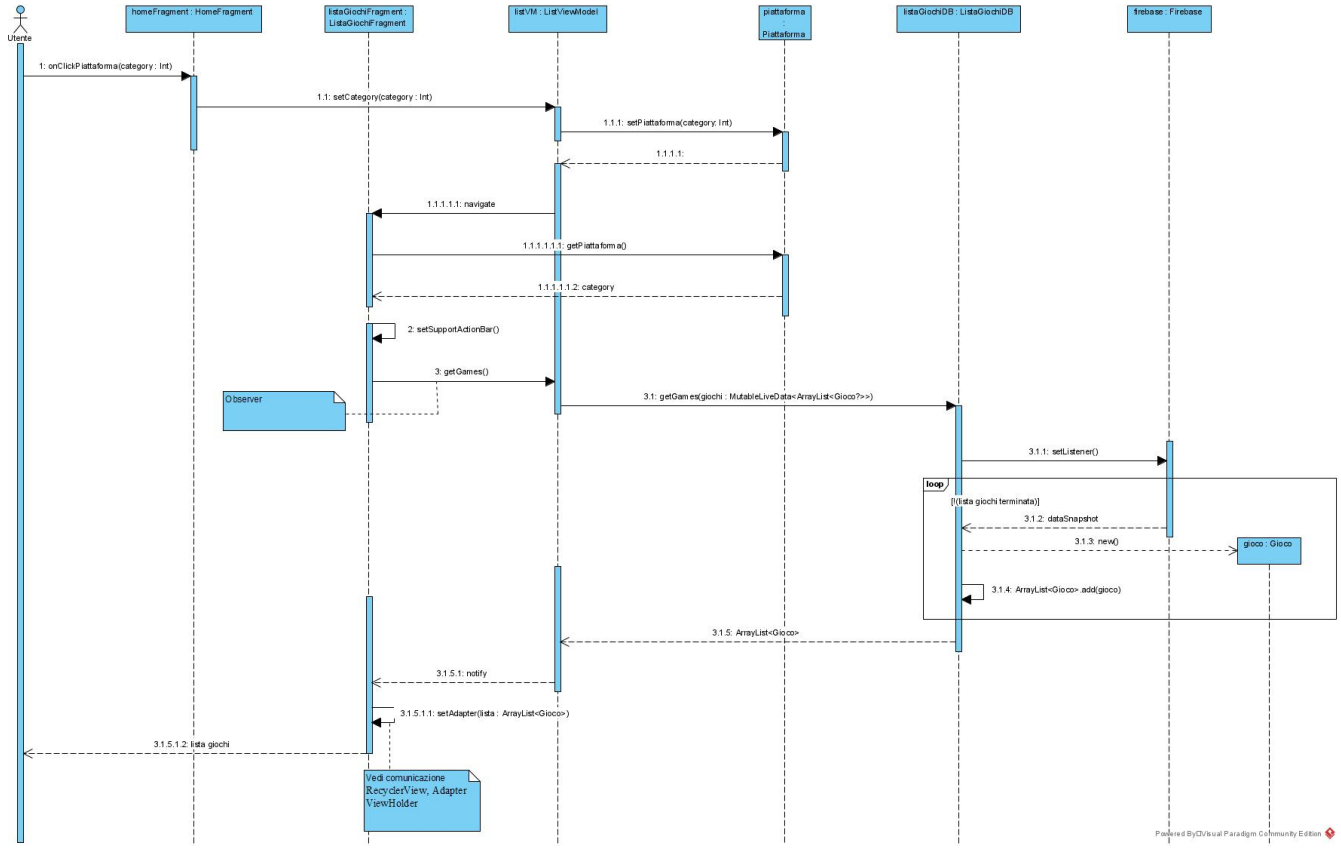


Il fragment listaGiochi recupera i giochi da visualizzare sfruttando il pattern observer illustrato in precedenza. Dopodiché crea un oggetto della classe adapter a cui passa il vettore di giochi, tale classe avrà il compito di associare gli attributi di ciascun gioco alle variabili del viewHolder. In questo modo automaticamente la RecyclerView sarà popolata dai vari oggetti di tipo rigaGiocoViewHolder.

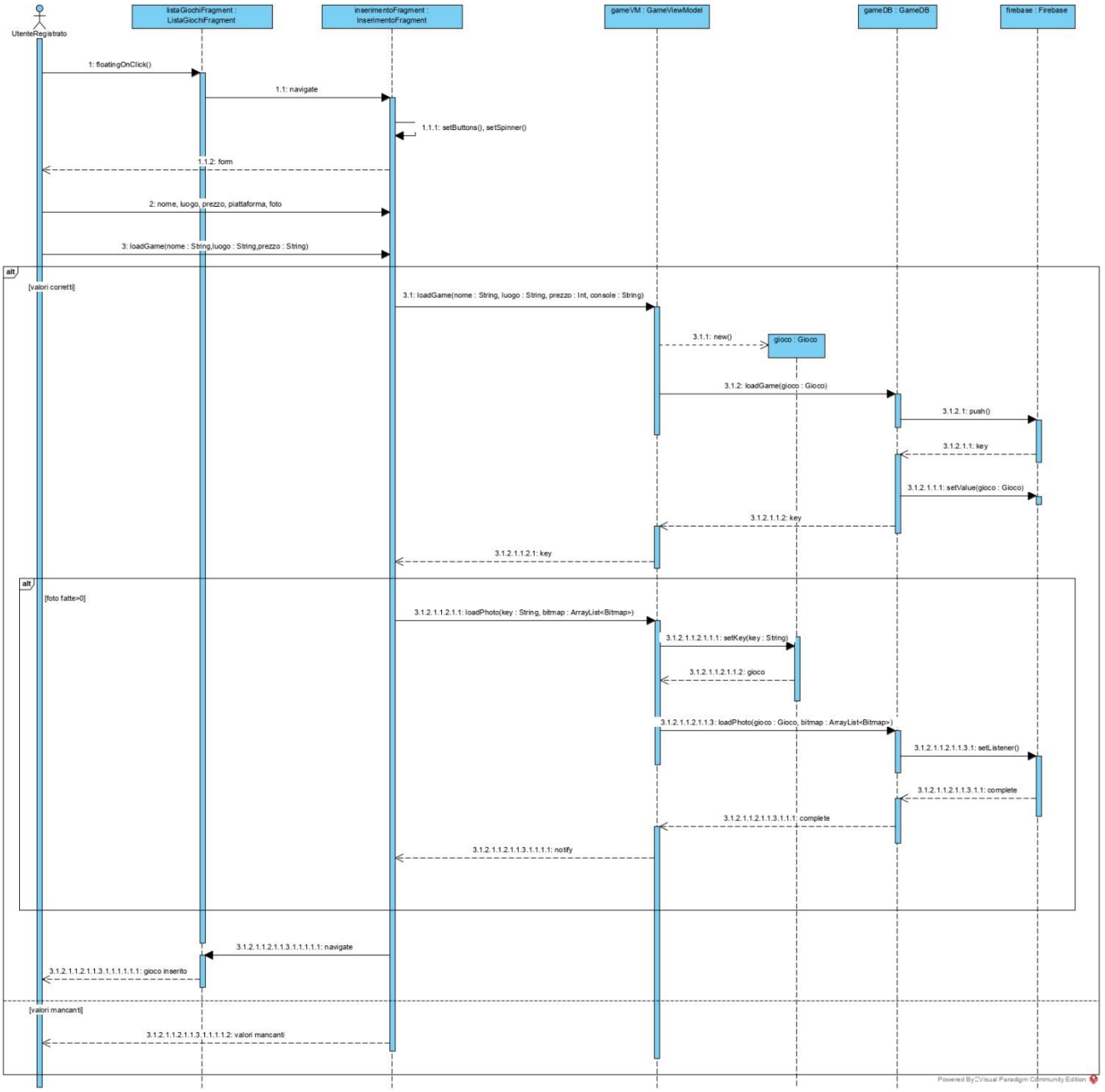
Acquista gioco



Visualizza lista giochi



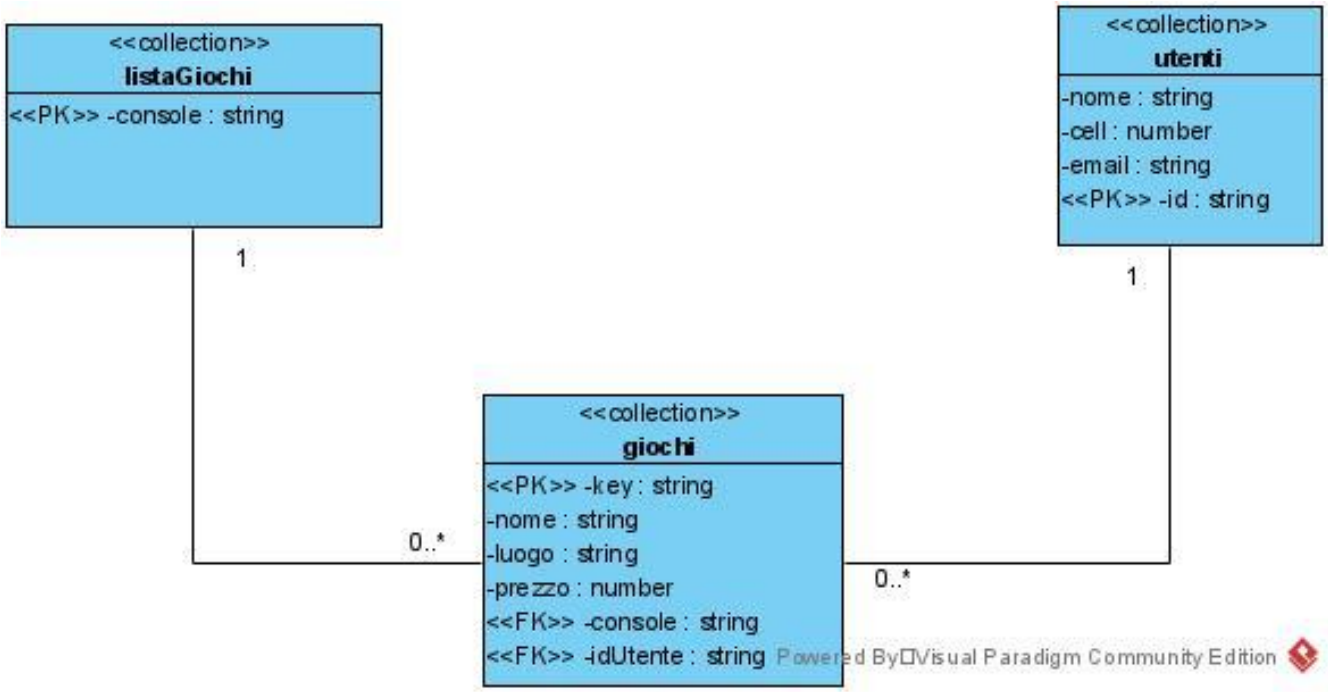
Inserisci gioco



Modello dei dati

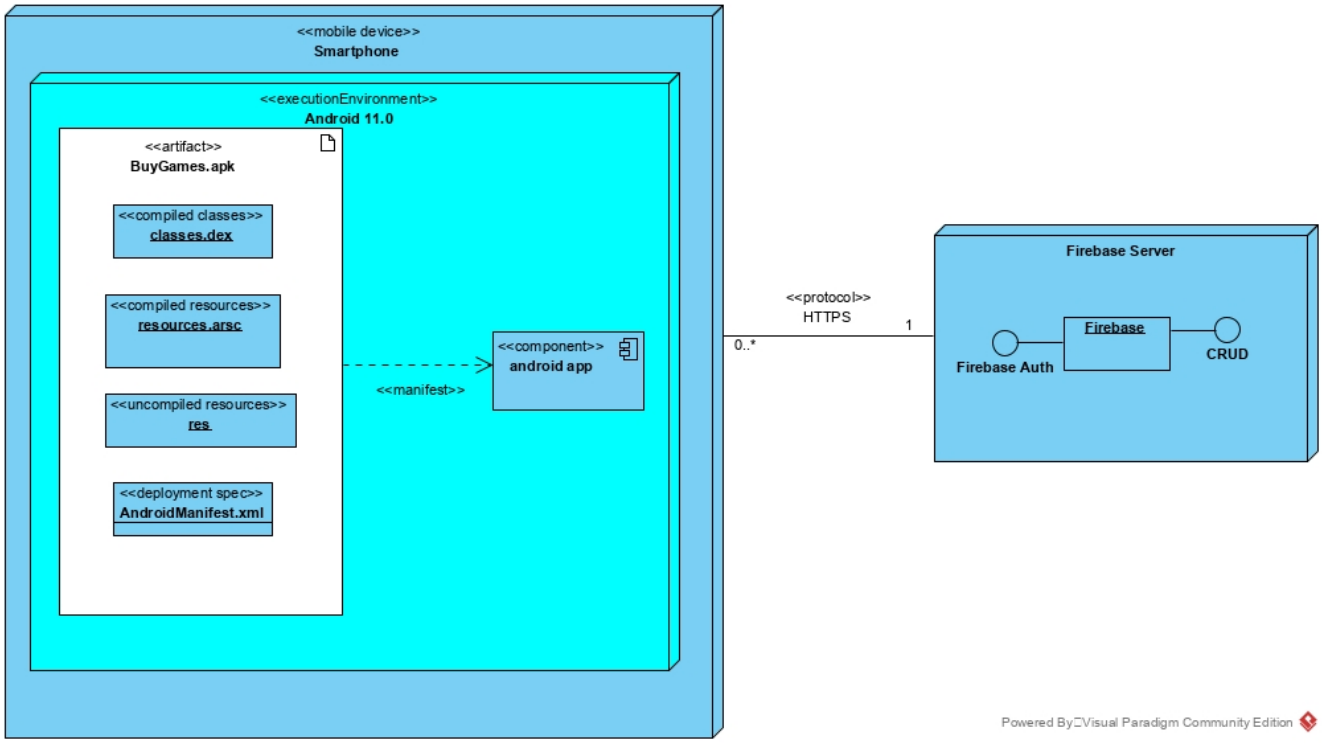
Di seguito viene riportato il modello dei dati memorizzati sul database e utilizzato dal sistema. Il database è costituito da una struttura JSON completamente personalizzabile che è stato organizzato in collezioni. Le collezioni vengono riportate di seguito.

Si noti come listaGiochi e utenti sono entrambe associate alla collection giochi, tali associazioni sono realizzate tramite i campi console e idUtente che svolgono il ruolo di foreign key.



Deployment diagram

Di seguito è riportato il Deployment Diagram del sistema, ossia un diagramma che espone l'architettura del sistema, in particolare i nodi sia in termini di hardware che software.



A livello di *deploy*, si evidenziano due nodi fisici:

- lo Smartphone, con un sistema operativo Android 9.0 o superiori, su cui sarà in esecuzione l'applicazione mobile;
- un server Firebase che sarà l'host del database del sistema software.

La comunicazione tra il nodo Smartphone e il database avviene tramite protocollo HTTPS.