StadiumLocator

Studente: Lutterotti Filippo
Ambiente: Android Studio – Kotlin

App che recupera una lista di stadi da un server PHP ospitato tramite XAMPP, e che gestisce modifiche alla lista durante l'uso dell'app, con aggiunte o eliminazioni. Utilizza il GPS per determinare la posizione dell'utente e calcola la distanza tra le coordinate degli stadi (fornite nel JSON dal server) e la posizione attuale dell'utente.

Ecco i miei file divisi per package:

- **MainActivity(View)**: È la pagina Home del progetto, che mostra la lista degli stadi e include i pulsanti per Eliminare, Aggiornare e Accedere alle Opzioni.
  I metodi principali sono:
  - *setupPermissionLauncher*: Registra il ActivityResultLauncher per gestire la richiesta dei permessi di localizzazione.
  - *checkAndRequestPermissions*: Controlla se i permessi di localizzazione sono stati concessi; in caso contrario, avvia la richiesta dei permessi.
  - *observeViewModel*: Osserva i cambiamenti nei dati del ViewModel, aggiornando la UI con la lista degli stadi e gestendo i feedback dell'utente.
  - *updateUI*: Aggiorna visivamente la lista degli stadi, creando dinamicamente le viste per ciascuno stadio e aggiungendole al layout.
  - *showSnackbar*: Mostra un messaggio di feedback all'utente utilizzando una Snackbar.

- **OptionsActivity(View)**: È la pagina delle Opzioni, che permette di aggiungere uno stadio al server.
  I metodi principali sono:
  - *observeViewModel:* Osserva i cambiamenti nei dati di feedback del ViewModel, mostrando un messaggio all'utente tramite Toast in base al feedback ricevuto.
  - *addStadiumButton.setOnClickListener*: Quando viene premuto, verifica se il nome dello stadio è stato inserito; se sì, chiama il metodo addStadium nel ViewModel, altrimenti mostra un messaggio di errore.

- **StadiumViewModel**: ViewModel che gestisce il collegamento tra la view (MainActivity e OptionsActivity) con le varie funzioni presenti nel package repository.
  I metodi principali sono:
  - *getStadiums*: Recupera la lista degli stadi dal StadiumRepository e aggiorna la LiveData _stadiums se la richiesta ha successo; altrimenti, imposta un messaggio di errore nel feedback.
  - *deleteStadium*: Rimuove uno stadio dalla lista richiamando il repository. Se la rimozione ha successo, aggiorna la LiveData _stadiums; in caso di errore, aggiorna il feedback.

    - *addStadium*: Aggiunge un nuovo stadio chiamando il repository. Se l'operazione è riuscita, aggiorna la lista degli stadi e il feedback con un messaggio di successo; in caso contrario, fornisce un messaggio di errore.

- **StadiumRepository**: Contiene le funzioni per prendere, aggiungere ed eliminare uno stadio dal server.
  I metodi principali sono:
    - *getNearbyStadiums*: Recupera gli stadi vicini utilizzando l'API Retrofit e calcola la distanza da ciascuno stadio in base alla posizione attuale dell'utente. Se ha successo, restituisce una lista di stadi ordinata per distanza; altrimenti, restituisce un messaggio di errore.
    - *checkLocationPermission*: Verifica se i permessi di localizzazione sono stati concessi.
    - *getLastKnownLocation*: Restituisce l'ultima posizione nota dell'utente utilizzando il FusedLocationProviderClient, se i permessi sono stati concessi.
    - *deleteStadium*: Invia una richiesta DELETE per rimuovere uno stadio dal server. Se la richiesta ha successo, restituisce un risultato positivo; altrimenti, restituisce un errore.
    - *addStadium*: Aggiunge un nuovo stadio recuperando prima la posizione corrente. Se la richiesta POST ha successo, restituisce lo stadio appena creato; in caso di errore, restituisce un messaggio di fallimento.

- **Stadium(Model)**: Definisce la struttura degli stadi, con variabili per nome, latitudine, longitudine e distanza.

- **ApiService(Network)**: Contiene i metodi per le chiamate API (GET, POST e DELETE).

- **RetrofitClient(Network)**: Installa il client per la connessione al server, dove deve essere specificato l'URL del sito di riferimento.

Ho utilizzato dei file xml per cambiare lo stile dell'applicazione e aggiungere delle Icon, ad esempio per il pulsante elimina e per l'Icona dell'applicazione.

Punti di forza

- **Architettura basata su MVVM**: L'adozione del pattern Model-View-ViewModel garantisce una chiara separazione delle responsabilità tra la logica di business e la presentazione, migliorando la testabilità e la manutenibilità del codice.
- **Gestione centralizzata dei permessi**: L'implementazione di un meccanismo per richiedere e verificare i permessi nella ViewModel e nelle Activity semplifica la gestione delle autorizzazioni, assicurando una maggiore coerenza e riducendo il rischio di errori.
- **Operazioni asincrone ben gestite**: L'uso delle coroutine consente di eseguire le operazioni di rete e I/O in modo non bloccante, mantenendo l'interfaccia reattiva e migliorando l'esperienza utente durante l'interazione con il server.
- **Feedback utente centralizzato e consistente**: L'approccio uniforme nella gestione dei feedback, attraverso Snackbar e Toast, assicura che l'utente riceva informazioni coerenti e immediate su successi o errori, migliorando la chiarezza dell'interazione.

- **Aggiornamento dinamico della UI con LiveData**: Grazie all'uso di LiveData per osservare i cambiamenti di dati, l'interfaccia utente si aggiorna automaticamente senza bisogno di interventi manuali, migliorando l'efficienza del rendering delle informazioni.

Possibili Migliorie

Per l'app in sé:

- Integrazione con mappe: Integrare una mappa interattiva dove gli utenti possano visualizzare la posizione degli stadi rispetto alla loro posizione attuale renderebbe l'esperienza di navigazione più visiva e accattivante.

Riguardante il codice:

- Miglioramento generale del codice: visionare se ci sono metodi migliori da quelli utilizzati e quindi già importabili facilmente anche.
- Idea del passaggio dei messaggi da Repository alla View: non se il passaggio dei messaggi sia stato svolto in modo corretto e se sia un buon metodo.