

# 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.

Nel mio progetto ho creato due file di activity per gestire le due pagine principali (Home e Options) e tre file di configurazione:

- **MainActivity:** 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:
  - **setupPermission:** Inizializza il meccanismo per richiedere i permessi.
  - **checkLocationPermission:** Gestisce i permessi di localizzazione.
  - **fetchLocationAndStadiums:** Salva la posizione ottenuta tramite *getLastKnownLocation* e avvia la chiamata di *fetchNearbyStadium*.
  - **getLastKnownLocation:** Ottiene la posizione e verifica i permessi tramite *checkLocationPermission*.
  - **fetchNearbyStadium:** Recupera la posizione e gli stadi tramite RetrofitClient con una chiamata GET e calcola la distanza.
  - **updateStadiumList:** Aggiorna completamente la lista degli stadi, chiamando *fetchLocationAndStadiums*. Può essere utilizzato anche solo per chiamare *fetchLocationAndStadiums*.
  - **updateUI:** Aggiorna visivamente la lista degli stadi.
  - **removeStadium:** Rimuove uno stadio dalla lista e viene chiamato da *deleteStadiumToServer*.
  - **deleteStadiumToServer:** Rimuove uno stadio dal server inviando una richiesta DELETE.
- **OptionsActivity:** È la pagina delle Opzioni, che permette di aggiungere uno stadio al server.  
I metodi principali sono:
  - **getLastKnownLocation:** Ottiene la posizione e chiama *addStadiumToServer*.
  - **addStadiumToServer:** Aggiunge uno stadio al server tramite una chiamata POST, includendo il nome e le coordinate dello stadio.
- **Stadium:** Definisce la struttura degli stadi, con variabili per nome, latitudine, longitudine e distanza.
- **ApiService:** Contiene i metodi per le chiamate API (GET, POST e DELETE).
- **RetrofitClient:** 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

- **Divisione per Activity:** L'organizzazione del progetto in base alle Activity migliora la struttura e la gestione delle diverse funzionalità, rendendo il codice più modulare e facile da mantenere.
- **Riutilizzo del Codice:** La creazione di funzioni riutilizzabili, come `updateStadiumList(input x)`, contribuisce a un codice più pulito e mantenibile.
- **Gestione delle Operazioni Non Bloccanti:** L'uso di `CoroutineScope` e `Dispatchers` permette di eseguire operazioni di rete e di IO in background, evitando il blocco del thread principale e migliorando la reattività dell'app.
- **Gestione degli Errori e Feedback:** L'implementazione di messaggi di errore e di feedback positivi tramite `Snackbar` e `Toast` fornisce un'esperienza chiara all'utente.
- **Aggiornamento Dinamico della UI:** Il metodo `updateUI()` gestisce l'aggiornamento dinamico dell'interfaccia utente, garantendo che i dati visualizzati siano sempre aggiornati.
- **Organizzazione del Progetto:** La suddivisione del progetto in file separati è ben strutturata e contribuisce a una maggiore chiarezza e gestibilità del codice.

## Possibili migliorie

Una cambiamento che si potrebbe fare è magari rendere riutilizzabile `getLastKnownLocation` sia per `MainActivity` che per `OptionsActivity`.

Utilizzare dei `ViewModel` e `LiveData` per aggiungere degli `Observer` per vedere se ci sono cambiamenti nei dati e lasciare che ci pensino loro l'aggiornamento della variabile.