Relazione Programmazione di sistemi mobili

Aloè Matteo & Strazzella Elia

**Introduzione**

Il presente elaborato descrive il processo di progettazione, sviluppo e implementazione di un’applicazione mobile nativa per Android, realizzata nell’ambito del corso universitario di *Programmazione di Sistemi Mobili*. L'applicazione, denominata **ParkFinder**, nasce con l’intento di rispondere all’esigenza di molti utenti di scoprire con facilità parchi pubblici nelle vicinanze, ricevere informazioni dettagliate su di essi e gestire una lista di preferiti.

**Obiettivi del progetto**

L’obiettivo principale del progetto è stato la realizzazione di un'applicazione mobile intuitiva, funzionale e orientata all’utente, capace di:

* **Individuare i parchi pubblici nelle vicinanze** dell’utente utilizzando i servizi di localizzazione del dispositivo.
* **Visualizzare i parchi sulla mappa**, offrendo così un'interfaccia grafica immediata e interattiva per la navigazione geografica.
* **Fornire informazioni dettagliate su ciascun parco**, quali immagini, indirizzo, distanza attuale, recensioni, e condizioni meteo aggiornate in tempo reale.
* **Consentire il salvataggio dei parchi preferiti** in una sezione apposita, con archiviazione in locale per permettere la consultazione anche in assenza di connessione.
* **Garantire una user experience fluida**, adottando soluzioni tecnologiche e architetturali all’avanguardia per Android.

**Descrizione dell’applicazione**

L’applicazione **ParkFinder** si presenta con una **struttura semplice e intuitiva**, suddivisa in tre sezioni principali: una schermata iniziale con mappa interattiva, una sezione dedicata alla visualizzazione dettagliata dei parchi selezionati e una sezione per la gestione dei preferiti.

**Funzionalità principali:**

* **Localizzazione e mappa interattiva**: l’applicazione utilizza i servizi di localizzazione del dispositivo per determinare la posizione corrente dell’utente. I parchi pubblici nelle vicinanze vengono recuperati tramite l’integrazione con le API di Google Places e mostrati su una mappa interattiva fornita dal Google Maps SDK, con l’ausilio di marker personalizzati.
* **Scheda informativa del parco**: cliccando su un parco, l’utente può accedere a una scheda contenente:
  + Immagini illustrative del luogo (caricate dinamicamente tramite la libreria Glide).
  + Distanza tra l’utente e il parco.
  + Valutazioni, solo se disponibili su Google.
  + Indirizzo completo, solo se disponibile su Google Maps.
  + Condizioni meteo aggiornate, recuperate da OpenWeatherMap API in base alle coordinate geografiche del parco.
* **Sezione preferiti**: ogni utente può salvare parchi nella propria lista dei preferiti. I dati vengono memorizzati in locale tramite SharedPreferences. La gestione dei preferiti è semplice, con possibilità di rimuovere elementi e visualizzarli ordinatamente.

**Architettura del sistema**

Per garantire una struttura solida, modulare e facilmente manutenibile, l’app è stata sviluppata seguendo il pattern architetturale **MVVM (Model-View-ViewModel)**, ormai standard nello sviluppo Android moderno.

**Struttura:**

* **Model**: contiene le classi dati (come Park, ecc.) e la logica per il recupero delle informazioni da sorgenti esterne (API) e da memoria locale.
* **ViewModel**: gestisce la logica di presentazione e funge da intermediario tra la View e il Model. Utilizza **LiveData** per garantire la reattività dell’interfaccia grafica e una gestione corretta del ciclo di vita dei componenti.
* **View (UI)**: composta da Activity e Fragment, si occupa dell’interfaccia utente e si aggiorna automaticamente al variare dei dati osservati nel ViewModel.

Il progetto prevede inoltre l’uso di **repository** per l’incapsulamento dell’accesso ai dati, una scelta architetturale che consente maggiore indipendenza tra componenti e facilita l’integrazione futura di nuove sorgenti dati.

Tecnologie Utilizzate

| **Tecnologia / Strumento** | **Descrizione / Funzione** |
| --- | --- |
| **Kotlin** | Linguaggio utilizzato per l’intero sviluppo dell’app. Offre modernità, sicurezza e piena compatibilità con Android. |
| **Google Places API** | Utilizzata per ottenere dettagli sui parchi pubblici in base alla posizione geografica dell’utente. |
| **Google Maps SDK** | Consente la visualizzazione dei parchi su una mappa interattiva, con marker personalizzati. |
| **Servizi di localizzazione** | Utilizzati per acquisire la posizione corrente dell’utente e calcolare la distanza rispetto ai parchi. |
| **OpenWeatherMap API** | Fornisce informazioni meteo in tempo reale in base alle coordinate geografiche del parco. |
| **Glide** | Libreria efficiente per il caricamento asincrono delle immagini. Supporta caching automatico e gestione ottimizzata della memoria. |
| **Retrofit + Moshi** | Retrofit per le richieste HTTP e Moshi per la serializzazione/deserializzazione dei dati JSON. Soluzione robusta per il networking. |
| **ViewModel + LiveData** | Componenti dell’architettura MVVM che assicurano una gestione efficiente del ciclo di vita e aggiornamento reattivo dell’interfaccia. |
| **Local Storage (SharedPreferences)** | Permette di salvare i dati localmente in modo semplice e veloce. Utilizzato per la gestione della lista dei preferiti. |
| **buildConfigField & ProGuard** | Utilizzati per proteggere le chiavi API e offuscare il codice durante la compilazione dell’app, migliorandone la sicurezza. |

Considerazioni Finali

Il progetto ha permesso di realizzare un'applicazione mobile completa, funzionale e ben strutturata. ParkFinder è in grado di offrire un servizio utile agli utenti, combinando la geolocalizzazione, la visualizzazione su mappa, l'accesso a informazioni dettagliate sui parchi e la possibilità di salvare quelli preferiti.

Grazie all'integrazione di diverse API e all’adozione di un’architettura solida, l’app risulta stabile, reattiva e facilmente estendibile. Le tecnologie utilizzate hanno reso possibile un’esperienza utente fluida e moderna, oltre a garantire buone prestazioni anche su dispositivi di fascia media.

Il progetto è stato sviluppato in modo modulare, permettendo un’eventuale evoluzione futura con l’aggiunta di nuove funzionalità o il miglioramento di quelle esistenti.