Documentazione del Progetto di Programmazione Mobile A.A. 2025-2026

Nome del Progetto: QuestMaster

Gruppo: Bulletin Bills

Membri:

•857321 Andrea De Luca

•938107 Viktor Pannagl

Indice

1. Introduzione

QuestMaster è un'applicazione Android progettata per trasformare l'esplorazione del mondo reale in un'avventura interattiva. Il suo obiettivo primario è fornire una piattaforma in cui gli utenti possano creare e partecipare a "quest" personalizzate, ovvero cacce al tesoro o percorsi a tappe basati sulla geolocalizzazione.

L'idea nasce dal desiderio di unire il mondo digitale del gaming con l'esperienza fisica dell'esplorazione urbana e naturale. In un'epoca in cui le interazioni sono sempre più mediate da uno schermo, QuestMaster incoraggia gli utenti a uscire e scoprire punti di interesse, parchi o angoli nascosti della propria città in modo divertente e coinvolgente. A differenza delle semplici app di navigazione, QuestMaster aggiunge uno strato di gamification, permettendo di creare narrazioni, sfide e ricompense legate a luoghi fisici.

Molte persone cercano nuovi modi per passare il tempo all'aperto, ma organizzare attività come una caccia al tesoro può essere complesso. QuestMaster mira a risolvere questo problema offrendo strumenti intuitivi per disegnare percorsi, definire enigmi e condividere queste avventure con gli amici o la community. La piattaforma gestisce l'avanzamento della quest, verifica il completamento delle tappe tramite GPS e mostra le istruzioni, lasciando agli utenti il piacere del gioco e della scoperta.

2. ArchitetturaLo sviluppo di QuestMaster adotta il pattern architetturale Model-View-ViewModel (MVVM), raccomandato da Google per la creazione di applicazioni robuste, scalabili e testabili. Questa architettura promuove una netta separazione delle responsabilità.

Il flusso di dati ed eventi segue questa struttura:

View (UI Layer) <--> ViewModel (Domain Layer) <--> Repository (Data Layer) <--> Data Sources

•View (UI Layer): Questo strato, composto da Activity e Fragment, è responsabile della visualizzazione dell'interfaccia utente e della cattura degli input dell'utente. Non ha alcuna conoscenza della logica di business o della provenienza dei dati. Si limita a "osservare" il ViewModel per ricevere i dati e a notificare al ViewModel gli eventi generati dall'utente. Nel nostro progetto, questo include HomeActivity, StartActivity, FeedFragment e QuestCreateFragment.

•ViewModel (Domain Layer): Il ViewModel funge da intermediario tra la View e la logica di business dell'applicazione. Il suo ruolo principale è quello di gestire i dati relativi alla UI in modo consapevole del ciclo di vita. Ciò significa che i dati contenuti nel ViewModel (ad esempio, la lista di quest) sopravvivono ai cambiamenti di configurazione, come la rotazione dello schermo, evitando perdite di dati e chiamate di rete superflue. Il ViewModel espone i dati alla View tramite LiveData e non ha riferimenti diretti ad Activity o Fragment, garantendo un'elevata testabilità e riusabilità.

•Model (Data Layer): Questo strato è rappresentato dal Repository Pattern. Il Repository è l'unica fonte di verità (Single Source of Truth) per i dati dell'applicazione. Astrae le fonti dei dati dal resto dell'app, il che significa che il ViewModel non sa (e non deve sapere) se i dati provengono da un server remoto, da un database locale o da una cache in memoria. Questa modularità è fondamentale per la nostra architettura, poiché ci consente di aggiungere o modificare le fonti dati senza alterare la logica di business. I nostri modelli dati sono definiti in classi come Quest.java, User.java e QuestLocation.java.

Questa architettura rende il codice più facile da manutenere, testare e scalare nel tempo.

3. Design e ImplementazioneIl design dell'app segue le linee guida del Material Design, garantendo un'interfaccia utente pulita, intuitiva e moderna, che risulta familiare sulla piattaforma Android.UI LayerL'interfaccia utente è implementata utilizzando il sistema di viste tradizionale di Android. L'applicazione è costruita attorno a una Single-Activity Architecture, in cui HomeActivity funge da punto di ingresso principale e contenitore per le varie destinazioni (Fragment). Questo approccio moderno è più efficiente in termini di memoria rispetto all'uso di più Activity e crea un'esperienza utente più coesa.

•Navigazione: La navigazione tra le schermate è orchestrata dal Jetpack Navigation Component. Questo strumento ci permette di definire tutti i percorsi e le transizioni di navigazione all'interno di un grafico visivo, rendendo il flusso dell'applicazione facile da comprendere e gestire da una posizione centralizzata.

•Layout e Viste: Le schermate sono strutturate come Fragment (es. FeedFragment, QuestCreateFragment) con i rispettivi layout definiti in file XML. Per visualizzare elenchi di contenuti dinamici, come il feed delle quest, utilizziamo RecyclerView. Questo componente è altamente ottimizzato per le prestazioni, garantendo uno scorrimento fluido anche con grandi quantità di dati, grazie al riciclo e riutilizzo delle viste. Adattatori personalizzati, come QuestAdapter e UserAdapter, vengono utilizzati per collegare i nostri modelli dati alle viste visualizzate nell'elenco.Domain LayerQuesto strato rappresenta il cuore logico dell'applicazione. I ViewModel sono responsabili di preparare e gestire tutti i dati richiesti dalla UI. Recuperano i dati dal Repository e li espongono come LiveData osservabili.

•Unidirectional Data Flow (UDF): Seguiamo un pattern UDF in cui i dati fluiscono in una sola direzione (dal ViewModel alla UI) e gli eventi nella direzione opposta (dalla UI al ViewModel). Quando i dati all'interno di un oggetto LiveData cambiano (ad esempio, vengono caricate nuove quest), questo notifica automaticamente il suo osservatore (il Fragment), che quindi aggiorna la UI. Questo modello reattivo rende lo stato dell'app prevedibile e molto più facile da debuggare.Data LayerIl Data Layer è costruito sul Repository Pattern, stabilendo un'unica fonte di verità.

•Repository: Il QuestRepository, ad esempio, fornisce un'API pulita per il domain layer (es. getQuests(), saveQuest(quest)).

•Data Sources: Internamente, il repository gestisce le chiamate a due tipi di fonti dati:1.Remote Data Source: Questo componente è responsabile di tutta la comunicazione di rete. Interagisce con i nostri servizi di backend su Firebase (Firestore) per salvare e recuperare quest, profili utente e altri dati dinamici.

2.Local Data Source: Prevediamo di utilizzare la libreria di persistenza Room per implementare un database locale. Questo fungerà da cache per i dati recuperati dalla rete, consentendo un accesso rapido e fornendo una modalità offline di base in cui gli utenti possono ancora visualizzare i contenuti caricati in precedenza. Il repository conterrà la logica per decidere se recuperare dati aggiornati dalla rete o servirli dalla cache locale.

4. FunzionalitàAutenticazione e RegistrazioneL'accesso all'app è gestito tramite Firebase Authentication. La StartActivity orchestra l'intero flusso di login e registrazione. Gli utenti possono:

•Registrarsi con un'email e una password.

•Effettuare il login con le proprie credenziali esistenti.

•Utilizzare il Google Sign-In per un'esperienza di autenticazione più rapida e snella.Creazione delle QuestGli utenti autenticati possono creare nuove quest tramite il QuestCreateFragment. Questo flusso è centrato su una mappa interattiva fornita dal Google Maps SDK.

•Progettazione Visiva: Gli utenti possono progettare visivamente la loro avventura toccando sulla mappa per posizionare dei waypoint.

•Definizione delle Tappe: Ogni waypoint rappresenta una tappa della quest, definita dal modello QuestLocation. I creatori possono arricchire ogni tappa con un titolo e un indizio descrittivo o un enigma.

•Salvataggio: Una volta completata la quest, l'intero oggetto Quest viene salvato su Firebase Firestore, rendendolo immediatamente disponibile per essere scoperto e giocato da altri utenti della community.Partecipazione alle QuestIl FeedFragment mostra un elenco di quest disponibili a cui gli utenti possono partecipare.

•Progressione: L'app mostra la prima tappa sulla mappa con il relativo indizio. Utilizzando il permesso ACCESS_FINE_LOCATION, l'app traccia la posizione GPS dell'utente.

•Completamento: Quando un utente si avvicina a un waypoint, l'app contrassegna quella tappa come "completata" e rivela automaticamente la successiva, creando un senso di progressione fluida fino al termine dell'avventura.

•Interazione: Gli utenti possono lasciare commenti sulle quest che hanno giocato (Comment.java), favorendo un senso di community.

Profilo Utente

Una sezione dedicata al profilo consente agli utenti di:

•Visualizzare e gestire le quest che hanno creato.•

Tenere traccia delle quest che hanno completato.

•Gestire le impostazioni dell'account ed effettuare il logout.

5. Testing

Per garantire la qualità del codice, è essenziale una strategia di test multi-livello:

•Unit Test: Vengono utilizzati per i ViewModel e i Repository. Usando JUnit e una libreria di mocking come Mockito, possiamo verificare la logica di business in completo isolamento dal framework Android. Ad esempio, un test unitario può confermare che un ViewModel chiami il metodo corretto sulla sua dipendenza Repository quando viene simulata un'azione dell'utente.

•Integration Test: Questi test verificano l'interazione tra diversi componenti. Ad esempio, scriveremo test di integrazione per confermare che le nostre query al database Room per la memorizzazione nella cache funzionino correttamente.

•UI Test: Utilizzando il framework Espresso, possiamo scrivere test automatizzati che simulano interi percorsi utente. Questi test vengono eseguiti su un dispositivo o emulatore e verificano che la UI si comporti come previsto. Ad esempio, un test della UI potrebbe simulare un utente che effettua il login, naviga alla schermata di creazione della quest, compila i dettagli e salva con successo la quest, verificando che in ogni passaggio venga mostrato lo stato corretto della UI.

6. Sviluppi Futuri

Per arricchire ulteriormente l'esperienza di QuestMaster, prevediamo di esplorare le seguenti funzionalità:

•Quest Private e Condivisione: Consentire agli utenti di creare quest private da condividere solo con gli amici tramite un link o un codice univoco.

•Indizi Multimediali: Permettere ai creatori di caricare immagini, audio o video come indizi per le tappe, utilizzando Firebase Storage per ospitare i file multimediali.

•Classifiche e Punti: Introdurre un sistema di punteggio basato sul tempo di completamento o sul numero di quest terminate per incoraggiare una competizione amichevole.

•Modalità Offline Migliorata: Consentire agli utenti di scaricare intere quest, inclusi dati della mappa e indizi, per un'esperienza di gioco completamente offline in aree con scarsa connettività.

•Integrazione con la Realtà Aumentata (AR): Utilizzare ARCore per visualizzare indizi, modelli 3D o aiuti di navigazione nel mondo reale attraverso la fotocamera del dispositivo, creando un'avventura più immersiva.

7. Appendice (Tecnologie Utilizzate)

•Linguaggi: Java, Kotlin

•Architettura: MVVM (Model-View-ViewModel)

•UI: Android View System, XML Layouts, Material Design 3

•Navigazione: Jetpack Navigation Component

•Asincronia: Kotlin Coroutines e Flow

•Networking e Backend:

•Firebase Authentication per la gestione degli utenti

•Firebase Firestore per lo storage di dati in tempo reale

•Firebase Storage (per futuri file multimediali)

•Database Locale: Room (per caching e supporto offline)

•Mappe: Google Maps SDK for Android

•Librerie Esterne:

•Glide per il caricamento efficiente delle immagini

•Google Play Services (Auth, Maps)

•Material Components for Android