

Relazione di Progetto

OceanTunes.
Give value to your music



Magliani Andrea - 894395

Paparella Nicolò - 899537

Perego Luca - 894448

Picco Nicolas - 894588

Gruppo TunaGold

Introduzione.....	3
Funzionalità.....	3
Descrizione Schermate.....	4
Login e Registrazione.....	4
Home.....	4
Pagine “Scopri di più”.....	4
Ricerca.....	5
Profilo.....	5
Dettaglio Canzoni.....	5
Librerie utilizzate.....	6
UI.....	6
Librerie Esterne.....	6
Librerie Importanti.....	6
Storage.....	6
Architettura.....	7
Auth Activity.....	7
Auth Nav.....	7
Main Activity.....	7
Nav Host.....	8
User Interface (UI).....	8
ViewModel.....	8
Repository.....	8
Storage.....	9
Design.....	10
Icona.....	10
Palette dei Colori.....	11
Pattern Utilizzati.....	12
Model-View-ViewModel (MVVM).....	12
Dependency Injection.....	12
Repository Pattern.....	12
Result Pattern.....	12
Singleton.....	13
Sviluppi Futuri.....	14

Introduzione

OceanTunes è un'applicazione che permette di valutare e salvare canzoni, creando dei sistemi di classifiche che permettono di comparare i brani più amati.

Tramite OceanTunes gli utenti possono tenere traccia dei loro brani più amati mettendoli tra i preferiti e dando voti, inoltre tramite i voti vengono stilate classifiche globali con cui gli utenti possono vedere le canzoni più popolari e votate.

Il progetto è stato realizzato interamente in Kotlin su Android Studio, IDE creato da JetBrains.

Funzionalità

L'applicazione offre la possibilità di navigare uno dei più grandi ed attivi archivi di canzoni al mondo, creando una propria lista di brani apprezzati tramite il tasto "Mi piace" o valutando la canzone in questione tramite l'intuitivo sistema di votazione a stelle.

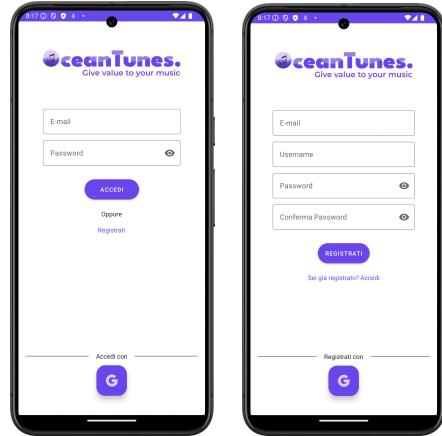
Inoltre è possibile confrontare i propri voti con un database aggiornato in tempo reale che permetterà ad ogni utente di vedere le valutazioni medie delle canzoni e sapere quanti utenti hanno il brano in questione tra i preferiti.

Per facilitare l'accesso all'applicazione, l'utente ha la possibilità di effettuare il "Login con Google", evitando di dover creare un account ad hoc.

Descrizione Schermate

Login e Registrazione

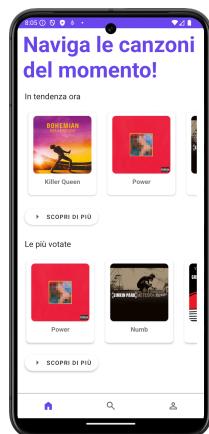
La pagina login permette all'utente di accedere all'applicazione. Come detto nel paragrafo precedente, in questa pagina si hanno due possibilità: l'accesso tradizionale tramite account con E-Mail e password oppure l'accesso con Google. In caso si scelga di effettuare l'accesso tramite E-Mail e password, è prima necessario creare un account tramite l'apposita pagina di registrazione.



Home

L'homepage è il principale punto di scambio con il bacino di utenza di OceanTunes, tramite questa pagina è possibile scoprire le **canzoni più apprezzate** dagli utenti e **quelle più votate** (che sia per la loro bellezza o meno) in due sezioni specifiche.

Vengono raggruppate le prime in rilevanza nei due caroselli, per scoprire poi le successive l'utente è invitato a cliccare il pulsante "Scopri di più".



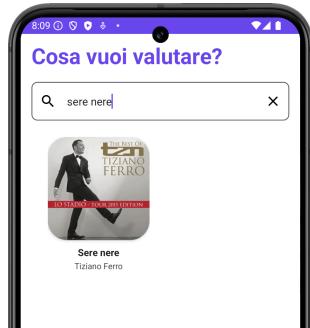
Pagine “Scopri di più”

Sono pagine che permettono di vedere più elementi di sezioni come “tendenze” o “preferiti”. Sono fatte per accedere subito ai brani ed espandere le viste. Vi è una funzione di sorting che consente di ordinare le canzoni per recenti o date [secondo la valutazione utente].



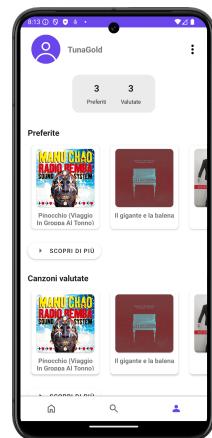
Ricerca

La ricerca consente di trovare le canzoni desiderate attraverso una griglia aggiornata in tempo reale secondo le keyword di ricerca inserite. Vengono mostrate le canzoni con la corrispondenza di titolo e artista migliore. È possibile in qualsiasi momento accedere alle ricerche recenti, che salva le canzoni recentemente consultate durante la ricerca.



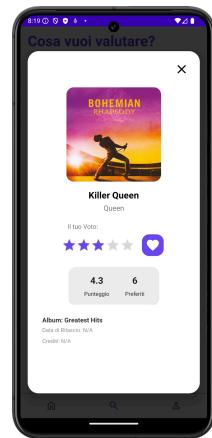
Profilo

La terza tab principale dell'app, Profilo, contiene informazioni su nome utente, quante canzoni l'utente ha valutato, quante ne ha messe nei preferiti. Vi sono poi due caroselli, uno con le canzoni **Preferite** e l'altro con le canzoni **Recentemente valutate**. Attraverso i pulsanti "Scopri di più" si può vedere una vista più estensiva di queste due. Vi è infine un menù a tendina per modificare il proprio nome utente e uscire dal proprio account (logout).



Dettaglio Canzoni

Una card che **compare in sovrapposizione** al contenuto della pagina corrente, con uno slide dal basso. Contiene informazioni su: **titolo, artista, il tuo voto, like** (preferiti), **punteggio medio**, quanti **hanno salvato la canzone nei preferiti**, **nome dell'album**, **data di uscita** e **crediti**. La card può essere chiusa in qualsiasi momento con l'apposito pulsante oppure premendo in una qualsiasi area in background.



Librerie utilizzate

UI

- Navigation
- Lifecycle
- Glide

Librerie Esterne

- Ktor
- Google Identity
- Google Services/Auth

Librerie Importanti

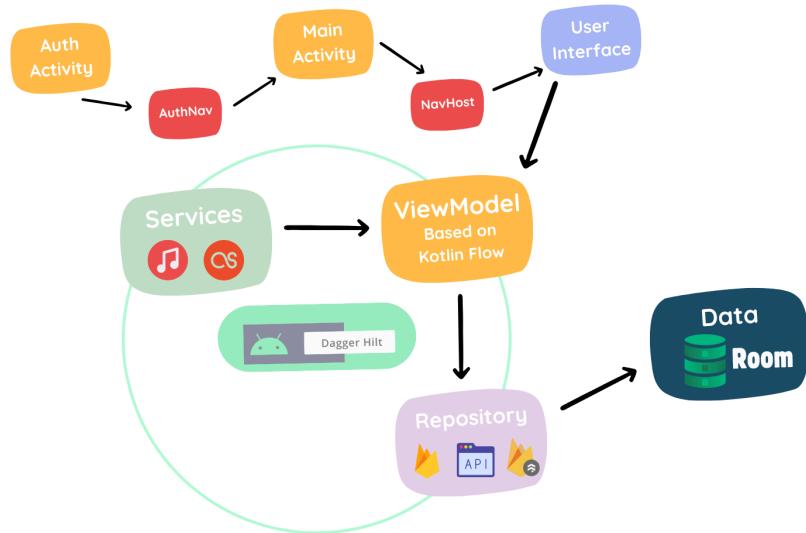
- Hilt.
- Lifecycle

Storage

- Room
- Firebase Firestore
- Firebase Auth

Architettura

L'applicazione segue l'architettura mostrata nel seguente grafico:



Auth Activity

Auth Activity è l'activity che gestisce l'autenticazione degli utenti tramite **Firebase Authentication**, includendo sia il login che la registrazione. Rappresenta il punto d'ingresso dell'applicazione per gli utenti non ancora autenticati.

Auth Nav

Auth Nav si occupa di gestire la navigazione dall'Auth Activity alla Main Activity.

Main Activity

Main Activity è il nucleo dell'applicazione, sviluppato utilizzando kotlin e seguendo gli standard di Material 3, per mantenere un design moderno e standard di accessibilità elevati.

Nav Host

Navigation è utilizzato da OceanTunes per gestire la navigazione nell'app e **NavController**, che definisce il punto di partenza dell'app e la navigazione in essa.

Le schermate principali includono: Login, Registrazione, Home, Ricerca, Profilo.

Inoltre l'applicazione include una navbar che permette di navigare le tre schermate principali dell'applicazione.

User Interface (UI)

L'architettura dell'interfaccia utente segue un modello standard android, MVVM (**Model-View ViewModel**) e il componente di navigazione permette la gestione delle schermate. I componenti che costituiscono l'app, come pulsanti, griglie e card sono state costruite come classi con attributi personalizzabili, in modo da essere riutilizzate con la massima flessibilità.

ViewModel

L'architettura di **ViewModel** in OceanTunes segue un pattern ben definito, garantendo la separazione della logica di business e dello stato della UI dalla View (Fragment/Activity), la sopravvivenza dei dati durante i cambiamenti di configurazione e la comunicazione reattiva con i repository tramite LiveData o Flow per aggiornamenti asincroni dei dati.

Repository

L'architettura di OceanTunes adotta il pattern repository in modo ben definito, possedendo repository principali come: SongRepository, UserRepository e LastFMRMRepository e altre.

AppModule si occupa della dependency injection, garantendo una separazione chiara delle responsabilità tra i componenti.

Ogni repository implementa un'interfaccia che definisce il modello per le operazioni sui dati, consentendo la flessibilità di cambiare l'implementazione sottostante senza influenzare il resto dell'applicazione.

Storage

OceanTunes adotta un'architettura che integra **Room** per la gestione dei dati in modo efficiente. Questo è utilizzato come database locale per la memorizzazione di informazioni, la gestione di operazioni CRUD utilizzando un'interfaccia DAO e supportando i flussi con Kotlin Flows.

L'utilizzo di un database locale permette di avere una sola fonte di verità, aderendo così al pattern Single Source of Truth (SSoT), come richiesto dall'architettura MVVM.

Design

OceanTunes è stata progettata seguendo le linee guida di Material Design, la filosofia di design supportata da Android.

Il design segue i principi di Material Design 3, generando un design moderno, coerente con gli standard e le applicazioni Android e mantenendo alta l'accessibilità.

La progettazione dell'applicazione è stata fatta tramite la creazione di mockup in Figma, questo ha permesso di creare un design uniforme lungo tutta l'applicazione, che poi è stato realizzato in Android Studio.



Icona

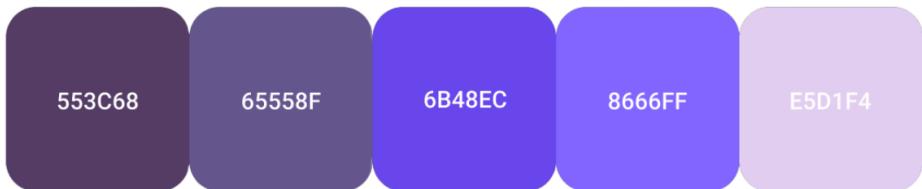
L'icona dell'applicazione è stata progettata per comunicare immediatamente l'idea della musica all'utente. Inoltre, l'ispirazione all'oceano nel logo (e quindi del nome associato all'applicazione) cercano di ricordare all'utente "il mare di brani" a sua disposizione.

Il logo è stato realizzato con Photoshop in diversi formati e lungo una serie di iterazioni che hanno portato all'ottenimento della paletta cromatica che caratterizza l'applicazione.



Palette dei Colori

La palettes di colori di è coerente lungo tutti i media legati all'applicazione, dal logo agli elementi di accento presenti in essa.



Pattern Utilizzati

Model-View-ViewModel (MVVM)

MVVM è un pattern architetturale che separa la logica di presentazione dalla logica di business.

- Il Model gestisce i dati, la validazione e le regole di presentazione della logica di business.
- Il View è responsabile dell'interfaccia utente, osserva il ViewModel per aggiornamenti e inoltra le interazioni dell'utente.
- Il ViewModel è il ponte che collega i due elementi elencati precedentemente, gestendo lo stato della UI e la logica di business senza fare riferimenti diretti alla View.

Questo approccio offre multipli vantaggi, tra cui la separazione delle responsabilità, la testabilità del codice senza UI, la riusabilità dei componenti e una gestione centralizzata dello status della UI.

Dependency Injection

Dependency Injection (DI) è un pattern di progettazione in cui le dipendenze vengono “iniettate” in una classe invece di essere create al suo interno. Questo riduce l'accoppiamento tra i componenti, rende il codice mantenibile e promuove la separazione delle responsabilità.

Repository Pattern

Repository è un pattern architetturale che estrae la logica di accesso ai dati dai ViewModel o da altri componenti. Questo permette di separare le fonti dei dati dalla logica di business.

Result Pattern

Result è un pattern per la gestione degli errori che utilizza un tipo Result personalizzato al posto della gestione delle eccezioni tradizionali. Ciò migliora la chiarezza del codice, evita il costo delle eccezioni e rende più esplicito il flusso dei dati, facilitando la gestione degli esiti.

Singleton

Singleton è un pattern creazionale usato per assicurare che una classe abbia una sola istanza e fornire un punto di accesso globale ad essa. Questa tecnica permette di evitare la creazione di istanze multiple che accedono allo stesso set di risorse condivise.

Sviluppi Futuri

In futuro si prevede il supporto continuato dell'applicazione tramite l'aggiunta di diverse feature, tra cui:

- **Connettività:** Maggiore interazione tra gli utenti grazie alla possibilità di aggiungere amici e visualizzare i loro profili.
- **Usabilità:** Introduzione di raccomandazioni basate sui gusti dell'utente e non solo in base alla popolarità e voti delle canzoni.
- **Snippet Audio:** Possibilità di ascoltare estratti di canzone, in modo da facilitarne il ricordo e la valutazione.
- **supporto di playlist:** Possibilità di dare voti alle playlist e avere quindi classifiche a parte per esse.
- **API più stabili:** Utilizzo di API più stabili e che contengano una quantità di brani maggiore