Proposta di progetto

Nome del progetto di laboratorio:

OtakuTube

Gruppo

Informazioni sul gruppo di laboratorio.

|  |  |
| --- | --- |
| Nome del gruppo: | OtakuTuber |
|  |  |
| Componenti: | \* Alessandro Scicolone  \* Daniela Magrì |

Date

Le date principali del documento.

|  |  |
| --- | --- |
| **Data di sottomissione della proposta di progetto** | 04/07/2024 |
|  |  |
| Data di accettazione della proposta di progetto | ##DATA ACCETTAZIONE## |

Descrizione breve

Descrizione della App in una frase.

|  |
| --- |
| Con questa applicazione si ha la possibilità di creare il proprio mondo anime senza confini: streaming gratuito, senza pubblicità invasiva, con una libreria personalizzata e la possibilità di salvare i tuoi preferiti. Tutto ciò che l’utente desidera, a portata di click. |

Definizioni

Di seguito la definizione dei termini, abbreviazioni e acronimi utilizzati.

|  |  |
| --- | --- |
| **Termine** | **Definizione** |
| Anime | Animazione giapponese |
| Otaku | Appassionato di anime, manga o videogiochi giapponesi |
|  |  |
|  |  |

Sommario

[Gruppo 2](#_Toc171353595)

[Date 2](#_Toc171353596)

[Descrizione breve 2](#_Toc171353597)

[Definizioni 2](#_Toc171353598)

[Sommario 3](#_Toc171353599)

[1 Contesto del progetto 5](#_Toc171353600)

[1.1 Situazione attuale 5](#_Toc171353601)

[1.2 Benefici e creazione di valore 5](#_Toc171353602)

[1.3 Obiettivi del progetto 5](#_Toc171353603)

[2 Profilo del progetto 6](#_Toc171353604)

[2.1 Ambito del progetto 6](#_Toc171353605)

[2.2 Profilo della soluzione da realizzare 6](#_Toc171353606)

[3 Vincoli e assunti (Requirements) 9](#_Toc171353607)

[3.1 Vincoli tecnologici 9](#_Toc171353608)

[4 Design 9](#_Toc171353609)

[4.1 MVVM 9](#_Toc171353610)

[4.2 Room 9](#_Toc171353611)

[4.3 API 9](#_Toc171353612)

[4.4 Player Video 9](#_Toc171353613)

[5 Implementazione 10](#_Toc171353614)

[5.1 Navigation 10](#_Toc171353615)

[5.2 Model 10](#_Toc171353616)

[5.3 Services 10](#_Toc171353617)

[5.4 Room 11](#_Toc171353618)

[5.5 Activity 11](#_Toc171353619)

[5.6 View Model 12](#_Toc171353620)

[5.6.1 Inizializzazione e Repository 12](#_Toc171353621)

[5.6.2 Operazioni sul DB 12](#_Toc171353622)

[5.6.3 Funzioni CRUD (Create, Read, Update, Delete) 12](#_Toc171353623)

[5.6.4 Funzioni di recupero dei dati 12](#_Toc171353624)

[5.6.5 Gestione dello stato 13](#_Toc171353625)

[5.6.6 Funzioni di utilità 13](#_Toc171353626)

[5.6.7 Funzioni di servizio privato 14](#_Toc171353627)

[5.6.8 Test della connessione 14](#_Toc171353628)

[6 Test 14](#_Toc171353629)

[6.1 Database test 14](#_Toc171353630)

[6.2 Server connection test 15](#_Toc171353631)

[6.3 UI test 15](#_Toc171353632)

[7 Conclusioni 15](#_Toc171353633)

# Contesto del progetto

## Situazione attuale

Attualmente su Play Store ci sono diverse applicazioni che riguardano lo streaming di anime, la principale è Crunchyroll. Con OtakuTube vorremmo cercare di colmare le seguenti mancanze:

* **Pubblicità invasiva**: due o più pubblicità nel mezzo della visione;
* **Molte delle funzionalità dell’applicazione Crunchyroll**, come lo streaming offline, **sono inibite per gli utenti che non pagano un abbonamento premium**. Inoltre, sempre senza quest’ultimo, **molti degli anime presenti non possono essere visti oppure essere visti solamente a metà.**

## Benefici e creazione di valore

OtakuTube si propone di avere al massimo una sola pubblicità ad inizio visione e di offrire tutte le sue funzionalità in modo completamente gratuito.

## Obiettivi del progetto

|  |  |
| --- | --- |
| MVVM | L’applicazione sarà strutturata utilizzando MVVM in modo tale da dividere il Model dalla View |
| Room  e Flows | L’applicazione userà Room per implementare la funzione di streaming offline e la libreria |
| Jetpack Compose | Per implementare l’UI dell’applicazione |
| API | L’applicazione chiamerà l’API per ottenere i video |
| Player Video | L’applicazione sfrutterà Player Video per ottenere i permessi necessari per la visualizzazione |

# Profilo del progetto

## Ambito del progetto

**Funzionalità presenti:**

1. **Material design**: per garantire la qualità e la praticità dell’UI;
2. **Libreria personalizzata**: consentire agli utenti di creare e gestire librerie dove possono aggiungere titoli che intendono guardare o che hanno già visto;
3. **Gestione del profilo utente**: consentire agli utenti di creare e gestire facilmente il proprio profilo utente;
4. **Streaming offline:** fornire la possibilità agli utenti di scaricare i video per guardarli offline;
5. **Categorie e raccomandazioni**: gli anime sono organizzati in categorie e fornisce raccomandazioni personalizzate agli utenti in base alle loro preferenze della community;
6. **Ricerca avanzata**: offrire una funzione di ricerca avanzata tramite dei filtri che consente agli utenti di trovare facilmente gli anime desiderati per titolo e genere;
7. **Modalità offline**: se non c’è connessione internet o i server sono irragiungibili, l’utente ha accesso solo alla visione delle sue librerie e del suo profilo utente;
8. **Server**: l’app si appoggia a 2 server già online, il principale serve per ottenere tutte i dati, compresi link per la visione degli episodi e il secondo server per poter ottenere i trailer degli specifici anime.

*Nota: al momento l’API a cui si appoggia il secondo server potrebbe essere in manutenzione, quindi potrebbe non riuscire ad ottenere i trailer;*

**Possibili funzionalità da aggiungere:** salvare i video scaricati direttamente dentro l’app e aggiungere la visualizzazione degli anime appena usciti.

## Profilo della soluzione da realizzare

L’applicazione è suddivisa in quattro sezioni principali:

1. **Sfoglia:** la prima sezione dell’applicazione dove appare una lista di anime consigliati, in base agli anime preferiti dalla community.
2. **Esplora:** la seconda sezione dell’applicazione dove vengono visualizzate diverse liste di anime suddivisi per genere, in modo tale da facilitare la ricerca.
3. **Libreria:** la terza sezione dell’applicazione dove l’utente può creare le sue liste personalizzate, e quelli visti di recente.
4. **Profilo:** la quarta sezione dell’applicazione dove vi sono i dati dell’utente, come nome e immagine profilo.

Ogni sezione principale corrisponde ad un modulo ed implementate con il framework Jetpack Compose. In aggiunta vi sono delle sezioni secondarie: quella che appare premendo e digitando sulla barra di ricerca \*, quella che appare premendo la card dell’anime \*\*, quella che appare premendo la card della playlist \*\*\**(l’utente avrà una lista “Favourite”* *di default)* e quella che appare premendo il pulsante “Episodes” \*\*\*\* (*selezionando un episodio apparirà infine il player video*). Anche per le sezioni secondarie corrisponde un modulo implementato con il framework Jetpack Compose.

Ogni componente sarà implementato in modo tale da essere riutilizzabile.

**1** **2** **3**

Immagine che contiene testo, schermata, Software multimediale, software

Descrizione generata automaticamenteImmagine che contiene testo, schermata, grafica, Software multimediale

Descrizione generata automaticamenteImmagine che contiene testo, schermata, grafica, Software multimediale

Descrizione generata automaticamente

**\*** **\*\*** **\*\*\***

Immagine che contiene testo, cartone animato, narrativa, grafica

Descrizione generata automaticamenteImmagine che contiene testo, schermata, Sito Web, software

Descrizione generata automaticamenteImmagine che contiene testo, schermata

Descrizione generata automaticamente

**4** **\*\*\*\***

Immagine che contiene testo, schermata, cartone animato, grafica

Descrizione generata automaticamente Immagine che contiene schermata, testo, viola

Descrizione generata automaticamente

*Ulteriori screen sono presenti all’interno della cartella “features”.*

# Vincoli e assunti (Requirements)

## Vincoli tecnologici

L’applicazione sarà disponibile per le versioni Android 10 e superiori.

# Design

## MVVM

Implementiamo il nostro ViewModel nel file *MyViewModel.kt*, salvando le variabili di cui vogliamo tenere traccia dello stato, cosi da aggiornare correttamente lo stato dei componenti @Composable che utilizzano quei dati.

Per esempio se abbiamo già caricato almeno 1 volta gli anime popolari (che compaiono nella sezione Zapping), settiamo la variabile *isZappingScreenLoaded* a *true*, e i dati caricati in *popularAnime*, cosi quando ritorneremo allo *ZappingScreen*, non dovremmo ricaricare di nuovo gli anime popolari, ma basterà leggere *isZappingScreenLoaded* per decidere se fare la chiamata al API o meno.

Per la modifica delle variabili, utilizziamo apposite *Set* in modo da rendere sicuro il suo accesso.

In questo modo il ViewModel fa da “mediatore” tra View e Model dividendo le due logiche.

Inoltre tiene traccia anche dei valori dei dati salvati su room, che utilizzando Flow, aggiorna i dati appena questi ultimi cambiano.

## Room

Per quanto riguarda Room, lo utilizziamo per salvare le preferenze dell’utente (nome e immagine profilo), ma soprattutto le playlist e i preferiti.

Quindi con le classi Entity, rappresentiamo le entità, cioè le tabelle, nei Dao definiamo le operazioni possibili e nei Repository come queste devono essere utilizzate.

## API

Per le chiamate API utilizziamo la libreria ktor per fare le chiamate http ai 2 server esterni.

## Player Video

Per il player video degli episodi utilizziamo la libreria ExoPlayer, caricando nella lista di video del player man mano gli episodi dell’anime selezionato, cosi da poter navigare tra gli episodi direttamente dal player.

Per il player video dei trailer, utilizziamo l’api documentata nella seguente repo: <https://github.com/PierfrancescoSoffritti/android-youtube-player>

Visto che i trailer sono link di youtube, che non sono accettati nel player video di ExoPlayer.

# Implementazione

Il progetto è stato organizzato in modo modulare, suddividendo il codice in diverse cartelle per garantire una struttura ordinata, migliorando così la leggibilità e l'usabilità complessiva.

## Navigation

navigation route "home\_screen"

navigation route "zapping\_screen"

navigation route "lib\_screen"

navigation route "profile\_screen"

navigation route "anime\_{name}\_{id}"

Abbiamo definito quattro route: le principali schermate delle sezioni e una schermata dedicata alla pagina dell'anime selezionato. Quest'ultima route richiede due parametri di input, ovvero il nome e l'id dell'anime, in modo che, una volta attivata, possiamo ottenere i dati corretti dai server.

Navighiamo tra le sezioni principali dell'applicazione tramite una "NavigationBar". Per maggiori dettagli, consultare il file ui/components/BottomNavigation.kt. Invece navighiamo all’interno dello screen dell’anime una volta che premiamo una card (*le card sono all’interno del file ui/components/Cards.kt*); ci sono diversi tipi di card, in particolare quelle che gestiscono l’evento di click con la navigazione alla schermata dell’anime della card selezionato sono: “AnimeCard”, “AnimePopularCard” e “AnimeCardPlaylist”.

Le schermate della lista degli episodi, del risultato della search, della lista degli anime nella playlist sono dialog, ovvero componenti che appaiono in risposta ad una determinata richiesta.

## Model

Nel model troviamo le classi che rappresentano i dati reperiti dai server che utilizza la nostra app.

## Services

Il file AnimeService.kt definisce un servizio per gestire le richieste HTTP relative agli anime, utilizzando la libreria **Ktor** per Android. Il servizio include metodi per ottenere informazioni sugli anime, episodi, trailer e generi da due server remoti, *URLNPM* e *URLPYTHON*. Utilizza **Gson** per la deserializzazione le risposte JSON.

La connessione ai server è verificata tramite il metodo *testConnection()*, che tenta di connettersi ai due URL entro un timeout di 10 secondi. In caso di successo, i metodi del servizio utilizzano Util.GET per effettuare le richieste HTTP e restituiscono i dati deserializzati. In caso di fallimento della connessione, i metodi restituiscono risultati vuoti o nulli.

## Room

Il file AppDatabase.kt definisce un database Room di nome "OtakuTubeDB".

Questo database include quattro entità: **PlaylistEntity**, **PlaylistAnimeRelationEntity**, **UserEntity** e **AnimeEntity**. Sono presenti tre DAO: **PlaylistDao**, **UserDao** e **AnimeDao**, per l'accesso ai dati delle rispettive entità.

Il database è implementato come singleton, garantendo un'unica istanza condivisa tramite il metodo *getDatabase(context: Context).* Questo metodo utilizza Room.databaseBuilder per creare l'istanza del database, abilitando la migrazione distruttiva con fallbackToDestructiveMigration(), che cancella e ricrea il database in caso di cambiamenti nella struttura delle entità.

## Activity

Il file MainActivity.kt definisce l'attività principale dell'applicazione "OtakuTube" utilizzando Jetpack Compose per la costruzione dell'interfaccia utente. La classe MainActivity estende ComponentActivity e gestisce un'istanza del MyViewModel, utilizzata per mantenere lo stato dell'applicazione.

Nel metodo *onCreate*, l'interfaccia utente viene impostata utilizzando il tema **ANITESTTheme** e viene chiamata la funzione *MyActivityContent*.

La funzione MyActivityContent gestisce l'interfaccia utente principale dell'attività. All'inizio, viene determinata la configurazione corrente delle risorse. Viene poi osservato lo stato della connessione all'interno di un blocco LaunchedEffect richiamando la funzione *testConnection()*; si può ricadere in tre casi:

1. La connessione dati del telefono è disattivata;
2. La connessione dati del telefono è attivata ma i server sono irraggiungibili;
3. La connessione dati del telefono è attivata e il server è raggiungibile.

Nei casi 1 e 2, si accede all’applicazione in modalità offline, ovvero vengono visualizzate solo le sezioni della libreria (*con delle limitazioni sulle funzionalità che richiedono l’utilizzo di internet*) e del profilo, che utilizzando il database non richiedono la connessione dati. Inoltre, nella *NavigationBar*, al posto degli item per navigare nelle altre sezioni vi è un altro item che al suo click ritenta la connessione. Quando poi la *testConnection()* restituirà il valore true, ossia avvenuta connessione, l’applicazione viene visualizzata con tutte le sue funzionalità.

La funzione SideEffect viene utilizzata per gestire le impostazioni della barra di stato e della barra di navigazione in base all'orientamento dello schermo. Se l'orientamento è in modalità paesaggio, le barre di stato e di navigazione vengono impostate su trasparente e viene abilitata la modalità immersiva, questo avviene quando viene aperto il video player per guardare gli episodi. Altrimenti, le barre vengono impostate su un colore specifico in base al tema dell’applicazione.

Inizialmente viene visualizzato uno schermo di caricamento con l'icona dell’app e il testo "OtakuTube". Appena la fase di caricamento è conclusa, viene eseguita la navigazione all'interno dell'applicazione utilizzando la funzione Navigation, passando il viewModel, il contesto e il route selezionato (*Screen.Home.route o Screen.Library.route in base allo stato della connessione*).

## View Model

La classe MyViewModel gestisce diverse operazioni asincrone relative agli anime, playlist e utenti.

### Inizializzazione e Repository

* Viene inizializzato un'istanza di AnimeService per gestire le chiamate di rete.
* I repositori PlaylistRepository, UserRepository e AnimeRepository vengono inizializzati utilizzando i DAO corrispondenti ottenuti dal database AppDatabase.
* Viene creata una playlist di default chiamata "Favourite" e un utente di default se non esistono.

### Operazioni sul DB

### Funzioni CRUD (Create, Read, Update, Delete)

* **Playlist**:
  + insertPlaylist(playlist: PlaylistEntity)
  + deletePlaylist(playlist: String)
  + deleteRelation(relation: PlaylistAnimeRelationEntity)
* **Anime**:
  + insertAnime(anime: AnimeEntity)
  + deleteAnime(anime: AnimeEntity)
  + deleteAll(): Elimina tutti gli anime dal database.
* **User**:
  + insertUser(user: UserEntity)
  + deleteUser(id: Int)

### Funzioni di recupero dei dati

* **Playlist**:
  + getPlaylist(name: String): Recupera una playlist con i relativi anime.
* **User**:
  + getUser(): Recupera l'utente corrente.
* **Anime**:
  + getAnime(id: String): Recupera un anime per ID.
  + getMaxInsertOrderAnime(): Ottiene l'ordine massimo di inserimento.
  + getOldestInsertedAnime(): Ottiene l'anime inserito più vecchio.

### Gestione dello stato

* **AnimeInfo**:
  + \_animeInfoTrailer e \_animeInfo gestiscono le informazioni sugli anime e i relativi trailer.
  + \_animeForGenres, \_popularAnime, \_genres e \_animeSearch gestiscono le liste degli anime per generi, popolari, generi e risultati di ricerca rispettivamente.
* **Episodi**:
  + \_currentEpisodeIndex, \_episodes, \_currentEpisode gestiscono l'indice e la lista degli episodi correnti.
* **Schermate**:
  + \_isSearchScreenOpen, \_isEpisodesButtonOpen, \_isAnimeScreenLoaded, \_isZappingScreenLoaded, \_isExploreScreenLoaded, \_isCategoryRowLoaded gestiscono lo stato di caricamento delle varie schermate.
* **Connessione**:
  + \_connection gestisce lo stato della connessione.

### Funzioni di utilità

* **Impostazione dello Stato**:
  + setIsLoadedAnimeScreen(flag: Boolean)
  + setIsLoadedZappingScreen(flag: Boolean)
  + setIsLoadedExploreScreen(flag: Boolean)
  + setIsLoadedEpisode(flag: Boolean)
  + setIsLoadedCategory(category: String, flag: Boolean)
* **Gestione degli Episodi**:
  + initEpisodes(n: Int): Inizializza una lista di episodi con elementi nulli.
  + addEpisodes(episode: Episode): Aggiunge un episodio alla lista.
  + setEpisode(episodeId: String): Imposta l'episodio corrente.
* **Gestione dei Filtri**:
  + addFilterRequest(genre: Genre): Aggiunge un filtro per genere.
  + removeFilterRequest(genre: Genre): Rimuove un filtro per genere.
  + removeAllFilterRequest(): Rimuove tutti i filtri.
* **Gestione dei dialog**:
  + openEpisodes(), closeEpisodes()
  + openSearch(), closeSearch()
* **Anime Info**:
  + setAnimeInfo(id: String): Imposta le informazioni sugli anime.
  + setAnimeInfoTrailer(name: String): Imposta i trailer degli anime.
* **Caricamento degli Anime**:
  + addAnimeByGenre(page: Number, genre: String)
  + addPopularAnime(page: Number)
  + setGenres()

### Funzioni di servizio privato

* getEpisode(episodeId: String): Recupera un episodio.
* getAnimeByGenre(page: Number, genre: String): Recupera una lista di anime per genere.
* getEpisodes(episodes: List<String>): Recupera una lista di episodi.
* getAnimeSearch(id: String): Recupera una lista di risultati di ricerca degli anime.
* getAllAnime(page: Number): Recupera una lista di tutti gli anime.
* getPopularAnime(page: Number): Recupera una lista di anime popolari.
* getAnimeInfo(id: String): Recupera le informazioni di un anime.
* getGenres(): Recupera la lista dei generi.
* getAnimeInfoTrailer(name: String): Recupera i trailer di un anime.

### Test della connessione

* testConnection(): Testa la connessione ai server e aggiorna lo stato della connessione.

# Test

Abbiamo svolto tre tipologie di test: i **test sulla connessione del server**, i **test sul database** e i **test sulla UI.**

## Database test

Il file AppDatabaseTest utilizza Room per testare la persistenza dei dati in AppDatabase. I test coprono tre funzionalità principali:

1. ***insertTest***: Verifica l'inserimento di una *PlaylistEntity* nella base di dati e controlla che la playlist sia correttamente salvata e recuperata. Inoltre verifica anche l'inserimento di un *AnimeDetail* nella playlist creata, verificando che l'anime sia correttamente associato alla playlist.
2. ***deleteTest***: Testa l'inserimento e successiva eliminazione di una *PlaylistEntity*, assicurandosi che la lista di playlist sia vuota dopo la rimozione.

Ogni test utilizza coroutine e Room per operazioni asincrone.

## Server connection test

I test nel file ServiceUnitTest verificano due aspetti del servizio *AnimeService*:

* Il primo test è la ***testConnectionTest*** che controlla che il metodo “testConnection()” restituisca true, confermando la connessione. Se il server è irraggiungibile sarà false, ma non è un errore se effettivamente il server non è raggiungibile, fino a quando i server diventano raggiungibili e restituisce true;
* Il secondo test è ***getAnimePopularTest*** verifica che il metodo “getPopularAnime(page)” ritorni una lista di anime non vuota. Abbiamo voluto testare se effettivamente riesce oltre che a connettersi al server, reperire dati da esso.

Entrambi i test utilizzano runBlocking per gestire le coroutine e assicurano il corretto funzionamento del servizio.

## UI test

Il file NavigationTest.kt contiene il test strumentale *testZappingScreen\_DisplayPopularText*, utilizzando Jetpack Compose per verificare che la schermata "Zapping" mostri correttamente il testo "Popular". Viene usato Mockito per creare un mock del ViewModel, configurato con dati di esempio come liste di anime ed elementi di navigazione.

Il test *testZappingScreen\_DisplayPopularText* configura l'interfaccia utente con questi mock e verifica che il testo "Popular" sia visualizzato correttamente utilizzando *composeTestRule*. Questo permette di verificare la corretta navigazione all'interno dell'app.

# Conclusioni

In conclusione, i requirments sono stati soddisfatti, come già detto il MVVM tramite il file MyViewModel.kt, rendendo più facile la scrittura in creazione del progetto, sia per le modifiche future, così da dividere le varie logiche.

Per quanto riguarda il requirement su JetPack Compose, abbiamo sfruttato le funzionalità del framework per poter dividere l’UI dell’app in vari componenti, così da essere riutilizzabili e leggibili.

Per le funzionalità dell’intera app, abbiamo sfruttato anche varie librerie (ktor, ExoPlayer, ecc..), API esterne e due server esterni che li abbiamo resi usufruibili potenzialmente da tutti e modificati\* in base alle nostre esigenze.

I test che come detto abbiamo fatto sulla navigazione, sul funzionamento di Room e sulle chiamate ai server esterni, permettendoci di controllare il corretto funzionamento delle principali funzioni.

\*Modificati poiché uno dei due server, è un API già fatta da un utente di Git, mentre il secondo è stato fatto interamente da noi appoggiandosi ad una libreria esterna.