# F.2 – Servizi Remoti e Retrofit

Finora abbiamo utilizzato una base dati presente all’interno della nostra app. Comunemente pero’, i dati di un’app vengono forniti da un servizio cloud remoto tramite connessione di rete.

Ci sono molti modi per interfacciarsi con un servizio remoto, il piu’ comune è sicuramente l’architettura **REST API**

Un'API (*Application Programming Interface*) è un insieme di regole predefinite che consentono a diverse applicazioni di comunicare tra loro.

REST (representational state transfer) è una serie di principi architetturali che permettono di standardizzare le operazioni di trasferimento dati network, e che descrivono il WEB.

## REST API service

<https://www.ibm.com/topics/rest-apis>

Le API REST sono formate da un server e un client. Ad esempio, il server è un servizio dati remoto, mentre il client è all’interno della nostra app.

Il server espone degli **endpoint:** degli [URI](https://en.wikipedia.org/wiki/Uniform_Resource_Identifier) dove riceve le richieste e invia le risposte. In un server remoto, questi sono una serie di indirizzi URL.

Il client invia richieste a questi endpoint per ottenere risposte specifiche per ogni endpoint.

Server e Client comunicano tramite richieste e risposte HTTP per eseguire funzioni di database standard come la creazione, la lettura, l'aggiornamento e l'eliminazione di record (noti anche come CRUD).

Ad esempio, il client di un'API REST può inviare al server una richiesta:

* POST per creare un record (C)reate
* GET per recuperare un record (R)etrieve
* PUT per aggiornare un record (U)pdate
* DELETE per eliminare un record (D)elete

Tutti i metodi HTTP possono essere utilizzati nelle chiamate API.

Le risposte sono comunemente in formato JSON.

## Retrofit

Retrofit è una libreria Client che consente di semplificare enormemente il processo di richiesta e risposta, integrando anche il processo di serializzazione e deserializzazione json tramite GSON in un unico flusso.

Interface

Retrofit Client

Client App

Un'implementazione di Retrofit è composta principalmente da due componenti:

* Un’interfaccia, dove vengono elencate le richieste specificando l’endpoint, il modello della data class che ci aspettiamo come risposta, e il tipo di richiesta HTTP (GET, POST…)
* Un client HTTP, che se necessario può essere configurato a nostra discrezione

In una tipica implementazione, il **Repository** invierà una richiesta a una Interfaccia Retrofit (che rappresenta una **data source**) invocando una delle sue funzioni. Queste saranno **suspend functions** e forniranno come risposta direttamente l’istanza di un oggetto.

Sarà Retrofit a occuparsi della gestione della chiamata HTTP e della deserializzazione della risposta.

### Setup Progetto

In build.gradle:app:



Il *converter* è la libreria che si occuperà della Ser\Deser – in questo caso, GSON.

## Interface

La prima parte dell'implementazione è dedicata all'interfaccia, che astrarrà gli endpoints del servizio remoto.

Andiamo sul sito del servizio che usiamo: The Movie DB, dove possiamo trovare la **reference (interface agreement)** dell’API

<https://developer.themoviedb.org/reference/intro/getting-started>

Vediamo che sulla destra vengono indicati gli endpoint piu’ popolari

A screenshot of a computer

Description automatically generated with low confidence

A noi interessa l’ultimo, /3/movie/now\_playing. Finora abbiamo utilizzato una risposta json di questo endpoint salvata nel file movies.txt. Adesso faremo in modo che ogni volta che si apre l’app, la lista venga aggiornata.

Trovando la chiamata nella lista, vedremo il suo indirizzo completo, e un esempio per essere invocata.

A screenshot of a computer

Description automatically generated with medium confidence

Nel nostro progetto android, crea un package *network* dentro il package *data.*   
Crea dentro di esso un’interfaccia chiamata *MovieNetworkAPI*

*A screen shot of a computer

Description automatically generated with medium confidence*

All’interno dell’interfaccia, aggiungi questa funzione per mappare l’endpoint:

@GET("/3/movie/now\_playing")  
suspend fun getMoviesNowPlaying():MoviesResponse

* L’annotazione *GET* identifica il tipo di chiamata HTTP da compiere, e l’endpoint dove farla
* La funzione è una **suspend fun**
* Il risultato è un MoviesResponse.

Per ottenere una chiamata funzionale, dobbiamo fare un altro step.

TMDB è una API esposta tramite HTTPS, quindi richiede una autorizzazione.

Aggiungi questa annotazione dopo GET per fornirgli un identificativo:

@Headers("Authorization: Bearer eyJhbGciOiJIUzI1NiJ9.eyJhdWQiOiJkOThkNjUyOWUwYmVjYmRkMmIzNDRhYTVmN2Q4M2VkYSIsInN1YiI6IjU4OTFhNjczOTI1MTQxMmRjZDAwNGYwMCIsInNjb3BlcyI6WyJhcGlfcmVhZCJdLCJ2ZXJzaW9uIjoxfQ.jl8QHdNFS3srkDw5\_n8ScNA5S608kFC2jC4m7UZmSiQ")

Questo è un **Access** **Token,** che di solito identifica un’app.

A screen shot of a computer program

Description automatically generated with low confidence

Un singolo token puo’ fare solo un certo numero di richieste al minuto\ora\giorno.

Per ottenere il tuo token, registra un account su TMDB: loggando, ti comparirà direttamente nella pagina della reference, sotto “Authentication”

## Retrofit Client

Ora creiamo l’implementazione di Retrofit che farà da ponte tra l’interfaccia e il client http.

Dentro *network,* crea una nuova classe RetrofitClient

Aggiungi una val *movieBaseAddress* che sarà una Stringa dove è indicato l’URL comune a tutti gli endpoint dell’API. Puoi trovare questo indirizzo controllando la reference di TMDB.

val movieBaseAddress = "https://api.themoviedb.org"

Quando si effettuerà una chiamata, l’endpoint specificato nell'interfaccia verrà concatenato a questo URL

Ora crea un’istanza di Retrofit, utilizzando il Builder:

val movieRemoteDataSource =  
 Retrofit.Builder()  
 .addConverterFactory(GsonConverterFactory.create(Gson()))  
 .baseUrl(movieBaseAddress)  
 .build()  
 .create(MovieNetworkAPI::class.*java*)

Questo builder indica:

* GSON come deserializzatore
* il base url che sarà utilizzato per effettuare le chiamate
* l’interfaccia che definisce gli endpoint
* se non si specifica il client http, retrofit ne costruirà uno standard

### Implementazione

Ora modifica MoviesRepository per includere l’API come sorgente dati, e invoca il builder da RetrofitClient.movieRemoteDataSource

A screen shot of a computer program

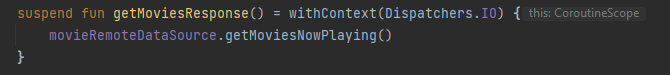
Description automatically generated with low confidence

Adesso aprendo l’app, quando la View richiederà i movies al ViewModel, getMoviesResponse invocherà una richiesta network e i movie verranno scaricati da TMDB.

L’app funziona, ma abbiamo invocato getMoviesResponse da viewModelScope, che è abbastanza “intelligente” da saltare da un thread a un altro quando necessario.

getMoviesResponse non mette limiti da dove può essere invocata, quindi potremmo richiamarla anche da un Main Thread. Questo manderebbe l’app in crash, perché getMoviewNowPlaying effettua una chiamata network, cosa proibita sul main thread dal sistema Android.

Facciamo dunque un’ultima modifica per rendere questa funzione *safe.*



*withContext* è una suspend function utilizzata per definire un Dispatcher, identificandone l’uso. In questo caso, utilizza Dispatchers.IO per una chiamata network.

Diversamente da *launch* che ritorna un job, withContext ritorna il contenuto delle operazioni al suo interno.

## HTTPClient

Quando abbiamo dichiarato getMoviesNowPlaying nell'interfaccia di retrofit, abbiamo dovuto aggiungere un header che ci ha permesso di autorizzarci al server.

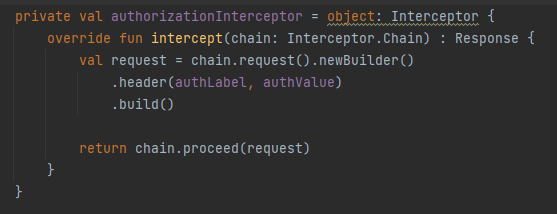
Per evitare di dover specificare ad ogni chiamata andremo adesso a modificare il client retrofit creando un client HTTP personalizzato.

Copia i parametri che avevamo inserito dentro l’annotazione @Header in MovieNetworkApi dentro due val in RetrofitClient. I duepunti sono il carattere di separazione.



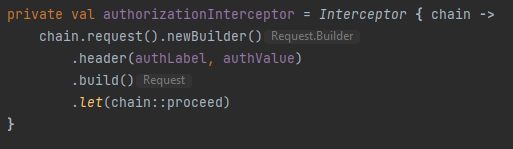
Ora rimuovi completamente l’annotazione @Header dalla chiamata getMoviesNowPlaying.

Torna in RetrofitClient e aggiungi un nuovo **Interceptor,** una classe del pacchetto okhttp3 che permetterà di modificare la chiamata in vari punti dell’esecuzione



Dopo aver ottenuto la request, l’interceptor creerà una nuova richiesta uguale, ma aggiungendo un header di autorizzazione. Infine, manderà la nuova richiesta modificata all’esecuzione della chiamata (chain)

Ottimizzazione di authorization interceptor:



Crea un nuovo HTTPClient con il builder OkHttpClient.Builder()

A screen shot of a computer program

Description automatically generated with medium confidence

* Aggiungi l’interceptor al client http
* Aggiungi il client http a retrofit

Lanciando l’app ora sarà il client http ad occuparsi dell’autorizzazione.

### Log Network calls

Un caso molto comune è quello di analizzare i log delle chiamate network per esaminare potenziali problemi. Aggiungeremo adesso un interceptor che ci permetterà di fare questo lavoro

In build.gradle:app aggiungi la seguente riga e fai un sync:

implementation 'com.squareup.okhttp3:logging-interceptor:4.9.3'

Ora aggiungi un HttpLogginInterceptor al client http:

A screen shot of a computer program

Description automatically generated with low confidence

* Il level identifica cosa deve loggare, con BODY loggherà headers e body.

Ora lanciando l’app e aprendo il Logcat:

A screenshot of a computer

Description automatically generated with medium confidence

## Esercizio

Cerca in TMDB un endpoint che fornisce i dati del cast, e integralo nell’app.

Nella pagina di dettaglio:

* Mostra l’immagine del regista - e fai capire che è il regista
* Mostra le immagini dei 5 attori più famosi, ed il loro ruolo nel film.
* Ordina gli attori secondo l’importanza del ruolo che hanno nel film: per primo il protagonista, poi il coprotagonista, ecc...