SporTrack

Progetto di Programmazione di Dispositivi Mobili

Ferrario Christian - 886230
Buratti Samuele - 879255
Bonfanti Davide - 873293

UNIVERSITÀ DEGLI STUDI DI MILANO BICOCCA

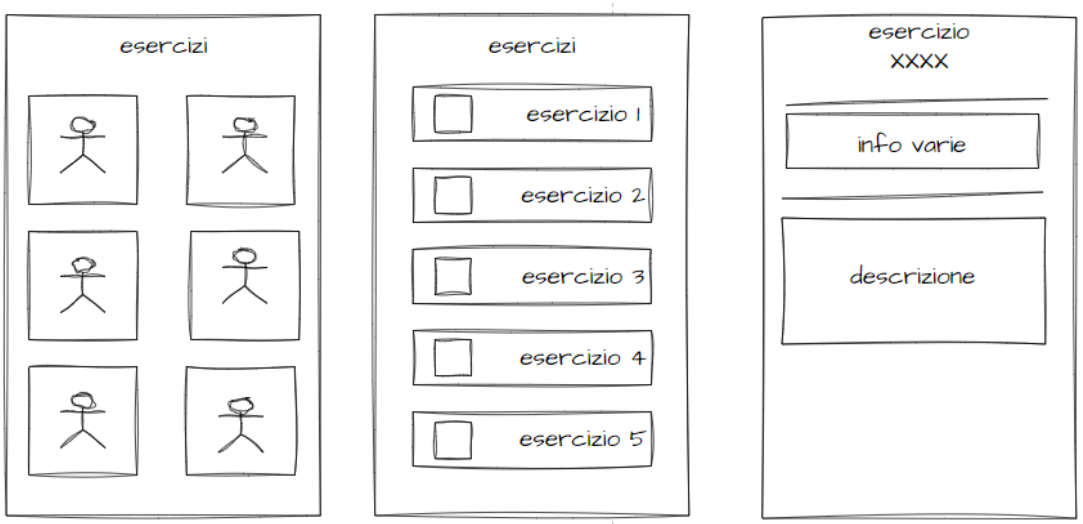
Indice

Introduzione

Il nostro progetto nasce dall'esigenza di fornire agli appassionati di fitness uno strumento semplice ed efficace per monitorare e migliorare le proprie performance in palestra. L'applicazione è stata progettata per consentire agli utenti di creare e personalizzare schede di allenamento, aggiungendo esercizi specifici e tenendo traccia dei progressi fatti nel tempo. Grazie a un'interfaccia intuitiva e funzionalità avanzate, l'app rappresenta un supporto ideale per chiunque desideri ottimizzare il proprio allenamento, raggiungere nuovi obiettivi e mantenere alta la motivazione.

Bozze

Le seguenti bozze rappresentano l'approccio iniziale al design dell'interfaccia utente della nostra applicazione Android e costituiscono la base visiva del progetto e forniranno il punto di partenza per iterazioni successive, con l'obiettivo di perfezionare ulteriormente l'esperienza utente.



Casi d’uso e panoramica

Registrazione, Login e Logout e Utente:

- L'utente può registrarsi creando un nuovo account fornendo le informazioni necessarie (email, password).
- L'utente può effettuare il login utilizzando le credenziali (email e password) per accedere alla propria area personale nell'app.
- L'utente può disconnettersi dal proprio account per tornare alla schermata di login.

Visualizzazione della Home Page:

- Dopo il login, l'utente può accedere alla Home Page, dove vengono visualizzati il calendario dei progressi e dei banner pubblicitari.

Creazione di Schede di Allenamento Personalizzate:

- L'utente può creare una scheda di allenamento con esercizi, specificando il numero di serie e ripetizioni per ciascuno.

Avvio e Monitoraggio dell'Allenamento:

- L'utente può avviare una scheda e monitorare l'allenamento con un timer, utilizzando i pulsanti per saltare (skip) una serie, mettere in pausa o passare alla serie successiva.

Consultazione degli Esercizi per Gruppo Muscolare:

- L'utente può selezionare un gruppo muscolare e visualizzare una lista di esercizi associati, con una descrizione e una pagina dedicata per ogni esercizio.

Modifica ed eliminazione di schede di allenamento:

- L'utente può modificare una scheda esistente, aggiungendo o rimuovendo esercizi.
- L'utente può eliminare una scheda di allenamento che non è più rilevante o aggiornata.

Visualizzazione e Modifica delle Impostazioni:

- L'utente può accedere alla pagina delle impostazioni (settings) dove può visualizzare le informazioni del proprio account o personalizzare le preferenze dell'app come il tema o la lingua dell’applicazione.

Api Ninja e Retrofit

Nell'applicazione, per ottenere gli esercizi mirati all'allenamento di specifici gruppi muscolari, abbiamo integrato l'apposita API del servizio **API Ninja**. Questa API ci permette di eseguire richieste e ci restituisce un array di esercizi, ciascuno contenente dettagli come il nome dell'esercizio, il gruppo muscolare target, il livello di difficoltà e le istruzioni.

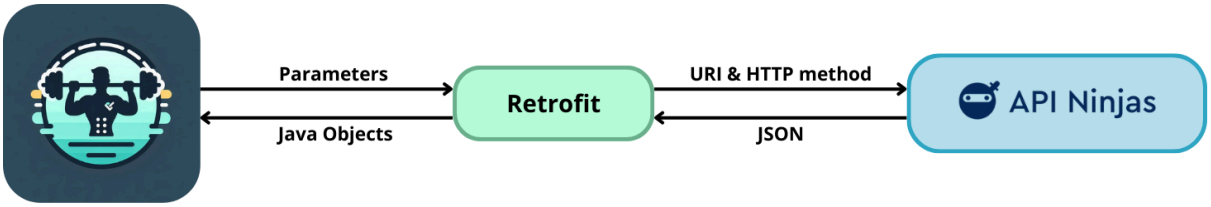
Per gestire le chiamate API e trasformare i dati ricevuti, abbiamo utilizzato **Retrofit**, una libreria Android per le richieste HTTP. Retrofit semplifica notevolmente l'interazione con le API, gestendo automaticamente il parsing del formato JSON in oggetti Java, che poi vengono utilizzati all'interno dell'app.

Funzionamento di Retrofit

1. **Configurazione e Chiamata API:** Utilizziamo Retrofit per configurare il client HTTP, definendo l'endpoint di API Ninja e i metodi per richiedere dati sugli esercizi. Un'interfaccia Java definisce i metodi di richiesta, nel nostro caso una *GET* per ottenere l'elenco di esercizi.

2. **Parsing del JSON:** Una volta effettuata la chiamata API, Retrofit si occupa di convertire automaticamente il JSON restituito in oggetti Java, utilizzando la libreria di parsing **GSON**. I campi del JSON vengono mappati sui campi corrispondenti delle classi Java, che rappresentano il modello dei dati (ad esempio, la classe *Exercise.java*).

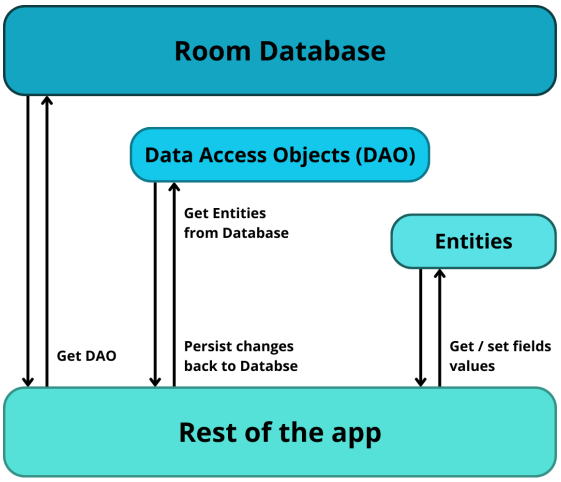
3. **Utilizzo degli Oggetti Java:** Gli oggetti Java risultanti vengono poi utilizzati direttamente nell'applicazione per visualizzare gli esercizi e le relative informazioni. Questo processo di conversione è automatico e permette di lavorare con i dati API come normali oggetti Java, rendendo più semplice la manipolazione e la visualizzazione.



ROOM Database

Per gestire il salvataggio e la persistenza offline dei dati nell'applicazione, abbiamo utilizzato **Room**, una libreria di Android che fornisce un'interfaccia astratta sopra SQLite. Room semplifica l'interazione con il database, permettendo di trasformare gli oggetti Java in righe del database e viceversa, senza dover gestire manualmente le complessità di SQLite.

Funzionamento di Room



Room segue un approccio basato su **Entity**, **DAO** (Data Access Object) e **Database**. Ecco come funziona il processo:

1. **Entity:** Gli oggetti Java che vogliamo salvare nel database vengono annotati con @Entity. Questa annotazione indica che la classe rappresenta una tabella nel database. Room si occupa di trasformare gli oggetti Exercise, WorkoutExercise, Schedule e ExerciseCompleted in righe delle rispettive tabelle.
2. **DAO (Data Access Object):** Le operazioni di lettura e scrittura sul database vengono gestite tramite DAO. In un DAO, definiamo metodi annotati per eseguire query SQL, come @Insert, @Update, @Delete e @Query. Room genera automaticamente il codice SQL necessario per interagire con il database, mappando i risultati delle query direttamente agli oggetti Java.
3. **Database:** Infine, l'applicazione interagisce con un'istanza del database, definita da una classe annotata con @Database, che collega le Entity e i DAO, anche in base ad altre regole, definite in apposite classi per la gestione delle relazioni tra le tabelle. Room si occupa della gestione completa della connessione al database e della sua creazione se necessario.

Quando un oggetto Exercise, WorkoutExercise o Schedule viene salvato nel database, Room lo converte automaticamente in una riga della rispettiva tabella. Analogamente, quando effettuiamo una query (ad esempio, per ottenere tutti gli esercizi per un determinato muscolo), Room esegue la query SQL, trasforma i risultati (che sono righe del database) negli oggetti Java corrispondenti e li restituisce all'applicazione.

Firebase come Backend

Per gestire l'autenticazione degli utenti e la persistenza dei dati in tempo reale, abbiamo scelto di utilizzare **Firebase**, un backend-as-a-service di Google, che offre soluzioni scalabili e facili da integrare.

Autenticazione con Firebase

Firebase Authentication è una soluzione flessibile e sicura per gestire l'accesso degli utenti all'applicazione. Abbiamo integrato Firebase Auth per gestire il login, la registrazione e la gestione delle sessioni utente, semplificando la gestione delle credenziali e delle politiche di sicurezza. Firebase Auth supporta diversi metodi di autenticazione, il che ci consente di offrire varie opzioni di accesso agli utenti.

Metodi di autenticazione utilizzati

Firebase offre diverse opzioni per autenticare gli utenti. Nella nostra applicazione, abbiamo implementato:

- **Autenticazione con email e password**: Gli utenti possono creare un account registrandosi con il proprio indirizzo email e una password sicura. Firebase gestisce la creazione dell'account, la crittografia delle password e il recupero dell'account in caso di password dimenticata, riducendo il rischio di vulnerabilità.
- ~~**Autenticazione tramite provider esterni**: Abbiamo integrato anche l'accesso tramite provider esterni come **Google.** Grazie a Firebase Auth, gli utenti possono accedere all'app utilizzando i loro account social esistenti, senza la necessità di creare un nuovo account, migliorando l'esperienza utente.~~

Gestione delle sessioni e sicurezza

Firebase gestisce in automatico l'autenticazione delle sessioni degli utenti, garantendo che l'accesso rimanga sicuro durante la navigazione nell'app. Le sessioni vengono mantenute anche dopo la chiusura dell'app grazie ai token di autenticazione sicuri, che Firebase genera e aggiorna periodicamente.

- **Gestione dei token**: Quando un utente effettua l'accesso, Firebase fornisce un token che identifica l'utente in modo univoco. Questo token viene utilizzato per accedere in sicurezza alle altre risorse, come il **Firebase Realtime Database** o **Firestore**, senza che l'utente debba reinserire le credenziali.
- **Email verification**: Per migliorare la sicurezza, Firebase offre l'opzione di verificare gli indirizzi email degli utenti, inviando loro una mail di conferma. Abbiamo disabilitato questa funzionalità per facilitare lo sviluppo

dell’applicazione, ma una volta terminata è possibile attivarla semplicemente..
- **Reset password**: Firebase gestisce anche il recupero delle credenziali, permettendo agli utenti di resettare facilmente la propria password tramite email in caso di smarrimento.

Una volta autenticato, l'utente può interagire con il **Firebase Realtime Database**. I privilegi di accesso possono essere personalizzati tramite regole di sicurezza basate sull'utente autenticato, consentendo agli utenti di modificare solo i propri dati, senza avere accesso all’intero database.

Firebase Realtime Database

Firebase Realtime Database è una soluzione di database NoSQL che consente di archiviare e sincronizzare dati tra client in tempo reale. Questo tipo di database è particolarmente utile per applicazioni che richiedono aggiornamenti immediati e continui, come la nostra app di allenamento, dove i dati possono essere condivisi e aggiornati tra diversi dispositivi.

Architettura del Database

Firebase Realtime Database memorizza i dati in formato JSON in un'unica struttura ad albero. Ogni dato è rappresentato come un nodo, e i dati possono essere letti e scritti da qualsiasi punto di quest’albero. Questa architettura flessibile ci consente di salvare varie tipologie di informazioni, come esercizi, schede, e profili utente.

Sincronizzazione in Tempo Reale

Una delle caratteristiche principali del Realtime Database è la sincronizzazione automatica e immediata dei dati tra tutti i client connessi. Quando un utente modifica un dato (come un nuovo esercizio aggiunto o un allenamento completato), queste modifiche vengono istantaneamente riflesse sui dispositivi di tutti gli utenti interessati. Ciò è possibile perché Firebase mantiene una connessione persistente tra l'app e il server, che ascolta e distribuisce gli aggiornamenti in tempo reale.

Questo meccanismo garantisce un'esperienza utente fluida e reattiva, dove i dati sono sempre aggiornati senza la necessità di ricaricare o fare polling continuo.

Sicurezza e Regole di Accesso

Per proteggere i dati e garantire un accesso controllato, Firebase offre un sistema di regole di sicurezza basate su espressioni che si applicano direttamente alla struttura JSON. Queste regole permettono di definire chi può leggere e scrivere in specifiche parti del database, in base a criteri come l'autenticazione dell'utente.

Ad esempio, possiamo configurare il database affinché solo un utente autenticato possa accedere ai propri dati personali, mentre altri utenti non possono visualizzare o modificare quelle informazioni.

Accesso Offline

Un altro vantaggio del Realtime Database è la **persistenza offline**. Anche se un dispositivo perde la connessione a Internet, Firebase continua a funzionare utilizzando una copia locale dei dati (cache). Gli utenti possono continuare a leggere e scrivere dati mentre sono offline, permettendo all'app di continuare a funzionare anche senza una connessione Internet, e rendendo i dati immediatamente disponibili localmente. Una volta ristabilita la connessione, Firebase sincronizza automaticamente tutte le modifiche effettuate, aggiornando il database remoto e gli altri client.

Questa funzionalità garantisce che l'app rimanga operativa in ogni condizione di rete, migliorando l'esperienza utente.

Librerie e Plugin

Oltre a Retrofit (con GSON) , Firebase, Material e Room nel nostro progetto abbiamo utilizzato le seguenti librerie:

ViewBinding:
ViewBinding è una libreria che consente di accedere in modo sicuro alle viste nel layout XML senza dover utilizzare findViewById(). Genera automaticamente classi di binding per ciascun file di layout, riducendo il rischio di errori di runtime dovuti a cast errati o viste mancanti.

DataBinding:
DataBinding permette di collegare direttamente il layout XML ai dati del ViewModel, migliorando l'efficienza nel gestire gli aggiornamenti dell'interfaccia utente. Utilizzando espressioni nel layout, consente una più stretta integrazione tra logica e UI, riducendo il codice boilerplate.

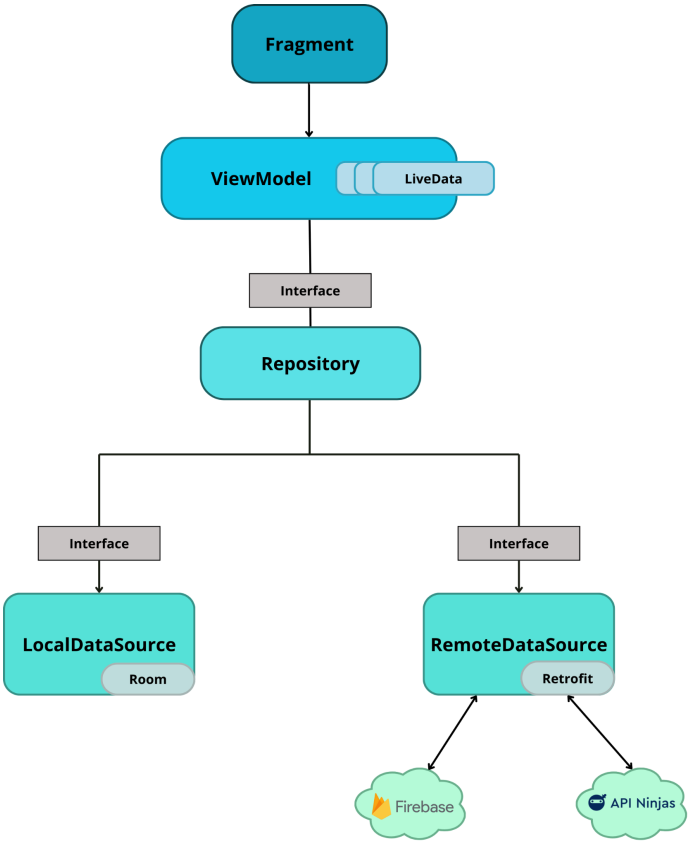
SafeArgs:
SafeArgs è un plugin della libreria Navigation per passare dati tra i fragment o le attività. Genera classi fortemente tipizzate per garantire che i dati vengano trasferiti senza errori e in modo sicuro, evitando problemi comuni con i bundle manuali.

RecyclerViewSwipeDecorator:
RecyclerViewSwipeDecorator è una classe di utility che semplifica l'aggiunta di sfondi, icone e etichette agli elementi di una RecyclerView durante lo swipe a sinistra o a destra. La libreria permette anche di personalizzare le icone e gli sfondi per le direzioni di swipe e di aggiungere etichette.

TimeIt:
TimeIt è una libreria che semplifica la gestione di cronometri e timer senza la necessità di gestire thread separati. Consente di creare e avviare cronometri e timer con poche righe di codice, supporta le funzionalità di pausa, ripresa e introduce dei listeners per i timer.

Architettura del sistema

L'architettura **MVVM** (Model-View-ViewModel) è un pattern ampiamente utilizzato nello sviluppo Android per separare la logica di business e la gestione dei dati dall'interfaccia utente, migliorando la modularità, la testabilità e la manutenibilità dell'applicazione. Nella nostra applicazione, i componenti chiave sono **Fragment**, **ViewModel**, **Repository**, **LocalDataSource**, e **RemoteDataSource**.



1. **Fragment**: Rappresentano le diverse schermate dell'interfaccia. Sono la parte visibile dell'applicazione, dove l'utente interagisce direttamente con la UI.
   I Fragment non gestiscono direttamente la logica o i dati, ma delegano queste responsabilità al ViewModel.
   Il Fragment si limita a ricevere gli input dall'utente e a osservare i dati presenti nel ViewModel. Questo avviene tramite l'osservazione dei LiveData, che aggiorna automaticamente la UI quando i dati cambiano.
2. **ViewModel**: Agisce come intermediario tra il Fragment e la Repository.
   Si occupa di mantenere i dati che la UI deve renderizzare e di gestire la logica legata all'interfaccia. Espone dati osservabili attraverso i LiveData.
   In pratica il ViewModel chiama le funzioni della repository, passandogli le *callback* che la repository dovrà chiamare per notificare lo stato della richiesta, in base al quale modificherà i dati contenuti nei LiveData, cambiando di conseguenza lo stato della UI.
   Il ViewModel è responsabile di mantenere uno stato consistente dei dati anche in caso di cambiamenti di configurazione.
3. **Repository**: È una componente chiave nell'architettura MVVM, che funge da ponte tra il ViewModel e le fonti di dati (locali e remote). Questo pattern centralizza la logica di accesso ai dati e consente al ViewModel di ignorare i dettagli su come e dove vengono recuperati. La Repository è responsabile di orchestrare la logica su quale fonte di dati utilizzare (locale o remota) in base alla disponibilità e al contesto. Ad esempio, nel nostro caso, la Repository

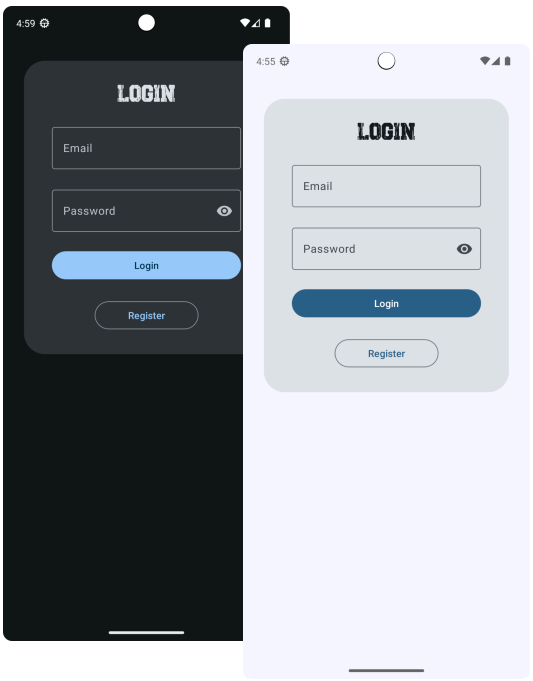
prova ad ottenere i dati dal **RemoteDataSource** (API o Firebase Database) e salvarli nel **LocalDataSource** (Room) per futuri utilizzi offline, e nel caso in cui la **RemoteDataSource** non fosse in grado di recuperare i dati la repository viene notificata tramite una *callback (es. onData Available()),* e li estrae dal database locale.

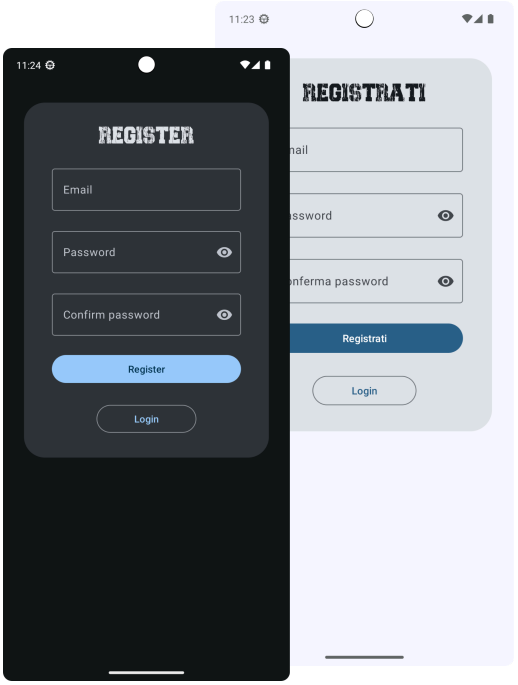
```
@Override 1 usage  FerrarioChristian
public void getWorkoutExercises(Long scheduleId, GetWorkoutExerciseCallback callback) {
    workoutExerciseRemoteDataSource.getWorkoutExercises(scheduleId,
            new GetWorkoutExerciseCallback() {  FerrarioChristian

                @Override  4 usages  FerrarioChristian
                public void onWorkoutExercisesLoaded(List<WorkoutExercise> workoutExerciseList) {
                    callback.onWorkoutExercisesLoaded(workoutExerciseList);
                    if (workoutExerciseList.isEmpty()) {
                        workoutExerciseLocalDataSource.deleteWorkoutExercises(scheduleId);
                    } else {
                        workoutExerciseLocalDataSource.updateWorkoutExercises(
                                workoutExerciseList);
                    }
                }

                @Override  FerrarioChristian
                public void onDataNotAvailable() {
                    workoutExerciseLocalDataSource.getWorkoutExercises(scheduleId, callback);
                }

                @Override  FerrarioChristian
                public void onFailure(String errorMessage) {
                    Log.e(TAG, errorMessage);

                }
            });
}
```

4. **LocalDataSource**: Si occupa della gestione dei dati persistenti salvati localmente sul dispositivo tramite il database Room.
5. **RemoteDataSource**: Si interfaccia con le fonti di dati esterne, come Firebase o servizi API come API Ninjas, utilizzando Retrofit per effettuare chiamate di rete. Questo consente all'applicazione di sincronizzare i dati e ottenere informazioni in tempo reale da fonti online. Il RemoteDataSource è anche responsabile della gestione degli errori di rete. In caso di fallimento della richiesta, il Repository utilizza una fonte alternativa (ad esempio, il database locale).

```
public class WorkoutExercisesRepository implements IWorkoutExercisesRepository {  2 usages  FerrarioChristian +1

    private static final String TAG = ExerciseRepository.class.getSimpleName();  1 usage

    private final IWorkoutExerciseDataSource.Local workoutExerciseLocalDataSource;  8 usages
    private final IWorkoutExerciseDataSource.Remote workoutExerciseRemoteDataSource;  4 usages

    public WorkoutExercisesRepository(IWorkoutExerciseDataSource.Local workoutExerciseLocalDataSource  1 usage
            , IWorkoutExerciseDataSource.Remote workoutExerciseRemoteDataSource) {
        this.workoutExerciseLocalDataSource = workoutExerciseLocalDataSource;
        this.workoutExerciseRemoteDataSource = workoutExerciseRemoteDataSource;
    }
```

Login Activity





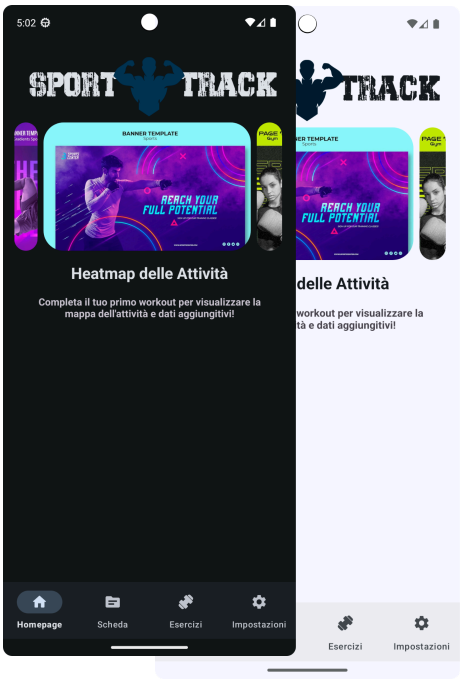
La pagina di login è una activity che permette all'utente di eseguire il login se si è a disposizione di un account o, in caso contrario, permette di effettuare la registrazione.

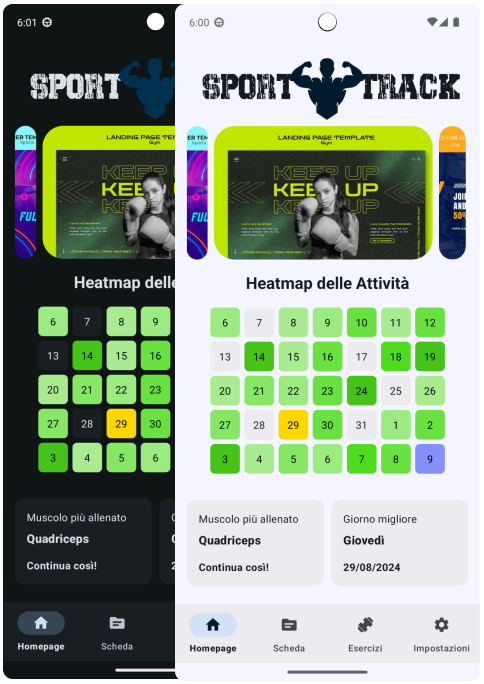
Una volta terminata la fase di autenticazione, l'activity viene distrutta e viene sostituita dalla MainActivityWithBottomNav.

Main Activity With Bottom Nav

Da questo momento l'applicazione si baserà su una struttura "One Activity Multiple Fragments".
Questa main activity contiene tutti i restanti fragment.

Home Page Fragment



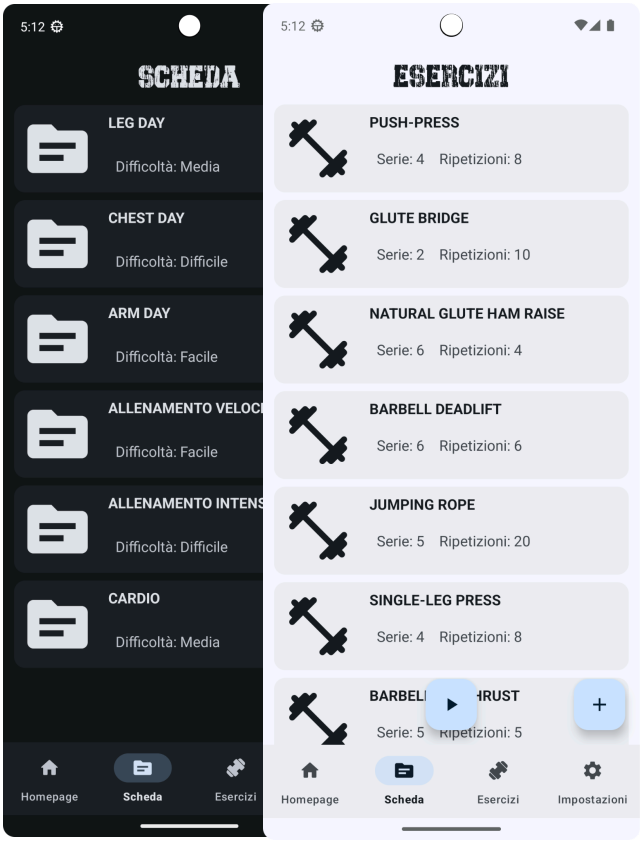


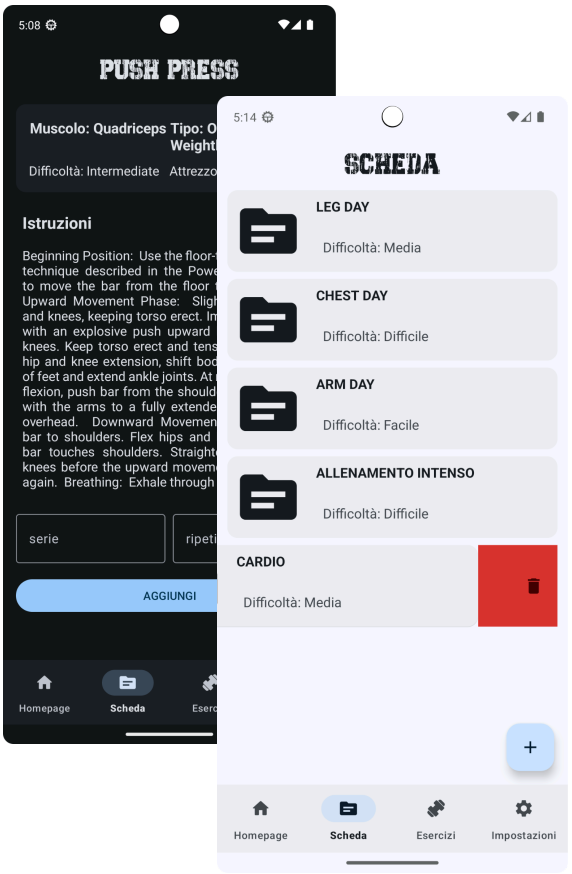
Nella Homepage, sotto al logo, troviamo un banner con immagini motivazionali e pubblicitarie. Al centro della pagina, è presente una sezione riepilogativa della situazione. Al primo accesso, l'utente è invitato a iniziare il primo allenamento per poter visualizzare questi dati. Una volta completato un esercizio, esso verrà visualizzato nella **HeatMap**, una mappa composta da celle che corrispondono ai giorni del mese. La tonalità di verde più scura indica un impegno maggiore dell'utente; il colore oro rappresenta il giorno con il maggior numero di esercizi svolti, mentre il colore azzurro indica la data odierna.

La HeatMap è dinamica: l'ultima cella corrisponde sempre alla data odierna e vengono mostrati i 35 giorni precedenti. Cliccando su ciascuna cella, un toast mostrerà il numero di esercizi completati in quella specifica data.

Nella parte inferiore dello schermo, le card mostrano il muscolo che l'utente ha allenato maggiormente e il giorno con le migliori prestazioni.

Scheda Fragment



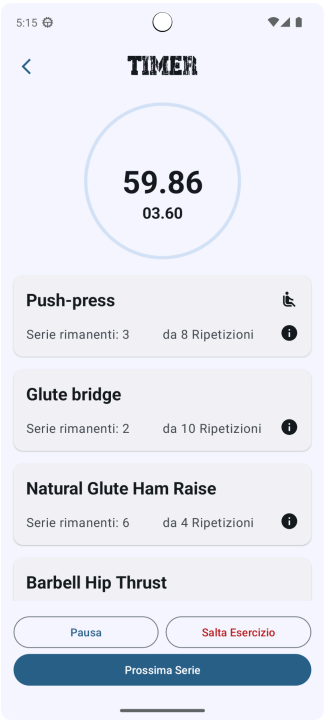


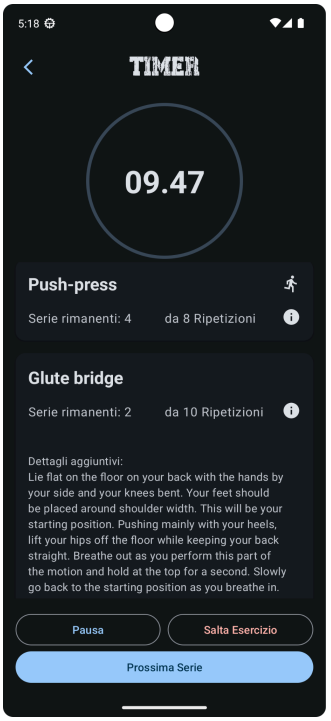
Il fragment "Scheda" permette agli utenti di creare e gestire piani di allenamento personalizzati. Al primo accesso a questa sezione, l'utente può creare una nuova scheda, definendo il livello di difficoltà e aggiungendo successivamente gli esercizi desiderati. Ogni scheda può includere uno o più esercizi, ciascuno con un numero specifico di serie e ripetizioni.

Per eliminare una scheda, è sufficiente trascinarla verso sinistra fino a far comparire un'icona a forma di cestino, quindi rilasciarla.

Se la scheda contiene uno o più esercizi, nella parte inferiore dello schermo apparirà un pulsante Play, che consente all'utente di avviare il timer e iniziare l'allenamento.

Timer Fragment





All'avvio del timer, l'utente viene portato a una nuova schermata che include un cronometro nella parte superiore, l'elenco degli esercizi contenuti nella scheda avviata e i tasti per il controllo dell'allenamento. Un'icona indica l'esercizio corrente e, se l'utente ha bisogno di rivedere come eseguire un esercizio, può cliccare sull'icona per espandere l'elemento e visualizzare le istruzioni.

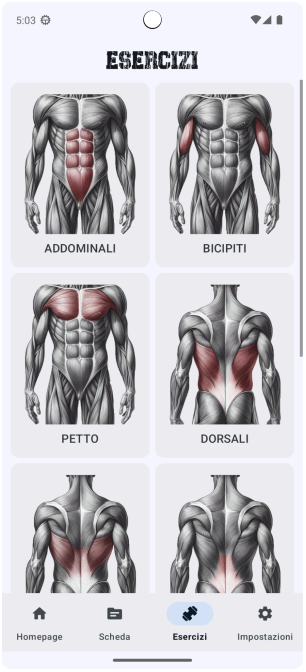
Sono presenti tre pulsanti principali: il tasto play/pause gestisce il cronometro, permettendo di avviarlo o fermarlo; il tasto "Salta Esercizio" consente di passare all'esercizio successivo nel caso in cui l'utente non voglia eseguirlo; il tasto "Prossima Serie" segnala al timer il completamento della serie corrente, riducendo il numero di serie rimanenti e avviando un piccolo stopwatch sotto il timer, che rappresenta il tempo di riposo consigliato per un corretto allenamento (indicato anche dall'icona di un uomo seduto).

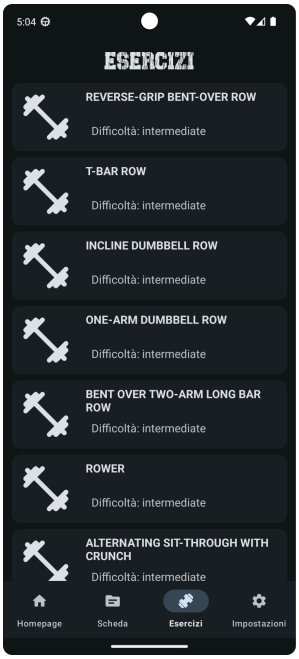
Una volta completate tutte le serie, l'esercizio corrispondente scompare dalla lista e viene salvato come completato, aggiornando le statistiche nella homepage indipendentemente dal fatto che l'intera scheda sia stata completata.

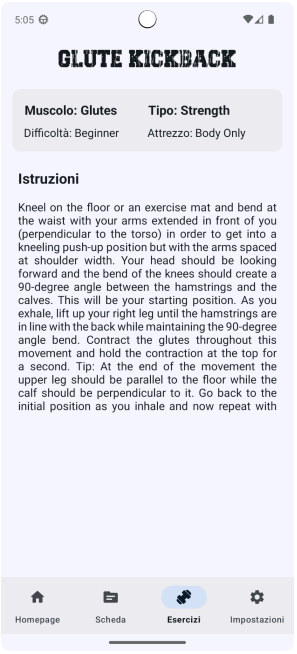
Vari controlli sono stati implementati per garantire che l'utente non compia azioni errate, come passare alla serie successiva se il timer è fermo, o viceversa.

Il timer è stato progettato per funzionare anche con l'app in background.

Esercizi Fragment





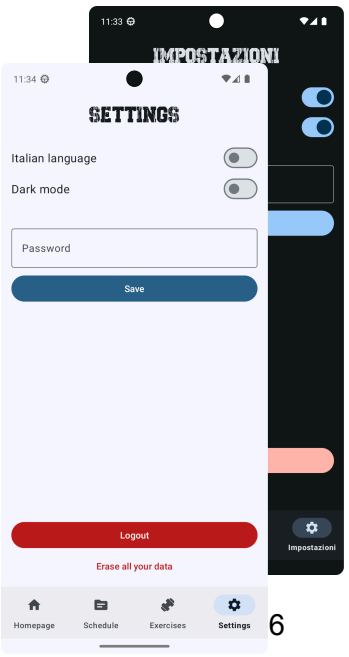


Il fragment "Esercizi" consente agli utenti di selezionare un gruppo muscolare e visualizzare una lista di esercizi correlati. Cliccando su un esercizio, è possibile accedere a una pagina dedicata che offre un breve sommario, con dettagli sull'esecuzione, il livello di difficoltà e gli eventuali strumenti necessari.

Settings Fragment

In questo fragment, l'utente ha la possibilità di cambiare la lingua dell'intera applicazione da italiano a inglese, e viceversa. È presente anche uno switch per attivare la modalità notte. Sotto a questi, è disponibile un campo per modificare la password.

Nella parte inferiore dello schermo, sono presenti due pulsanti: uno per effettuare il logout e un altro per eliminare tutte le schede e gli esercizi svolti, consentendo un nuovo inizio.

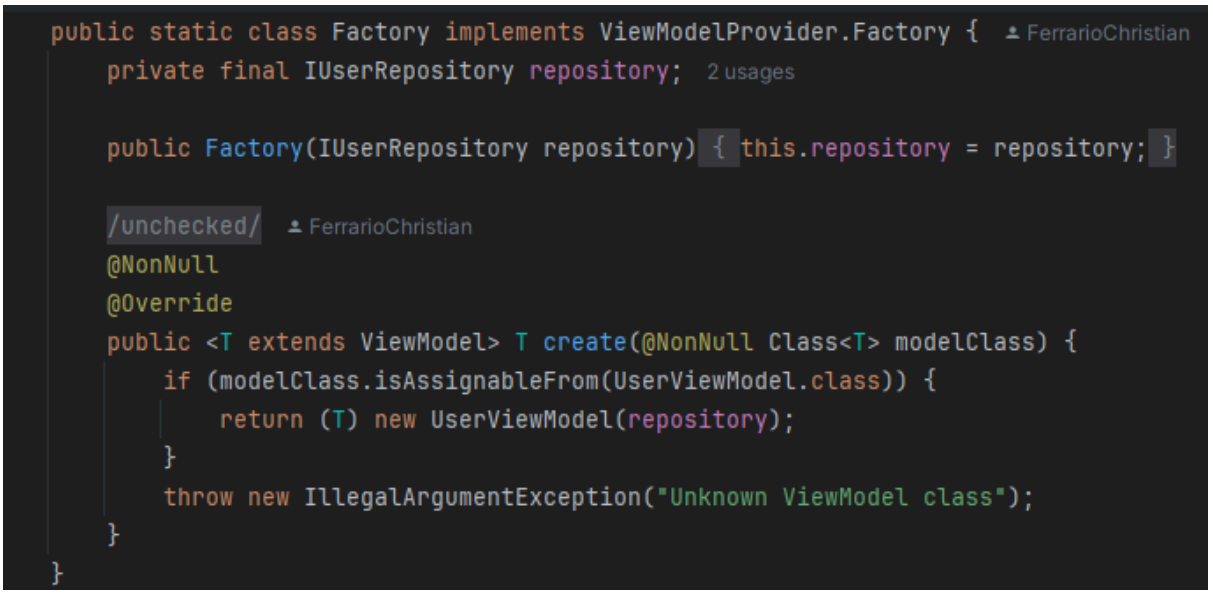


Pattern

Singleton

Il pattern Singleton è una soluzione utile per garantire che una classe abbia una sola istanza e fornisca un punto di accesso globale a quella istanza. Un ServiceLocator fornisce un modo per accedere a vari servizi e oggetti attraverso un'unica interfaccia. Utilizzandolo con il pattern Singleton, possiamo gestire centralmente e accedere ai servizi e alle dipendenze all'interno dell'applicazione migliorando la coerenza e la manutenibilità del codice.

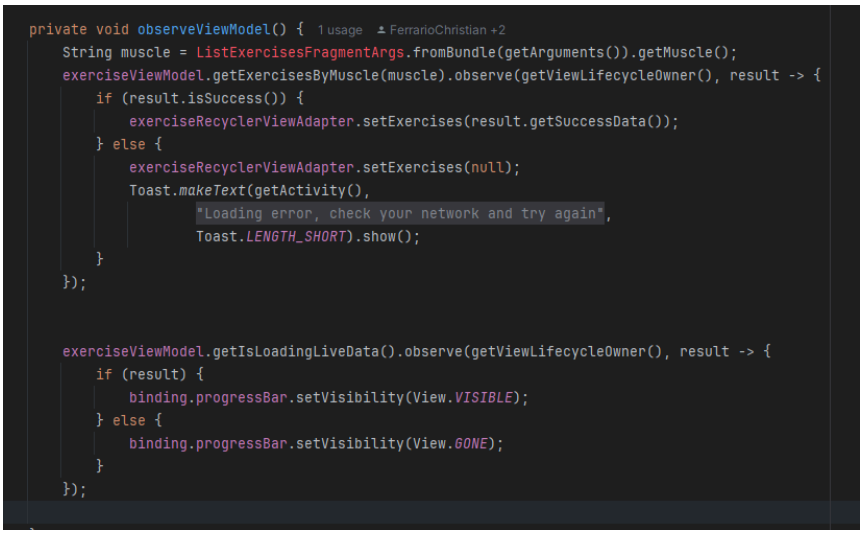
Factory



Il **Factory Pattern** è un pattern di design creazionale che fornisce un'interfaccia per la creazione di oggetti in una superclasse, ma permette alle sottoclassi di modificare

il tipo di oggetti che verranno creati. Questo pattern è utile quando non è noto a priori quale classe deve essere istanziata o quando le classi da creare devono essere modificate dinamicamente. In questo caso il pattern viene utilizzato per gestire la creazione di istanze di repository per l'accesso ai dati degli utenti.

observer



```
private void observeViewModel() {  1 usage   FerrarioChristian +2
    String muscle = ListExercisesFragmentArgs.fromBundle(getArguments()).getMuscle();
    exerciseViewModel.getExercisesByMuscle(muscle).observe(getViewLifecycleOwner(), result -> {
        if (result.isSuccess()) {
            exerciseRecyclerViewAdapter.setExercises(result.getSuccessData());
        } else {
            exerciseRecyclerViewAdapter.setExercises(null);
            Toast.makeText(getActivity(),
                    "Loading error, check your network and try again",
                    Toast.LENGTH_SHORT).show();
        }
    });


    exerciseViewModel.getIsLoadingLiveData().observe(getViewLifecycleOwner(), result -> {
        if (result) {
            binding.progressBar.setVisibility(View.VISIBLE);
        } else {
            binding.progressBar.setVisibility(View.GONE);
        }
    });
```

Il pattern Observer è un modello di design che definisce una dipendenza uno-a-molti tra oggetti in modo che quando un oggetto cambia stato, tutti i suoi dipendenti (osservatori) vengano notificati e aggiornati automaticamente. È molto utile quando hai bisogno di aggiornare diverse parti della tua applicazione in risposta ai cambiamenti di stato di un oggetto. In questo caso il ViewModel contiene un'istanza di LiveData che rappresenta i dati da osservare.
Gli osservatori, come il caso in esempio, si registrano per ricevere notifiche sui cambiamenti dei dati contenuti in LiveData.
Quando i dati cambiano, LiveData notifica automaticamente tutti gli osservatori registrati, eseguendo così il codice contenuto in essi.

Altre classi utili

Classe App

La classe App estende Application e svolge un ruolo cruciale nella gestione globale dello stato dell'applicazione e delle risorse condivise.

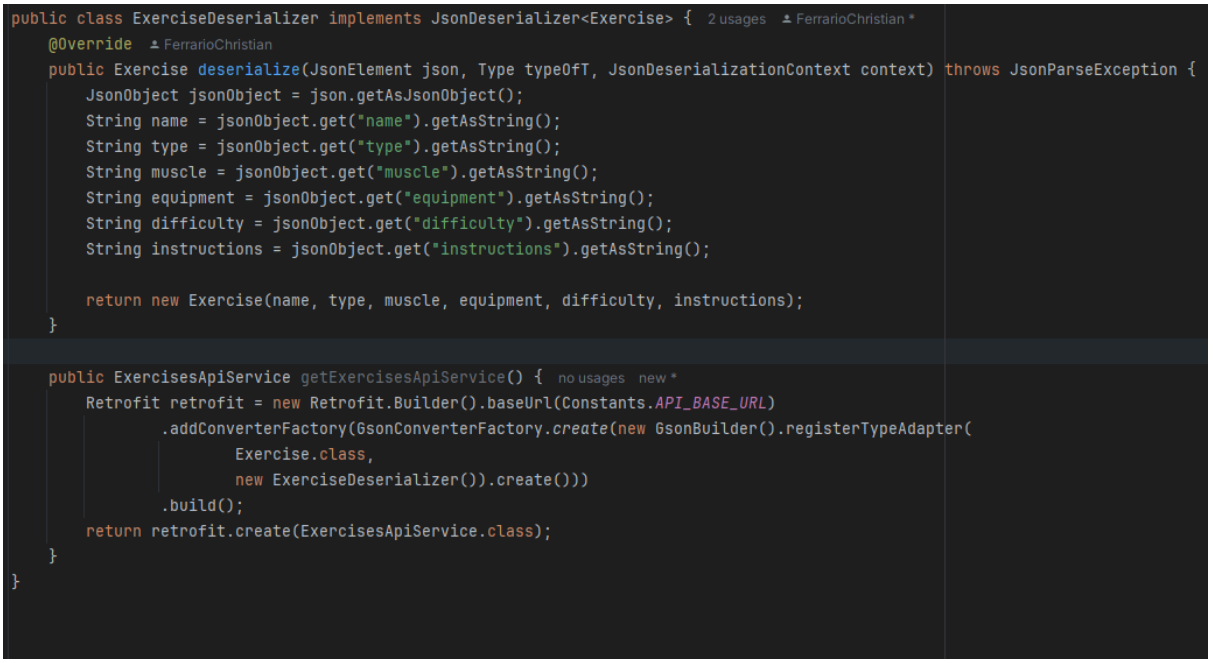
- **Gestione dell'istanza globale**: fornisce un punto di accesso globale all'istanza dell'applicazione tramite il metodo getInstance(). Questo è utile per accedere a componenti e risorse condivise in qualsiasi parte dell'app, senza dover passare esplicitamente l'istanza dell'app.
- **Accesso alle risorse**: La classe gestisce una variabile statica res di tipo Resources, che consente di accedere facilmente alle risorse dell'app (come stringhe, colori, dimensioni, ecc.) da qualsiasi punto dell'applicazione. Il metodo setRes(Resources res) è utilizzato per inizializzare questa variabile all'avvio dell'app e al cambio della lingua, per aggiornare le risorse Strings. Gli altri componenti dell'app, come Activity, Fragment, o ViewModel, possono accedere alle risorse condivise attraverso App.getRes(). Questo semplifica l'accesso a risorse comuni senza dover passare oggetti Resources esplicitamente.
- Configurazione di Firebase: All'interno del metodo onCreate(), la classe App abilita la persistenza offline per **Firebase Realtime Database** chiamando FirebaseDatabase.getInstance().setPersistenceEnabled(true). Questo assicura che i dati del database siano memorizzati localmente e disponibili anche quando il dispositivo è offline e vengano aggiornati correttamente in remoto una volta che l'applicazione torna online.

```
public class App extends Application {  G3N3T1C +1
    private static App INSTANCE;  2 usages
    private static Resources res;  2 usages

    public static App getInstance() { return INSTANCE; }

    public static Resources getRes() { return res; }

    @Override  FerrarioChristian
    public void onCreate() {
        super.onCreate();
        INSTANCE = this;
        res = getResources();
        FirebaseDatabase.getInstance().setPersistenceEnabled(true);
    }
}
```

Classe ExerciseDeserializer

La classe ExerciseDeserializer implementa l'interfaccia JsonDeserializer di Gson e ha lo scopo di personalizzare il processo di deserializzazione degli oggetti JSON in oggetti Java. La classe ExerciseDeserializer è progettata per convertire un oggetto

JSON, che rappresenta un esercizio, in un'istanza della classe Exercise, utilizzando un costruttore personalizzato e non sfruttando la corrispondenza uno a uno tra gli attributi della classe e le chiavi dell'oggetto JSON.
Nel nostro caso questo deserializer è passato alla Factory del servizio di Retrofit, in



```
public class ExerciseDeserializer implements JsonDeserializer<Exercise> {  2 usages  FerrarioChristian *
    @Override  FerrarioChristian
    public Exercise deserialize(JsonElement json, Type typeOfT, JsonDeserializationContext context) throws JsonParseException {
        JsonObject jsonObject = json.getAsJsonObject();
        String name = jsonObject.get("name").getAsString();
        String type = jsonObject.get("type").getAsString();
        String muscle = jsonObject.get("muscle").getAsString();
        String equipment = jsonObject.get("equipment").getAsString();
        String difficulty = jsonObject.get("difficulty").getAsString();
        String instructions = jsonObject.get("instructions").getAsString();

        return new Exercise(name, type, muscle, equipment, difficulty, instructions);
    }

    public ExercisesApiService getExercisesApiService() {  no usages  new *
        Retrofit retrofit = new Retrofit.Builder().baseUrl(Constants.API_BASE_URL)
                .addConverterFactory(GsonConverterFactory.create(new GsonBuilder().registerTypeAdapter(
                        Exercise.class,
                        new ExerciseDeserializer()).create()))
                .build();
        return retrofit.create(ExercisesApiService.class);
    }
}
```

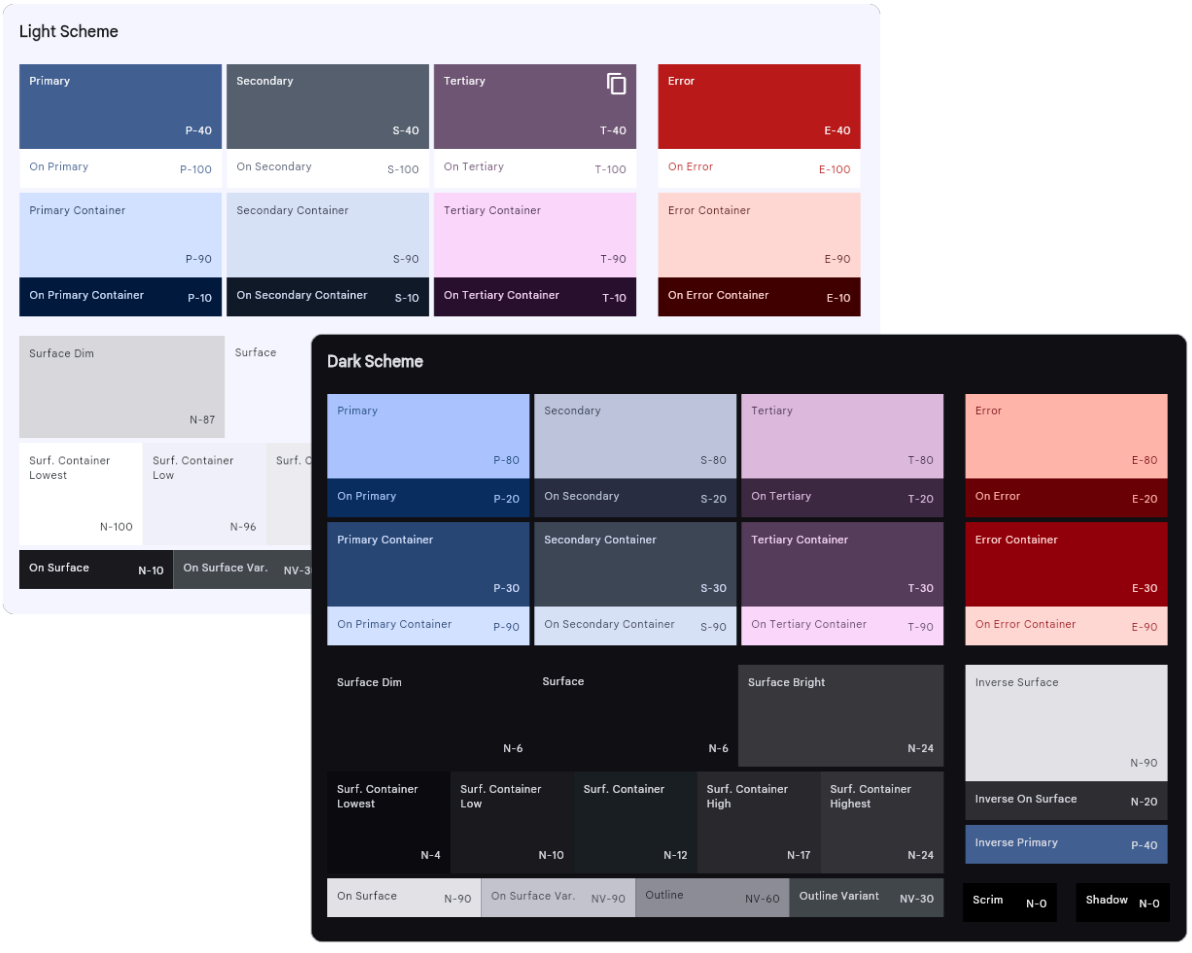
modo da convertire la risposta dell'API in oggetti Java, che verranno poi salvati sul database locale.

Design UI/UX

Come design system per la nostra applicazione abbiamo utilizzato il sistema di regole e la libreria dei componenti di Material Design 3.
Material Design 3 è l'ultima evoluzione del sistema di design creato da Google, progettato per fornire un set di regole e componenti per la creazione di interfacce utente intuitive, accessibili e visivamente coerenti. Questo sistema si basa su principi di semplicità, coerenza e flessibilità, offrendo strumenti per personalizzare il proprio design.

La libreria dei componenti, che include elementi come pulsanti, card, menu e sistemi di navigazione, è progettata per essere altamente modulare, scalabile e facile da implementare, rendendo lo sviluppo più efficiente e mantenendo la coerenza visiva tra le diverse piattaforme e dispositivi.

Material Design offre inoltre uno strumento chiamato Material Theme Builder, che permette di creare una palette di colori coerenti tra loro, ognuno con una specifica funzione all'interno del design. Nel nostro caso, abbiamo creato la palette a partire dai colori utilizzati nel logo, adattandoli sia al tema chiaro che al tema scuro.



Utilizzo dei colori

Material Design 3 introduce un sistema di colori avanzato e flessibile che permette di creare interfacce accessibili e coerenti. La gestione dei colori è suddivisa in categorie, ciascuna con un ruolo specifico nell'interfaccia utente. Le principali categorie di colori utilizzate nell'applicazione sono:

1. **Surface**: Questo colore rappresenta le superfici principali dell'interfaccia, come card, menu e fogli di dialogo. È generalmente un colore neutro, utilizzato per i contenuti di primo piano.
2. **OnSurface**: Questo colore viene applicato agli elementi che appaiono sopra le superfici (testo, icone, ecc.). Viene scelto per garantire un contrasto adeguato e migliorare la leggibilità.
3. **Background**: È il colore di sfondo principale dell'applicazione, spesso utilizzato per aree più estese come la schermata principale. Di solito, è simile al colore di superficie, ma si estende a livello dell'intera app.
4. **OnBackground**: Utilizzato per gli elementi di contenuto (testo, icone, ecc.) sopra il colore di sfondo. Come onSurface, il suo ruolo è garantire un contrasto sufficiente e leggibilità sugli sfondi.
5. **Primary**: È il colore principale che rappresenta il brand o l'elemento visivo distintivo dell'app. Viene usato per pulsanti, indicatori e altri elementi interattivi.
6. **OnPrimary**: Colore applicato sugli elementi posti sopra aree colorate con il colore Primary, come testo o icone sui pulsanti primari. Deve essere chiaro e ben leggibile.
7. **Error**: Questo colore viene applicato a componenti o testo che indicano errori o stati critici, come messaggi di errore, bordi di campi non validi o pulsanti che eseguono azioni potenzialmente dannose. Di solito, è un rosso acceso che cattura immediatamente l'attenzione.
8. **OnError**: È il colore utilizzato per gli elementi sovrapposti a un background di colore `Error`. Può essere bianco o chiaro per garantire un contrasto ottimale, migliorando la leggibilità del testo o delle icone su uno sfondo rosso.

Colori per le card

Le card rappresentano una superficie sopra la quale sono mostrati contenuti come testo o immagini. I colori usati per le card sono:

- **SurfaceContainer**: Il colore di base della card è il colore `surfaceContainer`, che è un colore neutro. Può variare in base al tema scelto (chiaro o scuro), ma è solitamente chiaro in un tema light e scuro in un tema dark.

- **OnSurface**: Gli elementi che si trovano sopra la card (testo, icone, ecc.) utilizzano il colore `onSurface`. Questo colore è pensato per avere un contrasto sufficiente con la superficie, garantendo una leggibilità ottimale.

Colori per i Button

I bottoni sono elementi interattivi che richiedono un focus visivo maggiore. I colori principali da usare per i bottoni sono:

- **Primary**: I bottoni principali (detti anche "filled button") utilizzano il colore `primary`. Questo colore rappresenta l'elemento distintivo e caratterizzante dell'interfaccia, spesso collegato all'identità visiva del brand o dell'app.
- **OnPrimary**: Il testo e le icone all'interno dei button primari devono utilizzare il colore `onPrimary`, che è generalmente bianco o chiaro, per garantire leggibilità sullo sfondo colorato.

Per i button secondari o meno prominenti (ad esempio i "outlined button" o "text button"):

- **Surface**: Il background dei button secondari può essere il colore `surface` o trasparente.
- **Primary**: Il colore del testo e delle icone è solitamente `primary` per mantenere una coerenza visiva con gli altri elementi interattivi dell'app.
- **OnSurface**: Nel caso di button che utilizzano `surface` come background, il colore `onSurface` viene utilizzato per testo e icone.

Nell'ambito dei **bottoni**, i colori **Error** e **OnError** vengono utilizzati per indicare azioni critiche o di avviso, come la cancellazione o l'annullamento di un'operazione importante.

- **Error**: Questo colore viene applicato al background dei bottoni che eseguono azioni critiche, ad esempio un pulsante "Elimina" o "Reset". Il rosso acceso del colore `Error` segnala chiaramente all'utente che l'azione ha conseguenze rilevanti o potenzialmente irreversibili.
- **OnError**: Il colore `OnError` viene utilizzato per il testo o le icone all'interno di questi bottoni. Solitamente è bianco o chiaro, per garantire un elevato contrasto e una buona leggibilità sullo sfondo rosso del bottone `Error`.

Questa combinazione di colori aiuta a rendere immediatamente riconoscibili le azioni critiche, migliorando la chiarezza e prevenendo errori accidentali.

Utilizzo degli spazi

In Material Design 3, la gestione degli spazi è fondamentale per garantire una disposizione ordinata e intuitiva degli elementi. Alcuni principi chiave sono:

- **Padding**: Definisce lo spazio interno tra il contenuto e i bordi di un componente, come pulsanti o card, per creare un layout più arioso e leggibile.
- **Distanza tra gli elementi**: La spaziatura tra componenti aiuta a mantenere una gerarchia visiva chiara, migliorando la navigazione e la comprensione dell'interfaccia. Le distanze variano a seconda dell'importanza degli elementi e del layout complessivo.
- **Dimensione minima di tocco**: Per garantire l'accessibilità, la dimensione minima di un'area di tocco è di 48x48 dp. Questo assicura che gli elementi interattivi, come pulsanti, siano facilmente cliccabili anche su schermi più piccoli.

Queste regole garantiscono un'interfaccia ben strutturata, usabile e accessibile.

Sviluppi Futuri

A Breve Termine

Aggiunta di esercizi personalizzati:

Al momento, l'app offre solo esercizi forniti dalla Ninja API, limitando così l'esperienza degli utenti esperti che desiderano inserire esercizi personalizzati. Implementare la possibilità di aggiungere esercizi propri potrebbe arricchire l'esperienza utente, offrendo una maggiore varietà e personalizzazione.

Possibilità di riordinare gli esercizi durante la sessione di allenamento

Con il crescente aumento di frequentatori in palestra, il sovraffollamento è diventato un problema comune. Per evitare che l'utente debba attendere per utilizzare i macchinari nell'ordine prestabilito, sarebbe utile poter riordinare la lista degli esercizi, mettendo al primo posto quelli che richiedono macchinari attualmente liberi.

Ulteriori grafici e statistiche

Analizzare i dati in modo approfondito è essenziale per comprendere i ritmi da seguire o mantenere al fine di ottimizzare il proprio percorso di allenamento. L'integrazione di ulteriori grafici, come un grafico a torta che mostra la distribuzione dei muscoli allenati o le schede più utilizzate, potrebbe rappresentare la chiave per migliorare le routine degli utenti.

Traduzione automatica in più lingue

L'API scelta fornisce i dati esclusivamente in lingua inglese. Sebbene la traduzione dei nomi dei muscoli possa essere gestita facilmente, le lunghe descrizioni degli esercizi possono rappresentare una difficoltà. Implementare un sistema di traduzione in tempo reale all'interno dell'app sarebbe molto utile per gli utenti che non sono a loro agio con lingue diverse dalla propria.

A Lungo Termine

Video Tutorial per Esercizi: Aggiunta di video dimostrativi per ogni esercizio, accessibili dagli utenti tramite un'interfaccia dedicata. Ogni esercizio avrà una pagina con il video associato e una descrizione dettagliata.

Area Shop: Integrazione di una sezione di shopping dove gli utenti possono acquistare prodotti dedicati al fitness, come attrezzature o integratori, con un carrello e opzioni di pagamento.

Conteggio delle calorie

È noto che per ottenere un fisico perfetto non basta allenarsi; anche la dieta gioca un ruolo fondamentale. Aggiungere una sezione dedicata alla gestione delle calorie e al bilanciamento delle assunzioni nutrizionali potrebbe elevare l'app dall'essere una delle tante disponibili a diventare l'unica necessaria per molti utenti.