

Relazione Laboratorio di Applicazioni Mobili
A.A. 2023-24

09/11/2024

Contents

1	Introduzione	3
2	Overview delle Funzionalità Implementate	3
3	Architettura dell'Applicazione	3
4	Struttura del Progetto	4
5	Implementazione dell'applicazione	4
5.1	View	4
5.2	Registrazione delle Attività	5
5.3	Tracciamento dei Passi	5
5.4	Rilevamento Automatico dell'Attività	6
5.5	BroadcastReceiver	6
5.6	Grafici	7
5.7	Notifiche Periodiche	7
5.8	Persistenza dei Dati	7
5.9	Calendario	8
6	Uno sguardo all'Interfaccia Utente	9
6.1	Login	9
6.2	HomePage	10
6.3	Grafici	10
6.4	Calendario	11
6.5	Profilo	12
6.6	Notifiche	13
7	Considerazioni Finali	14

1 Introduzione

Questa relazione descrive lo sviluppo dell'applicazione *Personal Physical Tracker*, progettata per dispositivi Android tramite Android Studio (*v. Ladybug 2024.2.1 Patch 1*). L'app è sviluppata in Kotlin (*v. 1.9.21*) utilizzando Jetpack Compose come toolkit per l'interfaccia utente e applicando il pattern architettonico MVVM (Model-View-ViewModel).

Si è cercato di rispettare i principi SOLID e le buone norme per lo sviluppo in Android apprese durante il progetto.

Nota: In questa relazione i termini "Progetto" e "Applicazione" saranno usati in modo intercambiabile.

2 Overview delle Funzionalità Implementate

L'applicazione *Personal Physical Tracker* fornisce una serie di funzionalità per il monitoraggio dell'attività fisica dell'utente. Tra queste:

- Monitoraggio delle attività (Walking, Driving, Sitting) con scelta manuale.
- Monitoraggio delle attività in maniera automatica tramite il riconoscimento dell'attività in corso ([Activity Recognition API](#))
- Conteggio dei passi ([Recording API](#)).
- Grafici relativi alle attività salvate ([MPAndroidChart](#)).
- Calendario per visualizzare le date in cui sono state registrate delle attività ([Calendar](#))
- Notifiche giornaliere per incoraggiare e ricordare il raggiungimento dei passi giornalieri (Notification Manager e Alarm Manager).

3 Architettura dell'Applicazione

L'applicazione segue l'[architettura MVVM](#), suddividendo l'interfaccia utente (View), la logica di gestione dei dati (ViewModel) e l'accesso ai dati (Repository).

- **View:** Schermate definite come *Screen*, ognuna delle quali è associata a un ViewModel.
- **ViewModel:** Contiene la logica di gestione dei dati e interagisce con le Repository.
- **Repository:** Classi che fanno da interfaccia per le ViewModel per l'accesso al database Room tramite l'utilizzo delle DAO.

L'iniezione delle dipendenze viene gestita con [Hilt](#), che facilita la creazione e il collegamento delle varie componenti del progetto.

4 Struttura del Progetto

Per un'organizzazione strutturale, il codice è suddiviso in vari package definito come quanto segue:

- **data**: Contiene le classi per la rappresentazione e la memorizzazione dei dati (incluso il DataStore).
- **di**: Contiene la classe `Modules` utilizzata da Hilt per la Dependency Injection.
- **room**: Include tutte le componenti per la creazione e gestione del database Room.
- **receiver**: Contiene le classi che implementano i *BroadcastReceiver*.
- **repository**: Include le repository che permettono ai ViewModel di interagire con il database Room.
- **service**: Contiene tutti i *Service*, incluse le classi che gestiscono il tracciamento in background e riconoscimento dell'attività.
- **ui**: Suddiviso in più package per gestire l'interfaccia utente, i ViewModel e le schermate (Screen).
- **utils**: Contiene funzioni e oggetti di supporto (helper) le quali vengono utilizzate da altre componenti.

5 Implementazione dell'applicazione

In questa sezione viene spiegato come sono state implementate alcune componenti e feature dell'applicazione, fornendo una panoramica della logica usata nella creazione dei componenti, senza andare nello specifico del codice usato.

5.1 View

Le *View* dell'applicazione sono realizzate utilizzando Jetpack Compose, che permette la creazione di interfacce utente tramite le *Composable*. Ogni schermata (o *Screen*) è associata a un ViewModel che gestisce la logica dei dati, garantendo una separazione tra l'interfaccia utente e la logica di business.

La navigazione tra le varie View viene gestita tramite la libreria [Navigation Compose](#), usando un Nav Host per la gestione delle destinazioni.

Le *View* utilizzano gli *state* di Compose per reagire automaticamente ai cambiamenti nei dati provenienti anche in alcuni casi dai ViewModel associati, aggiornando in tempo reale l'interfaccia grafica, migliorando la responsiveness dell'applicazione.

5.2 Registrazione delle Attività

La registrazione delle attività è la funzione base ed essenziale dell'applicazione. L'utente può scegliere manualmente il tipo di attività (*Walking*, *Running*, *Sitting*) tramite il Dialog che si apre quando l'utente clicca il Bottone Start del HomeScreen.

Dopo la scelta, il ViewModel associato segnalerà l'inizio dell'attività inviando un Intent al *TrackingService* con *Action START* passandogli anche la tipologia di attività negli Extra.

Quando l'utente clicca nuovamente il pulsante, verrà inviato un secondo Intent allo stesso Service, ma con l'*Action STOP*, facendo così terminare la registrazione dell'attività.

Il salvataggio dei dati verrà gestito direttamente tramite il Service che si collegherà al Database tramite il *TrackingDataDao* per il salvataggio dei dati.

Ogni registrazione viene associata a un record nel database tramite una *TrackingData* entity, che include il tipo di attività, l'ora di inizio e di fine, il numero di passi (nel caso di *WALKING*), e l'username dell'utente attivo in quel momento.

Durante tutta la fase di tracciamento, viene mostrato all'utente una notifica perenne (un ForegroundService implementato tramite il *TrackingService*) il quale tiene traccia del tempo trascorso dall'inizio dell'attività e si occupa di aggiornare la notifica ogni secondo, facendo visualizzare il tempo attuale.

5.3 Tracciamento dei Passi

Il tracciamento dei passi non è stato implementato utilizzando l'API di *Google Fit*, il quale risulta deprecato, ma sfruttando la *Recording API* che permette di recuperare i dati in modo efficiente dal punto di vista della batteria.

Questa API consente di recuperare il numero di passi effettuati dato un certo periodo temporale, senza l'utilizzo di un Account Google associato, i dati relativi ai passi sono conservati direttamente sul dispositivo dell'utente, accessibile tramite l'API.

Il conteggio dei passi viene effettuato in maniera automatica dal Sistema, mentre per ottenere le informazioni è necessario l'iscrizione all'API.

Quando l'utente avvia l'applicazione, viene iscritto quest'ultima al *Recording API*, così da permettere l'accesso ai dati registrati da quell'istante in poi. Il tracciamento sarà attivo anche quando l'app è in background in quanto sarà l'API a tracciare i passi effettuati e non la nostra applicazione.

Alla fine della sessione, verranno recuperati i passi effettuati dall'utente leggendo i dati tramite l'API, passandogli il periodo temporale (inizio e fine) dell'attività e raggruppandoli per giorni*.

I dati poi verranno salvati nel database, collegati all'utente e al periodo di tempo dell'attività registrata.

Per motivi logici dell'applicazione, dettati principalmente dall'eventuale "ritardo" che avviene dal momento in cui l'applicazione tenta di registrarsi all'API e dal costo "oneroso" di iscriversi e disiscriversi in continuazione, specialmente nel caso del rilevamento automatico dell'attività, si è deciso di registrare l'applicazione un'unica volta all'avvio e di disiscriversi quando l'applicazione viene terminata.

* Il raggruppamento può essere fatto anche per periodi più brevi o più lunghi, ma si è deciso di mettere 1 giorno così da evitare di creare più entries nel database senza motivo valido.

5.4 Rilevamento Automatico dell'Attività

L'applicazione integra l'API di Rilevamento dell'Attività (*Activity Recognition API*)^{*} per il riconoscimento automatico delle attività dell'utente. Questa API permette di identificare se l'utente sta camminando, correndo, guidando o è fermo, senza necessità di input manuali.

Quando viene avviata una sessione in cui viene scelta la modalità di tracciamento automatico, la ViewModel invierà un Intent esplicito a *ActivityDetectionService* il quale si occuperà di gestire il tracciamento dell'attività.

Il *ActivityDetectionService* utilizza un *ForegroundService* per monitorare le attività in background. Questo permette di continuare a tracciare le attività anche quando l'applicazione è chiuso in background.

Il Service ha solo due compiti, quello di iscrivere l'applicazione all'API definendo ogni quanto ricevere un Intent da quest'ultimo e quello di notificare all'utente, come implementato anche nel caso del tracciamento manuale, che è in corso la registrazione delle attività in maniera automatica tramite una notifica perenne. Mentre l'API è impostato a inviare un *PendingIntent* esplicito al nostro *BroadcastReceiver* ogni mezzo secondo (parametro modificabile) da quando si registra l'applicazione, il quale quest'ultimo gestirà poi il controllo dell'attività corrente individuato e l'eventuale registrazione dell'attività nel database.

L'*Activity Recognition API* stima l'attività tramite i sensori e ne fornisce una "confidenza" sulla correttezza del tipo di attività in corso. Si è deciso di impostare manualmente la confidenza al 70 percento (parametro modificabile), così da ignorare eventuali risultati in cui la confidenza così da aumentare la probabilità che l'attività rilevata sia corretta.

* Nel progetto viene utilizzato l'*Activity Recognition API* e non l'*Activity Recognition Transition API*. Questo è dovuto al fatto che la *Transition API* si appoggia all'*Activity Recognition API*, ma non è possibile impostare ogni quanto ricevere un Intent. Oltre a questo, dopo vari test sia su emulatore che su dispositivo reale, l'utilizzo della *Transition API* aveva qualche ritardo e non sempre puntuale con l'invio degli Intent.

5.5 BroadcastReceiver

Nel progetto troviamo 3 *Broadcast Receiver*, i quali hanno tutti l'obiettivo di catturare degli Intent specifici.

- **DetectionActivityReceiver:** Gestisce la ricezione degli Intent provenienti dalla Activity Recognition API e dall'applicazione stessa (per la terminazione manuale), salvando l'ultimo tipologia di attività registrata nel DataStore, così da poterlo controllare in un successivo Intent. Il Receiver salva un'attività nel database tramite il relativo DAO, solamente quando riceve un Intent relativa al rilevamento di un'attività diversa da quella precedentemente, ignorando tutti quei casi in cui la confidenza è minore del 70 percento.

- **NotificationReceiver:** Usato per ricevere gli Intent interni per gestire le notifiche periodiche (giornaliere). Viene usato poi al suo interno il Notification Manager per inviare le rispettive notifiche dopo aver effettuato i relativi controlli.
- **BootReceiver:** Usato per gestire le notifiche periodiche nel caso in cui il dispositivo si riavvii. Quando il dispositivo viene riavviato, riceverà un Intent con *ACTION BOOT COMPLETED*, il quale segnalerà al nostro Receiver di programmare tramite un Alarm Manager l'invio dei due Intent al *Notification Receiver* per le notifiche periodiche.

5.6 Grafici

I grafici vengono implementati con la libreria *MPAndroidChart* per rappresentare visivamente i dati dell'utente. In particolare, è stato scelto un grafico a barre (*BarChart*) per mostrare i passi effettuati nei vari mesi e un grafico a torta (*PieChart*) per le statistiche sulle tipologie di attività registrate.

Questi vengono aggiornati automaticamente ogni volta che l'utente registra nuove attività, grazie all'uso dei *LiveData* e agli *Observer* associati. I dati vengono presi tramite la ViewModel relativa, che si interfaccia con il Database attraverso i DAO, usando classi personalizzate come contenitori per i dati.

Si è deciso di non aggiungere altri grafici semplicemente dovuto al fatto che l'utilizzo di altri grafici era solo una questione di modificare le query SQL dei DAO per ottenere dei dataset dal database che rispettassero la tipologia di dati che i vari Grafici mostravano.

5.7 Notifiche Periodiche

Le notifiche periodiche avvengono tramite il sistema di notifica di Android con un *NotificationChannel* denominato "tracking", creato appositamente all'avvio dell'applicazione.

Sono state configurate due notifiche principali:

- Una notifica alle 9:00*, che incoraggia l'utente a iniziare la propria attività giornaliera.
- Una notifica alle 18:00*, che verifica se l'utente ha raggiunto l'obiettivo di passi giornaliero (impostato a 100* passi) e nel caso indica.

Le notifiche vengono gestite con un *Alarm Manager*, che assicura l'invio degli Intent anche se l'app è chiusa. Gli Intent vengono poi ricevuti dal *Notification Receiver* descritto sopra.

5.8 Persistenza dei Dati

Si è deciso di usare la libreria *DataStore*, che viene definita sul sito ufficiale di Android come "la soluzione moderna che si dovrebbe usare per l'archiviazione dati rispetto alle Shared-Preferences" e *Room* per quanto riguarda l'uso di un Database.

Datastore è una soluzione di archiviazione dati che consente di archiviare coppie chiave-valore, principalmente usato per salvare e accedere a dati fissi, come username attivo in questo momento, oppure valori che non necessitano di essere mantenuti o salvati in una tabella nel Database.

Room è un database locale, usato principalmente per salvare in forma strutturata (tramite tabelle), i vari dati che l'applicazione usa. Come i dati relativi alle attività svolte o l'elenco degli utenti iscritti sul dispositivo. Viene implementato DAO con le sue classi Standard (Database - Entity - DAO), come indicato anche dalla guida ufficiale Android. Viene però utilizzato una classe Converter che permette di Convertire i timestamp di tipo Long in valori Date, in quanto ROOM non accetta tutti i tipi di variabili, ma solo alcuni.

5.9 Calendario

Viene implementato tramite l'utilizzo della libreria [Calendar](#) di kizitonwose e permette di visualizzare in un calendario dinamico i giorni in cui sono state registrate delle attività.

Tramite il relativo ViewModel, prende i dati salvati nel database ROOM e li modifica tale da poterli usare per generare correttamente le date in cui risultano registrate attività per poi essere mostrate all'utente.

Viene implementato anche un Dialog che farà da filtro, il quale verrà passato alla View-Model così da consentire all'utente di decidere quali tipologie di attività visualizzare nel Calendario.

6 Uno sguardo all'Interfaccia Utente

In questa sezione verrà mostrato com'è graficamente l'applicazione e le sue varie View (Screen). La navigazione tra le varie View sono gestite tramite la Bottom Navigation Bar che viene generata all'inizio dell'applicazione, nella quale troviamo le 4 View Principali (Home-Page, Charts, Calendar, Profile).

Mentre troviamo una quinta View, quella del Login per l'inserimento dell'username, che compare solamente la prima volta o quando non è stata impostata ancora l'username attivo di default.

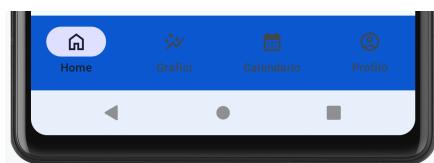


Figure 1: Bottom Navigation Bar

6.1 Login

Nella schermata di Login troviamo un campo di testo dove possiamo inserire il username e entrare direttamente come quell'utente.

Mentre sotto troviamo un elenco di Utenti già registrati nell'applicazione, dove possiamo entrare nuovamente come tali oppure eliminarli definitivamente dall'applicazione tramite apposito pulsante.



Figure 2: Login Screen

6.2 HomePage

Nella HomePage troviamo un semplice pulsante per iniziare la sessione di registrazione dell'attività, il quale ci fa comparire un Dialog con la quale ci permette di selezionare il tipo di Attività da registrare, o nel caso quello automatico.

Dopo aver iniziato la sessione, il bottone cambia testo e servirà per interrompere la sessione, terminando così la fine del tracciamento e salvando i dati nel Database.



Figure 3: Home Page

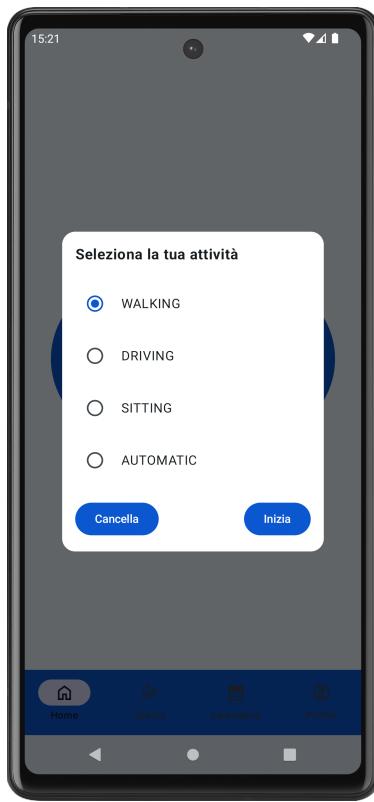


Figure 4: Dialog per la selezione dell'attività



Figure 5: Tracking Avviato nella Home Page

6.3 Grafici

La seconda View è quella dei grafici, dove possiamo vedere il grafico a Torta che ci indica la percentuale (in base al tempo) di Attività che abbiamo nel Database.

Mentre sotto troviamo un grafico a barre, dove ogni barra indica un mese e il totale dei passi registrati in quel dato mese.

Note: I valori visualizzati sotto sono stati inseriti manualmente nel database, così da avere più dati da visualizzare.



Figure 6: View Grafici

6.4 Calendario

La terza View è quella del calendario, nella quale possiamo visualizzare in modo semplice e chiaro le date (quelle in blu) in cui sono state registrate delle attività. Cliccando sulla data, comparirà in basso un elenco di tutte le attività di quella giornata.

Inoltre è possibile filtrare le date per attività, cliccando l'icona in alto a destra che farà comparire un Dialog con la quale possiamo decidere quali attività visualizzare nel calendario.

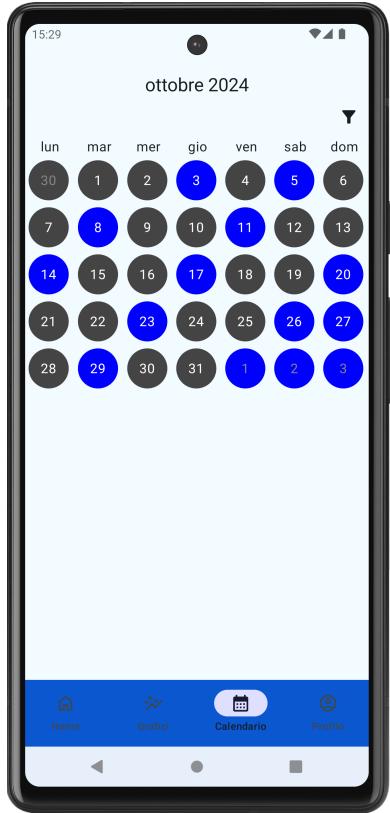


Figure 7: Calendario



Figure 8: Dialog per il Filtro delle attività



Figure 9: Attività della Giornata

6.5 Profilo

La View del profilo è abbastanza minimale, permette all'utente di visualizzare il proprio username e nel caso di fare il Logout.

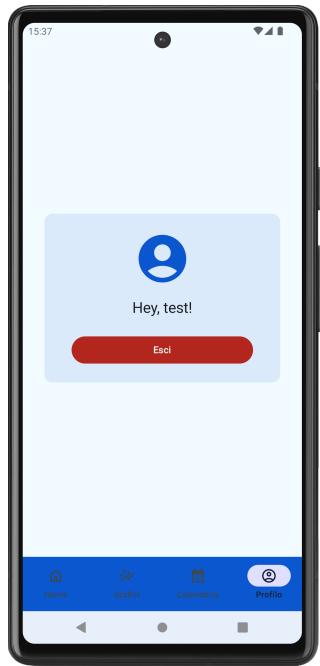


Figure 10: Profile Screen

6.6 Notifiche

L'applicazione ha 4 notifiche principali:

- Notifica di tracciamento manuale in corso.
- Notifica di tracciamento automatico in corso.
- Notifica periodica delle 9:00
- Notifica periodica delle 18:00

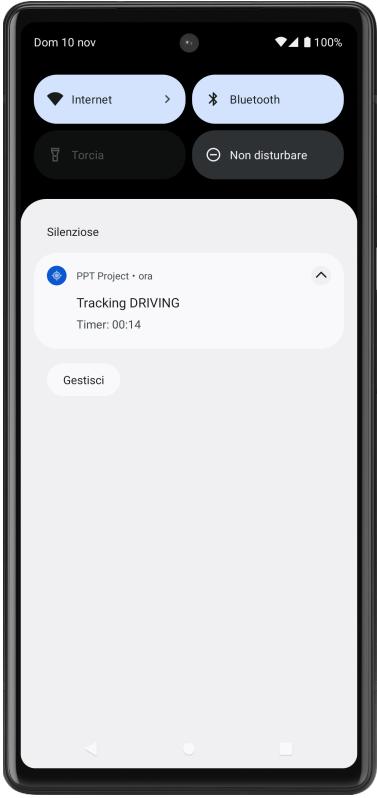


Figure 11: Notifica del Tracciamento Manuale in corso

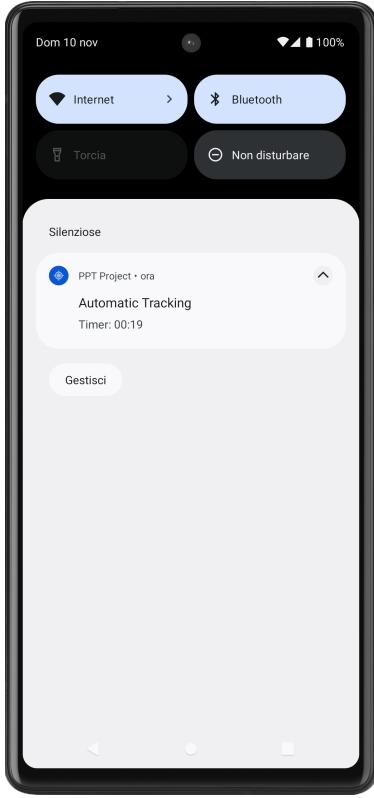


Figure 12: Notifica del Tracciamento Automatico in corso

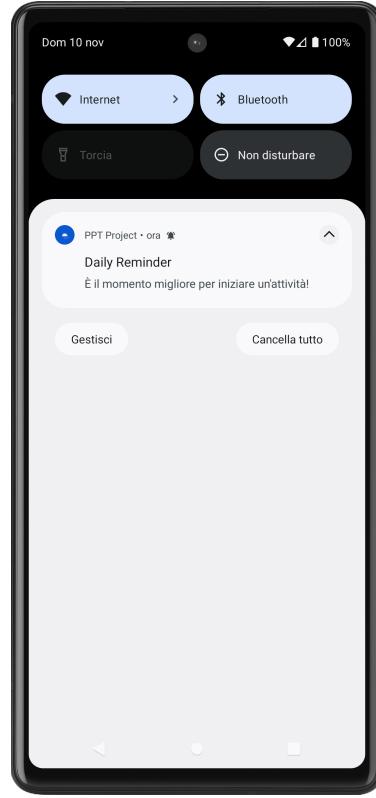


Figure 13: Esempio di Notifica Giornaliera delle ore 9:00

7 Considerazioni Finali

L'applicazione segue la lingua del dispositivo ma di default è settato a *English*, mentre se il dispositivo è in lingua Italiana, tutte le scritte e notifiche saranno in *Italiano*.

Il progetto è conservato su una repository su Github [[Link](#)], nella quale sono state salvate tutti i commit per la creazione di ogni sua singola parte, così da tenere traccia dei vari progressi fatti.

All'inizio del progetto si era deciso di non utilizzare Hilt per la gestione delle dipendenze, in quanto era più chiaro capire concettualmente e vedere come si collegavano i componenti tra di loro, ma a metà progetto si è deciso di implementare Hilt in quanto contribuiva a uno sviluppo più efficiente e una chiarezza del codice.

Per lo sviluppo di questa applicazione è stato essenziali varie guide trovate sul web, in particolar modo quella riferitasi a [Hilt](#), quello per il [riconoscimento dell'attività](#) anche se non è stato utilizzato propriamente quello nel tutorial.

In particolar modo è stato utile l'[Android Codelabs](#) nella quale si trovano vari tutorial per lo sviluppo di applicazioni con le varie metodologie (es. MVVM) e librerie (es. ROOM).