POLITECNICO DI TORINO

Corso di Laurea Magistrale in Ingegneria Informatica

Politecnico
di Torino

Titolo della tesi

Well being app - progetto di interazione con wearable e back end

Relatore

Prof. Maurizio MORISIO

Candidato

Enrico SCALABRINO

Anno Accademico 2021-2022

Alla mia famiglia, grazie di tutto

INDICE

SOMMARIO

Contesto:

Il progetto Well being app, nasce con lo scopo di realizzare una applicazione multi-piattaforma (Android/iOS) per assistere e monitorare (attraverso device esterni tipo smartwatches, wristbands) dati clinici e di salute per persone anziane (attività fisica, qualità e durata del sonno, pressione, pulsazioni) e proporre recommendations, derivate da letteratura scientifica specifica al caso. [1]

I primi mesi dell'attività di tesi sono stati dedicati all'analisi dell'offerta di mercato dei devices predisposti all'acquisizione dei dati e delle funzionalità che essi offrivano in termini di API disponibili, canali di comunicazione (BLE) e costo. Successivamente, scelto il device più idoneo (famiglia Fitbit), si è passati a progettare e adattare l'architettura della mobile app all'architettura back end già esistente.

Obiettivi:

Con la collaborazione del Professore Luigi Fontana e dell'Azienda Ospedaliera di Verona, il progetto ha come target due tipologie di pazienti: quello "sperimentale" e quello di "controllo". In particolare, l'applicazione cambia morfologia, e quindi presenta diverse UI, a seconda della tipologia di paziente: Nel caso di paziente sperimentale i dati raccolti sono presentati con un livello di bravura (attraverso l'uso di colori diversi) e con le conseguenziali recommendations che ne derivano. Nel caso di paziente di controllo, tali UI sono più neutre e non presentano recommendations.

Metodologia:

Una prima scelta iniziale era stata quella di sfruttare il canale di comunicazione che utilizzano gli wristbands per comunicare con le mobile applications, ovvero sfruttare il canale Bluetooth Low Energy (BLE) per estrapolare direttamente i dati senza applicativi terzi. Ma l'impossibilità di accedere ai dati presenti nel wristbands (poiché protetti a livello firmware) ha fatto abbandonare l'idea, che è virata verso l'utilizzo di Application programming interface (API) a disposizione del wristband scelto. Visto

l'esigenza di creare una applicazione mobile multipiattaforma e l'esistenza di un back end già consolidato ed a servizio di un Web Front-end oggetto di altra tesi (applicazione web per i dottori che monitorano i pazienti in questione), la scelta più naturale è stata di utilizzare il framework React-Native con linguaggio di programmazione Javascript, per avere una interazione facilitata con l'architettura di tipo REST preesistente. Una proposta di architettura alternativa (GraphQL) sarà discussa in un capitolo, come variante a quella adottata (REST) mettendo a confronto i vantaggi e gli svantaggi delle due architetture in questione.

Conclusioni:

Il lavoro svolto ha portato alla stesura di una versione iniziale dell'applicazione mobile, che è totalmente integrata nell'architettura complessiva esistente e implementa gran parte dei requisiti iniziali richiesti. Successivamente potrà essere estesa per tracciare dati sulla nutrizione, tipo, quantità e qualità dell'alimentazione quotidiana.

ELENCO DELLE FIGURE

ELENCO DELLE TABELLE

ACRONIMI

1. INTRODUZIONE

Il supporto scientifico che ha dato origine alla sperimentazione di questo progetto di tesi di laurea sono i contenuti trattati nel testo “The Path to Longevity: How to Reach 100 with the Health and Stamina of a 40-Year-Old” [1] di Luigi Fontana, professore ordinario di Medicina e Scienze nutrizionali presso l’Università di Brescia e la Washington University di St. Louis e scienziato riconosciuto a livello internazionale tra i massimi esperti nel campo del benessere e della longevità.

Il testo mette sotto i riflettori i parametri della vita quotidiana che hanno effetto in modo rilevante la salute delle persone, quali nutrizione, esercizio fisico, salute mentale e qualità del riposo, proponendo le “recommendations” intese come le buone norme di comportamento da seguire per ottenere uno stile di vita sano e longevo.

L’aumento delle aspettative di vita del genere umano è uno tra gli obiettivi più ambiziosi di cui si sta facendo carico il progresso tecnologico in ambito medico nel mondo, come confermato dalla presenza sul mercato di moltissimi dispositivi (wearables) che informano in tempo reale sulle regole e buone prassi connesse al mantenimento di uno stile di vita sano.

Tuttavia se l’uso di tali dispositivi è immediato e continuo da parte di nuove generazioni, le persone più anziane, invece, presentano notevoli difficoltà ad interagire con queste nuove tecnologie per quanto ne vengano a loro proposti l’uso e l’utilità.

Strettamente connesso a facilitare l’interazione tra utilizzatori in avanzata età e tali dispositivi c’è anche la modalità di comunicazione con i medici che seguono tali pazienti, ed a questo stanno mirando un gruppo di medici dell’ospedale di Verona con uno studio finalizzato a semplificare la comunicazione degli accorgimenti e degli stili di vita da adottare sulla base dei bati acquisiti da dispositivi.

La necessità di conservare dati relativi allo stato di salute dei pazienti e, soprattutto, la loro evoluzione nel tempo, al fine di valutarne successivamente l'andamento qualora aderiscano al monitoraggio, ha indicato la strutturazione della applicazione che verrà descritta nei capitoli successivi.

2. ANALISI DEL MERCATO

2.1. PANORAMICA DEI DISPOSITIVI WEARABLES

La scelta del dispositivo più adatto a raggiungere gli obiettivi che si pone il progetto è basilare per l'architettura e la soluzione tecnologica che deve avere l'applicazione.

Per le motivazioni introdotte precedentemente, si è reso necessario procedere attraverso una analisi dell'offerta che il mercato oggi propone.

In particolare, l'analisi è stata orientata alla ricerca di dispositivi facili da indossare e dotati di sensoristica idonea all'estrapolazione dei dati biometrici adatti alla fattispecie dei pazienti da monitorare.

I più comuni sensori di cui sono dotati i dispositivi indossabili vengono di seguito elencati:

- l'accelerometro, il sensore in grado di rilevare e/o misurare l'accelerazione.
- il giroscopio, che misura la velocità angolare per determinare l'orientamento del dispositivo;
- il GPS, per rilevare le coordinate dell'utilizzatore, utile soprattutto quando non è previsto l'utilizzo di uno smartphone;
- l'elettrocardiogramma (ECG) oppure (HeartRate Monitor), per misurare l'attività e la frequenza cardiaca;
- il saturimetro, che rileva l'ossigenazione del sangue;

Determinante per l'affidabilità e l'accuratezza delle misurazioni prodotte è ovviamente la qualità dei sensori così come il range e la sensibilità ne determinano, invece, il potenziale di efficacia degli algoritmi di analisi dei dati.

Proprio gli algoritmi utilizzati hanno un ruolo fondamentale per determinare l'apprezzamento nel mercato dei produttori di wearable device: dalla loro implementazione e dalla loro capacità di analisi dei dati diretti provenienti dai sensori per derivarne indirettamente informazioni più complesse dipende la qualità del wearable; si pensi ad esempio alle informazioni sulla qualità del sonno a partire da misure dirette sul movimento, sulla temperatura e sulla frequenza cardiaca.

Altre funzionalità più comuni offerte dai wearable sono:

- l'attività fisica, in particolare il numero di passi, la distanza percorsa, il dislivello. Tali funzioni tengono traccia dei minuti giornalieri trascorsi in modo sedentario e quelli in cui è stata rilevata un'attività, indicandone il livello di intensità. Alcuni derivano una stima del numero di calorie bruciate;

- la frequenza cardiaca. La misurazione costante del battito cardiaco permette di rilevare la propria frequenza cardiaca a riposo, il tempo passato in determinati range di frequenza che identificano il livello di attività, eventuali picchi anomali e il tempo impiegato per recuperare dopo uno sforzo;

- il sonno, sia per quanto concerne la quantità che la qualità. Tramite la combina- zione di alcuni parametri quali l'assenza di movimento, la frequenza cardiaca, la temperatura, la frequenza di respirazione si elabora una stima piuttosto affidabile dei minuti passati in ogni fase del sonno (leggera, profonda, REM), rilevando anche eventuali sospensioni.

I wearables sono oggi dispositivi largamente utilizzati per monitorare il proprio stile di vita in ampie fasce di età della popolazione, con tecnologie che permettono il rilascio di informazioni sempre più accurate senza avere necessariamente dimestichezza con il mondo digitale; il prezzo contenuto di questi dispositivi, nell'ordine delle decine di euro, ha agevolato la loro diffusione, in particolare tra i giovani e coloro che fanno attività sportiva.

Il loro utilizzo però nelle fasce di popolazione in età avanzata, appare problematico soprattutto con riguardo alla lettura dei dati: oltre alla limitata superfice visiva dei display la criticità maggiore riguarda la necessità di accedere alle informazioni attraverso il passaggio successivo di schermate che presentano numerosi elementi quali grafici, dati numerici e unità di misura.

Per questo tipo di utilizzatori, per quanto sopra detto, appare molto utile la possibilità di dotarsi di una applicazione che si interfaccia con il proprio smartphone (o iPhone) e che migliori la facilità di utilizzo e accesso alle informazioni raccolte dai wearables.

2.2. SCELTA DEL WEARABLE

L'analisi del mercato ha consentito di raccogliere uno scenario delle offerte tecnologiche che vengono sintetizzate in modo tabellare nella tabella sotto riportata:

Tabella 2-1 Confronto offerta wearables (aggiornamento 06/2022)

Dotazione dati forniti	Wearable				
Sensori	Fitbit inspire 2	Miband 5/6	Huawei band 6	Honor Band 6	AmazFit Band 5
Accelerometro	Si, 3 assi	Si, 3 assi	Si, 3 assi	Si, 3 assi	Si, 3 assi
Giroscopio	No	Sì	Sì	Sì	Sì
Heart Rate Monitor	Sì	Sì	Sì	Sì	Sì
Sangue O2 Monitor	No	Sì	Sì	Sì	Sì
Controllo telecamera smartph in remoto	No	Sì	Sì	Sì	Sì
Sensore caduta del dispositivo	Sì	No	No	No	No
Misure					
Contapassi	Sì	Sì	Sì	Sì	Sì
Monitoraggio delle calorie introdotte con il cibo	Sì	No	No	No	No
Monitoraggio calorie consumate	Sì	Sì	Sì	Sì	Sì
Misurazione Ritmo della attività sportiva	No	Sì	Sì	Sì	Sì
Distanza percorsa	Sì	Sì	Sì	Sì	Sì
Battito cardiaco	Sì	Sì	Sì	Sì	Sì
Saturazione O2	No	Sì	Sì	Sì	Sì
Monitoring stress	Sì	Sì	Sì	Sì	Sì
Moitoraggio sonno	Sì	Sì	Sì	Sì	Sì
Report della qualità del sonno	Sì	Sì	Sì	Sì	Sì
Capace di riconoscere heart rate irregolari	No	Sì	Sì	No	Sì
Monitera il BMI	No	Sì	Sì	Sì	Sì
Monitoraggio del peso	No	Sì	Sì	No	Sì
Connettività					
Durata batteria	10 giorni	14 giorni	14 giorni	14 giorni	14 giorni
Capacità batteria	/	125 mAh	180 mAh	180 mAh	125 mAh
Tempo ricarica	/	2 ore	1 ora	Sconociuto	Sconosciuto
Ricarica wireless	No	No	No	Si	Si
Compatibile Android	Sì	Sì	Sì	Sì	Sì
Compatibile iOS	Sì	Sì	Sì	Sì	Sì
Sincronizzazione wireless automatica	Sì	Sì	Sì	Sì	Sì
Bluetooth version	4	5	5	5	5
Bluetooth range funzionamento	Sconosciuto	10 metri	10 metri	Sconosciuto	Sconosciuto
Controllo chiamate in entrata	Sì	Sì	Sì	Sì	Sì
Può catturare foto	No	Sì	Sì	Sì	No
Valutazione media della app da parte degli utenti Apple Store	FitBit APP -> 4.5	Mi Fit -> 3.5 Zepp App -> 4.6	Huawei Health -> 2.6	Huawei Health -> 2.6	Zepp App-> 4.6
Valutazione media della app da parte degli utenti Google Play	FitBit APP -> 4	Mi Fit -> 3.8 Zepp App -> 4	Huawei Health -> 4.3	Huawei Health -> 4.3	Zepp App -> 4
Richiede account associato	Sì	Sì	Sì	Sì	Sì
Connessione	Bluetooth BLE	Bluetooth BLE	Bluetooth,BLE	Bluetooth,BLE	Bluetooth,BLE
Sincronizzazione cloud	Sì	Sì	Sì	Sì	Sì
Transfer dati / smartphone	FitBit App	Mi Fit App Zepp App	Huawei Health App	Huawei Health App	Zepp App
Transfer dati / server	Web API di tipo RestFul - FitBit Web API	Huami API (Zepp API) Google Fit API	Huawei Health kit API	Huawei Health kit API	Huami API (Zepp API) Google Fit API
Tipologia API	Restful	REST, Android, iOS	REST, Android	REST, Android	REST, Android, iOS
Costo	Circa 60 €	Circa 35 €	Circa 40 €	Circa 50 €	Circa 35 €

Oltre ai parametri precedentemente detti, sono importanti altre caratteristiche non secondarie: il supporto da parte dei produttori di app proprietarie, che permettono di capire se i dati hanno una misurazione affidabile, e soprattutto le API che essi forniscono, ovvero un insieme di definizioni e protocolli per la creazione e l'integrazione di software applicativi e infine il costo, fattore comunque importante da tenere in considerazione.

Come è visibile nella tabella sono diversi i produttori forniscono API; la prima parte del lavoro di tesi è stato quindi dedicato alla analisi di tali API per capire quali effettivamente fossero adatte al nostro caso.

Da una attenta analisi della offerta di mercato, quella più idonea a perseguire gli obbiettivi finali del progetto è apparsa la scelta di Zepp API [2].

Nonostante le diverse richieste inoltrate al produttore di queste API, non si è riusciti a ricevere risposta e quindi è maturata la necessità di abbandonare l'idea di utilizzare dispositivi di fascia bassa ma efficaci come Xiaomi mi band.

Come primo tentativo si è cercato di implementare in codice Android nativo [3] le API fornite da Google Fit.

Il risultato ottenuto in parte è stato soddisfacente, ma per i motivi che descriverò nel paragrafo successivo, si è deciso che non fosse la strada giusta da intraprendere.

La sintesi dei risultati di ricerca sopra descritti, fermo restando l'obiettivo di progettare un'unica App proprietaria in grado di avere una interazione diretta con un wearable, è stata quella che la direttrice della attività si è focalizzata sullo studio del protocollo di comunicazione tra dispositivo wearable e uno smartphone, attraverso il protocollo di comunicazione Bluetooth Low Energy (BLE) come meglio descritto nel capitolo che segue.

2.3. GOOGLE FIT

Prima di abbandonare definitivamente l'idea di utilizzare dispositivi di fascia medio-bassa come quelli ad esempio di Xiaomi, si è tentata la strada Google Fit. [4]

Google Fit fornisce API Android e REST per aiutare gli sviluppatori a creare app per monitoraggio salute e benessere in modo più intelligente.

L'obiettivo è stato quello di utilizzare le API di Google Fit per entrare in contatto con diverse altre app di sistemi proprietari (Xiaomi come esempio) gestendo allo stesso tempo un'unica integrazione.

Google Fit è un ecosistema aperto: consente agli sviluppatori di caricare dati relativi a salute e benessere in un repository centrale in cui gli utenti possono accedere ai loro dati da diversi dispositivi e app da un'unica posizione; l'utente potrà comunque accedere ai propri dati se esegue l'upgrade ad un nuovo dispositivo.

L'app per salute e benessere può memorizzare dati da qualsiasi dispositivo indossabile o sensore e accedere ai dati creati da altre app.

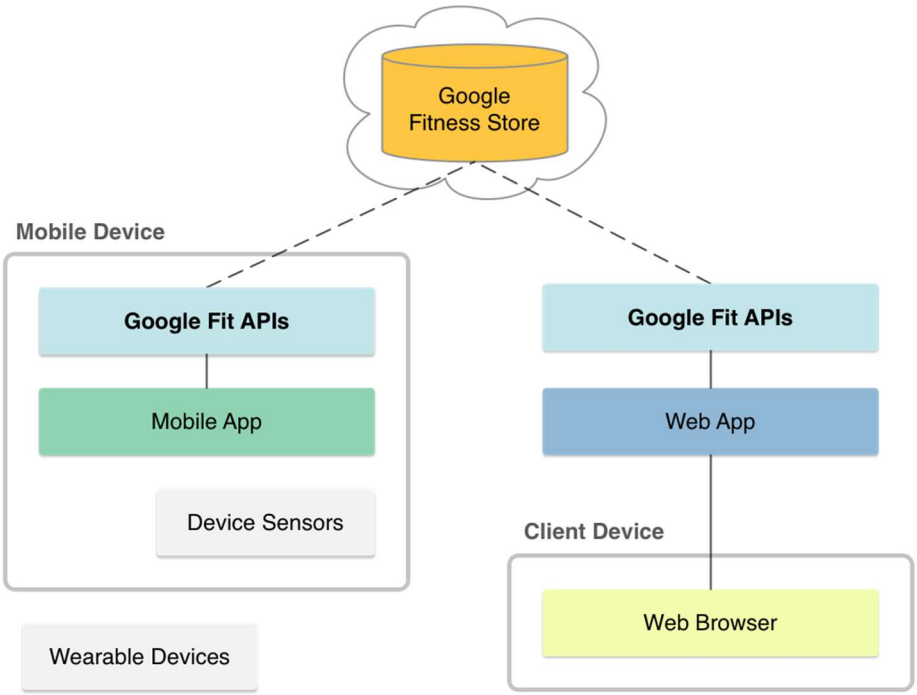


Figura 2-1 Panoramica della piattaforma

Google Fit è costituito dai seguenti componenti:

- articoli per il fitness: è un servizio cloud che memorizza i dati di salute e benessere utilizzando l'infrastruttura di Google. Le app su piattaforme e dispositivi diversi possono archiviare dati e accedere ai dati creati da altre app. Google Fit fornisce un set di API che semplificano l'inserimento di dati e di eseguire query sul negozio per l'attività fisica.

- Il framework dei sensori: definisce rappresentazioni di alto livello per sensori, tipi di dati, punti dati (data point) e sessioni; queste rappresentazioni semplificano l'utilizzo del negozio di fitness su qualsiasi piattaforma, come per esempio ottenere direttamente le coordinate spaziali dell'utilizzare sfruttando il giroscopio presente nel tracker; nel proseguo è presente un esempio grafico;
- Autorizzazioni e controlli utente: un insieme di ambiti di autorizzazione per richiedere l'autorizzazione dell'utente a lavorare con i dati relativi a salute e benessere.

  Google Fit richiede il consenso degli utenti per accedere ai dati relativi a salute e benessere definendo quindi ambiti OAuth [5] che vengono mappati a una serie di gruppi di autorizzazioni con privilegi di lettura e scrittura distinti: attività, corpo, posizione, alimentazione e tipi di dati correlati alla salute (i tipi di dati relativi alla salute vengono raggruppati in modo più granulare); ogni gruppo di autorizzazioni concede alle app l'accesso a un insieme di tipi di dati: ad esempio i valori dei passi raccolti dall'utente non richiedono particolari autorizzazioni, mentre i dati relativi alla frequenza cardiaca risultato più sensibili e richiedono privilegi autorizzativi più stringenti;

- API Google Fit: API Android e REST per accedere allo store per il fitness. Puoi creare app che supportano Google Fit su più piattaforme e dispositivi, come Android, iOS e app web.

Di seguito mostro un esempio di utilizzo di tali api, direttamente dalla repository ufficiale Google, con riguardo l'estrapolazione dei passi effettuati.

Prima di utilizzarle è necessario registrarsi come sviluppatore, ottenere un certificato per autenticare il client che le utilizza. Tale certificato è uno SHA-1 fingerprint, ottenuto attraverso il comando keytool con tale sintassi (Windows):

```
keytool -list -v \
-alias androiddebugkey -keystore
%USERPROFILE%\.android\debug.keystore
```

Il risultato come anticipato sarà uno SHA-1:

```
Certificate fingerprint: SHA1:
DA:39:A3:EE:5E:6B:4B:0D:32:55:BF:EF:95:60:18:90:AF:D8:0
7:09
```

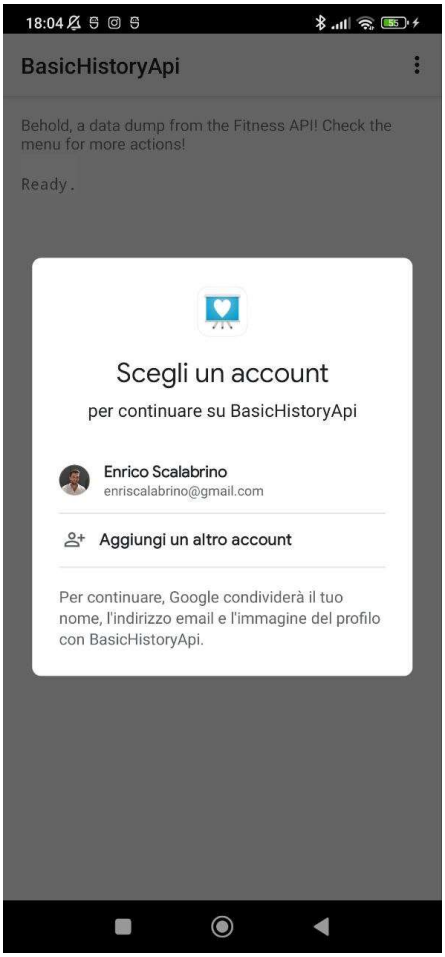


Figura 2-2 Autenticazione utente

L'autenticazione dell'utente si basa su **OAuth 2.0**, acronimo di "Open Authorization", è uno standard progettato per consentire a un sito web o a un'applicazione di accedere a risorse ospitate da altre applicazioni web per conto di un utente.

Ha sostituito OAuth 1.0 nel 2012 ed è ora lo standard de facto del settore per l'autorizzazione online. OAuth 2.0 fornisce l'accesso con consenso e limita le azioni che l'applicazione client può eseguire sulle risorse per conto dell'utente, senza mai condividere le credenziali dell'utente.

Dopo questi passi preliminari, è stato possibile ottenere facilmente dati quali i passi durante la giornata, divisi anche per settimana, che venivano visualizzati dal prototipo di applicazione Android, così come pure per il battito cardiaco (tenendo conto anche le necessarie autorizzazioni da implementare nel codice per quei dati ritenuti sensibili).

Figura 2-3 Passi letti durante la settimana

Un altro esempio sotto riportato è il framework dei sensori, nel quale si evince l'estrapolazione di oggetti come **DataPoint** che rappresentano le coordinate geografiche con relativa accuratezza associata.

Figura 2-4 Framework dei sensori

Nonostante queste API funzionassero anche con dispositivi da diverso tempo presenti nel mercato ed economicamente accessibili (ad esempio il test è stato effettuato con MiBand3 di Xiaomi) , il loro impiego imponeva l'utilizzo di diverse app necessariamente installate nello smartphone, tra cui :

- L'app proprietaria del wearable, in questo caso ZeppApp (Xiaomi);

- L'App GoogleFit, che agiva da ponte per estrapolare i dati presenti nel back end Xiaomi e fornirli attraverso le loro API Fitness;
- L' App di progetto, **(HealthApp)** quale destinataria di dei dati forniti da Google.

E' apparso subito evidente la difficolta di configurazione (in considerazione della destinazione della applicazione a persone anziane) che, unita alle dipendenze sopra elencate per ottenere i dati, ha orientato la decisione di abbandonare la strategie di progetto sopra detta e favorire quella di creare un'unica applicazione che fosse in grado di estrapolare i dati direttamente dal un wearable indossato dal paziente.

Al momento storico della scelta, le API Google richiedevano l'app Google Fit installata. Un upgrade dell'architettura successivo ha eliminato questo vincolo, permettendo l'utilizzo di tali api direttamente con l'ausilio di Google Play Services. evitando quindi le complicazioni di configurazione sopra dette.

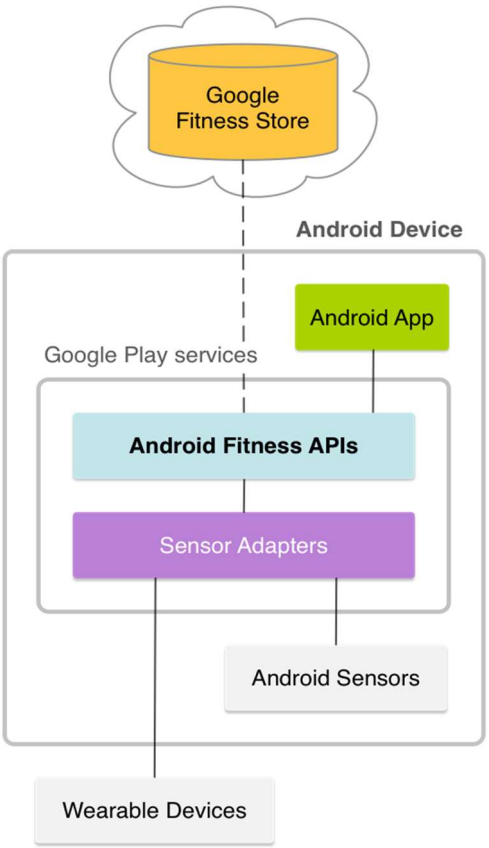


Figura 2-5 Upgrade architettura API Google

Una soluzione alternativa a tale problema poteva prevedere la scelta di utilizzare un wearable con WearOS, [6] cioè il sistema operativo per dispositivi indossabili di Google, questo avrebbe permesso una associazione diretta senza utilizzare app di terze parti (Zepp,Huawei come esempio).

Ultimamente la direzione di sviluppo tecnologico è quella di integrare questo unico sistema operativo in tutti i bracciali indossabili, ma lo scenario attuale vede questo OS disponibile solo per dispositivi di fascia alta che certamente non possono fare a caso per i nostri scopi.

Inoltre, una nota ufficiale presente nel sito **android developers** [7] con riguardo a tali API, riporta il completo ritiro dell'API Fit Android, informando che verrà disattivata entro la fine del 2024. Questo ha dato un indizio sul fatto che le relative API non erano certamente esaustive per un completo utilizzo adatto al nostro scopo. L'API in questione diventerà "**Health Connect**" progetto di API universale che raccoglie, a sua volta, diverse sotto API presente nella suite Fit.

3. BLUETOOTH LOW ENERGY

3.1. IL PROTOCOLLO

Attraverso una fase di ricerca e di studio sul tipo di connessione tra bracciali e smartphone si è individuato un protocollo di comunicazione potenzialmente idoneo.

La parte "Low Energy" dell'acronimo BLE ci dà un suggerimento: mentre il Bluetooth classico è progettato per la trasmissione di flussi continui di dati, come la riproduzione di musica, il BLE è ottimizzato per l'efficienza energetica.

Questo protocollo trova utilizzazione nei dispositivi wearable per sopperire alla loro capacità di autonomia limitata, quindi al fine di mantenere tale autonomia al più lungo possibile. Infatti, un dispositivo BLE può in genere funzionare con una piccola batteria per settimane, se non mesi o addirittura anni, il che lo rende perfetto per i casi d'uso basati su sensori o sull'Internet of Things (IoT); queste caratteristiche lo rendono adatto al fine di monitorare al meglio i valori biometrici di chi indossa il bracciale, scambiando periodicamente piccole quantità di dati.

Come il Bluetooth classico, anche il BLE opera nella banda di frequenza 2,4 GHz, ma le connessioni aperte durano pochi millisecondi per poi ibernarsi temporaneamente. Questa ottimizzazione permette di raggiungere un livello di consumo fino a 100 volte inferiore rispetto al comune Bluetooth, a fronte di un limitato bit rate di 125 Kb/s, 500 Kb/s, 1 Mb/s a seconda della versione utilizzata.

Prima di procedere alla descrizione del funzionamento del protocollo, è necessario definire alcuni principali attori coinvolti: [8]

- **Centrale/Client:**Un dispositivo che esegue la scansione e si connette alle periferiche BLE per eseguire alcune operazioni. Nel contesto dello sviluppo di applicazioni, si tratta in genere di un dispositivo Android.
- **Periferica/Server:**Un dispositivo che pubblicizza la sua presenza nella rete e a cui viene collegato un client per svolgere un compito. Nel contesto dello

sviluppo di applicazioni, si tratta in genere di un dispositivo BLE con cui si lavora, come un cardiofrequenzimetro.

- **Generic Attribute Profile (GATT):** Il profilo GATT è una specifica generale per l'invio e la ricezione di brevi dati noti come "attributi" su un collegamento BLE.

- **Profili:** Un profilo è una specifica per il funzionamento di un dispositivo in una particolare applicazione. Si noti che un dispositivo può implementare più di un profilo. Ad esempio, un dispositivo può contenere un cardiofrequenzimetro e un rilevatore del livello della batteria.

- **UUID(Universally Unique Identifier):** Identificatore univoco universale, numero a 128 bit utilizzato per identificare servizi, caratteristiche e descrittori.

- **Attribute Protocol (ATT):** Il GATT si basa sul protocollo ATT (Attribute Protocol). Questo viene anche chiamato GATT/ATT. ATT è ottimizzato per funzionare su dispositivi BLE. A tal fine, utilizza il minor numero possibile di byte. Ogni attributo è identificato in modo univoco da un UUID. Gli attributi trasportati da ATT sono formattati come caratteristiche e servizi.

- **Servizio GATT:** una raccolta di caratteristiche (campi di dati) che descrive una caratteristica di un dispositivo, ad esempio il servizio Informazioni sul dispositivo può contenere una caratteristica che rappresenta il numero di serie del dispositivo e un'altra che rappresenta il livello della batteria del dispositivo.

- **Caratteristica GATT:** un'entità contenente dati significativi che possono essere letti o scritti, ad esempio la caratteristica Stringa del numero di serie. Solitamente contiene il valore e una serie di descrittori Gatt.

- **Descrittore GATT:** un attributo che descrive la caratteristica a cui è collegato, ad esempio il descrittore “Client Characteristic Configuration” mostra se il client è attualmente sottoscritto alla notifica riguardo la

modifica del valore di una caratteristica Gatt.

- **Notifiche:** un mezzo per una periferica BLE per notificare alla centrale quando il valore di una caratteristica cambia. Non è necessario che la centrale confermi di aver ricevuto il pacchetto.

3.2. BLUETOOTH LE GATT

Il dispositivo Android che funge da elemento centrale può connettersi a più periferiche (dispositivi BLE esterni) contemporaneamente, ma ogni dispositivo BLE che funge da periferica può generalmente interagire solo con un dispositivo centrale alla volta. Il comportamento più comune è che quando un dispositivo BLE è collegato ad uno smartphone, smette di pubblicizzarsi nella rete come periferica disponibile perché non è più in grado di connettersi.

È utile chiarire la relazione tra un elemento centrale BLE (applicazione Android) e una periferica come ad una relazione **client-server**: il server (periferica) ospita un database **GATT** che fornisce informazioni a cui il client (centrale, app) accede tramite **BLE**.

In sintesi, ciò che si può fare con la comunicazione BLE può essere riassunto in 3 operazioni BLE comuni:

1. **Scrittura**: Il client (app) scrive alcuni byte in una funzione o descrittore sul server (dispositivo BLE). Il firmware del server elabora la scrittura ed esegue alcune operazioni lato server in risposta ad essa. Ad esempio, un termostato intelligente può avere una funzione che modifica la temperatura target quando viene scritta.
2. **Lettura**: Il client (app) legge il valore di una caratteristica o di un descrittore sul server (dispositivo BLE) e lo interpreta secondo un protocollo stabilito in precedenza. Un termostato intelligente può avere una caratteristica il cui valore rappresenta la temperatura target corrente.

3. **Notifica/indicazione**: Il client (app) si abbona a una caratteristica per ricevere notifiche o indicazioni e riceve una notifica dal server quando il valore della caratteristica cambia. Un termostato intelligente può avere una caratteristica notificabile che segnala le variazioni della temperatura ambiente quando viene sottoscritta.

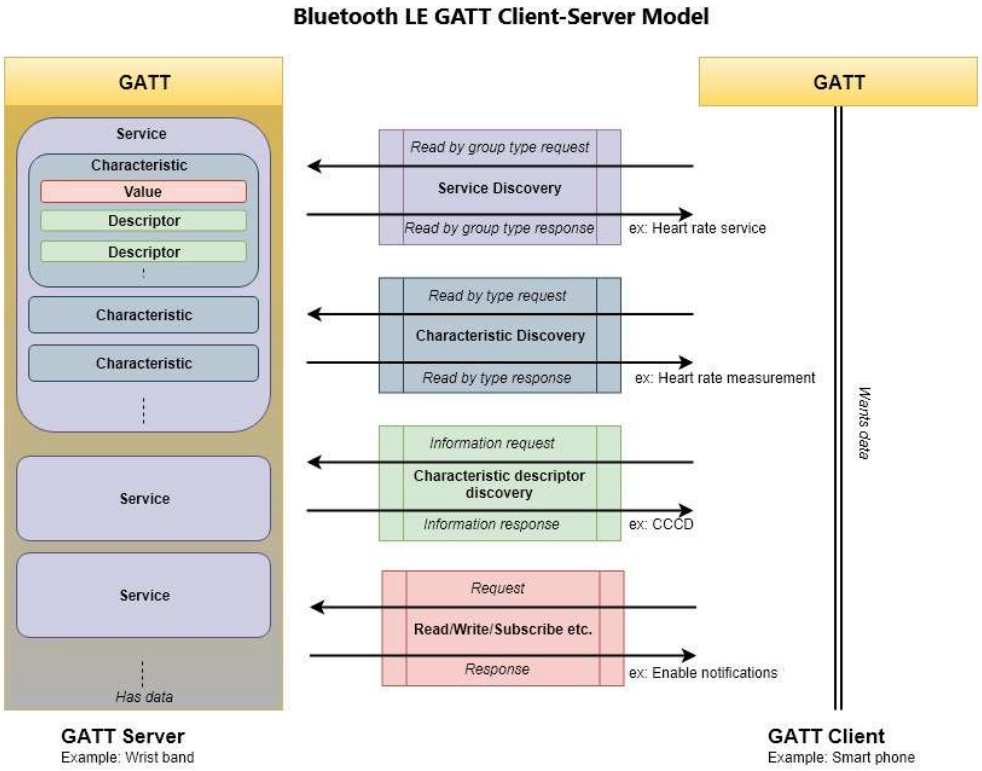


Figura 3-1 Modello Bluetooth LE GATT client-server

Avendo chiarito gli attori presenti in questa architettura, diventa più semplice comprendere lo scambio di dati tra le entità.

Lo smartphone, Gatt client, cerca dati provenienti dal Gatt Server, ovvero il nostro wristband (wearable); per fare ciò incomincia a mandare trame Service Discovery, per cercare i servizi disponibili e offerti dal server; una volta fatto questo il client riceverà dal server una lista di servizi disponibili, come ad esempio Heart Rate service, che è il servizio incaricato a descrivere il valore del battito cardiaco attuale.

Avendo a disposizione questo servizio possiamo chiaramente procedere con la lettura della caratteristica, che conterrà il valore effettivo ricercato e una serie di descrittori. Tralasciando i dettagli di basso livello del descrittore, che conterrà diversi campi a sua volta (**Handle,UUID,Permissions,Value**), per interpretare correttamente il valore della caratteristica bisogna convertire da esadecimale a decimale.

3.3. BLUETOOTH LE SERVICE

Consultando la documentazione ufficiale delle **API BLE**, [9] le stesse che verranno accorpate nelle nuove **API Health Connect,** [10] e provando ad implementare una soluzione in Java, si è riusciti ad ottenere un prototipo che consentiva di leggere diverse caratteristiche provenienti dal bracciale come in figura:

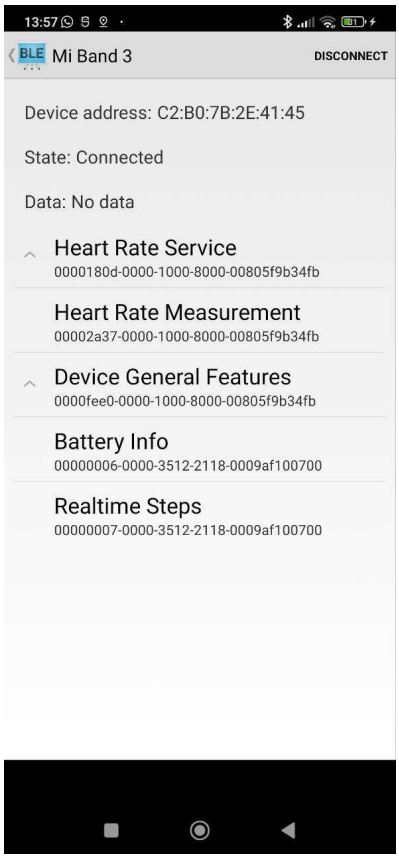


Figura 3-2 Prototipo con soluzione in Java

Si noti come sia stato necessario convertire i valori esadecimali provenienti dalla caratteristica di un servizio, analizzando interamente il payload e convertendolo in valore decimale. In questo esempio si è estrapolato il valore dei passi effettuati attraverso la conversione di una porzione, scartando altri elementi della trama.

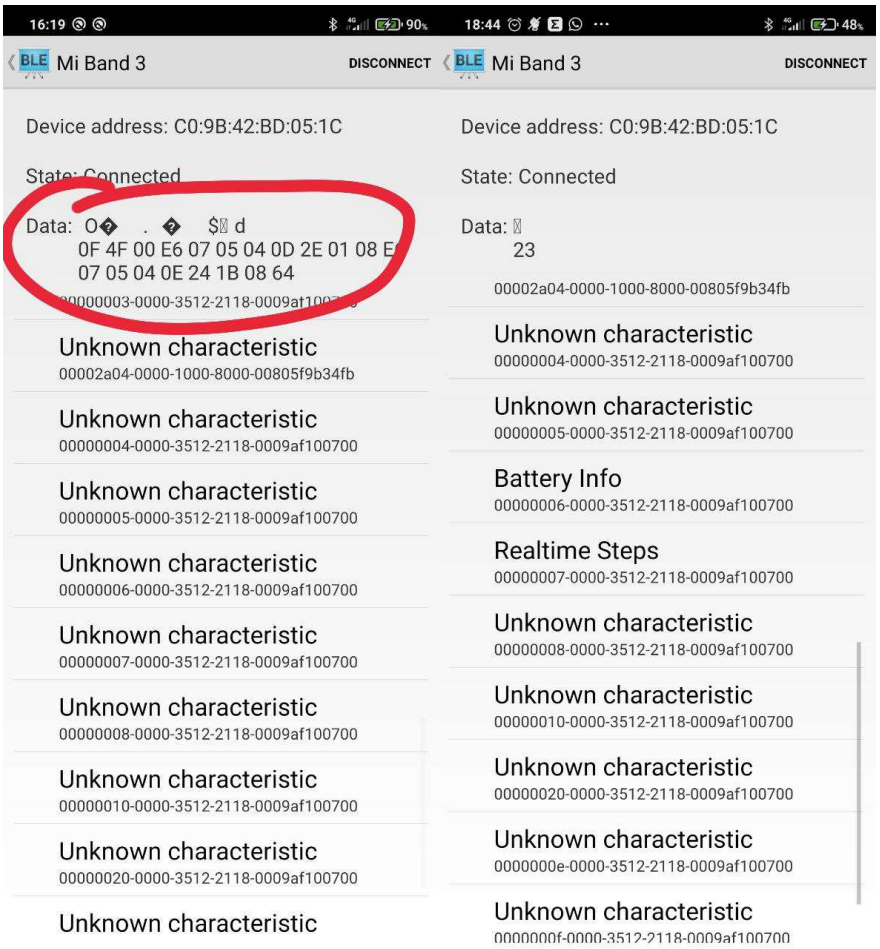


Figura 3-3 Dati caratterista dei passi

Infatti dopo un primo pairing con il braccialetto, è stato possibile ottenere i servizi che esso offriva tramite **BluetoothLeService**, una classe Java che estende la classe astratta **Service.**

```java
/**
 * Service for managing connection and data communication with a GATT server hosted on a
 * given Bluetooth LE device.
 */
public class BluetoothLeService extends Service {
    private final static String TAG = BluetoothLeService.class.getSimpleName();

    private BluetoothManager mBluetoothManager;
    private BluetoothAdapter mBluetoothAdapter;
    private String mBluetoothDeviceAddress;
    private BluetoothGatt mBluetoothGatt;
    private int mConnectionState = STATE_DISCONNECTED;

    private static final int STATE_DISCONNECTED = 0;
    private static final int STATE_CONNECTING = 1;
    private static final int STATE_CONNECTED = 2;

    public final static String ACTION_GATT_CONNECTED =
            "com.example.bluetooth.le.ACTION_GATT_CONNECTED";
    public final static String ACTION_GATT_DISCONNECTED =
            "com.example.bluetooth.le.ACTION_GATT_DISCONNECTED";
    public final static String ACTION_GATT_SERVICES_DISCOVERED =
            "com.example.bluetooth.le.ACTION_GATT_SERVICES_DISCOVERED";
    public final static String ACTION_DATA_AVAILABLE =
            "com.example.bluetooth.le.ACTION_DATA_AVAILABLE";
    public final static String EXTRA_DATA =
            "com.example.bluetooth.le.EXTRA_DATA";
```

Figura 3-4 Classe BLE Service

A titolo di esempio, la classe implementa diverse **callback** per gestire il flusso di informazioni ricevute dal bracciale, come si può evincere dallo snapshot del codice sottostante:

```
private final BluetoothGattCallback mGattCallback = new BluetoothGattCallback() {
    @Override
    public void onConnectionStateChange(BluetoothGatt gatt, int status, int newState) {
        String intentAction;
        if (newState == BluetoothProfile.STATE_CONNECTED) {

            intentAction = ACTION_GATT_CONNECTED;
            mConnectionState = STATE_CONNECTED;
            broadcastUpdate(intentAction);
            Log.i(TAG, msg: "Connected to GATT server.");
            //make auth

            // Attempts to discover services after successful connection.
            Log.i(TAG, msg: "Attempting to start service discovery:" +
                    mBluetoothGatt.discoverServices());

        } else if (newState == BluetoothProfile.STATE_DISCONNECTED) {
            intentAction = ACTION_GATT_DISCONNECTED;
            mConnectionState = STATE_DISCONNECTED;
            Log.i(TAG, msg: "Disconnected from GATT server.");
            broadcastUpdate(intentAction);
        }
    }

    @Override
    public void onServicesDiscovered(BluetoothGatt gatt, int status) {
        if (status == BluetoothGatt.GATT_SUCCESS) {
```

Figura 3-5 Callbacks

Inoltre tale classe comprendeva metodi fondamentali per notificare l'arrivo di nuovi messaggi, come **broadcastUpdate:**

```
private void broadcastUpdate(final BluetoothGattCharacteristic characteristic) {
    final Intent intent = new Intent(BluetoothLeService.ACTION_DATA_AVAILABLE);


    // This is special handling for the Heart Rate Measurement profile.  Data parsing is
    // carried out as per profile specifications:
    // http://developer.bluetooth.org/gatt/characteristics/Pages/CharacteristicViewer.aspx?u=org.bluetooth.characteristic.heart_rate_measurement.xml
    if (UUID_HEART_RATE_MEASUREMENT.equals(characteristic.getUuid())) {
        System.out.println("Hear Rate Entrato");
        int flag = characteristic.getProperties();
        int format = -1;
        if ((flag & 0x01) != 0) {
            format = BluetoothGattCharacteristic.FORMAT_UINT16;
            Log.d(TAG, msg: "Heart rate format UINT16.");
        } else {
            format = BluetoothGattCharacteristic.FORMAT_UINT8;
            Log.d(TAG, msg: "Heart rate format UINT8.");
        }
        final int heartRate = characteristic.getIntValue(format, offset: 1);
        Log.d(TAG, String.format("Received heart rate: %d", heartRate));
        intent.putExtra(EXTRA_DATA, String.valueOf(heartRate));
    } else {
        // For all other profiles, writes the data formatted in HEX.
        final byte[] data = characteristic.getValue();
        int i;
        if (data != null && data.length > 0) {
            final StringBuilder stringBuilder = new StringBuilder(data.length);
            for(byte byteChar : data)
                stringBuilder.append(String.format("%02X ", byteChar));
```

Figura 3-6 Broadcast update

ricevendo come parametro una caratteristica e ne capiva il contenuto attraverso il suo **UUID,** che è **standardizzato ed univoco.**

Una speciale gestione veniva adottata per la caratteristica del battito cardiaco, dato come detto in precedenza, considerato sensibile dalla piattaforma Google.

```
// Handles various events fired by the Service.
// ACTION_GATT_CONNECTED: connected to a GATT server.
// ACTION_GATT_DISCONNECTED: disconnected from a GATT server.
// ACTION_GATT_SERVICES_DISCOVERED: discovered GATT services.
// ACTION_DATA_AVAILABLE: received data from the device.  This can be a result of read
//                        or notification operations.
private final BroadcastReceiver mGattUpdateReceiver = (context, intent) -> {
        final String action = intent.getAction();
        if (BluetoothLeService.ACTION_GATT_CONNECTED.equals(action)) {
            mConnected = true;
            updateConnectionState("Connected");
            //invalidateOptionsMenu();
        } else if (BluetoothLeService.ACTION_GATT_DISCONNECTED.equals(action)) {
            mConnected = false;
            updateConnectionState("Disconnected");
            //invalidateOptionsMenu();
            clearUI();
        } else if (BluetoothLeService.ACTION_GATT_SERVICES_DISCOVERED.equals(action)) {
            // Show all the supported services and characteristics on the user interface.
            displayGattServices(mBluetoothLeService.getSupportedGattServices());
        } else if (BluetoothLeService.ACTION_DATA_AVAILABLE.equals(action)) {
            displayData(intent.getStringExtra(BluetoothLeService.EXTRA_DATA));
        }
};

// If a given GATT characteristic is selected, check for supported features.  This sample
// demonstrates 'Read' and 'Notify' features.  See
// http://d.android.com/reference/android/bluetooth/BluetoothGatt.html for the complete
// list of supported characteristic features.
```

Figura 3-7 Broadcast Receiver

Successivamente un Broadcast Receiver, opportunamente dichiarato dentro una classe di tipo Activity, potrà ricevere gli aggiornamenti scoperti dal servizio GATT.

Nonostante una buona base per sfruttare pienamente il protocollo BLE attraverso le sue API, la documentazione ufficiale riguardo le stesse è sembrata poco esaustiva per gestire situazioni secondarie ma importanti come ad esempio **il Bounding,** che nell'immaginario comune è associato al **Pairing,** ma a livello tecnico non è un sinonimo.

Per **Pairing** si intende lo scambio di chiavi di crittografia temporanee che consente lo scambio di chiavi di crittografia a lungo termine, il **Bounding** si occupa a far accadere ciò.

Quest'ultimo era diventato importante perché si notava che la connessione tra lo smartphone e il bracciale non era mantenuta nel tempo, bisognava implementare esplicitamente il bounding attraverso consenso biometrico dell'utente nel bracciale.

Inoltre si notava che i dati erano disponibili solamente per versioni di bracciali alquanto antiquate (Mi Band 3) mentre per bracciali moderni come il Mi Band 5/6 i dati non erano disponibili.

Con una ricerca sul web ci si è imbattuti in un progetto completamente Open Source con diversi anni di sviluppo alle spalle: **GadgetBridge.** [11]

GadgetBridge è una applicazione che ha come scopo di unificare la gestione delle informazioni fitness provenienti da diversi dispositivi anche di brand diversi, in un unico posto. Cosa molto comoda, considerando che esiste una app proprietaria per ogni famiglia di dispositivo wearable.

Studiando il codice sorgente completamente pubblico e disponibile si è capito che la logica utilizzata era proprio derivata in parte dalle API BLE, ed inoltre supera i problemi riguardo il Bounding.

Rendendoci conto della difficoltà tecnica importante, e del fatto che dispositivi moderni non potevano essere sfruttati con queste API per **via di protezioni a livello firmware,** protetti con chiavi di crittografie ottenibili solamente nel sito del produttore e non automaticamente, si è definitivamente deciso di abbandonare il progetto BLE a fronte di API proprietarie di un prodotto specifico, **Fitbit**. [12]

4. ARCHITETTURA COMPLESSIVA DEL PROGETTO

Il lavoro di ricerca svolto ha confermato che l'obiettivo finale doveva rimanere quello di produrre una applicazione mobile il più semplice possibile, evitando qualsiasi tipo di problema di configurazione a cura del paziente, evitando quindi che gli utilizzatori finali dovessero stressarsi psicologicamente ad usare l'applicazione; al contrario occorreva creare un prodotto che li aiuta o li notifichi del loro percorso di miglioramento quotidiano.

Questo scenario ha indotto a prendere definitivamente la strada per la scelta di **Fitbit**.

4.1. FITBIT

Fitbit fornisce una raccolta di Web API che è sembrata subito la più completa possibile nel parco prodotti attuale; inoltre i dati disponibili sono precisi e dettagliati.

Alcuni esempi degli endpoint presenti:

- **Frequenza respiratoria** : fornisce la media dei respiri al minuto dell'utente durante la notte.
- **Sonno**: fornisce informazioni sulle abitudini di sonno dell'utente.
- **SpO2 :** fornisce i livelli di ossigeno nel sangue dell'utente**.**
- **Punteggio di fitness cardio (VO2 Max):** indica la velocità massima o ottimale alla quale il cuore, i polmoni e i muscoli dell'utente possono utilizzare efficacemente l'ossigeno durante l'esercizio.

- **Serie temporale della frequenza cardiaca:** restituisce i valori della frequenza cardiaca dell'utente e della frequenza cardiaca a riposo. Gli endpoint delle serie temporali possono essere utilizzati per osservare le tendenze.

- **Variabilità della frequenza cardiaca**: fornisce la media quadratica della stanza dei valori successivi delle differenze registrate durante il periodo di sonno dell'utente.

- **Temperatura**: restituisce la temperatura interna e cutanea dell'utente.

Anche in questo caso, per poter sfruttare tali endpoints, è necessario registrarsi come sviluppatore nel loro sito, registrare la propria applicazione e seguire il flusso **OAuth2** che era presente anche nelle API Google, ed infine possedere chiaramente un tracker Fitbit**.**

L'utilizzo e le implementazioni effettive di tali API nel nostro back end sono state oggetto di un lavoro di tesi recente ed inoltre questo argomento sarà alla base di un elaborato di tesi successivo al presente.

Un dispositivo wearable che soddisfi la dotazioni di API prima elencate è ad esempio **Fitbit Inspire2,** fermo restante che tali API sono disponibili per una vasta gamma di prodotti Fitbit.

Caratteristiche Fitbit Inspire2:

- Offre più di 20 profili sportivi pre-configurati.
- È dotato della funzione **SmartTrack**, che rileva automaticamente l'inizio di un esercizio ed identifica di che tipo di attività si tratta per poter iniziare la registrazione dei dati.
- **Non è dotato di GPS integrato**, ma ti permette di utilizzare lo smartphone per far sì che i dati di distanza, ritmo e mappa del percorso siano precisi.
- Grazie al sensore ottico al polso Fitbit Inspire 2 offre dati sul tempo che passi in ogni fascia di frequenza cardiaca e ti permette di programmare

allenamenti in base a queste fasce. Alla fine della sessione otterrai un punteggio del tuo livello di forma cardiovascolare.

- Comprende inoltre funzionalità per il nuoto, dispone infatti di un profilo specializzato a ciò ed è in grado di resistere fino a 50 metri di profondità
- Una funziona interessante è la possibilità di guidare l'utente a intraprendere esercizi di respirazione.
- Durata batteria circa 10 giorni di utilizzo, che può variare a seconda dell'uso

Figura 4-1 Fitbit Inspire2

Fitbit Inspire2 è un bracciale fitness che copre tutte le necessità sia di registrazione dell'attività giornaliera, sia di monitoraggio del sonno di notte. [13]

Durante la giornata, offre funzioni di **conteggio dei passi, distanza percorsa**, **calorie bruciate** e include **avvisi per inattività** se passi troppo tempo senza muoverti. E, ovviamente, è dotato della funzione di rilevamento e **controllo ininterrotti della frequenza cardiaca**.

Quando dormi, registra qualsiasi tipo d'interruzione del sonno e identifica le varie fasi del sonno, tra cui la fase REM.

Diverse informazioni provenienti da queste funzionalità potranno poi essere consultate in forma grafica anche nell'applicazione mobile presentata in questo elaborato.

4.2. SCELTA DEL FRAMEWORK: REACT-NATIVE

La figura riportata nella pagina successiva rappresenta una fotografia dell'architettura complessiv [14] a del progetto HealthApp; l'architettura comprende tre attori principali:

a) **HealthApp:** la nostra applicazione mobile nata dalla collaborazione di due tesisti, tra cui il sottoscritto. Abbiamo scelto di utilizzare come linguaggio di programmazione **JavaScript,** e framework operativi come **React-Native ed Expo,** allo scopo di creare una applicazione **cross-platform** (Android/iOS) e tradurre il codice prodotto automaticamente in **codice nativo** in fase di deployment.

   Infatti le componenti **React wrappano il codice nativo** esistente e interagiscono con le API native tramite il paradigma dichiarativo dell'interfaccia utente di React e JavaScript.

   I principi di funzionamento di React Native sono praticamente identici a quelli di React, tranne per il fatto che React Native non manipola il DOM attraverso il Virtual DOM. Viene eseguito in un processo in background (che interpreta il JavaScript scritto dagli sviluppatori) direttamente sul dispositivo finale e comunica con la piattaforma nativa tramite dati serializzati su un ponte asincrono.L'applicazione si occupa di mostrare dati biometrici, dati riguardanti attività fisica (passi), dati riguardo la frequenza cardiaca a riposo e media, dati derivati dal sonno dei **Pazienti,** oltre una serie di altre funzionalità aggiuntive che verranno descritte nel paragrafo successivo. Ovviamente si occupa anche di inviare al back end proprietario aggiornamenti di stato riguardo tali dati.

b) **HealthWebApp:** applicazione web che raccoglie tutte le informazioni relative ai Dottori dei Pazienti, e anche dei Pazienti stessi. Utile ai Dottori per gestire al meglio i pazienti e monitorarli nel loro percorso salutare.

c) **HealthCore:** Spring Boot Application che rappresenta il back end proprietario dell'intera architettura. A livello implementativo la soluzione scelta è stata utilizzare**, nel layer Data del framework Spring**, **Spring Data JPA.** [15] Come database l'implementazione scelta è stata **H2 Database.** [16]

   Il compito principale del sottoscritto è stato di creare le fondamenta della applicazione mobile **HealthApp**, allo scopo di collegare interamente la nuova applicazione con il back end o core già esistente, cercando di creare una struttura solida e scalabile. Lo studio e implementazione delle componenti grafiche e psicologiche dell'uso dell'App è oggetto di un'altra tesi, che verrà pubblicata dall'altro tesista coinvolto nello sviluppo di questa mobile application.

   HealtCore inoltre gestisce la completa integrazione delle **API Fitbit** nel nostro ecosistema, a tal riguardo per ottenere tali dati sono stati forniti degli endpoint specifici opportunamente elaborati lato mobile, al fine di mostrare correttamente i progressi fisici e salutari del paziente.

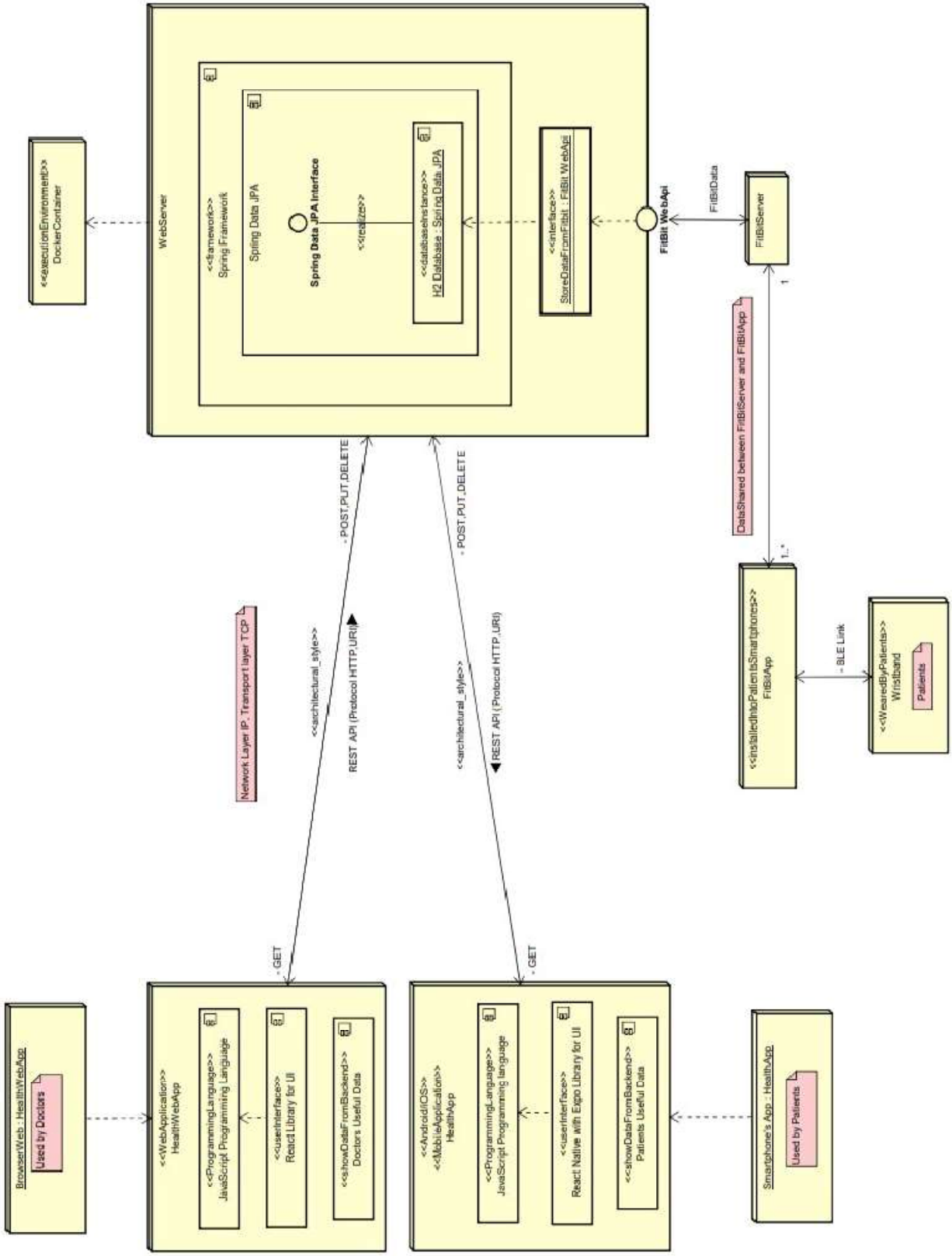


Figura 4-2 Deployment Diagram architettura complessiva progetto HealthApp

Date queste premesse, e date le scelte tecnologiche pre-esistenti alla nascita della applicazione mobile, la scelta di **React-Native, Expo e Javascript** ci è sembrata quella più naturale.

4.3. INTEGRAZIONE CON IL BACK END ESISTENTE

React-native offre, tramite le sue librerie native, tanti elementi chiave che hanno permesso un rapido sviluppo dell'applicazione in costruzione.

Uno di questi è sicuramente la **Navigazione** ed uno dei modi per implementarla è lo **StackNavigator** oggetto fondamentale dell'applicazione, raccoglie dentro tutte le schermate visive utilizzate nell'applicazione chiamate **StackScreen.**

È stato uno dei primi elementi pensati ed implementati, perché permette una maggiore scalabilità e un controllo puntuale delle pagine presenti nell'app.

```
const App = () => {
    return (
        <NavigationContainer>
            <Stack.Navigator initialRouteName='Home'>
                <Stack.Screen
                    name="HomePage_s"
                    component={Tabs}
                    options={{
                        title: "Benvenuto",
                        headerBackTitleVisible: false,
                        headerShown: false,
                    }}
                />

                <Stack.Screen
                    name="Home"
                    component={SignInPage}
                    options={{
                        title: "Benvenuto in HealthApp!",
                        headerShadowVisible: false,
                        headerTitleAlign: "center",
                        headerTitleStyle: {
                            fontSize: 16
                        }
                    }}
                />
                <Stack.Screen name="Impostazioni" component={Impostazioni} options={{
                    headerShadowVisible: false,
                    headerTitleAlign: "center",
                    headerTitleStyle: {
                        fontSize: 16
                    }
                }}/>
                <Stack.Screen name="Profilo" component={Profilo} options={{
                    headerShadowVisible: false,
```

Figura 4-3 Stack Navigator

Grazie a questi oggetti è possibile navigare facilmente in qualsiasi punto logico del codice richiamando l'oggetto **navigation** nativo di react, come ad esempio in questo snippet di codice:

```
await doLogout().then(() =>  navigation.navigate('Home'));
```

Dopo aver fatto un logout, naviga nella home page per il login. La stessa **Home** dichiarata nello **StackScreen** precedente.

Appare evidente il vantaggio tecnico di utilizzare React-Native, la navigazione tra pagine è completamente integrata nel framework.

Un altro aspetto fondamentale è la completa possibilità di creare **Componenti**, sia ovviamente nell'aspetto stilistico, ma soprattutto in quello funzionale.

Queste componenti possono basarsi interamente da elementi base presenti nel linguaggio (come ad esempio **Text,** che è un elemento che renderizza un testo nello schermo) oppure a sua volta essere una elaborazione complessa di più componenti.

In questo modo è possibile costruire veri e propri "mattoni" logici e funzionali della nostra applicazione ("casa").

Il vantaggio principale è che una volta definiti, è possibile chiamarli in qualsiasi momento nel codice, aumentando notevolmente la rapidità di sviluppo e la scalabilità.

```
import React from 'react'
import {TextInput, StyleSheet} from "react-native";
const s = require("../../core/styles");
const  CustomInput = ({value,setValue,placeholder,secureTextEntry,onFocus,error,keyboardType, numericInput}) => {
    return (
                    <TextInput
                        onFocus={onFocus}
                        placeholder={placeholder}
                        selectionColor={"grey"}
                        style={numericInput? s.input_num : styles.input}
                        value={value}
                        onChangeText={setValue}
                        secureTextEntry={secureTextEntry}
                        error={error}
                        keyboardType={keyboardType}
                    />
    )
}

const styles = StyleSheet.create({

    input : {
        width: "100%",
        borderWidth:1,
        marginBottom:20,
        borderRadius: 20,
        padding:10
    }
})
export default CustomInput
```

Figura 4-4 CustomInput

In questo esempio di codice è stato creato un **CustomInput** che personalizza ed estende un **TextInput** presente nativamente in React-Native.

La keyword **export default CustomInput** permette di esportare la visibilità di tale oggetto nel nostro progetto, e quindi il conseguente utilizzo in altre pagine.

```
return (

    <SafeAreaView style={styles.container}>
        <View style={styles.container_header}>
            <Image
                style={styles_.logo}
                source={require('../../../assets/logo.png')}
            />
        </View>
        <View style={styles.root}>
            <CustomInput placeholder="Username"  value={username} setValue={setUsername}
                         onFocus={() => handleError( error: null,  input: 'username')} error={errors.username}  />
            <CustomInput placeholder="Password"  value={password} setValue={setPassword} secureTextEntry={true}
                         onFocus={() => handleError( error: null,  input: 'password')} error={errors.password} />
          <View style={styles.loginButton}>
            <CustomButton  onPress={
                validate
            } button={"first"} text={isLoading ? "Accedi" :
                (loggedUser?"Loggato":<ActivityIndicator/>)
            }
            />
          </View>
        </View>
    </SafeAreaView>
);
```

Figura 4-5 Render del Login

Come in questo esempio, questa è la porzione di codice della pagina del Login dell'applicazione, abbiamo sfruttato la definizione del nostro elemento **CustomInput** per creare un campo di inserimento testo con la nostra logica e funzionalità, e **CustomButton** , un Bottone custom che segue la stessa logica, ridefinisce il Button nativo di React estendendolo di funzionalità adatte ai nostri scopi, che viene richiamato quando necessario.

Figura 4-6 Pagina Login

Questa schermata introduce un altro aspetto chiave. Il collegamento con il nostro back end. Il linguaggio offre **API per il Networking.** [17]

Molte app per dispositivi mobili devono caricare risorse da un URL remoto.

Nel caso volessimo effettuare una richiesta di tipo POST ad un endpoint REST, il codice che realizza questo obiettivo può essere scritto con poche righe, come è evidenziato nella figura che segue, che segnala al server una richiesta di login:

```
async function login(username, password,ip_add){
    const formData  = new FormData();
    formData.append( name: 'username', username);
    formData.append( name: 'password', password);

    console.log(`http://${ip_add}:8080/login`)
     return new Promise ( executor: (resolve, reject) => {
        fetch( input: `http://${ip_add}:8080/login`,  init: {
            method: 'POST',
            body: formData
        })
            .then(async (response : Response ) => {
                const user = response.json()
                if (response.ok) {
                    resolve(user);
                    let cookie = response.headers.get('set-cookie').split( separator: ";")[0].split( separator: "=").pop();
                    console.log("Cookie:" +cookie)
                    try {
                        await SecureStore.setItemAsync(
                             key: 'cookie',
                            cookie
                        );
                    } catch (error) {
                        // Error saving data
                        reject(error)
                    }
                } else {
                    reject(user)
                }
            })
            .catch(() => { reject ( reason: {'error': 'Cannot communicate with the server or cookie error'})
            })
    })
}
```

Figura 4-7 Richiesta di login al back end

dove l'elemento chiave è il metodo **fetch;** esso richiede come parametro un URL arbitrario che rappresenta l'endpoint richiesto e come secondo parametro il metodo con cui richiediamo tali dati.

Il metodo inoltre gestisce per noi tutte le problematiche derivate dal networking e rende trasparenti al programmatore tutte le astrazioni coinvolte (ad esempio l'estrazione dei socket). Sono proprio tali fetch ad aggiornare le viste dell'applicazione

I valori ricevuti dal server, o i valori inoltrati al server, sono stati gestiti attraverso l'utilizzo di **Hooks.**

L'Hook è un moderno paradigma del linguaggio volto a risolvere problemi tipici di programmazione, con una sintassi semplice, come ad esempio l'hook **useState().**

```
const [state, setState] = useState(initialState);
setState(newState);
```

Attraverso il metodo **setState()** riusciamo ad aggiornale lo stato della variabile **state,** il valore aggiornato verrà visualizzato al prossimo **render()** della pagina.

Questo meccanismo è stato utilizzato ampiamente in diverse parti del codice sorgente, un esempio è il seguente:

```
export default function Analisi({navigation}) {


    const [isLoading, setLoading] = useState( initialState: true);
    const [bloodValues,setBloodValues] = useState( initialState: []);

    const getBloodValuesById = () => {
        fetch( input: `http://${global.enrico}:8080/api/patients/${global.id}/bloodAnalysis` ) Promise<Response>
            .then((response : Response ) => response.text()) Promise<string>
            .then((json : string ) =>{
                let bloodAnalysisArray = JSON.parse(json);
                setBloodValues(bloodAnalysisArray)}) Promise<string>
            .catch((error) => {
                console.log(error.message);
                throw error}) Promise<string>
            .finally( onFinally: () => {
                setLoading( value: false)
            });
        }



    useEffect( effect: () => {
        getBloodValuesById();
    }, deps: []);
```

Figura 4-8 Richiesta dati analisi del sangue al back end

In questo pezzo di codice, la funzione **getBloodValuesById** ha il compito di raccogliere le analisi del sangue di un paziente specifico attraverso l'endpoint specificato come parametro.

Nel frammento di codice sopra detto si possono riassumere tutti gli elementi del linguaggio citati precedentemente, come la richiesta di fetch dei dati, ma anche la gestione di tali dati attraverso gli **hook useState,** i quali sono incaricati di

aggiornare la vista sia nel caso ci sia ancora una fase di caricamento della risorsa, sia quando è arrivata la **Promise** e può essere consumata.

Usando l'hook **useEffect**, si comunica a React che il componente coinvolto deve fare qualcosa dopo il rendering; React ricorderà quindi la funzione che hai passato (la chiameremo il nostro "effetto") e la chiamerà in seguito dopo aver eseguito il rendering degli elementi della pagina. Quindi quando accediamo alla pagina relativa alle analisi del sangue, questa pagina si aggiornerà dinamicamente mostrando gli elementi provenienti dal server.

4.4. PROPOSTA DI ARCHITETTURA ALTERNATIVA: GRAPHQL

Durante lo sviluppo dell'applicazione mobile, ho maturato una coscienza più profonda delle potenzialità del linguaggio, e ho incominciato a guardare alternative architetturali che potessero facilitare lo sviluppo di codice senza compromessi.

Sulla base argomentative studiate su un testo di riferimento [18], ho appreso che è possibile creare **un unico ecosistema client-server** con JavaScript perno centrale e altri framework attorno di supporto.

Con le conoscenze che ho avuto modo di maturare durante lo sviluppo dell'App, mi sono chiesto se ci fosse stata la possibilità di costruire un'architettura alternativa per la generazione di API.

Questo obiettivo si è concretizzato con **GraphQL API query language** [19] che è una specifica open source, inizialmente sviluppata da Facebook nel 2012.

Il vantaggio di GraphQL è quello che permette al client **di richiedere in maniera precisa solo i dati che effettivamente servono**, semplificando drasticamente e limitando il numero delle richieste necessarie per ottenere tali risorse.

Questo permette di migliorare notevolmente le prestazioni del server quando dobbiamo inviare dati al client mobile, visto che mandiamo solo i dati di cui si ha effettivamente bisogno.

L'architettura attualmente nel progetto prevede l'utilizzo **di Representational State Transfer (REST) APIs**.

L'architettura REST è stata e continua ad essere il formato dominante per la creazione di API.

Quest'ultime differiscono profondamente da GraphQL (come vedremo), perché si basano principalmente su **URL**.

La robustezza dei tool presenti in GraphQL e il possibile aumento drastico delle prestazioni fanno ritenere quest'ultima la scelta preferita per sistemi moderni.

Esso si basa su pochi ma concreti concetti fondamentali:

- **Schemi:** uno schema è una rappresentazione scritta dei nostri dati e delle interazioni tra essi; richiedendo uno schema GraphQL **forziamo un contratto** con le nostre API; questo perché esse devono ritornare dati e devono avere una forte relazione con lo schema definito, **risultando un linguaggio fortemente tipizzato.**

- **Risolutori (Resolvers):** le nostre API agiscono da risolutori, infatti esse eseguiranno esattamente l'azione che il loro nome indica. Prima dobbiamo definirli nel nostro schema e dopo possiamo implementare la logica nel nostro codice JavaScript. Le nostre api conterranno due tipi di risolutori: **Queries, Mutations.**

- **Queries :** una richiesta query specifica i dati provenienti da una API, nel suo formato desiderato. Ad esempio se definiamo un tipo "Pizza" con la seguente sintassi **type Pizza { id: ID size: String slices: Int }** l'operazione di query ritornerà un oggetto, contenente solo le porzioni di dato effettivamente richieste dall'utente. Una operazione di query non modifica mai dati, accede solamente ad essi.

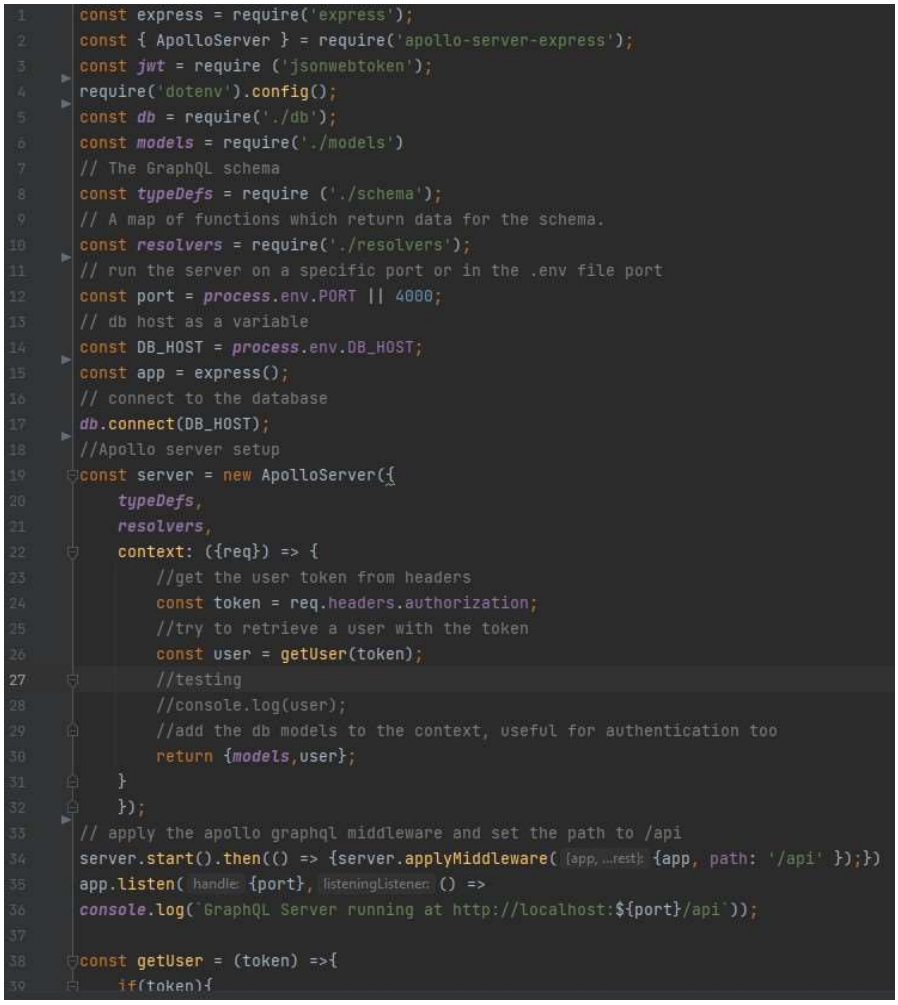
- **Mutations:** usiamo le mutazioni quando vogliamo modificare i dati nella nostra API. Se vogliamo modificare ad esempio il numero di slices nella pizza basta effettuare una operazione di mutazione. La mutazione solitamente restituisce un oggetto, tipicamente il risultato finale dell'azione appena performata.

- **Subscriptions:** come le query, le sottoscrizioni consentono di recuperare dati; a differenza delle query, le sottoscrizioni sono operazioni di lunga durata che possono cambiare il loro risultato nel tempo; possono mantenere una connessione attiva al server GraphQL (più comunemente tramite WebSocket), consentendo al server di inviare aggiornamenti al risultato della sottoscrizione**.**

  Le sottoscrizioni sono utili per notificare al client in tempo reale le modifiche ai dati di back end, come la creazione di un nuovo oggetto o l'aggiornamento di un campo importante.

GraphQL quindi potrà sostituire il nostro server che utilizza il paradigma REST; per fare ciò una possibile implementazione può essere l'utilizzo di **Apollo Server** [20]**,** che è una libreria open source basata su GraphQL ed è compatibile con un grande numero di **server frameworks,** come **Node.js** e **Express**, fornendo inoltre nativamente una interfaccia grafica che aiuta lo sviluppatore a costruire le API.

Esempio di implementazione del server:



```
const express = require('express');
const { ApolloServer } = require('apollo-server-express');
const jwt = require ('jsonwebtoken');
require('dotenv').config();
const db = require('./db');
const models = require('./models')
// The GraphQL schema
const typeDefs = require ('./schema');
// A map of functions which return data for the schema.
const resolvers = require('./resolvers');
// run the server on a specific port or in the .env file port
const port = process.env.PORT || 4000;
// db host as a variable
const DB_HOST = process.env.DB_HOST;
const app = express();
// connect to the database
db.connect(DB_HOST);
//Apollo server setup
const server = new ApolloServer({
    typeDefs,
    resolvers,
    context: ({req}) => {
        //get the user token from headers
        const token = req.headers.authorization;
        //try to retrieve a user with the token
        const user = getUser(token);
        //testing
        //console.log(user);
        //add the db models to the context, useful for authentication too
        return {models,user};
    }
    });
// apply the apollo graphql middleware and set the path to /api
server.start().then(() => {server.applyMiddleware( {app, ...rest} {app, path: '/api' });})
app.listen( handle: {port}, listeningListener: () =>
console.log(`GraphQL Server running at http://localhost:${port}/api`));

const getUser = (token) =>{
    if(token){
```

Figura 4-9 Apollo GraphQL

Con poche righe di codice abbiamo creato un server pronto ad ascoltare richieste di tipo GraphQL.

Da qui è possibile interrogare il server con **queries e mutations.**

Esempio query:

Selezioniamo solamente le caratteristiche che ci interessano, ignorando altri elementi come i dati delle analisi del sangue o le note associate al paziente.

Figura 4-10 Selezione mirata attributi richiesti

E si procede con l'operazione di query:

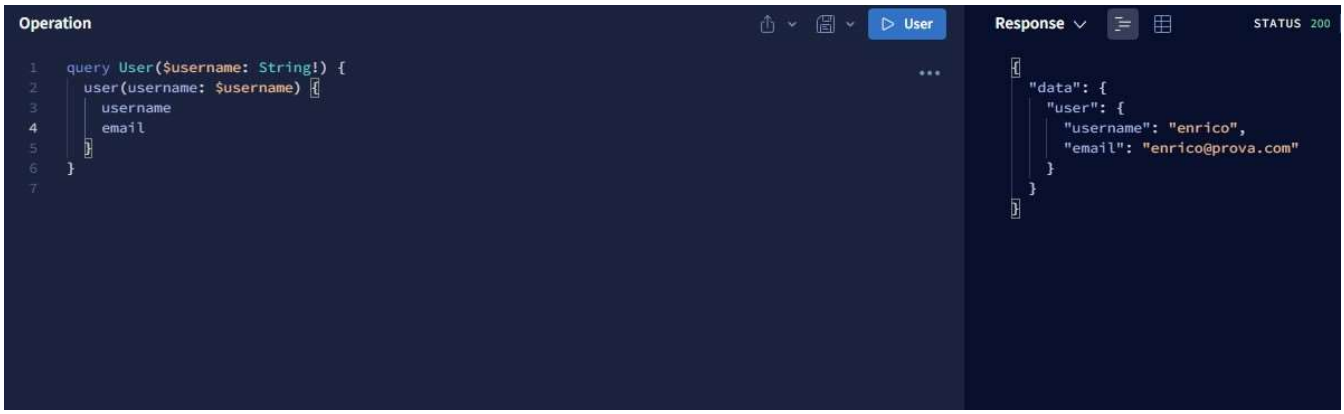


Figura 4-11 Risposta puntuale alla query

La risposta è secca, otteniamo **solo quello che abbiamo richiesto**; questo paradigma è completamente **duale** rispetto **alle fetch REST**.

In quest'ultimo ambito è il programmatore che deve selezionare le parti di dati che interessano, come fatto in questo pezzo di codice estratto dal progetto attuale:

```
Patient.js
class Patient{

    constructor(id =undefined, firstName, lastName, email, ssn, height, gender, birthDate, doctor,experimental, username){
        this.id = id;
        this.ssn = ssn;
        this.height = height;
        this.gender = gender;
        this.birthDate = birthDate;
        this.doctor = doctor;
        this.firstName = firstName;
        this.lastName = lastName;
        this.email = email;
        this.experimental = experimental;
        this.username = username;
    }

    static from(json){
        return new Patient(json.id, json.firstName, json.lastName, json.email, json.ssn, json.height, json.gender,
            json.birthDate, json.doctor.id,json.experimental,json.user.username);
    }
}

export default Patient
```

Figura 4-12 Classe necessaria a filtrare la risposta JSON

Risulta evidente la fatica che deve affrontare il programmatore front-end nel dover mappare solamente i valori che interessano effettivamente dalla risposta JSON ricevuta dall'endpoint REST.

Con le API GraphQL il programmatore front-end può usare la stessa sintassi utilizzata precedentemente nella piattaforma Web Apollo ed ottenere in maniera ottimizzata lo stesso risultato.

Questa differenza si nota maggiormente quando abbiamo bisogno **di dati incapsulati tra loro:** in REST è necessario fare richieste separate e multiple per ottenere l'informazione ricercata; in GraphQL basta richiedere l'elemento incapsulato direttamente nella prima query.

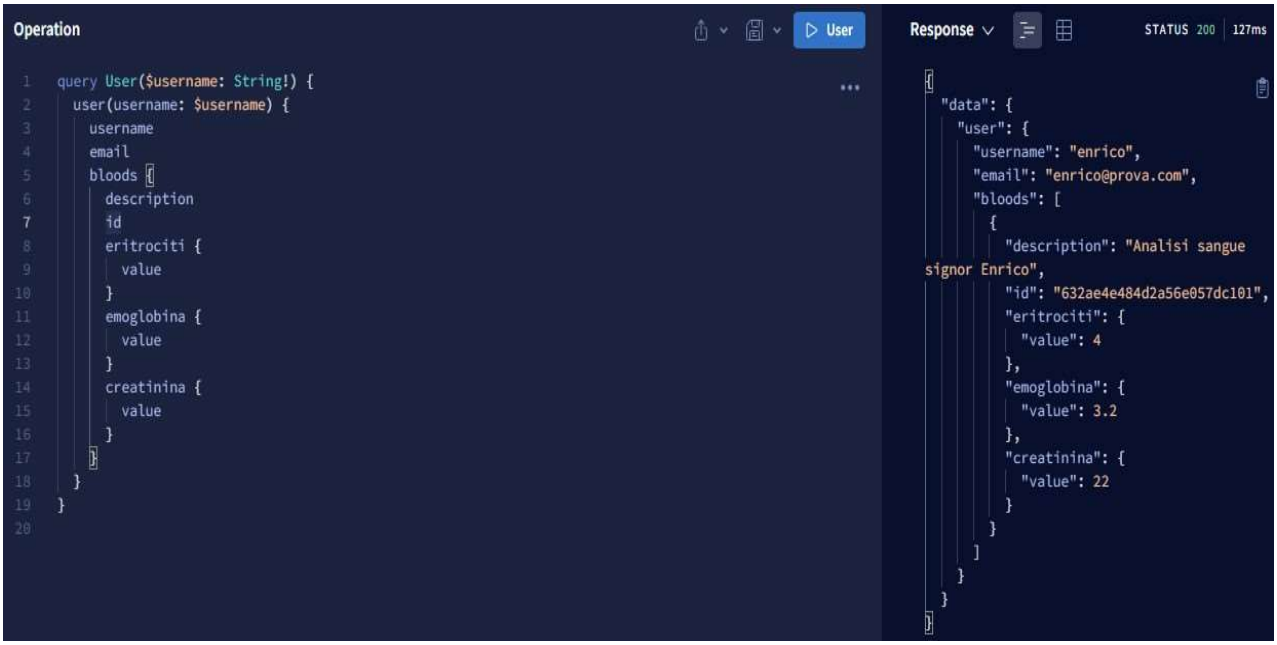


Figura 4-13 Esempio di singola query con dati incapsulati

Abbiamo ottenuto parte delle analisi del sangue dell'utente Enrico, selezionando solo campi specifici richiesti come eritrociti, emoglobina, creatinina; da notare che tutto è stato fatto con un'unica operazione di query, senza richieste multiple per recuperare le informazioni salvate in differenti risorse, evitando anche il caso in cui una richiesta è costretta ad attendere l'esito di un'altra.

Questo esempio mostra il vantaggio tecnico di GraphQL per questo tipo di operazioni, con prestazioni sia lato server che client migliorate.

Inoltre, permette di far lavorare in maniera indipendente teams di sviluppo front-end con quelli di back end, cosa con REST impossibile. Il team di sviluppo front-end dipende da quello back end per poter gestire le API.

Si evita l'overfetching e l'underfetching di risorse che invece può accadere in REST.

Come svantaggi possibili di questa tecnologia c'è la richiesta di **un'elevata curva di apprendimento** iniziale**;** inoltre le richieste di tipo REST possono essere cachate nativamente dallo strato HTTP, mentre GraphQL non prevede nessun meccanismo di caching esplicito e dipende da librerie terze per fare ciò.

5. MOBILE APP

HealthApp 1.0 è pronta: il collegamento con diversi endpoint forniti dal back end sono stati completamente implementati e gestiti attraverso i meccanismi introdotti precedentemente.

Tra le opzioni disponibili al momento troviamo:

- la possibilità di **login** e **logout** di un paziente. (come mostrato in figura 4.5) discernendo fin da subito la tipologia di paziente coinvolto:
  - di tipo **sperimentale:** colori dell'interfaccia marcati per fornire un **feedback visivo** importante al paziente che si accorgerà immediatamente dei suoi risultati parziali raggiunti.
  - di tipo **controllo:** colori dell'interfaccia tenui in quanto tali pazienti avranno difficoltà a capire la semantica dei colori utilizzata, quindi verranno mostrate solamente informazioni trasparenti;

  A seconda della tipologia di paziente coinvolta, il codice renderizzerà le pagine dinamicamente.
- Implementazione di una **HomePage semplificata** per gli utenti:

Figura 5-1 Home page post login

Tale interfaccia comprende **quattro macro aree** principali:

a) **Il monitoraggio del sonno** diviso per giorno, settimana e mese. Questi dati sono presi direttamente dagli endpoint del nostro back end che a sua volta ottiene i dati dagli endpoint Fitbit manipolandoli e semplificandoli.

I dati vengono raccolti attraverso un bracciale Fitbit indossato dal paziente, e rappresentati attraverso grafici semplificati.

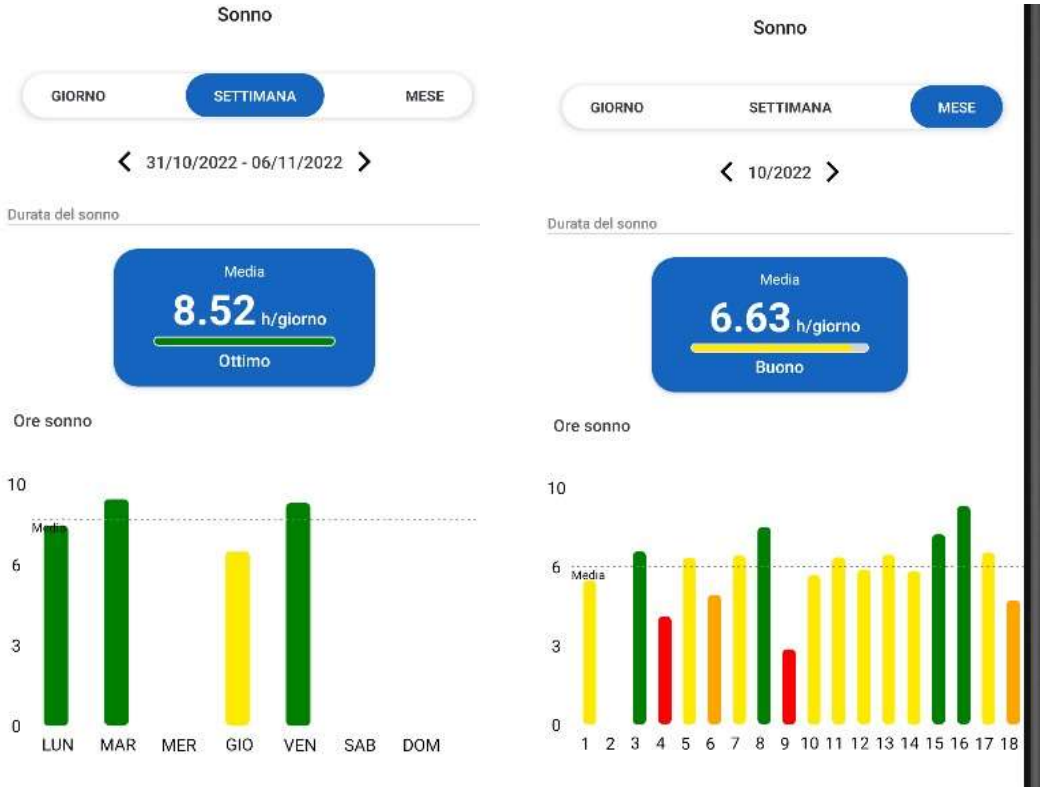


Figura 5-2 Monitoraggio sonno settimanale e mensile

b) Il monitoraggio dell'attività fisica anch'esso suddiviso in giorno, settimana e mese, si occupa di raccogliere tutti i dati relativi **ai passi effettuati** e alle misurazioni automatiche **del battito cardiaco**, anche qui i dati vengono rappresentati in grafici riassuntivi dell'andamento.

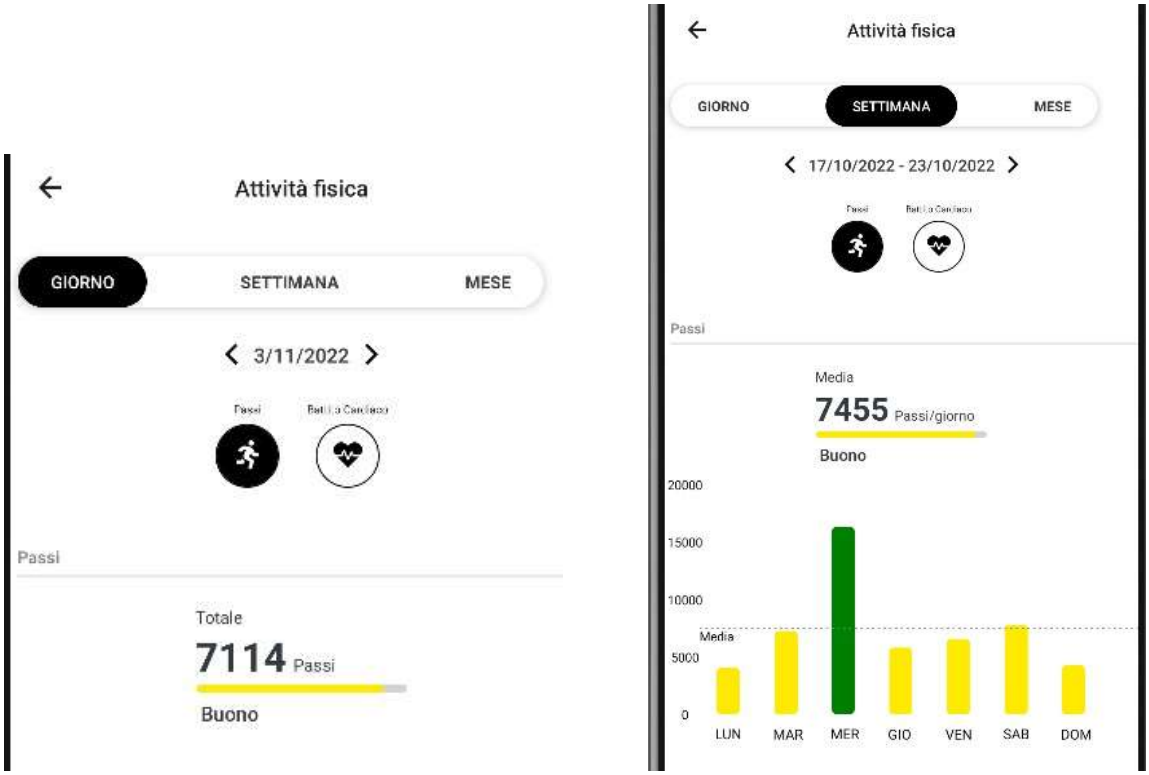


Figura 5-3 Monitoraggio attività fisica e cardiaca

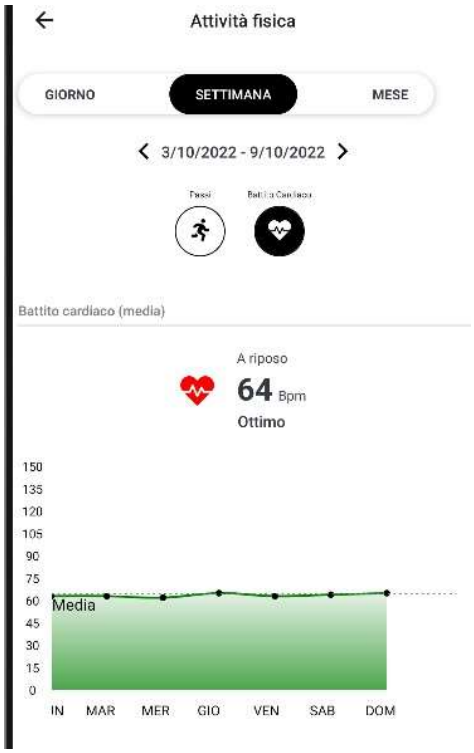


Figura 5-4 Monitoraggio battito cardiaco

In entrambe le sezioni è importante notificare **l'efficacia degli Hook** di React native **useState e useEffect** usati massicciamente per aggiornare il render delle pagine durante la navigazione tra le date e cioè durante lo scorrimento in avanti o indietro riguardo i giorni, settimane e mesi, con conseguente aggiornamento dinamico dei grafici coinvolti.

c) La **sezione Alimentazione,** che contiene attualmente solamente i questionari sottoposti ai pazienti relativi a questa tematica, e che, a seconda delle risposte pervenute fornisce dei feedback utili per suggerire le "**recommendations**" che verranno implementate in una versione successiva dell'applicazione; infatti queste ultime richiedono diversa logica implementata lato back end e che allo stato attuale della produzione di questo elaborato di tesi, non è ancora disponibile.

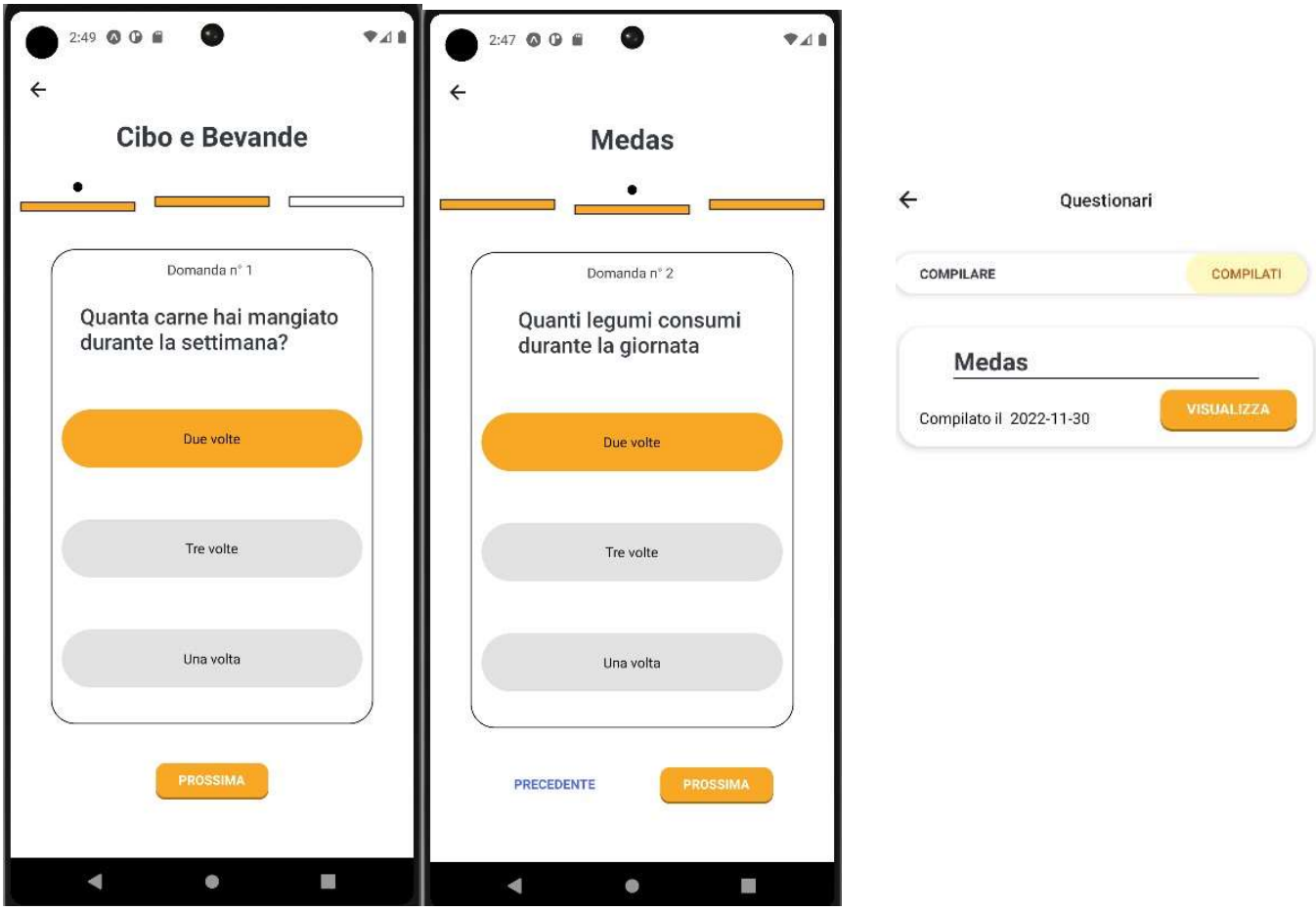


Figura 5-5 Questionari alimentazione e fine compilazione

La sezione **Consigli**, come già detto in precedenza, è ancora da implementare.

Sono inoltre disponibili anche altre sezioni aggiuntive: **inserimento e visualizzazione analisi del sangue.**

Infatti è prevista la possibilità da parte del paziente, in una sezione apposita delle impostazioni, di introdurre personalmente un referto medico riguardo i valori più comuni di un'analisi del sangue effettuata come mostrato in figura seguente:

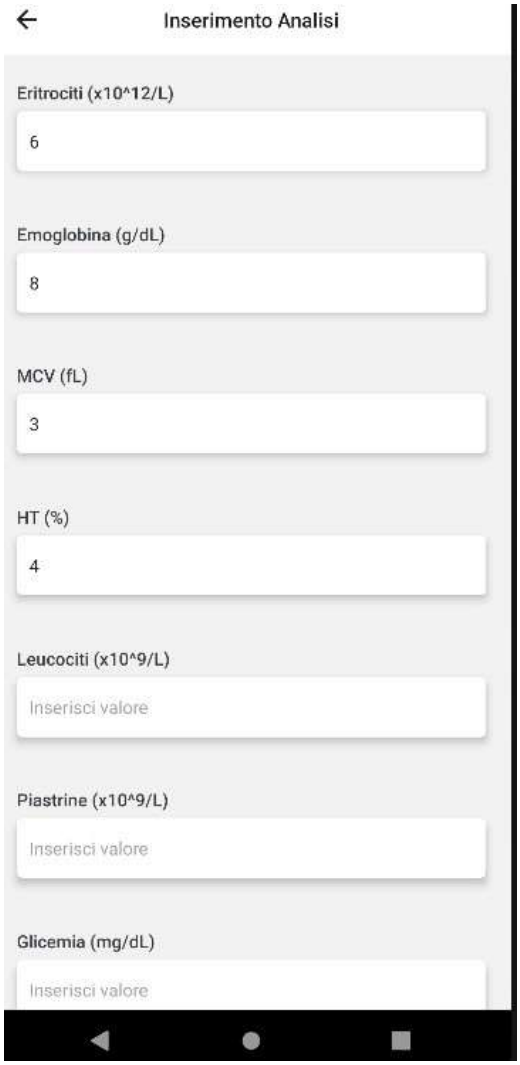


Figura 5-6 Inserimento dati analisi sangue

E' altresì possibile consultare gli storici delle analisi effettuate divisi per data e visualizzarli puntualmente cliccando su *"Visualizza":*

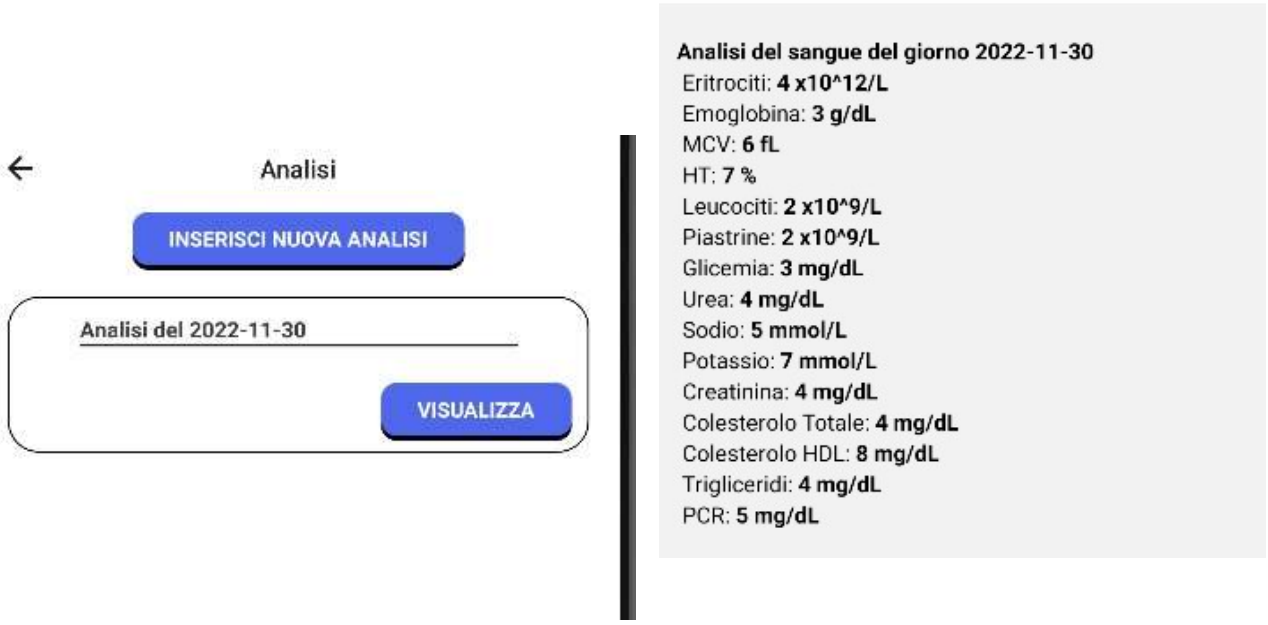


Figura 5-7 Storico dati analisi sangue

Un'altra sezione aggiuntiva implementata è la **Visualizzazione Profilo Utente e grafico Peso,** per visualizzare le variazioni sul peso corporeo a mezzo di un grafico sottostante alle informazioni del profilo, con la possibilità di modificare lo stesso

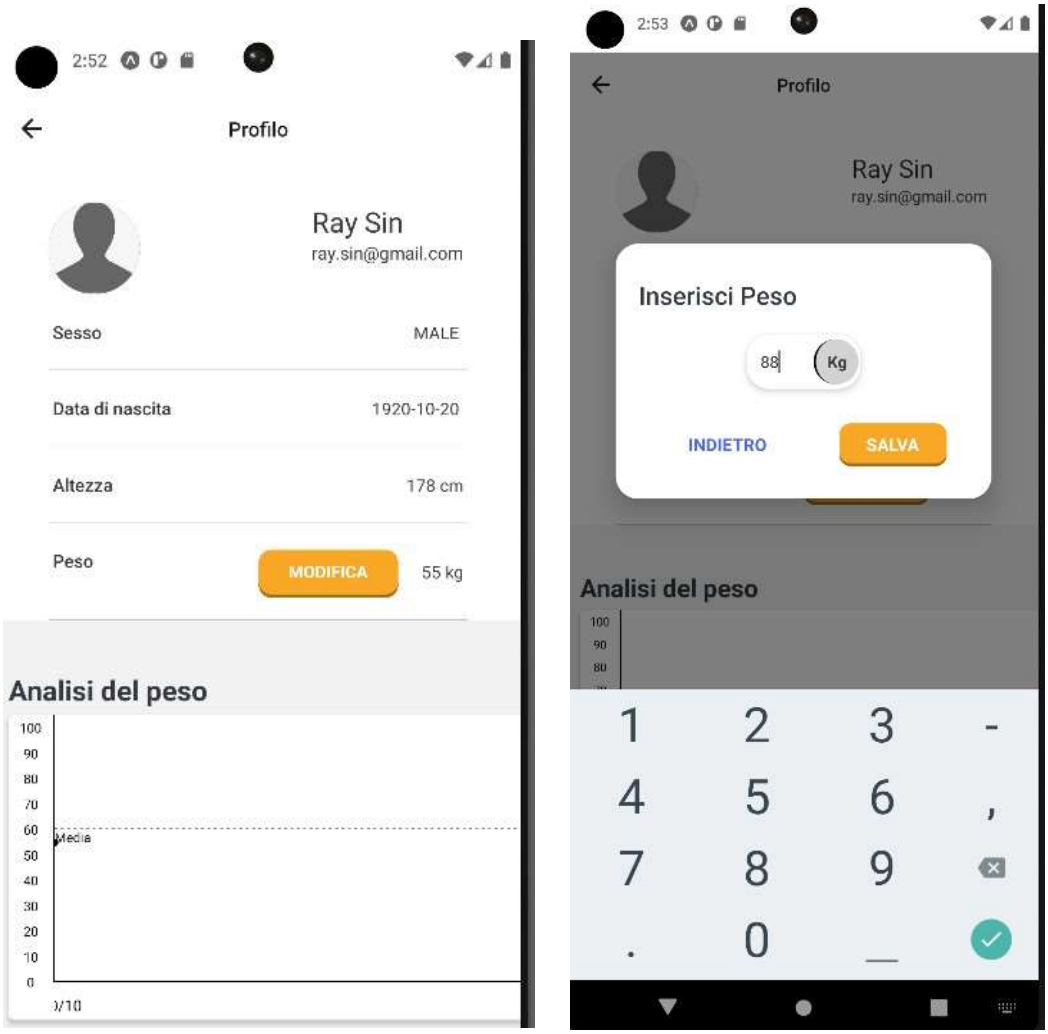


Figura 5-8 Monitoraggio peso corporeo

Una volta aggiornato il peso ad una certa data, un punto aggiuntivo verrà inserito al grafico che si aggiornerà formando così una curva di andamento.

Alla data di stesura di questo elaborato quelle sopra descritte sono le funzionalità principali dell'applicazione.

Come spunti successivi per il progetto c'è sicuramente l'implementazione della sezione consigli al momento in cui sarà approntata lato back end, e l'introduzione di una sezione dedicata all'interno della sezione Alimentazione, con la possibilità di inserimento manuale dei pasti consumati durante la giornata, con possibilità di tenere traccia delle calorie ingerite, dei macronutrienti e dei micronutrienti, rappresentandoli adeguatamente in un grafico semplificato.

6. DEPLOYMENT

L'intera applicazione è stata sviluppata con l'ausilio del framework **Expo,** che offre un tool per la distribuzione molto efficace chiamato **Expo Application Services (EAS)** con una **CLI (command line interface).** Tale comando da terminale permette la compilazione del codice scritto in React-Native nel codice nativo della piattaforma (Android per esempio).

La procedura per fare ciò è semplicissima e consiste nei seguenti steps da eseguire in un terminale:

1) Comando per installare il tool

```
npm install --global expo-cli eas-cli
```

2) Creare un account Expo e loggarsi sempre da terminale:

```
expo login
```

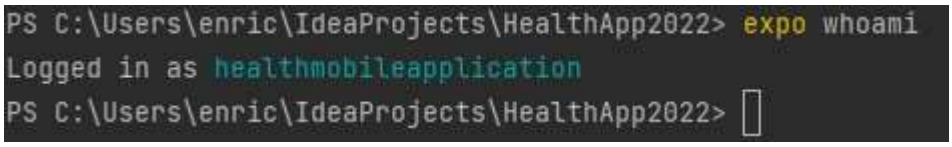


Figura 6-1 Expo login

Se vogliamo creare un file con estensione **.apk** dobbiamo creare un file di configurazione json all'interno del progetto, chiamato **eas.json ,** dichiarando le regole di costruzione, a secondo del caso che si tratti di un apk di produzione o un apk di preview, quindi di sviluppo.

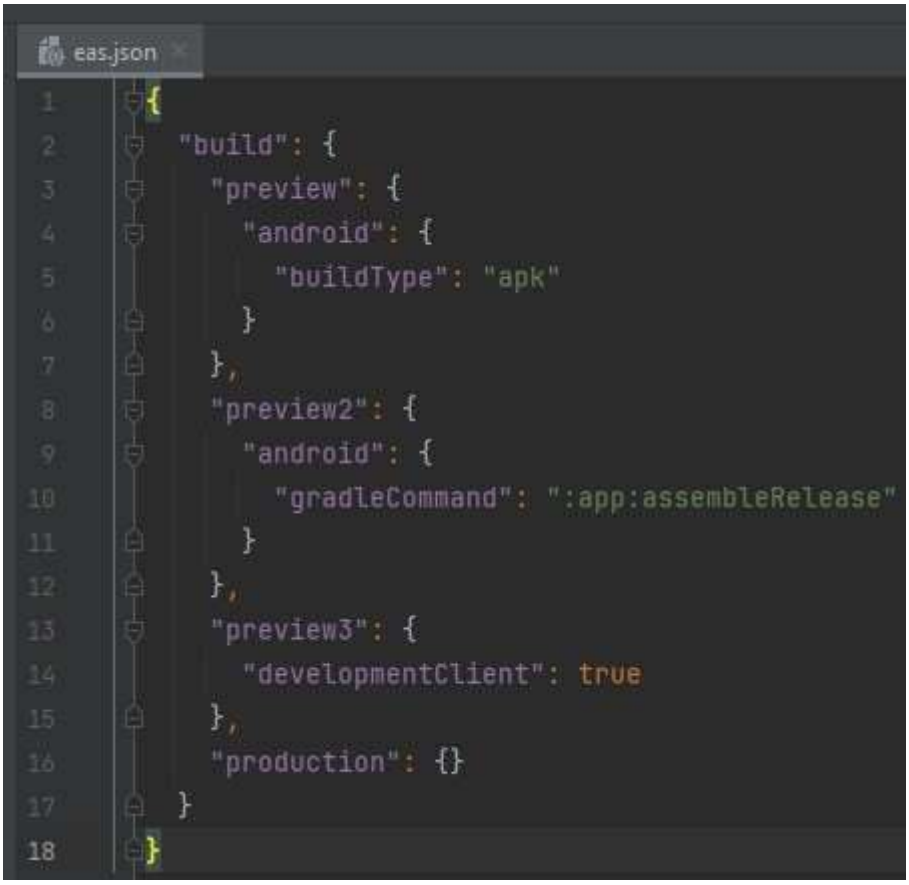


Figura 6-2 Configurazione EAS

3) Una volta seguiti i passi precedenti è possibile lanciare da terminale il comando chiave:

```
eas build -p android --profile preview
```

che farà partire la fase di compilazione.

Sarà possibile seguire lo stato di avanzamento della creazione dell'APK direttamente nella piattaforma web di Expo, dove successivamente si potrà scaricare il file .apk prodotto.

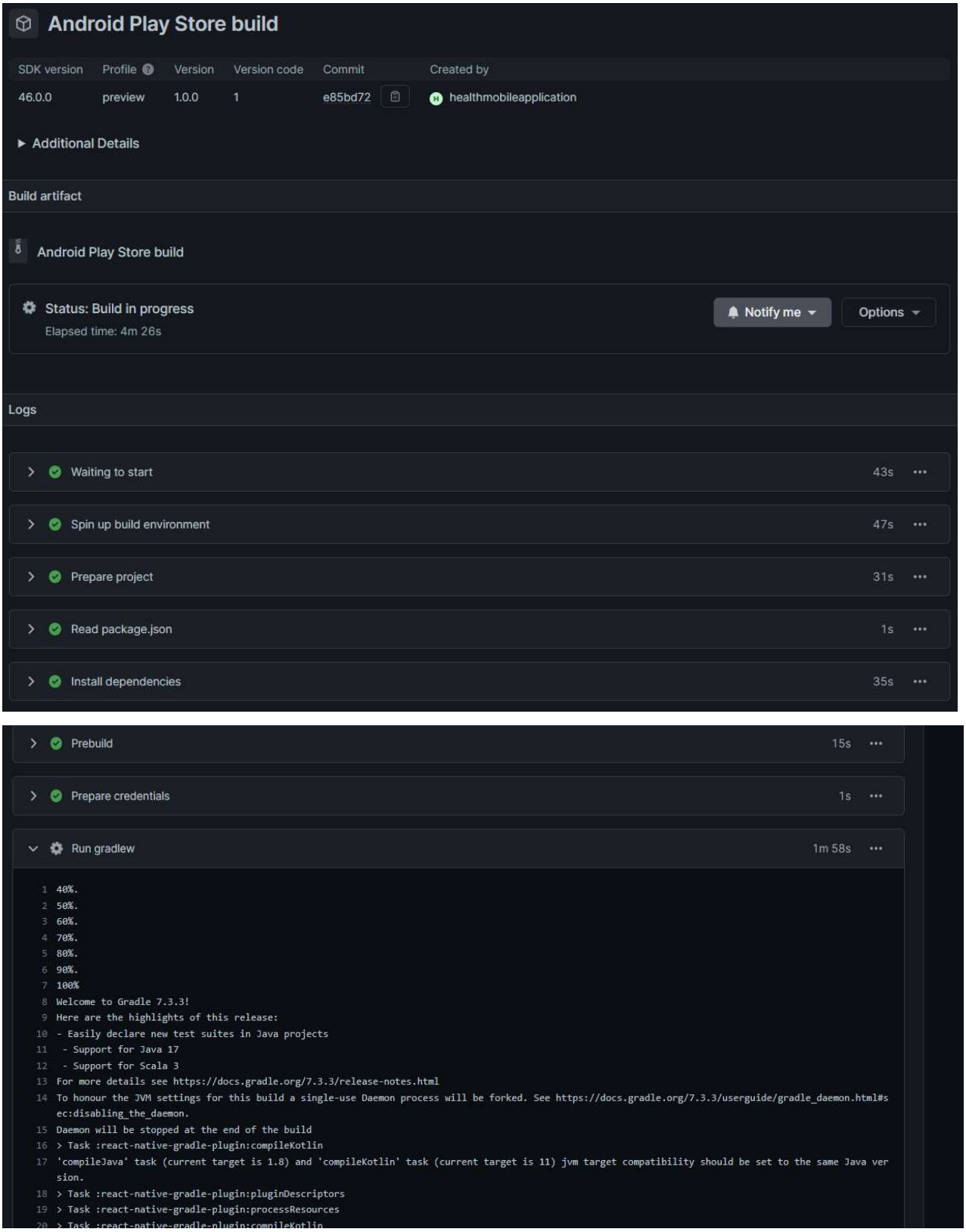


Figura 6-3 Fase di richiamo di gradle

Da notare che durante le fase di compilazione viene richiamato **Gradle** [21] che è un tool di building automatico per diversi linguaggi di programmazione, tra cui **Java e Kotlin,** linguaggi di programmazione utilizzati per la creazione di una applicazione **Android**.

Una volta che la procedura giunge al termine, se nel frattempo non ci sono stati errori di compilazione, sarà possibile scaricare e distribuire l'apk appena generato:

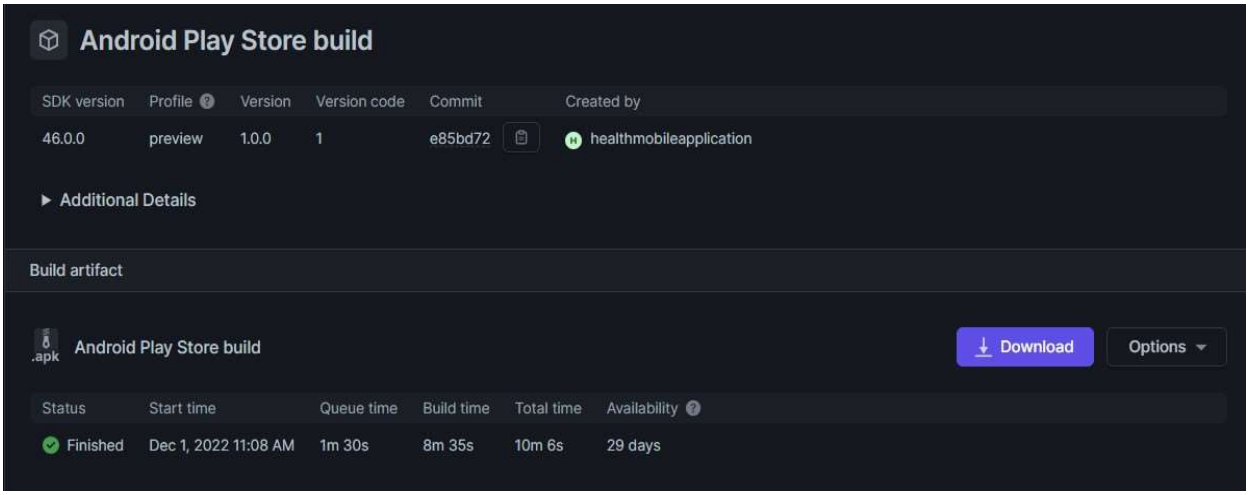


Figura 6-4 Fine compilazione e download apk

CONCLUSIONI

Il percorso per la realizzazione della **versione 1.0** dell'applicazione **HealthApp** ha richiesto necessariamente una fase di corposa ricerca preliminare, per capire cosa effettivamente il mercato attuale offrisse in questo settore tecnologico e quale fosse il metodo più efficace per raggiungere gli scopi voluti dal progetto.

Nella prima fase si è tentato di comunicare direttamente con il wearable con l'obiettivo di essere totalmente indipendenti da **API terze**; però i tentativi descritti in questo elaborato hanno fatto riflettere su quanto potesse essere conveniente questo sforzo tecnico a fronte di un servizio già funzionante e ottimale come quello offerto da **Fitbit**.

Una volta scelto il bracciale adatto e adottando l'ecosistema attorno ad esso con le relative API offerte, si è presentato il momento di implementare la nostra applicazione mobile e legarla con il **back end REST preesistente.**

L'obiettivo iniziale è stato così pienamente raggiunto, pur rimanendo chiaramente ampie possibilità di sviluppi futuri per estendere le funzionalità.

L'elaborato descrive il processo di analisi che ha portato al prodotto attuale; in particolare per **HealthApp 1.0** l'obiettivo principale era quello di legare l'applicazione all'ecosistema, garantendo una certa funzionalità base data dalla visualizzazione dei dati provenienti da Fitbit, e di generare dati utili forniti dai pazienti quali ad esempio questionari, tracciamento del peso corporeo, analisi del sangue. Tutto al fine di avere più feedback o informazioni per creare un prodotto in linea con le aspettative iniziali e il più possibile scalabile in termini di funzionalità offerte.

Questa esperienza non solo mi ha formato dal punto di vista tecnico, imparando linguaggi e frameworks nuovi, come quelli adottati per lo sviluppo dell'app, ma ha fatto crescere in me la consapevolezza che spesso il lavoro che stai portando a termine si deve interfacciare con lavori in corso di sviluppo o implementati da terzi.

CONCLUSIONI

E’ stata pure l’occasione per aumentare in me la consapevolezza di dare valore determinante al lavoro in team, facendolo oggi ritenere uno strumento necessario per raggiungere gli obiettivi prefissati.

Conserverò questa consapevolezza anche su lavori futuri che il percorso professionale mi metterà davanti, avendo interiorizzato la convinzione che la componente collaborativa sia importante in egual misura alle competenze tecniche acquisite nel percorso dei miei studi.

BIBLIOGRAFIA

[1] M. P. F. Luigi, The Path to Longevity: How to Reach 100 with the Health and Stamina of a 40-Year-Old, Sydney: Hardie Grant Books, 2020, p. 318.

[2] Z. API. [Online]. Available: https://docs.zepp.com/docs/reference/app-json/. [Consultato il giorno 07 2022].

[3] S. C. J. A. Lauren Darcey, Introduction to Android Application Development: Android Essentials, Addison-Wesley Professional, 2015.

[4] G. Developers, «Panoramica della piattaforma,» 05 11 2022. [Online]. Available: https://developers.google.com/fit/overview. [Consultato il giorno 10' 11 2022].

[5] O. Organization, «OAuth.net,» 06 2022. [Online]. Available: https://oauth.net/. [Consultato il giorno 06 2022].

[6] Google, «WearOS,» [Online]. Available: https://wearos.google.com/#uniquely-you. [Consultato il giorno 06 2022].

[7] Google. [Online]. Available: https://developers.google.com/fit/android. [Consultato il giorno 06 2022].

[8] Punchthrough. [Online]. Available: https://punchthrough.com/android-ble-guide/. [Consultato il giorno 07 2022].

[9] Google. [Online]. Available: https://developer.android.com/guide/topics/connectivity/bluetooth/ble-overview. [Consultato il giorno 07 2022].

[10] Google. [Online]. Available: https://developer.android.com/guide/health-and-fitness/health-connect. [Consultato il giorno 08 2022].

[11] Freeyourgadget. [Online]. Available: https://github.com/Freeyourgadget/Gadgetbridge. [Consultato il giorno 08 2022].

[12] Fitbit. [Online]. Available: https://dev.fitbit.com/build/reference/web-api/. [Consultato il giorno 08 2022].

[13] T. Cardiofrequenzimetro, «Fitbit Inspire2,» [Online]. Available: https://topcardiofrequenzimetro.com/bracciale-fitbit-inspire-2-specificazioni-recensione-opinioni/. [Consultato il giorno 11 2022].

[14] L. C. Craig Larman, Applicare UML e i pattern, Pearson, 2020.

[15] Spring. [Online]. Available: https://spring.io/projects/spring-data-jpa. [Consultato il giorno 05 2022].

[16] Mozilla. [Online]. Available: https://www.h2database.com/html/main.html. [Consultato il giorno 05 2022].

[17] React. [Online]. Available: https://reactnative.dev/docs/network. [Consultato il giorno 09 2022].

[18] A. D. Scott, JavaScript Everywhere, Oreilly & Associates Inc, 2020, p. 320.

[19] G. FOUNDATION, «GraphQL,» 2012. [Online]. Available: https://graphql.org/. [Consultato il giorno 09 2022].

[20] A. GraphQL. [Online]. Available: https://www.apollographql.com/docs/apollo-server/. [Consultato il giorno 15 10 2022].

[21] Gradle. [Online]. Available: https://gradle.org/. [Consultato il giorno 10 10 2022].