Politecnico di Torino

Politecnico di Torino

Corso di Laurea Magistrale in Ingegneria Informatica
A.a. 2021 / 2022
Sessione di Laurea Ottobre 2022

HealthApp – Applicazione per il monitoraggio del benessere e dello stile di vita dei pazienti

Relatore:
Prof. Maurizio Morisio

Candidato:
Mirco Vigna

Sommario

HealthApp è un programma applicativo nato con l'obiettivo di sostenere uno studio ideato da una squadra di medici dell'ospedale di Verona. Il fine ultimo della sperimentazione è fornire ai partecipanti, tutti anziani reduci da un periodo di ospedalizzazione, una dimostrazione dei benefici ottenuti incentivandoli a mantenere uno stile di vita sano e attivo anche a casa propria.

Lo studio si basa sulla compilazione reiterata nel tempo di alcuni questionari medici, finalizzati al calcolo dell'MPI (Multidimensional Prognostic Index) un indice prognostico di eventi negativi che dà una misurazione della fragilità clinica della persona. Al fine di rendere l'analisi più completa, si è pensato di dotare i pazienti di bracciali "fitness tracker" dai quali vengono recuperati alcuni dati sul benessere della persona, come, ad esempio, il battito cardiaco e la qualità del sonno, e sull'attività fisica svolta.

Il progetto si pone l'obiettivo di informatizzare l'acquisizione e la conservazione di tali dati: un'applicazione web permette ai medici la gestione delle anagrafiche dei pazienti, la creazione dei questionari e la loro compilazione. Inoltre, essa è in grado di raccogliere ed elaborare i dati ottenuti dai tracker e di mostrarli in grafici di riepilogo, insieme con gli indici sull'andamento positivo o negativo del benessere del paziente.

Tali indici sono pensati per incentivare il paziente a seguire le indicazioni dei medici e per dimostrare come un miglioramento di questi nel tempo coincida con un maggior benessere testimoniato dai questionari e dal valore dell'MPI.

Indice

Capitolo 1

Introduzione

Alla base del progetto sperimentale su cui si è lavorato vi sono i concetti trattati nel testo "*The path to Longevity: The Secrets to Living a Long, Happy, Healthy Life"* [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. Egli in questo libro, affronta, uno ad uno, gli aspetti della vita quotidiana che influenzano in maniera più rilevante la salute delle persone, quali nutrizione, esercizio fisico, salute mentale e qualità del riposo, indicando per ognuno di essi le buone norme di comportamento da seguire per massimizzarne gli effetti positivi.

I continui progressi della tecnologia e della scienza, soprattutto in ambito medico, hanno infatti permesso un aumento considerevole dell'aspettativa di vita nel mondo. Questo aspetto, sicuramente positivo, ha però portato alla luce una questione che fino a qualche decennio fa non era tenuta in grande considerazione: in quali condizioni di salute, fisica e mentale, arriveremo all'età senile? La longevità non può più essere un traguardo fine a sé stesso, anzi, rende sempre più importante la gestione dell'evoluzione dello stato di salute con l'aumento dell'età: l'aspettativa di vita dovrebbe crescere di pari passo alla "vita media in buona salute".

Oggigiorno le nuove generazioni dispongono di moltissimi strumenti per informarsi sulle regole e buone prassi connesse al mantenimento di uno stile di vita sano. Negli ultimi anni sono moltiplicati anche i mezzi tecnologici grazie ai quali monitorare costantemente il proprio stato di salute. Le persone più anziane, invece, si trovano disarmate nell'affrontare il problema, per mancanza di competenze nell'utilizzo di queste nuove tecnologie o perché ne ignorano l'esistenza.

Una squadra di medici dell'ospedale di Verona sta lavorando ad uno studio sperimentale proprio per porre rimedio a tale problema. Il progetto, che si svolgerà con un gruppo di pazienti anziani, si pone l'obiettivo di comunicare con loro in maniera semplice ed efficace, trovando strumenti adatti e di facile utilizzo per incentivare le persone ad attuare uno stile di vita sano, nella quotidianità e nelle proprie abitazioni.

Da un punto di vista tecnologico, questo ha richiesto il supporto di un sistema informatico che permetta ai medici di conservare dati relativi allo stato di salute dei pazienti e la loro evoluzione nel tempo, al fine di valutarne l'andamento a seguito di un'adesione, completa o parziale, ai consigli ricevuti sui comportamenti corretti da seguire.

1.1 Il problema della buona longevità

L'Organizzazione Mondiale della Sanità (OMS, o WHO in inglese), nelle sue "Stime Globali sulla Salute" del 2019 [2], ha documentato una crescita della longevità negli ultimi vent'anni a livello globale. Lo sviluppo scientifico ha influenzato positivamente la qualità della vita, permettendo un incremento dell'aspettativa di vita media (ALE – Average Life Expectancy) da 66,8 anni a 73,4 anni, nel periodo compreso tra il 2000 e il 2019.

Lo stesso studio riporta, però, anche un altro importante indice, la HALE (Healthy Average Life Expectancy), cioè l'aspettativa di vita media in piena salute, che incrementa da 58,3 anni a 63,7 anni nelle stesse due decadi. È interessante notare che mentre il primo indice ha mostrato un aumento di 6,6 anni, quest'ultimo è cresciuto ad un ritmo inferiore, salendo nello stesso periodo di soli 5,4 anni.

I dati appena presentati evidenziano il problema di quella che si può definire "buona longevità". Le previsioni dicono che l'HALE è destinato a discostarsi sempre di più dall'indice di longevità, in quanto l'invecchiamento porta ad una maggiore probabilità di incorrere in eventi negativi quali malattie cardiovascolari o degenerative. Diventa, quindi, ancora più importante capire come si possa non solo incrementare i propri anni di vita, ma anche rimanere in salute il più a lungo possibile.

Infatti, fin dalla giovane età diventa necessario preoccuparsi del proprio stile di vita per avere un invecchiamento sano e godere degli ultimi anni di vita in salute. Anche se la medicina moderna ci permette di sopravvivere pur rimanendo affetti da malattie importanti, la qualità della vita che viviamo deve essere il parametro principale da tenere in considerazione.

Negli ultimi anni, l'attenzione verso uno stile di vita sano è diventato un argomento molto discusso e affrontato dai diversi mezzi di comunicazione, sia quelli più moderni (ad esempio, i social network), sia quelli tradizionali (come la televisione), tanto che risulta piuttosto complesso destreggiarsi in una così vasta mole di dati e distinguere informazioni utili da quelle false o infondate. Inoltre, lo scoglio più grande risulta incentivare quegli atteggiamenti positivi che richiedono impegno e producono risultati favorevoli a lungo termine. La psicologia umana, infatti, accoglie favorevolmente ricompense immediate, possibilmente con il minimo sforzo; eppure, è proprio la costanza nel seguire regole che influenzano le principali attività quotidiane che porta miglioramenti nella qualità di vita e una riduzione della probabilità di incorrere in malattie invalidanti in età avanzata. Si tratta di un concetto difficile da assimilare con determinazione senza aiuti esterni.

Inoltre, la situazione si complica quando i soggetti cui viene richiesto di modificare il proprio stile di vita appartengono ad una classe di età geriatrica. Si tratta in questo caso di dover affrontare, non senza resistenza, abitudini ben radicate nell'individuo intervenendo al fine di stimolare un cambiamento positivo, efficace e duraturo.

Se si immaginasse il benessere come un iceberg, il raggiungimento di risultati tangibili e immediati ne rappresenterebbe solo la punta. Il segreto della dedizione ad uno stile di vita sano risiede nella parte immersa dello stesso blocco di ghiaccio che si raggiunge solamente con la perseveranza nel mettere in pratica le suddette buone abitudini. Seppur con minore evidenza nel breve termine, è proprio questo mantenimento a preservare il nostro organismo da rischi maggiori di morbilità future. Immergersi nelle profondità occupate dall'iceberg richiede coraggio e un grande senso di responsabilità verso sé stessi. Per tale ragione, una guida è indispensabile.

1.1.1 Gli strumenti attuali

L'avvento degli smartphone ha reso possibile al grande pubblico il monitoraggio della propria attività fisica, ottenendo dei *feedback* immediati sul proprio livello di fitness. Avere in tasca un computer in miniatura ha permesso l'accesso a informazioni illimitate e personalizzate, nonché fruibili nell'immediato, anche dalla palestra o durante una corsa. Inoltre, i molteplici sensori (accelerometro, GPS, etc.) di cui sono dotati i moderni cellulari, hanno dato modo di sviluppare un'immensa gamma di applicazioni dedicate al fitness. Oggi, ad esempio, ciascuno di noi, in totale autonomia e senza l'intervento di nessun esperto del settore, ha la possibilità di:

- avere accesso a librerie e collezioni di esercizi fisici, per perdere peso, per aumentare la massa muscolare o per altri fini specifici;
- tenere traccia delle proprie sessioni di corsa, dei percorsi effettuati, delle distanze coperte e dei tempi impiegati, potendo registrare in modo oggettivo e istantaneo i miglioramenti personali;
- mantenere un *food diary* per monitorare quantità, qualità e varietà della propria nutrizione;
- ricevere dei promemoria, durante tutta la giornata, per ricordarsi di assumere acqua in quantità sufficiente;
- essere introdotti nel mondo della meditazione, con esercizi guidati per comprendere l'importanza della salute mentale.

Negli anni le app hanno subito sviluppi, diventando sempre più intuitive e facilmente fruibili. Inoltre, sono stati introdotti metodi per incentivarne l'utilizzo, con tecniche di *gamification* e con la creazione di *community* dedicate. Nel primo caso, si possono ottenere *badge* e riconoscimenti personalizzati, ricompense digitali e, in casi più rari, premi materiali. Sfruttando il lato "social", invece, si ha un incentivo ulteriore a raggiungere alcuni obiettivi, che possono essere poi condivisi sui vari social network con gruppi di persone che seguono gli stessi allenamenti e/o che partecipano agli stessi eventi sportivi.

Queste app si sono dimostrate un ottimo strumento per aumentare la consapevolezza di milioni di persone sul benessere e sul proprio stato di salute. Non sono, però, prive di limiti: nella maggior parte dei casi, esse richiedono una partecipazione attiva e proattiva nella consultazione dei dati e nell'attivazione delle funzionalità. Le persone devono ritagliare del tempo dalla loro giornata da dedicare al loro utilizzo. Inoltre, richiedono di portare sempre con sé il cellulare, strumento che non può essere utilizzato per monitorare alcuni tipi di sport.
Per ovviare al problema sono stati, quindi, realizzati strumenti passivi dedicati al benessere. I "*wearable*", dispositivi indossabili intelligenti, quali orologi, bracciali, anelli ed occhiali, forniscono buona parte delle funzionalità in modo continuo e silente, senza necessità di interazione da parte dell'utilizzatore. Il mercato propone una grande varietà di dispositivi di questo genere, che va dai più economici, acquistabili con qualche decina di euro, a strumenti professionali, con caratteristiche che permettono di impiegarli anche in ambito medico e i cui costi sono decisamente più elevati.

I principali parametri di differenziazione tra i dispositivi sono la varietà e, soprattutto, l'accuratezza dei dati generati; quest'ultima, a sua volta, dipende da due fattori: la sensoristica a bordo e gli algoritmi di analisi dei dati grezzi.

Tra i diversi sensori che i dispositivi indossabili possono avere in dotazione, i più comuni sono:

- l'accelerometro, per misurare le accelerazioni subite dal dispositivo, di norma a seguito di un movimento dell'utilizzatore;
- il giroscopio, che misura la velocità angolare per determinare l'orientamento del dispositivo;
- l'altimetro, che determina l'altitudine a cui ci si trova;
- il termometro, per misurare la temperatura dell'utilizzatore o dell'ambiente circostante;
- il GPS, per rilevare le coordinate dell'utilizzatore, utile soprattutto quando non è previsto l'utilizzo di uno smartphone;
- l'elettrocardiogramma (ECG), per misurare l'attività e la frequenza cardiaca;

- il saturimetro, che rileva l'ossigenazione del sangue;
- l'elettrodermografo (EDG), misura la conduttività elettrica della pelle per calcolare il livello di sudorazione.

La qualità dei sensori determina l'accuratezza delle misurazioni, quindi l'affidabilità delle informazioni prodotte. La loro varietà, invece, aumenta il potenziale di efficacia degli algoritmi di analisi dei dati. Il successo imprenditoriale dei produttori di *wearable device* dipende in larga parte dalla bontà degli algoritmi utilizzati per analizzare i dati provenienti dai sensori e per derivarne di più complessi. Le informazioni relative al sonno sono un perfetto esempio di dati ricavati a partire da misure grezze sul movimento, sulla temperatura e sulla frequenza cardiaca. Le funzionalità più comuni offerte dai *wearable* riguardano:

- 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 combinazione 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 *wearable* sono strumenti decisamente utili per monitorare il proprio stile di vita e non comportano alcuno sforzo di utilizzo. Inoltre, la tecnologia avanza e le analisi dei dati diventano sempre più accurate. Il prezzo dei dispositivi di livello base è contenuto, nell'ordine delle decine di euro, ma questi offrono comunque funzionalità sufficienti per una buona percentuale di

persone. Per tali ragioni, il loro utilizzo è largamente diffuso, in particolare tra le nuove generazioni.

L'adozione dei *wearable* nelle fasce di popolazione in età avanzata, invece, è decisamente più limitata. Anche se il loro impiego non richiede competenze ed è perfettamente adatto anche a persone prive di conoscenza in campo digitale, la criticità maggiore riguarda la lettura dei dati. Infatti, la mole di dati prodotti necessita di essere mostrata attraverso interfacce grafiche complesse a causa della presenza di numerosi elementi sullo schermo: grafici, dati numerici e unità di misura. Pur essendo perfettamente funzionali per un utente medio, diventano poco leggibili e confusionarie per persone non abituate ad interfacciarsi con le schermate digitali.

1.2 Lo studio

Il progetto applicativo nasce con lo scopo di sostenere uno studio sperimentale medico ideato da un gruppo di medici dell'ospedale di Verona, rappresentati dal Dr. Vincenzo Di Francesco, direttore dell'Unità Operativa Complessa Geriatria III Borgo Trento, in collaborazione con il Sindacato Pensionati Anziani (SPI) della Confederazione Generale Italiana del Lavoro (CGIL). Un centinaio di pazienti volontari in età avanzata, recentemente ospedalizzati a causa di un trauma o di un intervento chirurgico, a seguito della dimissione dal monitoraggio ospedaliero per continuare il percorso di recupero in autonomia nelle proprie abitazioni, verranno sottoposti ad un'indagine sulla loro salute psicofisica per un periodo di sei mesi. Il monitoraggio avverrà tramite l'esecuzione di check-up, la sottomissione di questionari periodici, sia medici sia informativi, e la rilevazione di alcuni parametri indicativi dello stile di vita.

I check-up verranno eseguiti all'inizio e alla fine del periodo di studio per determinare lo stato clinico di partenza del paziente e la sua evoluzione.
Alcuni dei questionari sottoposti avranno valenza puramente informativa e da questi verranno ricavati indicatori riguardanti dati la cui rilevazione è difficilmente automatizzabile. Un esempio sono i test nutrizionali che forniscono indicazioni su eventuali discostamenti da una sana alimentazione. Altri questionari, invece, avranno valenza medica validata. Merita un cenno il

più importante dello studio, chiamato Multidimensional Prognostic Index (MPI). Si tratta di un indice prognostico di eventi negativi, quali mortalità, ospedalizzazione, istituzionalizzazione, basato sull'esecuzione di una valutazione multidimensionale mediante questionario.

Diverse versioni di MPI sono state sviluppate e verificate in differenti contesti clinici (ospedale, RSA, medicina generale, popolazione generale) e in numerosi paesi in Europa, America, Asia e Australia. La versione utilizzata nello studio è quella sviluppata e validata inizialmente nell'anziano ospedalizzato [3].

L'MPI è calcolato grazie ad un algoritmo matematico che include informazioni relative ad otto domini, tra cui le attività basali e strumentali della vita quotidiana, lo stato cognitivo, lo stato nutrizionale, il rischio di lesioni da decubito o motilità, la multimorbilità, la politerapia e lo stato abitativo del soggetto (da solo, in RSA o in famiglia). L'output dell'MPI può essere espresso sia come indice numerico continuo da 0 (assenza di rischio) a 1 (massimo rischio) sia in tre gradi di rischio di mortalità: basso (MPI-1), moderato (MPI-2) o severo (MPI-3) secondo opportuni *cut-off*.

Numerosi studi clinici nazionali ed internazionali condotti in oltre 50.000 soggetti anziani hanno dimostrato che l'MPI è molto accurato e calibrato nel predire la mortalità, ed altri eventi negativi nell'anziano quali il rischio e la durata di ricovero in ospedale, in RSA o la necessità delle cure domiciliari, il rischio di caduta e di sviluppare depressione. L'MPI inoltre è molto sensibile alle variazioni nel tempo della fragilità multidimensionale dell'anziano, sia durante un ricovero in ospedale sia in ambito ambulatoriale, esprimendo in maniera numerica e complessiva la fragilità multidimensionale dell'anziano [4]. L'MPI è risultato molto preciso in termini di validità, affidabilità e fattibilità per definire l'anziano fragile nella pratica clinica [5] ed è per questo che oggi l'MPI è uno degli strumenti più comunemente utilizzati per identificare e misurare la fragilità dell'anziano in ospedale e nell'ambito delle cure primarie [6].

Infine, ogni paziente sarà dotato di un bracciale smart per il fitness, scelto tra i vari disponibili nel mercato, con il quale verranno raccolti i dati riguardanti l'attività fisica giornaliera, la frequenza cardiaca, a riposo e sotto sforzo, la quantità e la qualità del sonno. I dati verranno immagazzinati ed elaborati in modo da offrire semplici indicatori di immediata lettura per valutarne l'andamento corrente rispetto ai parametri ottimali di uno stile di vita sano.

Tutte le informazioni relative ai soggetti saranno consultabili nell'applicativo web e, in un secondo momento grazie a sviluppi futuri, su un'applicazione mobile.

I volontari sottoposti allo studio saranno suddivisi in egual misura in due sottogruppi: uno sperimentale e uno di controllo. La raccolta dati avverrà per entrambi i gruppi nel medesimo modo. Le differenze tra il gruppo sperimentale e il gruppo di controllo riguarderanno la quantità di *feedback* forniti. I primi riceveranno riscontri frequenti sull'andamento dei dati, potranno vedere gli indicatori del proprio stile di vita espressi in modo semplice e con un formato a colori intuitivo, con gradazioni dal verde (indicatore positivo) al rosso (indicatore negativo). Gli sviluppi futuri dell'applicazione mobile da consegnare ai pazienti si concentreranno, infatti, sui riscontri appena citati. Essi prevederanno inoltre, promemoria giornalieri sotto forma di notifiche sugli smartphone, al fine di ricordare loro i buoni comportamenti da seguire per migliorare la propria salute.
Il gruppo di controllo, invece, potrà prendere visione dei propri dati ma non riceverà alcun tipo di riscontro o promemoria.

Il fine è dimostrare che, se correttamente stimolate ad assumere comportamenti riconosciuti sani, le persone più anziane, in genere riluttanti al cambiamento, possano beneficiare di un recupero più veloce e proficuo. La speranza, in futuro, sarà ridurre i tempi di ospedalizzazione e permettere il recupero clinico in ambienti più confortevoli.

1.3 L'architettura applicativa

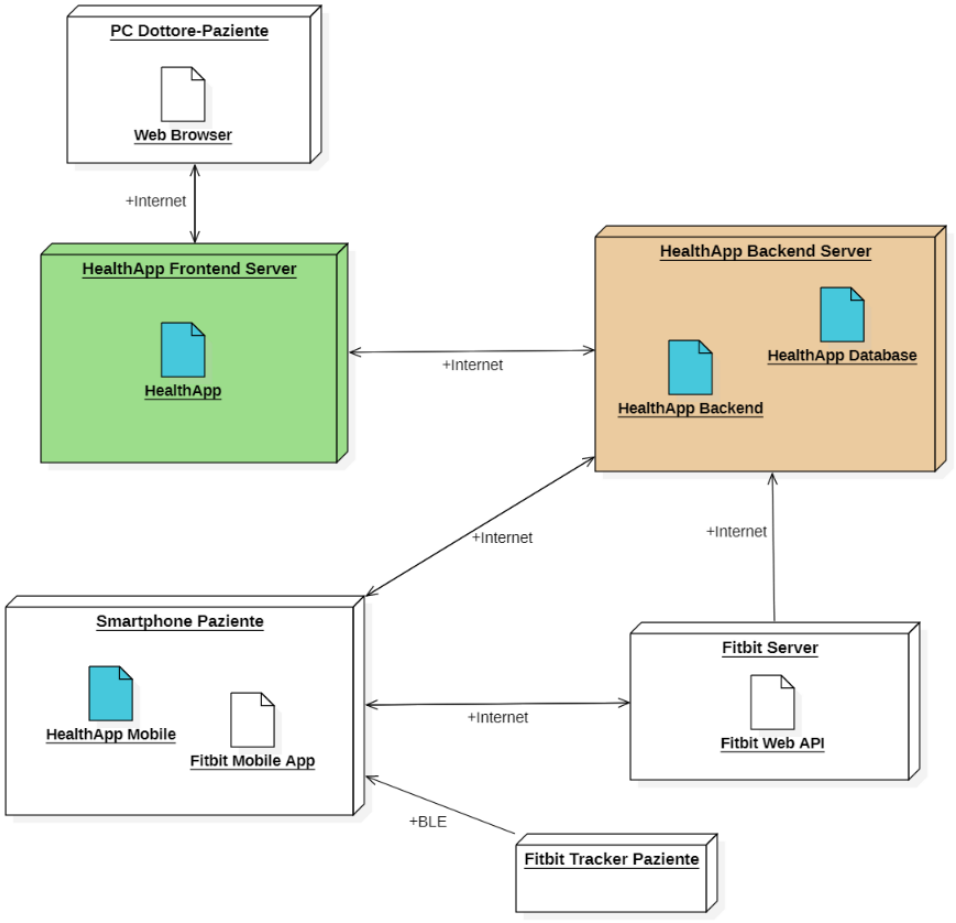
HealthApp è il programma applicativo sviluppato a supporto dello studio sperimentale. Si tratta di un'applicazione web, eseguibile mediante web browser, che si pone l'obiettivo di informatizzare tutti i dati riguardanti lo studio e consente di effettuare le operazioni basilari CRUD (*create*, *read*, *update*, *delete*) per la gestione persistente dei dati relativi ai tre principali soggetti coinvolti: dottori, pazienti, questionari.

Inoltre, per ogni paziente, il quale viene dotato di un dispositivo *wearable*, si persistono dati sul benessere come l'attività fisica giornaliera, la qualità/quantità del sonno e l'andamento medio

della frequenza cardiaca. Viene mostrata una *dashboard* con indicatori sullo stile di vita ottenuti dall’elaborazione di tali dati. Una routine di sincronizzazione dei dati viene eseguita periodicamente per aggiornare il database con i bracciali e garantire la freschezza dei dati sull’applicazione web.
L’applicazione è accessibile sia dai pazienti sia dai dottori, con diversi livelli di autorizzazione.

Nella figura 1.1 è rappresentato il diagramma architetturale dell’applicazione. HealthApp è accessibile sia ai pazienti sia ai dottori attraverso un web browser. *Il front-end* interagisce mediante le REST API esposte dal *back-end* con il database, recuperando dati da mostrare o scrivendone di nuovi. *Front-end* e *back-end* sono eseguite su *server* separati, pertanto, comunicano attraverso il protocollo HTTP.

Figura 1.1



Nell'immediato futuro è previsto lo sviluppo di un'applicazione HealthApp per smartphone ad uso esclusivo dei pazienti, per implementare le funzionalità di promemoria e ricompense destinate, in particolare, al gruppo sperimentale.

1.3.1 La connessione con il tracker

Una delle criticità più importanti affrontate nel progetto riguarda il metodo di acquisizione dei dati ottenuti dal *tracker*.
L'idea originale ambiva a sviluppare una versione zero di un'applicazione mobile che avesse funzionalità di interazione a basso livello con un *tracker* a scelta. L'applicazione, installata negli smartphone dei pazienti, prevedeva un'interfaccia grafica minima per guidare il paziente nella fase di autenticazione e configurazione dell'accoppiamento con il bracciale. Dopodiché, un servizio in *background*, responsabile della sincronizzazione periodica, avrebbe aperto la connessione con il bracciale, elaborato e trasformato i dati in conformità alle specifiche esposte dalle API del *back-end* e, infine, avrebbe salvato le informazioni nel database.

L'intenzione era fornire il servizio di diversi *adapter* che permettessero la connessione con modelli differenti, in modo da poter supportare *tracker* di vari marchi. Anche se la tecnologia alla base della comunicazione è la stessa (Bluetooth), la codifica dei dati è diversa da produttore a produttore. A fronte di diversi formati dei dati ricevuti in input, ogni *adapter* avrebbe prodotto in uscita informazioni modellate in accordo con il *data model* del nostro database.

Attraverso una fase di ricerca e di studio sul tipo di connessione tra bracciali e smartphone abbiamo identificato con più precisione il protocollo di comunicazione. Si chiama Bluetooth Low Energy (BLE) ed è una delle tecnologie principali dell'Internet of Things (IoT).

Come si intuisce dal nome, la differenza principale rispetto al classico Bluetooth riguarda il consumo. Infatti, esso è utilizzato per interagire con dispositivi dotati di batterie a capienza limitata con i quali avvengono scambi periodici di 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 tornare allo stato di *sleep mode*. 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.

Il passo successivo è stato comprendere i formati e le procedure utilizzate per il trasferimento dei dati. A definirle è il Generic Attribute Profile (GATT) [7] associato al *wearable* che utilizza come protocollo di trasporto l'Attribute Protocol (ATT).
Si tratta di una connessione *client-server* in cui il *tracker* assume il ruolo di GATT *server* e si predispone per ricevere richieste di lettura/scrittura di piccole quantità di dati, chiamati attributi. Il *client*, che nella nostra configurazione coincide con lo smartphone dell'utente, è responsabile di iniziare la transazione.

Gli attributi sono pezzi di informazioni indirizzabili che contengono dati utente (o metadati) concettualmente collocati nel *server* e accessibili (o potenzialmente modificabili) dal *client*.
La tabella 1.2 mostra alcuni esempi di attributi rappresentati in forma tabellare.

Tabella 1.2

Handle	Type	Permissions	Value	Length
0x0201	$UUID_1$ (16 bit)	R, no security	0x180A	2
0x0202	$UUID_2$ (16 bit)	R, no security	0x2A29	2
0x0215	$UUID_3$ (16 bit)	R/W, authorization	"string"	23
0x030C	$UUID_4$ (128 bit)	W, no security	{0xFF, 0xFF, 0x00, 0x00}	4
0x030D	$UUID_5$ (128 bit)	R/W, authentication	36.43	8
0x031A	$UUID_1$ (16 bit)	R, no security	0x1801	2

Ogni attributo è definito da:

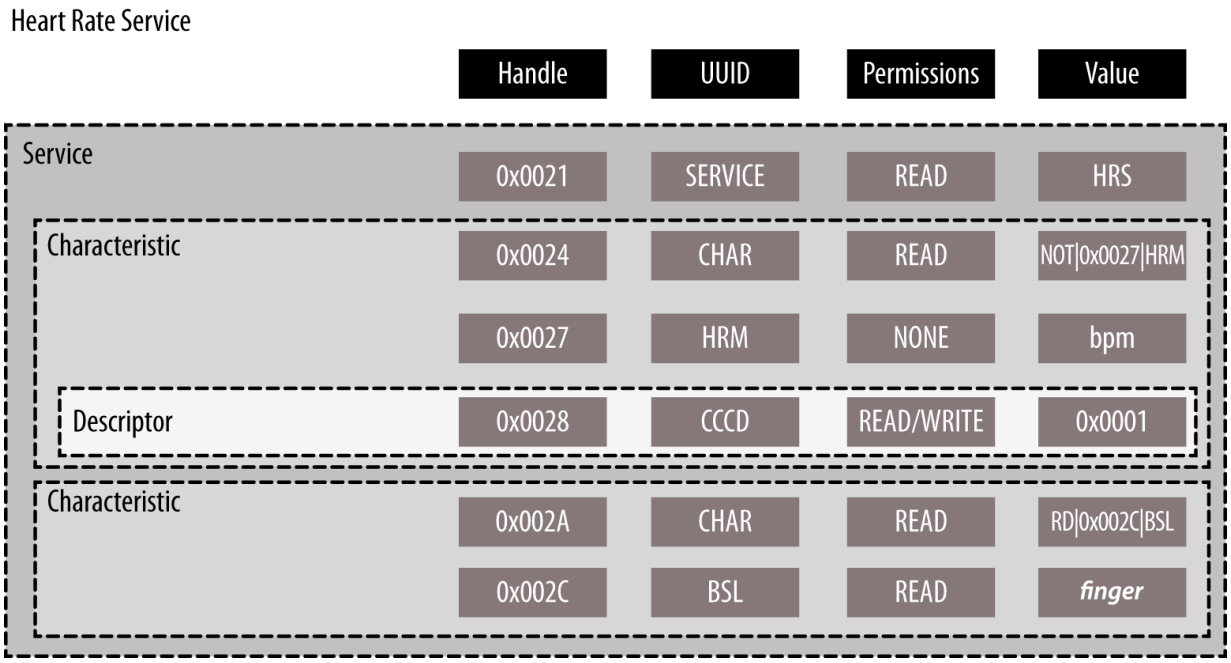
- Handle: è un id univoco di 16 bit che permette l'indirizzamento dell'attributo. Il suo valore rimane costante anche tra connessioni diverse.
- Type: un UUID (Universal Unique Identifier) che definisce il tipo di attributo, cioè il significato da attribuire al valore. L'id può essere di 16, 32 o 128 bit. I formati più corti sono utilizzati esclusivamente per tipi di attributi standard, definiti dalle specifiche della Bluetooth SIG, *16-bit UUID Numbers Document [8]*, l'organizzazione internazionale che

sovrintende lo sviluppo degli standard Bluetooth. Il formato da 128 bit è, invece, a disposizione dei produttori per identificare attributi specifici del dispositivo.

- Permissions: è un metadata che stabilisce quali operazioni (lettura e/o scrittura) sono ammesse sull'attributo e che requisiti di sicurezza sono necessari. L'accesso ad attributi sensibili può richiedere l'autorizzazione dell'utente. In alcuni casi può anche essere richiesta una connessione criptata per interagire con il dispositivo.
- Value: valore effettivo dell'attributo. Non ci sono restrizioni sul tipo di dato che può contenere.
- Length: lunghezza del campo *value*, limitata ad un massimo di 512 byte.

Gli attributi possono essere classificati in: servizi, caratteristiche e descrittori, i quali permettono una strutturazione gerarchica.

Figura 1.3



Tutti gli attributi concettualmente correlati sono raggruppati in servizi, ognuno dei quali può contenere zero o più caratteristiche. A loro volta, ogni caratteristica può contenere zero o più descrittori.

Il primo attributo di un servizio è anche chiamato *service declaration* e ha sempre uno UUID definito dagli standard Bluetooth SIG. Il suo valore corrisponde all’effettivo UUID del servizio esposto che può essere un servizio standard oppure custom.
Le caratteristiche possono essere considerate come contenitori di dati utente. Anch’esse prevedono un primo attributo di dichiarazione che indica l’inizio della caratteristica e fornisce metadati relativi ai successivi valori. Metadati aggiuntivi sono dichiarati mediante i descrittori.

Gli attributi di dichiarazione per i servizi e le caratteristiche vengono utilizzati dal *client* per una prima fase di scoperta dei dati esposti dal dispositivo. Solamente dopo aver recuperato gli UUID e i metadati è possibile accedere ai dati utente.

Dopo la fase di studio del protocollo si è tentato di aprire una connessione con un *tracker* modello Xiaomi Mi Band 3, riuscendo correttamente a identificare i servizi esposti e gli indirizzi ai quali recuperare il numero di passi giornalieri e la frequenza cardiaca. Tali attributi sono infatti identificati rispettivamente dagli UUID 0x2B40 e 0x180D indicati nelle specifiche della Bluetooth SIG [8]. Questo è stato un vantaggio non indifferente, in quanto i costruttori dei dispositivi sono liberi di scegliere di utilizzare sia UUID *custom* sia quelli standard; se la scelta fosse ricaduta sulla prima opzione avrebbe reso più complicato comprendere il tipo di dato dell’attributo. I bracciali prodotti da Fitbit, ad esempio, utilizzano identificatori *vendor-specific* per la quasi totalità dei dati utente. Sarebbe stato necessario un importante sforzo di *reverse engineering* per decifrare il tipo di attributi esposti.

In accordo con le specifiche richieste dai dottori, abbiamo inizialmente individuato tre dati strettamente necessari: il numero di passi, la frequenza cardiaca a riposo e i minuti di sonno, con granularità giornaliera. Tuttavia, sono state riscontrate due importanti problematiche.

La prima riguarda i dati derivati, ad esempio il sonno. Secondo le ricerche svolte, la totalità dei dispositivi di fascia medio-bassa adatti allo scopo dello studio, non eseguono i calcoli di derivazione in locale, all’interno del dispositivo, bensì sui *server* proprietari dei costruttori. Per sfruttare le piene potenzialità dei *tracker* è sempre necessario installare sul proprio smartphone l’applicazione associata. In questo modo, i dati grezzi, che si possono definire “primitivi”, direttamente misurati/calcolati dal *wearable* vengono mandati ai *server* che, oltre a

mantenerne lo storico, li processa tramite algoritmi proprietari per derivare dati più complessi. Tornando all'esempio del sonno, per definire se l'utilizzatore sta dormendo sono necessari, perlomeno, rilevazioni dell'accelerometro e della frequenza cardiaca.

Il successo imprenditoriale delle società produttrici di *wearable* dipende in buona parte dall'efficacia e dall'accuratezza degli algoritmi che utilizzano e dalla gestione di enormi quantità di dati che mantengono nei loro *server*. Processare i dati direttamente all'interno dei dispositivi avrebbe una serie di svantaggi importanti per le aziende:

- necessità di dotare i wearable di chip più performanti e costosi;
- minore autonomia a causa di un aumento dei calcoli processati;
- maggior esposizione delle proprietà intellettuali, in quanto gli algoritmi dovrebbero essere eseguiti nei software di ogni dispositivo;
- gli utenti potrebbero essere liberi di non installare l'applicazione associata.

Pertanto, interagire a basso livello con il *wearable* non avrebbe permesso di ottenere le informazioni relative al sonno. Questo primo problema è di possibile risoluzione attraverso l'ideazione e lo sviluppo di un algoritmo per riconoscere le fasi del sonno, tuttavia, questa opzione avrebbe richiesto uno studio molto complesso. Essa rimane, comunque, una soluzione per possibili sviluppi futuri.

La seconda problematica, invece, è difficilmente risolvibile. Il dispositivo Xiaomi Mi Band 3 è del 2018 e sembra essere uno degli ultimi *wearable* con il quale è possibile interfacciarsi senza requisiti di sicurezza. Infatti, i dispositivi indossabili moderni sono più evoluti nell'ambito della protezione dei dati dell'utente. Recuperare i dati sensibili richiede processi di autorizzazione e, inoltre, questi vengono criptati. Le applicazioni proprietarie dei produttori sono le uniche che possono interfacciarsi adeguatamente con i *tracker*.
A causa dei problemi citati si è reso necessario cercare delle soluzioni alternative.

Sono stati contattati i principali produttori di bracciali *smart* per il fitness, come Fitbit® (d'ora in poi: Fitbit) e Zepp® (d'ora in poi: Zepp), per provare ad ottenere strumenti di sviluppo che

permettessero di interagire con i *wearable* ad un livello più alto. In particolare, Zepp, una delle aziende leader nel mercato degli *smart wearable* e dei fitness *device*, sviluppa l’app Zepp Life associata sia ai dispositivi Amazfit® (d’ora in poi: Amazfit), di cui è anche la diretta produttrice, sia ai prodotti Mi Band di Xiaomi, altro colosso dell’industria.
Zepp avrebbe messo a disposizione lo Zepp SDK, una libreria per ambienti mobile che offre API per interagire con buona parte dei dispositivi delle due compagnie. Il processo per ottenere l’accesso all’SDK prevede diversi passaggi per accreditarsi come sviluppatori Zepp e per dimostrare gli utilizzi futuri della libreria. Sfortunatamente, non avendo più ricevuto comunicazioni da Zepp, non è stato possibile continuare a percorrere questa possibile soluzione. Negli ultimi mesi il sito web di Zepp è stato sottoposto ad un *re-branding* e i riferimenti all’SDK sono stati rimossi; è probabile che le strategie aziendali fossero in un periodo di transizione.

L’unico strumento che le società produttrici mettono a disposizione per sviluppatori esterni sono le web API. Sia Fitbit che Zepp forniscono una serie di *endpoint* REST dai quali si possono recuperare, a valle di un processo di autenticazione e autorizzazione, i dati relativi agli account associati ai *tracker*. Le web API di Fitbit sono la soluzione che si è scelto di adottare.

Dal momento che interfacciarsi direttamente con dispositivi prodotti da terzi implica i problemi già discussi, la cui risoluzione è difficilmente praticabile in tempi consoni con l’aspettativa di inizio della fase di studio, l’espediente utilizzato consiste nell’aver interagito con i *server* terzi (quelli di Fitbit come prima implementazione) che mantengono i dati utente. In questo caso, i dati sono già elaborati e pronti per essere recuperati e adattati al *data model* del nostro database.

Il carattere empirico di tale lavoro ha comportato la necessità di rivedere in itinere la soluzione ipotizzata all’inizio; ciò ha comportato diverse limitazioni e cambiamenti al progetto:

- i pazienti sono stati costretti ad installare l’app associata al *tracker* scelto. Infatti, l’app è indispensabile per sincronizzare i dati dal *tracker* ai *server* Fitbit. Inoltre, ogni paziente ha dovuto creare un account Fitbit associato.

- L’applicazione mobile HealthApp è stata ridimensionata con funzionalità duali nella versione web e con implementazioni aggiuntive per quanto riguarda le raccomandazioni e gli incentivi. In questo modo decade la necessità di implementare la logica di connessione con il *tracker*.
- Si è perso il controllo dei dati dell’utente, dipendendo, così da terzi.

Pur rimanendo una soluzione funzionale rispetto alle specifiche richieste, le strade che si aprono per miglioramenti e sviluppi futuri sono molte. Il progetto è ambizioso e si può estendere anche a tesi indirizzate verso lo sviluppo hardware per progettare *tracker ad hoc*.

Capitolo 2

Il back-end

Il *core* dell'applicazione permette ai due *client* dell'architettura applicativa, l'applicazione web e quella mobile, l'accesso alle istanze dei modelli di dominio del progetto. Il *back-end*, identificabile come il *server* dell'architettura, consiste in un'applicazione Spring Boot che implementa il *data access layer*: modellizza le informazioni da persistere, definisce ed espone le REST API che permettono ai *client* di eseguire operazioni di lettura e scrittura sui dati contenuti nel database, gestisce i requisiti di sicurezza, l'autenticazione degli utenti e i permessi di accesso.
In aggiunta, si interfaccia con i *server* Fitbit ed esegue la routine di sincronizzazione dei dati degli account Fitbit associati ai pazienti. Lo scopo è creare una ridondanza di informazioni basilari sul fitness per lo studio all'interno del database, in modo da rimanere indipendenti da terzi per quanto concerne lo storico dei dati sui quali verranno eseguite analisi al termine dello studio.

Il *framework* Spring (sviluppato in Java) fornisce un set di astrazioni e meccanismi che consentono di istanziare, assemblare e gestire il ciclo di vita dei componenti elementari di un'applicazione software. Allo sviluppatore rimane il compito di definire le classi dei componenti, chiamate *bean*, ed etichettarle mediante le annotazioni messe a disposizione da Spring. Il framework, in base alle assunzioni derivate dall'ispezione dei tipi e delle annotazioni presenti sui *bean*, ha lo scopo di generare il codice per gestire le dipendenze tra le classi.

Da Spring sono derivati diversi sotto-progetti rappresentabili come librerie pronte per essere integrate nell' applicazione, al fine di aggiungere specifiche funzionalità. Spring Boot è un esempio di modulo: esso consente di creare applicazioni Spring autonome che possono essere

eseguite immediatamente senza necessità di configurazioni, alleggerendo in modo drastico il carico dello sviluppatore.

Il codice del *back-end* è stato sviluppato in Kotlin, un moderno linguaggio di programmazione multi-paradigma in continua diffusione, specialmente nell'ambito dello sviluppo di applicazioni mobile Android. Derivando da Java è perfettamente interoperabile con esso.

2.1 Il modello dei dati

Il primo passo è stato individuare quali sono le informazioni scambiate o processate dal sistema e come modellizzarle in classi.

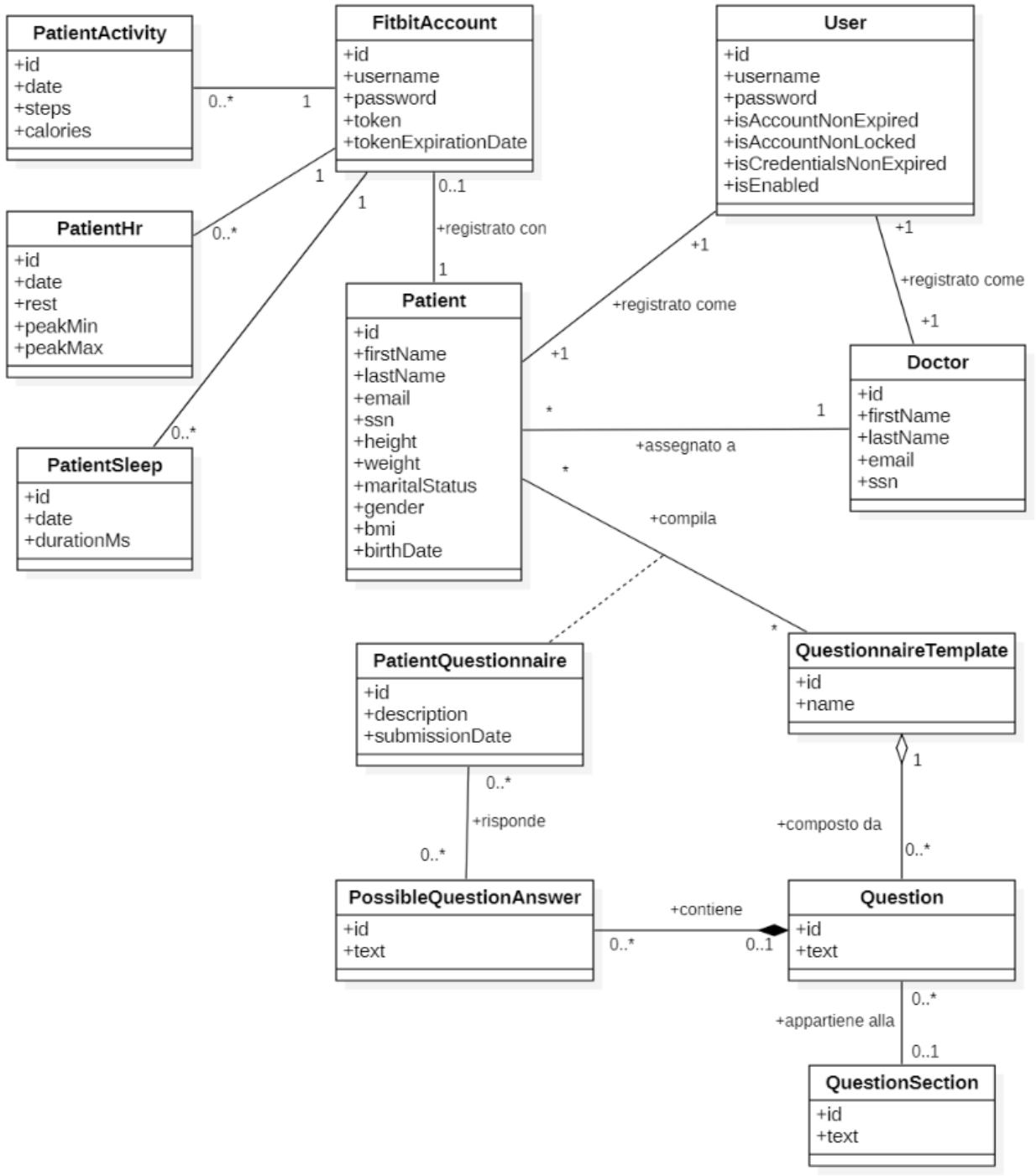
Gli *stakeholders* del sistema sono due: pazienti e dottori. Ognuno di essi è definito dalla rispettiva anagrafica ed è univocamente associato ad un utente del sistema, al quale vengono attribuiti uno username e una password.
I pazienti sono anche associati ai rispettivi account Fitbit, necessari per ricavare i dati sul fitness, vale a dire i dati sull'attività giornaliera (passi compiuti e calorie consumate), la frequenza cardiaca a riposo (insieme ai picchi minimi e massimi) e il numero di minuti di sonno al giorno.

Per quanto concerne la gestione dei questionari, le classi principali prese in considerazione sono le domande e le possibili risposte (si presuppone la presenza di sole domande a risposta multipla). Il *template* di un questionario si costruisce attraverso un *array* di domande, ognuna delle quali contiene un *array* di possibili risposte.

Infine, la compilazione dei questionari è definita dal paziente associato ad essa, dalla data di compilazione, dal *template* del questionario utilizzato e dall'*array* delle risposte attribuite ad ognuna delle domande.

In figura 2.1 è stato sviluppato un diagramma UML (Unified Modelling Language) che consente di avere una visione completa e chiara delle classi di dominio, degli attributi e delle rispettive relazioni; tale diagramma viene anche chiamato "modello concettuale".

Figura 2.1


PatientActivity
+id
+date
+steps
+calories
FitbitAccount
+id
+username
+password
+token
+tokenExpirationDate
User
+id
+username
+password
+isAccountNonExpired
+isAccountNonLocked
+isCredentialsNonExpired
+isEnabled
PatientHr
+id
+date
+rest
+peakMin
+peakMax
Patient
+id
+firstName
+lastName
+email
+ssn
+height
+weight
+maritalStatus
+gender
+bmi
+birthDate
Doctor
+id
+firstName
+lastName
+email
+ssn
PatientSleep
+id
+date
+durationMs
PatientQuestionnaire
+id
+description
+submissionDate
QuestionnaireTemplate
+id
+name
PossibleQuestionAnswer
+id
+text
Question
+id
+text
QuestionSection
+id
+text
0..*
1
0..1
+registrato con
+registrato come
+assegnato a
+compila
+risponde
+composto da
+contiene
+appartiene alla

2.1.1 I casi d'uso

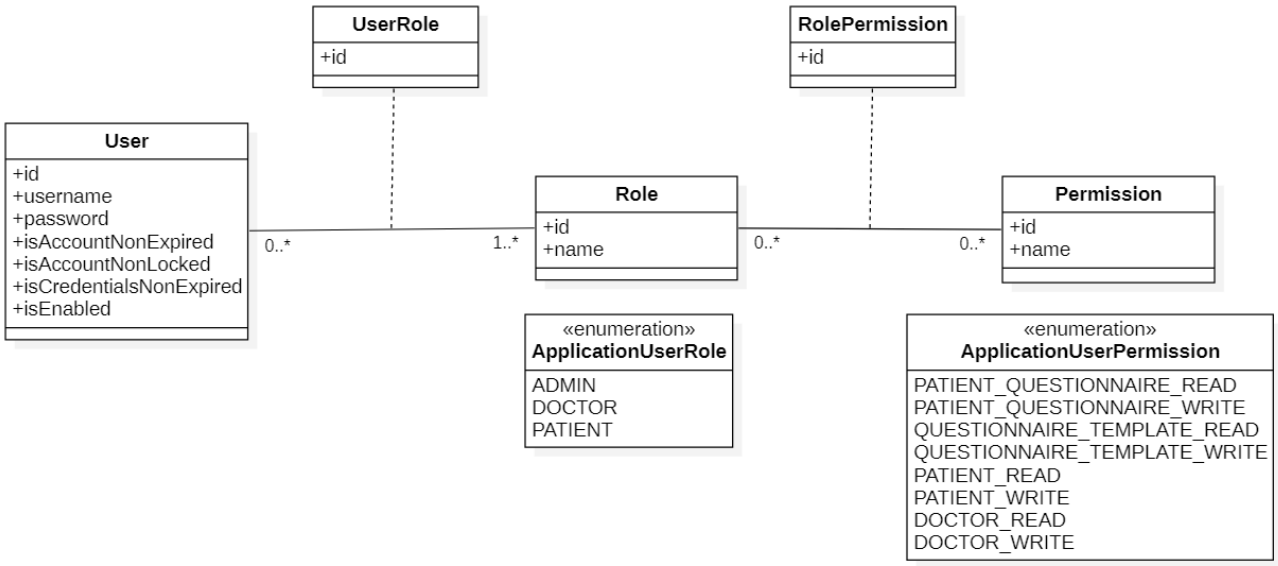
Ogni utente del sistema, che sia paziente o dottore, è caratterizzato da un ruolo. Nei sistemi informatici, l'attribuzione dei ruoli consente di diversificare i permessi di accesso ai dati o, più in generale, di autorizzare l'esecuzione di determinate attività ad alcune tipologie di utenti.

I ruoli previsti nell'applicazione HealthApp sono tre: paziente, dottore, amministratore. Per ogni ruolo viene specificata la lista dei permessi di accesso, in lettura e/o scrittura, su ciascuna delle entità del modello dei dati.

In figura 2.2 si è esteso il modello concettuale aggiungendo le classi relative ai permessi e ai ruoli:

- ad ogni utente possono essere attribuiti uno o più ruoli;
- ad ogni ruolo è attribuito un set di permessi.

Figura 2.2



Nella tabella 2.3 sono elencati i casi d'uso, vale a dire le attività che il *back-end* è in grado di gestire, associate ai ruoli che sono autorizzati ad eseguirle.

Tabella 2.3

Caso d'uso	Ruoli
Creazione nuovo dottore	Amministratore
Lettura lista dottori	Amministratore
Lettura singolo dottore per id	Amministratore, Dottore
Lettura lista pazienti associati al dottore	Amministratore, Dottore
Creazione nuovo paziente	Amministratore, Dottore
Lettura lista pazienti	Amministratore, Dottore
Lettura singolo paziente per id	Amministratore, Dottore, Paziente
Creazione template questionario	Amministratore, Dottore
Lettura lista template questionari	Amministratore, Dottore, Paziente
Lettura singolo template questionario per id	Amministratore, Dottore, Paziente
Creazione nuova compilazione per questionario, per paziente	Amministratore, Dottore, Paziente
Lettura lista compilazioni questionari, per paziente	Amministratore, Dottore, Paziente
Lettura singola compilazione per questionario, per paziente	Amministratore, Dottore, Paziente
Cancellazione compilazione per questionario, per paziente	Amministratore
Lettura numero di passi giornalieri, per utente, per data	Amministratore, Dottore, Paziente
Lettura frequenza cardiaca a riposo, per utente, per data	Amministratore, Dottore, Paziente
Lettura minuti di sonno giornalieri, per utente, per data	Amministratore, Dottore, Paziente

2.1.2 L'utilizzo delle Fitbit API

Ogni paziente verrà dotato di un tracker e di un account Fitbit per il monitoraggio dei valori sul *fitness*. Col fine di aumentare l'indipendenza da server terzi, *nel back-end* è attiva una routine periodica che va a recuperare, ogni otto ore, i dati aggiornati e li persiste all'interno del database. In questo modo la piattaforma HealthApp si può costruire uno storico di dati sul quale eseguire analisi, senza dover più fare affidamento a Fitbit per quanto riguarda i dati passati.

Le WEB API esposte da Fitbit sono descritte al sito web https://dev.fitbit.com/build/reference/web-api/. In particolare, l'applicazione sfrutta gli endpoint relativi all'attività fisica, alla frequenza cardiaca e al sonno.

Le applicazioni che utilizzano le Fitbit API devono usare il protocollo descritto in "*OAuth 2.0 Authorizazion Framework*" [9] per avere l'autorizzazione all'accesso ai dati di un utente Fitbit. Questo documento descrive come permettere ad un'applicazione terza di ottenere un accesso limitato ad un servizio HTTP che espone delle risorse, per conto del proprietario di quelle stesse risorse.

Il protocollo OAuth 2.0 prevede diversi tipi di flussi di autorizzazione. In particolare, Fitbit raccomanda di usare il flusso descritto in "*Authorization Code Grant Flow with Proof Key for Code Exchange*" (PKCE) [10] che richiede di creare dinamicamente una chiave criptabile casuale ("*code verifier*") con la quale viene risolto il "*code challenge*" per verificare l'autorizzazione del *client*.

Prima di procedere con l'autorizzazione, l'applicazione deve essere registrata su https://dev.fitbit.com/apps tramite un account sviluppatore. Durante la fase di registrazione vengono definiti due valori che saranno necessari per la verifica dell'autorizzazione, ovvero una chiave segreta (il Code Verifier) e un Callback URL. Di seguito sono descritte le fasi necessarie per ottenere l'autorizzazione.

1. **Generare il *Code Verifier* e il *Code Challenge***: il *Code Verifier* è un valore casuale di 43-128 caratteri crittografato tramite SHA-256 [11] e codificato in Base64 [12] (con *padding* omessi) per ottenere il *Code Challenge*.

2. **Richiedere l'autorizzazione ai dati di un utente Fitbit**: utilizzando un browser web bisogna effettuare una richiesta all'*endpoint* per la pagina di autorizzazione Fitbit.
   All'URL bisogna aggiungere i seguenti parametri di *query:*
   - *client_id*: corrisponde all'id dell'applicazione registrata;
   - *scope*: lista di collezioni dati richieste all'applicazione;
   - *code_challenge*: generato nel primo punto;
   - *code_challenge_method*: S256;
   - *response_type*: code.

   Di seguito un esempio di URL per la pagina di autorizzazione Fitbit completo di parametri:

   *https://www.fitbit.com/oauth2/authorize?**client_id**=ABC123&**response_type**=code&**code_challenge**=<code_challenge>&**code_challenge_method**=S256&**scope**=activity%20heartrate%20location%20nutrition%20oxygen_saturation%20profile%20respiratory_rate%20settings%20sleep%20social%20temperature%20weight*

   Il seguente link mostra una finestra dove l'utente deve autenticarsi con le proprie credenziali Fitbit e confermare il permesso all'accesso dei dati.

3. **Recuperare l'Authorization Code**: dopo che l'utente ha autorizzato l'accesso ai propri dati, Fitbit effettua un re-indirizzamento al Callback URL definito in fase di registrazione dell'applicazione. Nel caso di HealthApp, l'url punta al *back-end* e dai parametri si può estrarre il codice.
   https://softeng.polito.it/health-core/callback?code=<authorization_code>#_=_

4. **Scambiare l'Authorization Code con l'Access Token:** una POST viene eseguita all'endpoint per la generazione del token passando i seguenti parametri di *query*:
   - *client_id*: corrisponde all'id dell'applicazione registrata;
   - *code*: l'Authorization Code;
   - *code_verifier*: il Code Verifier;
   - *grant_type*: authorization_code.

Di seguito un esempio di URL per ottenere il token:

*https://www.fitbit.com/oauth2/token?**client_id**=ABC123&**code**=<authorization_code>&**code_verifier**=<code_verifier>&**grant_type**=authorization_code*

5. **Ricevere l'Access Token:** il token endpoint restituisce un oggetto Json che include: l'id dell'utente che ha autorizzato l'accesso, l'Access Token con il quale eseguire le query ai dati utente tramite le Fitbit Web API, il Refresh Token utilizzato per aggiornare l'Access Token quando scade, la durata di validità del token e la lista di collezione dati autorizzati ad ottenere.

Le prime due fasi sono eseguite nel *front-end* durante la fase di creazione di un nuovo paziente: il dottore, dopo aver compilato il form con i dati anagrafici del paziente dovrà procedere ad autorizzare l'accesso ai dati Fitbit; tramite un bottone viene avviato il flusso di autorizzazione che genera l'URL del punto 2 e mostra la finestra di login Fitbit. Dopo aver immesso le credenziali dell'account Fitbit associato al paziente ed aver autorizzato l'accesso, il re-direzionamento avviene nel *back-end*, il quale gestirà le fasi successive del processo per salvare l'Access Token e il Refresh Token nel database e associarli all'utente. In questo modo, tutte le successive chiamate alle Fitbit Web API sono autorizzate grazie al token.

2.2 Spring framework

Spring è uno dei *framework* più popolari tra quelli derivati dalle specifiche introdotte dalla *community* Jakarta EE, in precedenza Java EE (Enterprise Edition), per lo sviluppo di applicazioni distribuite e servizi web di alta qualità. Esso si basa su tre concetti chiave:

- Inversion of Control (IoC), uno dei principi più importanti della programmazione a oggetti; le dipendenze dirette tra oggetti di classi diverse vengono eliminate e sostituite dall'introduzione delle interfacce, le quali astraggono il comportamento atteso.
- Dependecy Injection (DI), un *pattern* di programmazione responsabile di implementare le dipendenze tra oggetti di classi diverse a tempo di esecuzione invece che a tempo di

compilazione. Spring ispeziona i nomi delle classi e delle interfacce, i tipi e le annotazioni che il programmatore introduce nel codice per creare un contenitore software al cui interno vengono descritte tutte le dipendenze. All'esecuzione dell'applicazione gli oggetti sono iniettati nel codice sotto forma di proprietà.

- Aspect Oriented Programming (AOP), un paradigma di programmazione che distingue quelle funzionalità dell'applicazione diffuse orizzontalmente su tutta la *code-base*, come il *logging* o la sicurezza. La logica delle funzionalità è raggruppata in un unico punto, detto modulo, mentre la loro applicazione è eseguita in modo dichiarativo.

Il *framework* mette a disposizione una varietà di moduli destinati a specifiche funzionalità che lo sviluppatore può decidere se includere o meno. Di seguito verranno approfonditi quelli utilizzati nel *back-end* di HealthApp.

2.2.1 Spring Data JPA

Spring Data è un modulo derivato da Spring e nato con l'obiettivo di elaborare un modello standard per l'implementazione del *data access layer,* il livello responsabile di fornire gli strumenti per accedere e modificare le tabelle del database scelto, relazionale o non relazionale. Sono disponibili diversi sotto-moduli a seconda del tipo di database.

In questo specifico caso è stato selezionato H2, un Database Management System scritto in Java con tecnologia *in-memory*: il database viene incapsulato direttamente all'interno dell'applicazione Java. Si tratta di una scelta finalizzata a semplificare il processo di sviluppo dell'applicazione nella prima fase; anche se in produzione potrebbe essere necessario virare verso una tecnologia più strutturata e performante.

Spring Data fornisce diversi tipi di approcci per la gestione della persistenza dei dati: Spring Data JDBC, Spring Data JPA, Spring Data R2DBC.
La scelta è ricaduta sull'opzione JPA (Java Persistence API) che espone un'interfaccia applicativa utile a costruire sistemi ORM basati su oggetti Java. L'Object-Relational Mapping consiste nell'utilizzare strumenti software per associare le tabelle presenti nel database (relazionale) alle classi di dominio definite nel modello concettuale di figura 2.1, anche chiamate entità. JPA

include il progetto Hibernate ORM che implementa questa strategia traducendo automaticamente i *record* delle tabelle in istanze di oggetti Java, e viceversa.
La dualità tabella-entità non è scontata, in quanto:

- gli oggetti possono essere polimorfici, i *record* di una tabella no;
- gli oggetti possono essere duplicati e avere strutture dati complesse, i *record* devono essere univoci e composti da semplici attributi con valori alfa-numerici;
- gli oggetti sono referenziati da un indirizzo di memoria che può variare, i *record* dalle loro chiavi che devono rimanere stabili;
- gli oggetti usano puntatori per avere relazioni con altri oggetti, i *record* devono usare le chiavi esterne.

Il *framework* gestisce le differenze in autonomia grazie alle annotazioni che il programmatore inserisce nel codice. Di seguito un esempio di definizione della classe che rappresenta i dottori:

```
@Table
@Entity(name = "doctor")
data class Doctor(
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    val id: Long? = null,

    @Column(name = "first_name")
    val firstName: String,

    @Column(name = "last_name")
    val lastName: String,

    val email: String,
    val ssn: String,

    @OneToMany(mappedBy = "doctor")
    @JsonIgnore
    val patients: List<Patient>,

    @OneToOne(fetch = FetchType.EAGER, cascade = [CascadeType.ALL])
    @JoinColumn(name = "user_id", nullable = false)
    val user: User
)
```

Ogni classe annotata con @Table e @Entity introduce una nuova tabella nel database. Tutti gli attributi di tipi elementari (numerico, booleano, stringa) della classe rappresentano le colonne

(l'annotazione @Column è utilizzata per definire un nome diverso alla colonna) e un attributo deve essere annotato con @Id per specificare quale valore deve assumere il ruolo di chiave primaria.
Gli attributi UDT (User Defined Type), vale a dire quelli che hanno come tipo un'altra classe di dominio, devono essere etichettati tramite le annotazioni di relazione @OneToOne, @ManyToOne, @OneToMany e @ManyToMany. Il *framework* e il DBMS implementano i meccanismi per rappresentare le relazioni nel database e lo fanno mediante le chiavi esterne, l'aggiunta di colonne alla tabella o eseguendo *join* tra tabelle diverse, in modo trasparente allo sviluppatore.

JPA fornisce, per ogni entità, un insieme di operazioni generali di basso livello per accedere, persistere, aggiornare e rimuovere i *record* dalle tabelle. Raramente, lo sviluppatore ha la necessità di impiegare funzioni di così basso livello: lo scenario tipico, infatti, consiste nell'utilizzare classi chiamate Repository per aumentare l'astrazione delle funzioni CRUD.
Per ogni tabella del database viene implementata un'interfaccia, annotata con @Repository, che estende la classe JpaRepository, tipizzata con la classe mappata alla tabella e il tipo elementare della chiave primaria.

```
@Repository
interface DoctorRepository: JpaRepository<Doctor, Long> {
    fun findDoctorByUser(user: User): Doctor
}
```

Spring genera, in autonomia, i metodi per attuare le operazioni basilari sulla tabella. Tali metodi sono richiamabili mediante convenzioni nominali:

- *findAll*() (oppure *getAll*(), le parole chiavi *find* e *get* sono intercambiabili), ritorna una lista con tutti gli oggetti presenti nella tabella in questione;
- i metodi con prefisso *findBy,* permettono di eseguire semplici query su attributi elementari;
- *save(<entity>),* inserisce un nuovo record in tabella;
- *delete(<entity>),* elimina il record mappato con l'oggetto passato in input.

Come si evince dall'esempio DoctorRepository, lo sviluppatore può integrare l'interfaccia con metodi che richiedono *query* più complesse, le quali possono necessitare di condizioni multiple, ordinamento dei risultati, *join* con altre tabelle.

A questo livello vengono trattate le entità, vale a dire le istanze di classi Java che rappresentano le tabelle; attraverso i *repository* definiamo i metodi di accesso e lettura del database sottostante. Non viene eseguita alcuna logica applicativa, la quale viene implementata al livello logico superiore.

2.2.2 Spring Web MVC

I livelli superiori del *back-end* sono sviluppati seguendo il pattern architetturale *Model-View-Controller* per mezzo dell'omonimo *framework* di Spring. Tale strategia permette di separare la logica di presentazione dell'applicazione da quella di business.

Il modulo è progettato intorno ad un *DispatcherServlet* che ha il compito di gestire le richieste HTTP effettuate dai *client*, inoltrandole ai rispettivi *handler*. In questo caso viene istanziato un *web server* Tomcat il cui *container* è in ascolto alla porta 8080.

Gli *handler* responsabili di accogliere e soddisfare le richieste dei client sono definiti in classi chiamate Controller, al cui interno vengono determinati gli *endpoint* esposti dall'applicazione.

```
@RestController
@RequestMapping("/api/doctors")
internal class DoctorController(val doctorService: DoctorService) {

    @GetMapping
    fun getAllDoctors(): List<Doctor> {
        return doctorService.getAllDoctors()
    }

    @GetMapping("/{doctorId}")
    fun getDoctor(@PathVariable doctorId: Long): Doctor {
        return doctorService.getDoctor(doctorId)
    }

    @GetMapping("/{doctorId}/patients")
    fun getAllPatientForADoctor(@PathVariable doctorId: Long): List<Patient>{
        return doctorService.getAllPatientForDoctor(doctorId)
    }

    @PostMapping
    fun createDoctor(@RequestBody doctorJson: DoctorJson): Doctor {
        return doctorService.createDoctor(doctorJson)
    }
}
```

Come mostrato nell'esempio del DoctorController, è necessario etichettare la classe con l'annotazione @RestController e @RequestMapping, specificando l'URL da chiamare per accedere agli *endpoint* esposti. Ogni metodo corrisponde ad un handler, annotato con il corrispondente metodo HTTP ed un eventuale *path* aggiuntivo che può essere parametrizzabile. Quando un *client* esegue una richiesta, il *servlet* cerca tra i controller un *handler* che mappa lo stesso URL e lo stesso metodo (GET, PUT, POST, DELETE, ecc.) della richiesta.

Di seguito viene mostrato un riepilogo dei *controller* forniti dal *back-end* di HealthApp, specificando per ognuno le API esposte:

- DoctorController:
    - GET */api/doctors* per ottenere la lista dei dottori;
    - POST */api/doctors* per aggiungere un dottore;
    - GET */api/doctors/{doctorId}* per ottenere un dottore dal suo id;
    - GET */api/doctors/{doctorId}/patients* per ottenere i pazienti associati ad un dottore.
- PatientController:
    - GET */api/patients* per ottenere la lista dei pazienti;
    - POST */api/patients* per aggiungere un paziente;
    - GET */api/patients/{patientId}* per ottenere un paziente dal suo id.
- UserController:
    - GET */api/users/{username}* per ottenere un paziente o un dottore dall'username.
- QuestionnaireTemplateController:
    - GET */api/questionnaires/templates* per ottenere la lista dei questionari;
    - POST */api/questionnaires/templates* per aggiungere un questionario;

    - GET */api/questionnaires/templates/{questionnaireTemplateId}* per ottenere un singolo questionario dal suo id.

- PatientQuestionnaireController:

    - GET */api/patients/{patientId}/questionnaires* per ottenere la lista delle compilazioni di un paziente;

    - POST */api/patients/{patientId}/questionnaires* per aggiungere una nuova compilazione di un paziente;

    - GET */api/patients/{patientId}/questionnaires/{questionnaireId}* per ottenere una compilazionedi un paziente;

    - DELETE */api/patients/{patientId}/questionnaires/{questionnaireId}* per cancellare una compilazione di un paziente.

- PatientActivityController:

    - GET */api/patients/{patientId}/activities/steps* per ottenere il numero di passi effettuati da un paziente per un range di date;

    - GET */api/patients/{patientId}/activities/calories* per ottenere il numero di calorie consumate da un paziente per un range di date.

- PatientHrController:

    - GET */api/patients/{patientId}/hrs/peak* per ottenere i picchi di frequenza cardiaca di un paziente per un range di date;

    - GET */api/patients/{patientId}/hrs/rest* per ottenere la frequenza cardiaca media a riposo di un paziente per un range di date.

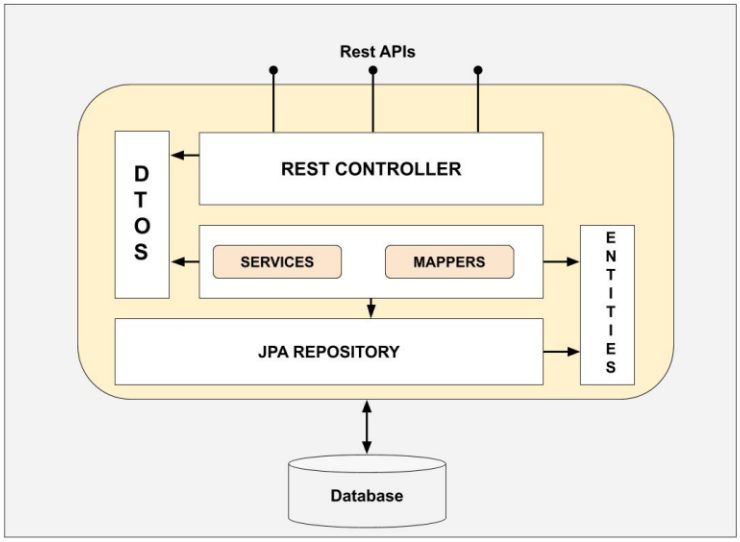
- PatientSleepController:

    - GET */api/patients/{patientId}/sleep/duration* per ottenere il numero di secondi in fase di sonno di un paziente per un range di date.

Ogni metodo di un *controller* può ricevere parametri in *input* attraverso le tre modalità che il protocollo HTTP permette: parametri di percorso, parametri di *query* e corpo della richiesta.
I metodi POST, PUT e PATCH permettono di allegare alla richiesta oggetti più complessi rappresentati per mezzo di DTO (Data Transfer Object).
Un Data Transfer Object è un oggetto che trasporta le informazioni tra i processi, in questo caso tra le API e il *back-end*. Come il sistema di trasmissione REST prevede, per passare dei dati nel corpo delle richieste HTTP si possono usare una grande varietà di *content-type*.
Tutte le API esposte dal *back-end* che prevedono un corpo accettano il *content-type* *application/json,* vale a dire un oggetto *json* sotto forma di stringa. I DTO possono essere validati automaticamente da Spring utilizzando annotazioni del tipo @NotNull, @Min, @Max, in modo da aggiungere un controllo sui dati che il *back-end* riceve in *input*.

I metodi dei *controller* richiamano funzioni di un ultimo componente che agisce da collante tra i *controller* stessi e i *repository*: il servizio.
All'interno delle classi annotate con @Service è gestita la logica applicativa dell'applicazione. Esse ricevono in input i DTO, li elaborano e li trasformano per ottenere le entità da dare in pasto ai *repository*. Inoltre, sono applicate alcune funzionalità che Spring mette a disposizione, ad esempio le politiche di sicurezza, di autenticazione e di autorizzazione.
In figura 2.4 vi è uno schema riepilogativo della struttura del *back-end*.

Figura 2.4



2.2.3 Spring Security

Il modulo di sicurezza che Spring mette a disposizione permette di definire meccanismi di protezione dell'applicazione in maniera dichiarativa, lasciando l'effettiva implementazione al *framework* stesso.

Agli *endpoint* esposti dai vari *controller* sono applicati dei filtri di accesso che consentono ai soli utenti autorizzati di ricevere la risposta, bloccandola, invece, in caso di mancato permesso. Il controllo dell'autorizzazione può avvenire a livello di URL della chiamata, oppure, come metodi di accesso alle entità.
Ogni possibile utilizzatore dell'applicazione deve essere associato ad un utente e deve essere previsto un meccanismo di autenticazione che consenta al *back-end* sia di accertarsi dell'identità dell'individuo e sia di applicare i filtri di accesso.
HealthApp sfrutta un comune meccanismo di autenticazione tramite username/password in cui le credenziali degli utenti sono mantenute criptate all'interno di una tabella del *database*, accessibile mediante la UserRepository.

```
class JpaUserRepository(private val userRepository: UserRepository) : UserDAO
{
    override fun selectUserByUsername(username: String): User? {
        return userRepository.findUserByUsername(username)
    }
}
```

Ogni tentativo di autenticazione da parte di un *client* richiede una chiamata POST all'*endpoint /login*, passando nel corpo della richiesta un oggetto *json* con username e password. La password viene confrontata con quella dell'oggetto di classe User ritornato dal metodo *selectUserByUsername*: un'eventuale corrispondenza garantisce il successo del processo di autenticazione. A valle di un corretto riconoscimento, Spring Security lega l'utente autenticato alla sessione HTTP instaurata con il *client*, in modo da garantire l'autenticazione per le future richieste.

La logica di sicurezza dell'applicazione è interamente definita in una classe derivata dalla classe di libreria WebSecurityConfigurerAdapter ed etichettata con le annotazioni @Configuration e @EnableWebSecurity

```
@Configuration
@EnableWebSecurity
class ApplicationSecurityConfig(
    private val userDetailsService: UserDetailsService,
    private val passwordEncoderAndMatcher: PasswordEncoder
) : WebSecurityConfigurerAdapter() {

        …
}
```

All'interno dell'ApplicationSecurityConfig viene definito il metodo per configurare la sicurezza per le richieste HTTP tramite la classe di libreria HttpSecurity, la quale offre alcune API per definire in modo compatto i filtri da applicare agli URL e per configurare gli *endpoint* di login/logout.

```
override fun configure(http: HttpSecurity) {
    http.cors()
        .and()
        .csrf().disable()
        .authorizeRequests()
        .antMatchers("/", "index", "/css/*", "/js/*")
        .permitAll()
        // USER ENDPOINTS
        .antMatchers(HttpMethod.GET, "/api/users/**").authenticated()
        // DOCTORS ENDPOINTS
        .antMatchers(HttpMethod.GET, "/api/doctors/**")
        .hasAuthority(ApplicationUserPermission.DOCTOR_READ.permission)
        .antMatchers(HttpMethod.POST, "/api/doctors")
        .hasRole(ApplicationUserPermission.DOCTOR_WRITE.permission)
        // PATIENTS ENDPOINTS
        .antMatchers(HttpMethod.POST, "/api/patients")
        .hasAuthority(ApplicationUserPermission.PATIENT_WRITE.permission)
        .antMatchers(HttpMethod.GET, "/api/patients/**")
        .hasAuthority(ApplicationUserPermission.PATIENT_READ.permission)
        …
```

Per ogni *endpoint*, il cui metodo e URL combaciano con quelli definiti negli *antMatchers,* viene indicato:

- se è richiesta un'autenticazione dell'utente;
- se l'accesso deve essere consentito solo ad utenti con specifici ruoli;
- se l'accesso deve essere consentito solo ad utenti con specifici permessi.

Il vantaggio nell'utilizzare il modulo Spring Security è notevole: infatti, si può lasciare allo sviluppatore il compito di dichiarare le *policy* di sicurezza che vuole attuare nel sistema,

consegnando l'implementazione al *framework* stesso. In questo modo, una parte sensibile di ogni sistema informatico non è lasciata completamente alle conoscenze del programmatore, il quale può avere lacune in un ambito che richiede un livello di specializzazione particolarmente alto.

Capitolo 3

Il front-end

È stata sviluppata un'interfaccia web per consentire ai medici e ai pazienti di interagire con le informazioni contenute nel *server*. Si è scelto di seguire un moderno *pattern* di sviluppo per le applicazioni web, chiamato Single-page-application: la pagina HTML non viene generata e mandata dal *server* ad ogni richiesta ricevuta, al contrario, è il *client* stesso ad auto-generare la pagina che deve mostrare grazie al codice JavaScript che ne definisce il *layout* e l'aspetto grafico. In questo modo il *back-end* delega la maggior parte del lavoro ai *browser* sui quali viene eseguita l'applicazione, rendendo il caricamento della pagina più fluido e meno dipendente da eventuali sovraccarichi del *server*.

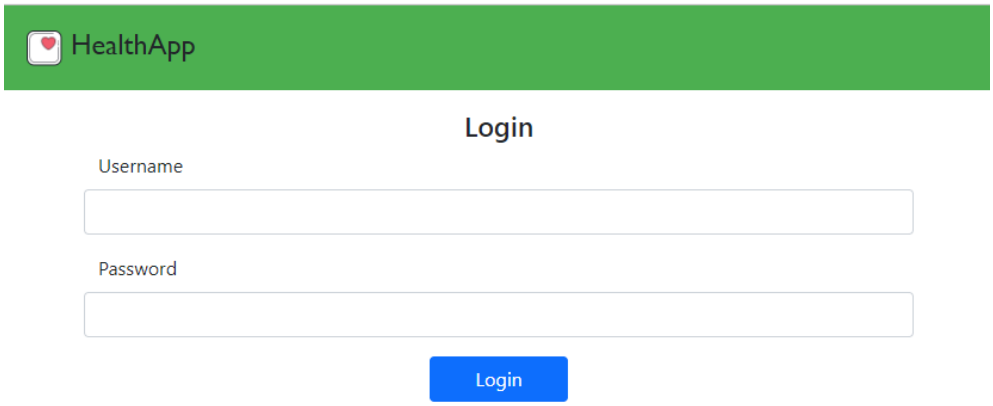
Per lo sviluppo del front-end si è utilizzato uno dei più importanti framework per lo sviluppo di interfacce web in JavaScript: React. Il programmatore può sfruttare un approccio dichiarativo con cui deve definire i componenti da visualizzare e dichiarare dove mostrarli, senza manipolare in modo esplicito il Document Object Model (l'oggetto che definisce la struttura della pagina web). React semplifica l'ambiente del *browser* uniformando i metodi DOM e processando in modo automatico eventi e aggiornamenti dell'interfaccia. Inoltre, la libreria è ampiamente estendibile grazie alla grande varietà di *plugin* e di moduli aggiuntivi disponibili, finalizzati ad un incremento delle funzionalità.

3.1 La struttura dell'applicazione web

L'applicazione assume un ruolo di interfaccia per la gestione dei pazienti e dei loro dati: la struttura, così come le funzionalità offerte, varia in modo dinamico a seconda del tipo di utente autenticato. Sono previsti tre ruoli: amministratore, dottore e paziente.

La prima pagina presenta il *form* per l'autenticazione mediante username e password (Figura 3.1).

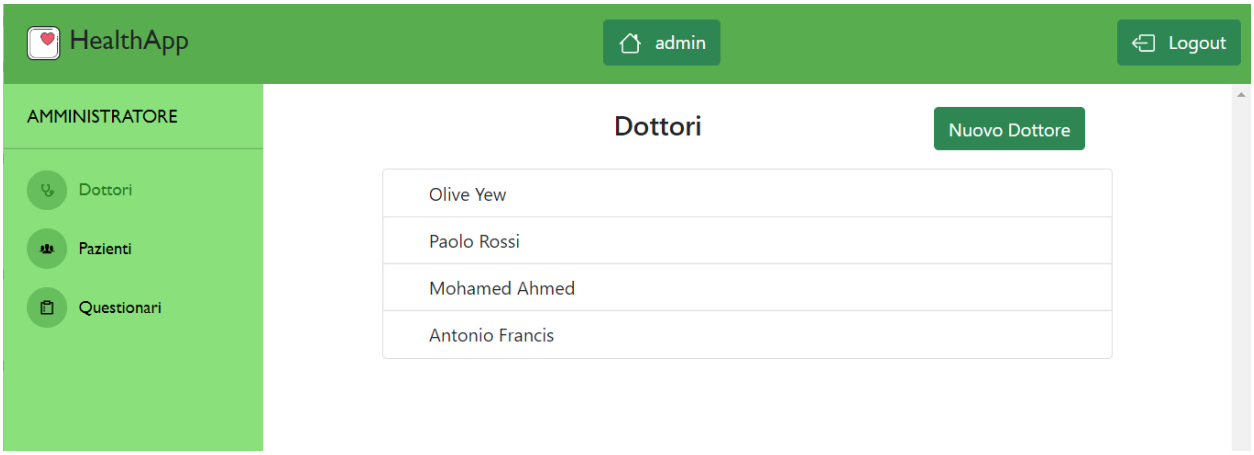
Figura 3.1



A valle del processo di autenticazione, l'applicazione riconosce il ruolo dell'utente e si adatta mostrando l'interfaccia opportuna. Lo scheletro del *layout* è costante per ogni pagina ed è sempre composto dai seguenti elementi (Figura 3.2):

- una barra superiore che mostra il logo dell'applicazione, un'indicazione sul tipo di utente autenticato e il bottone per il *logout*;
- una barra di navigazione laterale che permette di entrare nelle diverse sezioni messe a disposizione a seconda del tipo di utente;
- un corpo centrale che mostra il contenuto in accordo con la voce del menu che è stata selezionata.

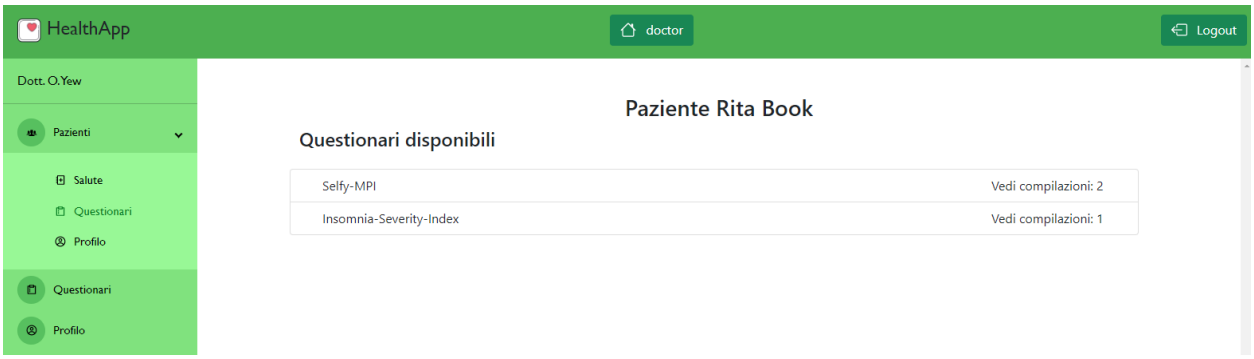
Figura 3.2



Un utente di tipo Admin ha un accesso completo a tutte le API per ottenere la lista completa degli oggetti delle tre principali classi di dominio: dottori, pazienti e questionari. Inoltre, è l'unico ruolo che ha l'autorizzazione a creare una nuova istanza di dottore.

Un utente di tipo Doctor ha accesso alla lista dei pazienti a lui assegnati e ha la possibilità di creare un nuovo paziente. Per ogni paziente può monitorare l'andamento dei dati sul benessere, il profilo anagrafico e le compilazioni dei questionari nel tempo (in Figura 3.3 è visualizzata la lista dei questionari e il numero di compilazioni per un paziente).

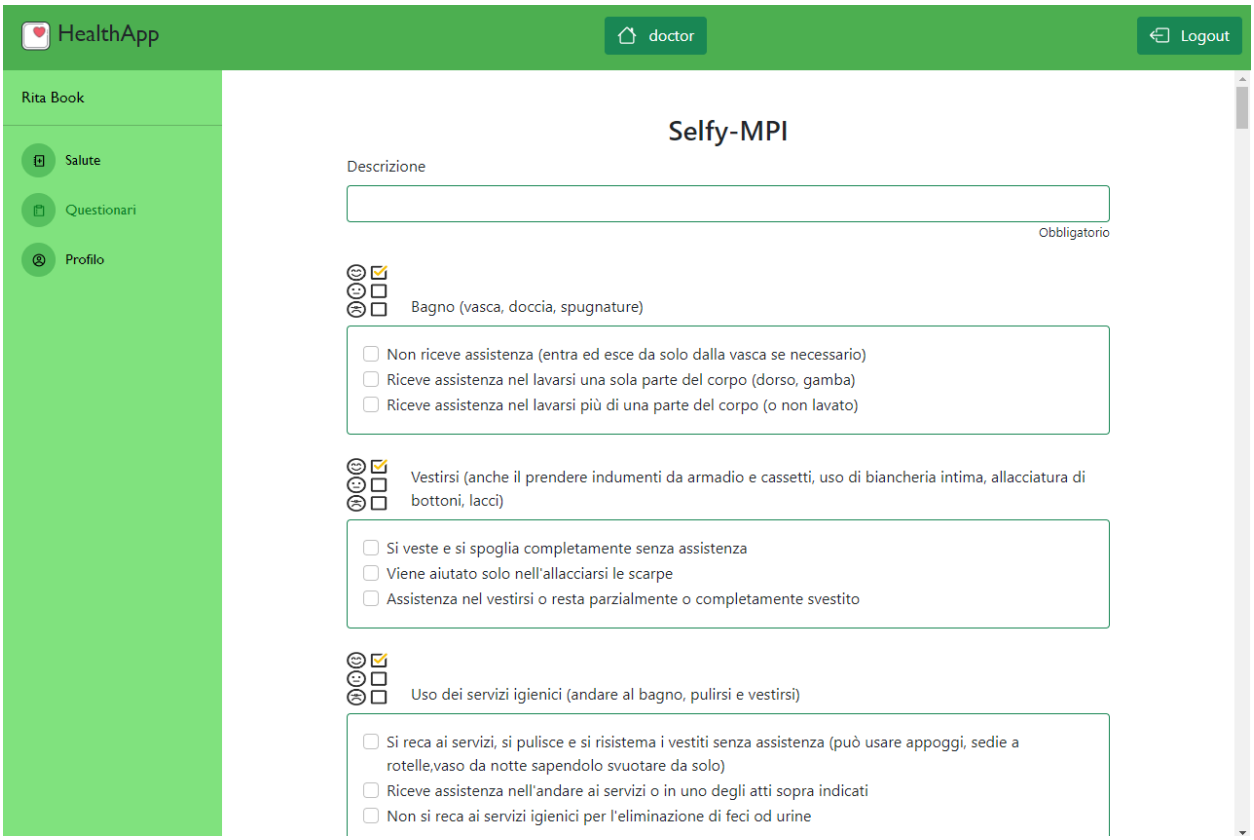
Figura 3.3



Alcuni tipi di questionari con valenza medica, come ad esempio quello relativo al calcolo del MPI, non possono essere compilati in autonomia dai pazienti (per specifica richiesta dei medici). Pertanto, il dottore è l'unico utente ad avere l'autorizzazione a compilare qualsiasi tipo di

questionario, per conto dei pazienti (in Figura 3.4 vi è un esempio di questionario). Infine, ha accesso al proprio profilo anagrafico.

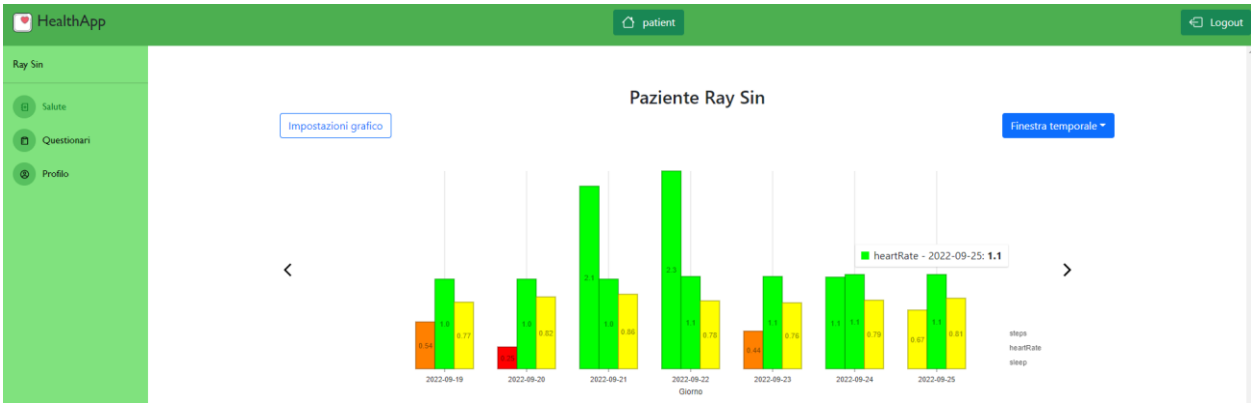
Figura 3.4



Un utente di tipo Patient vede la *dashboard* riepilogativa sull'andamento dei valori ottenuti dal tracker e una lista limitata dei questionari che può compilare in autonomia.

La sezione che mostra lo stato di salute di un paziente è costituita da un grafo a barre verticali (Figura 3.5).

Figura 3.5



Si tratta di un grafo temporale parametrizzabile: lungo l'asse delle ascisse è indicata la finestra temporale scelta, che può essere giornaliera, settimanale o mensile; per ogni unità temporale sono mostrate tre barre verticali indicatrici dell'andamento dei valori che vengono recuperati attraverso il *tracker*.

L'altezza e il colore della barra sono indicatori sull'andamento positivo o negativo. Le informazioni di partenza ricavate per mezzo del bracciale smart, per ogni paziente, sono il numero di passi giornalieri, la frequenza cardiaca media a riposo (giornaliera) e il numero di minuti di sonno giornalieri.

Per determinare l'andamento di tali dati sono necessari dei valori di riferimento. Si è voluta dare la possibilità ai medici di definire in modo dinamico i parametri ottimali: il grafo presenta un bottone di impostazioni dal quale si possono cambiare i criteri di valutazione, adattandoli allo stato di salute e all'età del paziente.

L'altezza degli indicatori mostra, pertanto, la percentuale del valore in questione rispetto al dato ottimale: i valori compresi tra zero e uno indicano un comportamento non conforme ai suggerimenti dei medici, mentre quelli maggiori di uno mostrano un andamento positivo.

Le barre sono colorate con una scala di colori che va dal rosso (vicino allo zero) al verde (sopra l'uno) per evidenziare in modo più efficace il comportamento del paziente.

Cambiando la finestra temporale il grafo si adegua in modo dinamico: aggrega i dati giornalieri per settimana o per mese e rielabora gli indicatori. In questo modo, sia i dottori sia i pazienti hanno una visione chiara dell'evoluzione dello stile di vita di questi ultimi, potendolo, inoltre,

confrontare con le compilazioni dei questionari al fine di identificare eventuali corrispondenze tra i periodi di buona salute e l'andamento positivo dei valori.

3.2 React

Il *framework* React è una libreria JavaScript *open-source* per lo sviluppo di interfacce utente basate su componenti grafici. Il programmatore è supportato sia nella gestione degli stati dell'applicazione sia nel rendering degli elementi grafici sul DOM.

React è ampiamente espandibile tramite librerie aggiuntive importabili. Per quanto concerne l'aspetto grafico, ad esempio, è stata utilizzata la libreria React-Bootstrap che permette di integrare alcuni componenti grafici avanzati, pronti all'uso, all'interno di un'applicazione React. Infatti, Bootstrap è un CSS (Cascading Style Sheets) framework che mette a disposizione template di elementi per la costruzione di UI (User Interface), come *form*, bottoni, barre di navigazione e contenitori per la gestione del *layout*. Tali elementi, nella loro versione standard, si presentano già con un ottimo aspetto estetico e, in aggiunta, la modifica di alcuni attributi ne permette la personalizzazione. In questo modo, il programmatore deve fare poco ricorso a fogli di stile *ad hoc* per raggiungere risultati grafici di buon livello.

I concetti chiave di React sono due: i componenti (o viste) e gli stati.
Ogni nodo dell'albero DOM è un componente che può essere di tipo contenitore, quindi, prevedere alcuni componenti figli, o di tipo foglia. Ogni componente, a sua volta, può essere composto da componenti annidati. I componenti padre possono determinare la configurazione di quelli figli mediante il passaggio delle *props*, ad esempio, valori di tipi primitivi o oggetti JavaScript, oppure delle funzioni (*callback*) per accedere ai metodi del padre.
Lo stato di un componente è definito da un insieme di variabili locali, eventualmente inoltrate agli elementi figli tramite props, le cui variazioni producono un automatico *re-rendering* del DOM. Per definire il valore di uno stato vengono utilizzate particolari funzioni di React chiamate *hook*.

Il modello React può essere rappresentato come un flusso di dati unidirezionale che attraversa tre elementi:

- lo **stato,** passato alle viste e ai componenti figli;
- le **viste** che possono innescare delle azioni;
- le **azioni** che producono un aggiornamento dello stato.

Infine, il cambiamento dello stato viene propagato alle viste e ne provoca un *re-rendering* (il ciclo ricomincia dal primo punto).

Oltre al JavaScript, React mette a disposizione la sintassi JSX che permette di scrivere un codice HTML-like per definire i componenti e i loro attributi in modo più snello, tramite l'utilizzo dei tag.

3.2.1 I componenti di HealthApp

Durante il processo di definizione dei componenti per il *front-end* si è seguito un approccio finalizzato al loro riutilizzo. Molti elementi, infatti, vengono riproposti più volte, ma con piccole differenze. Ad esempio, le *home-page* per i diversi tipi di utenti hanno la stessa struttura che include la barra di navigazione laterale e un corpo principale, oppure, il componente che mostra il dettaglio di un paziente (PatientDetails) è utilizzato anche come *form* per creare una nuova istanza di paziente: la *props* 'viewDetails' passata dal componente padre è un booleano che definisce se la vista deve essere configurata in modalità "lettura" o in modalità "creazione".

Per ognuno dei tre tipi di utente è stata implementata una home-page nella quale sono definiti un insieme di stati finalizzati a determinare la configurazione delle viste sottostanti e i dati da visualizzare. Ogni *home-page* presenta uno stato 'activeContent' di tipo enumerativo che indica la voce selezionata nella barra di navigazione laterale e, pertanto, il componente da visualizzare nel corpo principale.
Gli stati che rappresentano i dati provenienti dal database sono popolati tramite la chiamata di funzioni asincrone alle API esposte dal back-end. Ogni *home-page* ha bisogno dei seguenti dati da propagare alle viste figlie:

- HomeAdmin, necessita degli stati ‘doctorList’, ‘patientList’, ‘questionnaireList’ per contenere rispettivamente la lista dei dottori, dei pazienti e dei possibili questionari presenti nel database.
- HomeDoctor che richiede gli stati ‘doctor’ (collegati al profilo del dottore autenticato), ‘patientList’ e ‘questionnaireList’.
- HomePatient che ricorre allo stato ‘patient’, il quale è associato al profilo del paziente autenticato. I dati relativi alle compilazioni dei questionari e ai dati sul benessere sono contenuti negli stati dei componenti di livello gerarchico più basso, come UserSurveys e PatientHealthStatus.

Nel file API.js sono definite tutte le possibili chiamate agli *endpoint* esposti dal *back-end*, sotto forma di funzioni asincrone che sfruttano la Fetch API di React. Esse permettono di effettuare richieste HTTP indicando il tipo di richiesta e l’URL. Il programmatore deve gestire i possibili codici di stato HTTP della risposta e, in caso di successo (200 OK), mappare il corpo della risposta JSON in un oggetto di una classe JavaScript.
Di seguito due esempi di una chiamata POST per creare un nuovo paziente e una chiamata GET per ottenere la lista dei dottori.

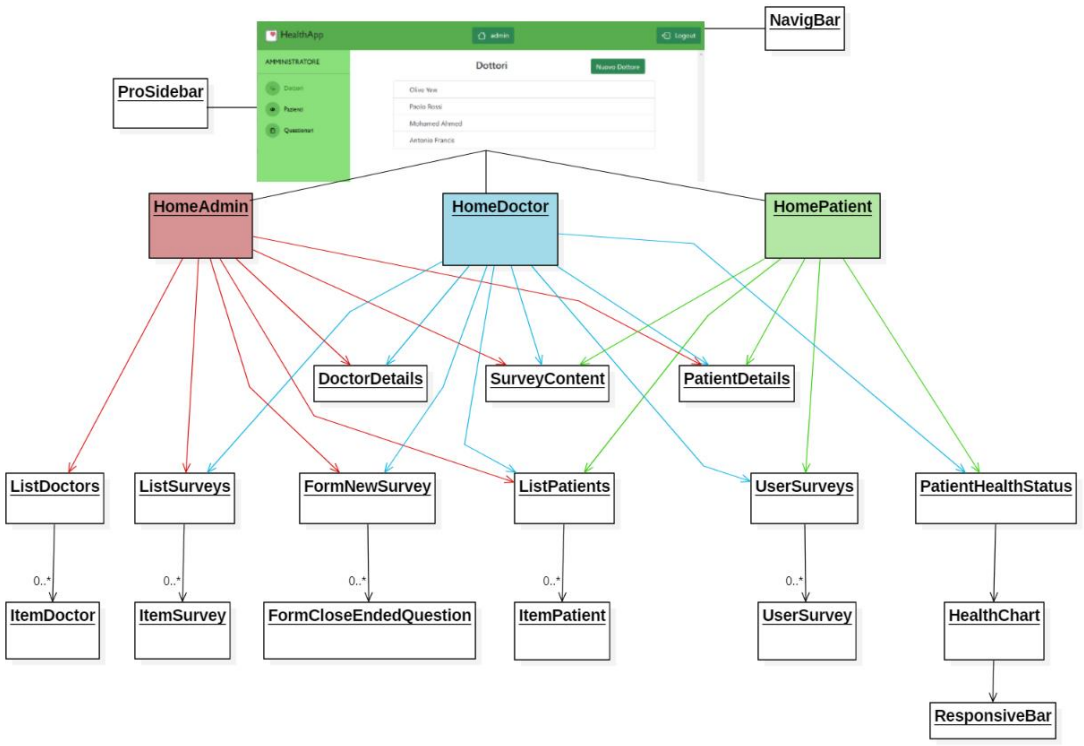
```
async function addPatient(patient) {
    return new Promise ((resolve, reject) => {
        fetch(`/health-core/api/patients`, {
            method: 'POST',
            headers: {'Content-Type': "application/json"},
            body: JSON.stringify(patient)
        })
        .then((response) => {
            const patientId = response.json()
            if (response.ok){
                resolve(patientId);
            } else {
                reject(patient);
            }
        })
        .catch(err => { reject ({'error': 'Cannot communicate with the
server'})})
    })
}
```

Il metodo *JSON.stringify()* è utile per serializzare un oggetto di una classe JavaScript (Patient in questo caso) in un oggetto JSON formattato tramite stringa, come richiesto dal Content-Type della chiamata.

```
async function getDoctors() {
    const response = await fetch('/health-core/api/doctors');
    const doctorsJson = await response.json();
    if (response.ok){
        return doctorsJson.map((s) =>
            Doctor.from(s)
        );
    } else {
        throw doctorsJson;
    }
}
```

Per ogni classe di dominio che definisce i modelli dell'applicazione è stato implementato un metodo statico *from(…)* che prende in input un oggetto JSON ritornando un'istanza della classe. In figura 3.6 viene mostrato un diagramma che rivela i principali componenti React definiti nell'applicazione; le frecce indicano la possibile relazione gerarchica padre-figlio che può avvenire tra i vari componenti, nelle diverse pagine.

Figura 3.6



Per alcuni componenti di complessità maggiore sono state importate librerie di terze parti:

- la barra di navigazione laterale, la quale è definita mediante il componente ProSidebar che deriva dalla libreria esterna React Pro Sidebar, la cui *repository* è pubblicata su GitHub al link https://github.com/azouaoui-med/react-pro-sidebar;
- il grafico a barre verticali per la visualizzazione dei dati sul *fitness* rilevati dal *tracker* è stato implementato attraverso la libreria Nivo, anch'essa pubblicata su GitHub al link https://github.com/plouc/nivo.

3.2.2 React Router

L'approccio Single-page-application nello sviluppo di un'applicazione web richiede che, quest'ultima, sia in grado di auto-generare la pagina HTML e di popolarla con i dati provenienti dalle risposte del server. La libreria React Router permette di configurare delle regole di *routing* per specificare quali componenti visualizzare a seconda dell'URL richiesto; inoltre, consente la navigazione tra le diverse viste e il salto a URL specifici.

Le regole di *routing* sono definite nei tag Route che richiedono di indicare l'attributo *path* (l'URL per il quale la *route* in questione è selezionata) e il componente da mostrare, configurato con le opportune *props*.

Il *front-end* di HealthApp gestisce una lista di *route* suddivise in tre parti:

- le *route* che iniziano con il percorso */admin/* e che producono il *rendering* del componente HomeAdmin. Ad esempio:

```
<Route exact path="/admin/patients" render={() =>
  <HomeAdmin activeContent={PATIENT}/>
}/>
```

- Le *route* che iniziano con il percorso */doctors/* le quali elaborano il *rendering* del componente HomeDoctor. Ad esempio:

```
<Route exact path="/doctors/:doctorId/profile" render={() =>
  <HomeDoctor activeContent={DOCTOR}/>
}/>
```

- Le *route* che iniziano con il percorso */patients/* associate al *rendering* del componente HomePatient. Ad esempio:

```
<Route exact path="/patients/:patientId/questionnaires" render={() =>
  <HomePatient activeContent={QUESTIONNAIRE}/>
}/>
```

La valorizzazione della *prop* 'activeContent' configura quale contenuto caricare nel corpo principale della *home-page*.

Tutte le *route* definite servono contenuti statici generati dall'applicazione React. Al contrario, se un URL non è intercettato da alcuna *route*, probabilmente, si tratta di una richiesta da inoltrare al *back-end* per recuperare dei contenuti dinamici. Tale operazione spetta al *proxy*.

Capitolo 4

Il deployment

Durante l'intera fase di sviluppo, l'applicazione React del *front-end* e quella Spring del *back-end* sono state eseguite in locale sulla stessa macchina, ed erano raggiungibili, rispettivamente, attraverso la porta 3000 e la porta 8080.
Nel file di configurazione 'package.json' dell'applicazione React è stato definito un *proxy* che permettesse di inoltrare alcune specifiche richieste al *back-end* (ad esempio per recuperare la lista dei pazienti, dei dottori o dei questionari).

```
"proxy": "http://localhost:8080"
```

L'aggiunta di questa riga di codice esegue un re-direzionamento di tutte le chiamate URL (esclusivamente quelle con intestazione Accept diversa da '*text/html*') non direttamente gestite da React verso l'*endpoint* del *back-end*. In particolare, tutte le *fetch* verso URL con percorso /health-core/api/ richiedono di essere servite dal *server* Spring.

Questa semplice soluzione è perfettamente funzionale durante lo sviluppo, ma non praticabile in fase di produzione, durante la quale le due applicazioni sono eseguite su macchine diverse.
La fase finale riguarda il *deployment* (o distribuzione) dell'applicativo, che consiste nell'individuazione dei *server* pubblici, raggiungibili da chiunque attraverso il web, sui quali ospitare l'esecuzione di *front-end* e *back-end*.

Il gruppo di ricerca Softeng del dipartimento DAUIN (Department of Control and Computer Engineering) del Politecnico di Torino, ha messo a disposizione i propri *server* per ospitare l'applicazione HealthApp.

4.1 Docker container

Entrambe le applicazioni, React e Spring, richiedono di essere compilate, insieme alle loro dipendenze, prima di poterle eseguire. La fase di compilazione è un processo strettamente legato alla macchina *host* e al suo sistema operativo. A differenza della fase di sviluppo, durante la quale il computer in cui viene eseguita l'applicazione è sempre lo stesso (quello del programmatore), nella fase di distribuzione sarebbe utile non avere dipendenze dalla macchina ospitante.

Nell'industria software si sono utilizzate per anni le Macchine Virtuali (VM): un'emulazione di un computer, compreso il sistema operativo, su cui è possibile installare tutte le dipendenze necessarie per eseguire il software. Tuttavia, diventa difficile usare le VM come strumento di *delivery*, in quanto si dovrebbe estrarre l'intera immagine, compreso il sistema operativo, e tale operazione renderebbe il processo lungo e poco pratico.

Il processo di *deployment* di un software si è evoluto con i *container*, diventati popolari grazie a Docker, una piattaforma che fornisce un insieme di prodotti PaaS (Platform as a Service) per lo sviluppo e il rilascio di applicazioni in *container*, per l'appunto.
L'immagine di un *container* è una sorta di involucro dell'applicazione, di tutte le sue dipendenze, del codice applicativo e delle librerie; essa è indipendente dall'ambiente esterno e pronta per essere facilmente spostata ed eseguita correttamente su qualsiasi sistema *host*.

Per il rilascio e la distribuzione di HealthApp si sono utilizzati i Docker Container, eseguiti sui *server* di Softeng Polito. Sono state create due immagini separate, una per l'applicazione React e una per il *back-end* Spring. La definizione delle immagini avviene attraverso la configurazione di un Dockerfile, nel quale sono indicati i passaggi necessari per installare il codice sorgente e le dipendenze, a partire da una *container image* già esistente (la quale emula un sistema *host* sottostante).

Di seguito il Dockerfile per il *back-end*:

```
FROM adoptopenjdk/openjdk11:alpine-jre
ARG JAR_FILE=target/*.jar
COPY ${JAR_FILE} app.jar
EXPOSE 8080
ENTRYPOINT ["java","-jar","/app.jar"]
```

Viene specificata l'immagine di partenza (Alpine è una versione di Linux leggera, semplice e sicura, adatta alla distribuzione software) alla quale aggiungere il *layer* dell'applicazione Spring Boot. Il file Jar che contiene il codice compilato e le dipendenze, pronto per essere eseguito, viene copiato nel sistema e attraverso il comando ENTRYPOINT si definiscono le istruzioni del Command Prompt per lanciare l'applicazione.

Di seguito il Dockerfile del *front-end*:

```
FROM node:12-alpine as build-stage
WORKDIR /app
COPY package.json .
COPY package-lock.json .
COPY . .
RUN npm i
RUN npm run build

FROM nginx:1.15-alpine
COPY --from=build-stage /app/build/ /usr/share/nginx/html
COPY --from=build-stage /app/nginx.conf /etc/nginx/conf.d/default.conf
```

In questo caso, la creazione dell'immagine avviene in due fasi:

- nella prima, identificata come 'build-stage' viene definita un'immagine di partenza, vengono copiati file di configurazione e l'intera cartella del codice sorgente, infine, vengono eseguiti i comandi *install* e *run build* per installare le dipendenze ed eseguire l'applicazione;
- nella seconda, viene implementato un web *server* Nginx sul quale è copiata la precedente immagine.

Una volta create le due immagini, esse possono essere pubblicate sulla *repository* ufficiale DockerHub, per essere scaricate ed eseguite ovunque. A questo punto, le due applicazioni sono pronte e funzionanti singolarmente, tuttavia, non sono correttamente configurate per

comunicare tra loro. Infatti, in fase di sviluppo, il *client* riusciva a raggiungere il *server* grazie al *proxy* nativo di React che re-indirizzava le chiamate alle API verso l'endpoint localhost:8080 che esponeva il *back-end*. In produzione, *client* e *server* girano su due *host* diversi raggiungibili da URL specifici (con dominio Softeng).

4.1.1 Nginx web server

Il *proxy* nativo React non è utilizzabile in produzione, pertanto è stato necessario trovare un'altra soluzione che svolgesse la funzione di *reverse proxy*. Come si evince nella fase due del Dockerfile del *front-end*, l'applicazione React viene copiata dentro un'istanza di un *web server* Nginx. Oltre a servire i contenuti statici dell'applicazione web, Nginx è usato anche per funzionalità di *reverse proxy*, HTTP *cache* e *load balancer*. Il file nginx.conf ne definisce la configurazione:

```
server {
    listen 3000;

    location /health-core/ {
        proxy_pass https://softeng.polito.it/health-core/;
    }

    location / {
        root /usr/share/nginx/html;
        index index.html index.htm;
        try_files $uri $uri/ /index.html =404;
    }
}
```

Il *server* si mette in ascolto sulla porta 3000.

Grazie alla prima voce *location*, tutte le richieste HTTP del *client* che presentano un URL con primo segmento di percorso /health-core/ (ovverosia, tutte le *fetch* alle API del *back-end*), sono inoltrate all'indirizzo del *back-end server*. La risposta è mandata a Nginx e ritornata al browser.

La seconda voce *location*, invece, riguarda tutte le altre richieste effettuate dal browser, la cui risposta è statica perché auto-generata dall'applicazione React.

I due container del *front-end* e del *back-end* sono in esecuzione sui *server* Softeng Polito, rispettivamente agli indirizzi:

https://softeng.polito.it/health-client

https://softeng.polito.it/health-core

CAPITOLO 5

Conclusioni

HealthApp è un progetto in continua evoluzione. Negli ultimi mesi si sono susseguiti diversi incontri con i medici per provare ad individuare le richieste e definire le specifiche. Comprendere quali informazioni siano rilevanti e quali no è un percorso complesso, dipendente da diversi fattori e che richiederà qualche mese di sperimentazione pratica dell'applicazione su studi reali. Infatti, gli sviluppatori non possiedono le conoscenze mediche per decidere in autonomia come elaborare i dati, in particolare le misurazioni derivate dai *tracker*, per estrapolare degli indicatori sulla qualità di vita. Anche se la tecnologia dei *wearable* è arrivata ad un livello di sensibilità e accuratezza elevato, definirli strumenti medici è eccessivo, pertanto, la differenza deve farla l'interpretazione dei dati.
Dall'altro lato, i medici, difficilmente riescono a visualizzare un modo efficace di rappresentare digitalmente la grande mole di informazioni proveniente dai *tracker*. Servirebbe una sinergia continuativa tra le due figure professionali per ottimizzare le potenzialità dell'applicazione.

Nelle prime fasi di sviluppo è stato proposto ai medici l'inserimento di una *dashboard* più dettagliata rispetto a quella attuale, contenente diverse strutture grafiche che permettessero di rappresentare in modo efficace i dati sul benessere del paziente. Tuttavia, il rischio sarebbe stato di duplicare le informazioni che tutti possono trovare sulle app proprietarie dei dispositivi *wearable*. Per i medici sarebbero state informazioni fuorvianti e lontane dallo scopo dell'applicazione, pertanto, si è scelta una soluzione più snella e con un impatto visivo immediato.

Nei prossimi mesi inizierà il primo ciclo di studio su un gruppo di pazienti. Questo primo test avrà un notevole impatto sugli sviluppi futuri: i medici, utilizzando quotidianamente

l'applicazione, potranno avere un'idea più chiara sulle specifiche da implementare e i miglioramenti da attuare. I pazienti avranno dei *feedback* la cui efficacia dovrà essere misurata, e, eventualmente, migliorata, per raggiungere l'obiettivo prefissato: dimostrare che un miglioramento della qualità di vita e del benessere personale va di pari passo con il miglioramento degli indicatori.
Per questa ragione, conclusioni puntuali sull'efficacia dell'applicazione si avranno al termine del primo ciclo di studio.

5.1 Sviluppi futuri

Le basi sulle quali si fonda il progetto permettono diverse evoluzioni dell'applicazione in futuro. La prima, attualmente in fase di sviluppo, prevede l'integrazione di una applicazione mobile nell'eco-sistema HealthApp, indirizzata in modo particolare ai pazienti. Questa, oltre a mostrare un livello di dettaglio maggiore sui rilevamenti dei tracker, introdurrà due funzionalità: il monitoraggio della nutrizione ed un servizio di raccomandazioni.

L'educazione alimentare è uno dei fattori di maggiore rilevanza quando si tratta del benessere delle persone. Infatti, nel libro di L. Fontana [1] vi è un intero capitolo dedicato alla corretta alimentazione ed ai suoi benefici. Tuttavia, avere una rilevazione affidabile sull'assunzione giornaliera di determinati cibi è un processo complicato e di difficile automazione.
I recenti studi nell'ambito dell'intelligenza artificiale forniscono le prime soluzioni per riconoscere gli ingredienti dei piatti da una semplice immagine; i limiti restano comunque evidenti specialmente nel caso di preparazioni culinarie più complesse che modificano la forma originale degli alimenti.
Nell'applicazione mobile, l'acquisizione delle informazioni alimentari avverrà tramite questionari esposti attraverso una UI immediata e semplice, finalizzata a ridurre al minimo lo sforzo necessario per garantire l'inserimento giornaliero e costante dei dati.

Per quanto concerne la seconda funzionalità, i pazienti appartenenti al gruppo di controllo saranno abilitati a ricevere eventi di raccomandazioni sotto forma di notifiche, in modo da aumentare gli stimoli a seguire le indicazioni per un corretto stile di vita. Le raccomandazioni

saranno generate da una logica attuata sul *back-end*, al termine di un processo di analisi dei dati di ogni utente volto a rilevare eventuali mancanze.

Per quanto concerne la connessione con i *wearable*, la dipendenza da terzi è ancora troppo importante. L'attuale soluzione richiede la creazione di un account Fitbit per ogni paziente e la gestione del processo di autenticazione per recuperare i dati dai loro *server*. Pur considerando che la logica è estendibile anche ad altri produttori e a diversi dispositivi, l'ideale sarebbe connettersi a basso livello con il *tracker* per evitare di usare API terze. Per realizzare tale obiettivo, bisogna, però, sviluppare algoritmi per l'interpretazione dei dati grezzi provenienti dai sensori del dispositivo e risolvere le problematiche di criptazione e identificazione degli attributi del GATT *server*, eventualmente ricercando sul mercato dispositivi più accessibili. Un'altra possibile evoluzione sarebbe integrare, nel gruppo di sviluppo di HealthApp, personale con competenze hardware per progettare un dispositivo *wearable ad hoc*.

Con il tempo, l'applicazione potrebbe estendersi e diventare a tutti gli effetti una piattaforma sulla quale mantenere diversi dati dei pazienti in fase di recupero dopo un periodo ospedaliero. Dopo aver raggiunto una base di dati consistente si potranno effettuare analisi approfondite atte a migliorare le politiche di gestione dei feedback, con il fine ultimo di trovare modi sempre più efficaci per persuadere le persone ad avere una maggiore cura del proprio benessere fisico.

Bibliografia

[1] Fontana L., *The Path to Longevity: The Secrets to Living a Long, Happy, Healthy Life*, Hardie Grant, 2020

[2] Department of Data and Analytics - Division of Data, Analytics and Delivery for Impact WHO, *WHO methods and data sources for life tables 1990-2019*, 2020

[3] Pilotto A., Ferrucci L., Francheschi M., *Development and validation of a Multidimensional Prognostic Index for 1-year mortality from the Comprehensive Geriatric Assessment in hospitalized older patients*, Rejuvenation Res 2008; 11: 151-61

[4] Pilotto A., Custodero C., Maggi S., Polidori MC., Veronese N., Ferrucci L., *A multidimensional approach to frailty in older people*, in *Ageing Res Rev 2020*; 60: 101047

[5] Warnier RM., van Rossum E., van Velthuijsen E*., Validity, Reliability and Feasibility of Tools to Identify Frail Older Patients in Inpatient Hospital Care: A Systematic Review*, in *J Nutr Health Aging 2016*; 20: 218-30

[6] Dent E., Martin FC., Bergman H*., Management of frailty: opportunities, challenges, and future directions,* in *Lancet 2019*; 394: 1376-1386.

[7] Townsend K., Cufì C., Akiba, Davidson R., *Getting Started with Bluetooth Low Energy,* O'Reilly Media, 2014

[8] Bluetooth SIG, *16-bit UUID Numbers Document*, 2022

[9] D. Hardt, *RFC-6749 The OAuth 2.0 Authorization Framework, IETF, Microsoft*, October 2012

[10] N. Sakimura, *RFC-7636 Proof Key for Code Exchange by OAuth Public Clients, IETF, Nomura Research Institite Google*, September 2015

[11] D. Eastlake 3rd, *RFC-6234 US Secure Hash Algotithms (SHA and SHA-based HMAC and HKDF), Huawei T. Hanssen AT&T Labs*, May 2011

[12] S. Josefsson, *RFC-4648 The Base16, Base32, and Base64 Data Encodings, Network Working Group,* 2006

Ringraziamenti

Ringrazio tutti coloro che, durante questo lungo percorso, sono stati un supporto morale, ognuno a modo proprio, chi tramite consigli, chi tramite lo svago.

In particolare, vorrei ringraziare quelle persone che hanno speso il loro tempo per aiutarmi negli ultimi mesi.
Grazie a Stefano e Andrea perché, pur non essendo dovuto, vi siete sempre resi disponibili.
Grazie a Martina perché, pur essendo dovuto, ti sei impegnata e interessata quasi come se fosse un tuo traguardo; è stato apprezzato, grazie.

Infine, un ringraziamento speciale va alla persona che più mi ha spinto a perseverare per raggiungere l'obiettivo di quando ero ragazzino. Grazie a Diego, perché solo tu conosci le difficoltà attraversate e l'impegno che ci ho messo.