UNIVERSITÀ DEGLI STUDI DI BERGAMO

Dipartimento di
Ingegneria

Corso di laurea in
Ingegneria Informatica

Classe n. LM-32 - Ingegneria informatica

Soluzione software per la raccolta dei parametri vitali da dispositivi wearable via API e per il follow-up clinico.

Candidato:
Andrea Roggeri

Matricola n.
1079033

Relatore:
Chiar.mo/Chiar.ma Prof. (Prof.ssa) Nome Cognome

Correlatore (se applicabile):
Dott. (Dott.ssa) Nome Cognome

Anno Accademico 2024/2025

Ringraziamenti

Desidero ringraziare il Prof. Angelo Gargantini e la Prof.ssa. Silvia Bonfanti, relatore e correlatore di questa tesi, per la disponibilità e cortesia dimostratemi, e per tutto l'aiuto fornito durante la stesura.

Un sentito ringraziamento ai miei colleghi ed amici, per essermi stati vicini sia nei momenti difficili, sia nei momenti felici.

Vorrei infine ringraziare con affetto la mia famiglia per il sostegno ed il grande aiuto che mi hanno dato durante tutto il percorso universitario e per avermi permesso di perseguire i miei obiettivi.

Andrea Roggeri

Abstract

Questa soluzione rappresenta una piattaforma web progettata specificatamente per semplificare il rilevamento e la valutazione dei parametri vitali nel contesto sanitario. Progettata con Python e Flask, fornisce un sistema composto destinato a medici e personale sanitario che vogliono seguire, analizzare e valutare gli impatti delle terapie elaborate mediante monitoraggio dei parametri vitali.

La piattaforma include funzionalità come la connessione ai device wearable (Fitbit) mediante API, visualizzazione dei trend temporali e la generazione di rapporti clinici personalizzati. L'architettura modulare permette una chiara separazione della gestione dei dati, dell'interfaccia utente e delle funzionalità di sicurezza, fornendo scalabilità e manutenibilità del codice.

Tra le sue funzionalità include un sistema di osservazioni cliniche che permette ai dottori di annotare interpretazioni e note su specifiche sessioni di registrazione dei parametri vitali e favorisce la correlazione tra le terapie somministrate e la risposta fisiologica del paziente. La piattaforma permette inoltre una condivisione del paziente tramite identificatori univoci (UUID), consentendo la consultazione fra specialisti di discipline differenti, rispettando nel frattempo standard di sicurezza e tracciabilità mediante un sistema di audit completo.

Questa tesi esplora la progettazione, implementazione e validazione della soluzione concepita e il suo impatto potenziale su una concreta applicazione in campo sanitario.

Indice

Capitolo 1

Introduzione

1.1 Contesto e motivazione

Negli ultimi anni, grazie all'avvento del COVID-19, si è assistito ad un'incremento della diffusione di tecnologie a distanza. In questo contesto, anche la telemedicina ha visto uno sviluppo considerevole. Si stima che la sua adozione sia raddoppiata in seguito alla pandemia nei paesi Ocse[1]. Per molti medici però, sopratutto nel contesto italiano nel quale più che in altri paesi la rottura con il passato è difficoltosa, l'adozione di tecnologie digitali per il monitoraggio dei pazienti risulta ancora poco affermata[2]. In molti casi, come ad esempio nel decorso post-operatorio, la rilevazione dei parametri vitali dei pazienti risulta quantomeno essenziale per una corretta formulazione di evenutali terapie e suggerimenti per una migliore guarigione. Risulta quindi chiaro che l'implementazione nel contesto sanitario di un sistema di monitoraggio remoto dei parametri vitali dei pazienti possa essere un grande aiuto per migliori diagnosi e terapie. È però necessario soffermarsi su alcuni aspetti che potrebbero rendere difficile l'adozione di tali tecnologie. Infatti, nonostante la disponibilità di dispositivi wearable, esistono ancora barriere significative all'integrazione di questi dispositivi nell'uso quotidiano in ambito clinico. Tra queste barriere possiamo trovare:

- Interoperabilità limitata tra dispositivi e sistemi clinici esistenti
- Frammentazione dei dati su piattaforme proprietarie non facilmente accessibili
- Mancanza di strumenti che facilitino l'analisi clinica dei dati raccolti
- Preoccupazioni relative alla sicurezza e alla privacy dei dati sanitari
- Necessità di standardizzazione nella raccolta e nell'analisi dei dati

In questo scenario, emerge la necessità di una piattaforma software integrata che permetta di colmare il divario tra i dispositivi wearable attualmente in commercio e i sistemi sanitari, consentendo ai medici di accedere, analizzare e valutare efficacemente i dati dei pazienti per migliorare diagnosi, trattamenti e follow-up clinico.

1.2 Obiettivi del progetto

Il progetto VitaLink nasce con l'obiettivo di sviluppare una piattaforma web in grado di fornire ai medici e agli operatori sanitari la possibilità di integrare nel loro flusso di lavoro un nuovo strumento, capace di semplificare e migliorare quello che è il monitoraggio dei parametri vitali dei pazienti. La piattaforma si propone di affrontare le problematiche sopra menzionate, fornendo un sistema integrato e modulare per la raccolta, l'analisi e la valutazione dei dati sanitari dei pazienti.

- **Raccolta e integrazione dei dati**: Sviluppare una piattaforma capace di interfacciarsi con le API di dispositivi wearable (inizialmente Fitbit nativamente) per raccogliere i dati senza la necessità di salvarli in maniera permanente, ma solo per il tempo necessario alla consultazione.
- **Visualizzazione e analisi**: Creare un'interfaccia intuitiva user-friendly che permetta ai professionisti sanitari di visualizzare, analizzare e interpretare i dati raccolti, identificando possibili anomalie e permettendo di formulare diagnosi e terapie adeguate.
- **Follow-up clinico strutturato**: Implementare un sistema di osservazioni cliniche che consenta ai medici di documentare interpretazioni e mettere in correlazione dati analizzati e eventuali terapie in atto.
- **Condivisione sicura dei dati**: Realizzare un meccanismo di condivisione dei pazienti tra diversi specialisti, garantendo sicurezza e privacy.
- **Generazione di report**: Sviluppare funzionalità per la creazione di report personalizzati che integrino osservazioni, note mediche e dati analizzati.
- **Scalabilità e flessibilità**: Progettare un'architettura modulare che permetta future implementazioni di altri dispositivi e facilitino l'integrazione di nuove funzionalità.

Il progetto si propone, in ultima battuta, di sviluppare un sistema integrabile senza difficoltà in sistemi già esistenti(come ad esempio Google Cloud e Amazon AWS) in maniera da rendere il più semplice possibile l'adozione.

1.3 Struttura della tesi

La tesi è strutturata in otto capitoli principali che seguono i principi dello sviluppo dell'ingegneria del software, seguiti da appendici tecniche.

- Il **Capitolo 1** introduce il contesto di sviluppo del progetto, le motivazioni e gli obiettivi.
- Il **Capitolo 2** esamina lo stato dell'arte delle piattaforme sanitarie digitali e le tecnologie di integrazione con dispositivi wearable.
- Il **Capitolo 3** approfondisce l'analisi dei requisiti funzionali e non funzionali, presentando casi d'uso e user stories.
- Il **Capitolo 4** descrive la progettazione del sistema, includendo l'architettura, i design pattern e il modello dei dati.
- Il **Capitolo 5** documenta l'implementazione, dettagliando lo stack tecnologico e la struttura del codice.
- Il **Capitolo 6** tratta il testing e la validazione del sistema.
- Il **Capitolo 7** illustra le strategie di deployment.
- Infine, il **Capitolo 8** valuta i risultati ottenuti e propone sviluppi futuri.

Le appendici includono un glossario tecnico, esempi di codice sorgente e diagrammi UML dettagliati.

Capitolo 2

Stato dell'arte

2.1 Piattaforme web per la salute

2.1.1 Soluzioni esistenti nel mercato

Ad oggi, nell'anno 2025, esistono numerose soluzioni che si propongono per fare da intermediarie tra i dispositivi wearable e i professionisti sanitari. Un'analisi di queste piattaforme risulta necessaria per comprendere le funzionalità da sviluppare e da offrire.

- **Validic**: *"Validic guida la trasformazione digitale con il più ampio ecosistema di dispositivi sanitari connessi, perfettamente integrati nelle cartelle cliniche elettroniche (EHR). Offriamo la soluzione di monitoraggio remoto dei pazienti più completa, con risultati comprovati su larga scala e un'IA generativa all'avanguardia."* [3].
- **Human API**: *"Human API è una piattaforma di dati sanitari controllata dai consumatori che offre ai tuoi utenti un modo semplice per connettersi e condividere i propri dati sanitari con la tua azienda, piattaforma o applicazione sanitaria. I nostri clienti aziendali (sia aziende sanitarie che start-up) utilizzano il nostro prodotto per creare e fornire app e servizi sanitari incentrati sui consumatori."* [4].
- **Health Mate di Withings**: *"Withings Health Mate è il modo migliore per tenere traccia dell'attività, del sonno, del peso e molto altro. Vedrai le tendenze, i progressi e riceverai coaching per aiutarti a migliorare nel tempo. Qualunque sia il tuo obiettivo di salute, troverai supporto nell'app Health Mate."* [5].
- **Mywellness**: *"Eliminando le barriere fra gli ambienti scelti dall'utente per praticare movimento e attività fisica, mywellness® cloud offre una gamma completa di*

applicazioni web e mobile, alle quali è possibile accedere dalle attrezzature Technogym e da qualsiasi dispositivo personale, che consente all'utente di gestire il proprio stile di vita e all'operatore di accedere a strumenti professionali per svolgere il proprio business in modo più efficace." [6].

- **Wellmo**: *"Wellmo è una piattaforma ecosistemica come servizio (PaaS). Con Wellmo, un orchestratore può rapidamente immettere sul mercato e migliorare costantemente servizi sanitari brandizzati e coinvolgenti forniti dai propri partner. Le funzionalità chiave della piattaforma Wellmo includono l'integrazione con centinaia di dispositivi sanitari connessi, un'app mobile e un'interfaccia web white label, un sistema di gestione dei contenuti, API di integrazione e analisi di utilizzo e risultati. Il percorso del cliente, la personalizzazione, la logica e i contenuti sono tutti configurabili, il che rende la creazione e il miglioramento dei servizi agili, convenienti e altamente produttivi. Wellmo, o una società di consulenza, può aiutare un orchestratore a implementare, migliorare e gestire i servizi."* [7].

- **LiVA Healthcare**: *"La nostra piattaforma digitale consente ai team sanitari di offrire programmi di stile di vita basati su prove concrete, aiutando i pazienti a gestire patologie a lungo termine attraverso comprovate tecniche di modifica del comportamento."* [8].

- **Doccla**: *"I servizi di monitoraggio clinico di Doccla offrono una supervisione continua, garantendo che qualsiasi segno di peggioramento venga rapidamente identificato e gestito. Il nostro servizio non solo migliora gli esiti clinici per i pazienti, ma riduce anche il carico clinico per i team sanitari, gestendo il monitoraggio di routine e implementando protocolli di escalation concordati. In qualità di fornitore regolamentato dal CQC, garantiamo i più elevati standard di assistenza e sicurezza."* [9].

2.1.2 Limiti delle soluzioni attuali

Nonostante la varietà di piattaforme disponibili, si riscontrano diverse limitazioni comuni:

- **Costo**: La maggior parte di queste soluzioni risulta essere chiusa come sistema, richiedendo costi piuttosto elevati per l'utilizzo.

- **Complessità di utilizzo**: Molte piattaforme, concentrandosi sul fornire servizi complessi e ricchi di funzionalità, trascurano l'aspetto della semplicità che può essere rilevante in un contesto di utilizzo per personale non specializzato nell'uso di tecnologie digitali.

- **Focus limitato**: Alcune piattaforme riducono la raccolta dati a categorie di parametri specifiche, senza offrire una visione olistica.

In questo contesto, VitaLink si pone l'obiettivo di colmare queste limitazioni e di sviluppare un sistema semplice ma efficace, in grado di gestire ogni tipo di dispositivo o di parametro vitale e sopratutto senza costi legati alla licenza.

2.2 Tecnologie per lo sviluppo web

2.2.1 Framework backend per applicazioni sanitarie

Esistono sul mercato molti framework per la gestione del backend; tra questi possiamo trovare[10]:

- **Django**: Uno dei framework più popolari per lo sviluppo di applicazioni web in Python.
- **Express**: Un framwork minimalista per Node.js, molto utilizzato per API RESTful.
- **Flask**: Un framework leggero per Python molto documentato e facile da usare, senza che ciò faccia mancare sicurezza e flessibilità.
- **ASP.NET**: Un framwork modulare open-source scritto in C Sharp.

2.2.2 Tecnologie frontend per la visualizzazione di dati clinici

Per il frontend, esistono una varietà di tecnologie che semplificano di molto la creazione di pagine responsive e interattive, permettendo agli sviluppatori di concentrarsi maggiormente sullo sviluppo effettivo del backend. Tra queste troviamo:

- **Bootstrap**: Il framework probabilmente più popolare per la creazione di pagine responsive e interattive. [11]
- **Flutter**: Un framework sviluppato da Google e ora open-source, per lo sviluppo cross-platform di soluzioni mobile, web e desktop. [12]
- **React**: Sviluppata da Facebook, è una libreria JavaScript per la creazione di interfacce utente. È molto popolare per la creazione di applicazioni web reattive. [13]

2.2.3 Database e persistenza dei dati sanitari

Dal punto di vista legislativo, in UE, è necessario rispettare le normative GDPR [14] per la privacy e la protezione dei dati personali. Soluzioni che che permettono di garantire il soddisfacimento di tali imposizioni risultano essenziali. Tra DBMS più comuni per la conservazione dei dati possiamo trovare:

- **MySQL**: Uno dei DBMS relazionali più popolari al mondo. Progettato per massimizzare velocità, scalbilità e affidabilità. [15]
- **PostgreSQL**: Un'altro DBMS relazionale. A differenza di MySQL, supporta una maggiore flessibilità per quanto riguarda i tipi di dati, scalabilità e integrità dei dati. [16]
- **MongoDB**: Un DBMS non relazionale(noSQL) orientato ai documenti. Permette di memorizzare in maniera efficente dati non strutturati o semi strutturati. [17]

2.3 Integrazione con dispositivi di monitoraggio della salute

2.3.1 Protocolli di comunicazione per dispositivi wearable

I dispositivi wearable, al giorno d'oggi, nella maggioranza dei casi necessitano ancora di sfruttare il collegamento con un dispositivo dotato di collegamento a internet per poter caricare i dati raccolti sul cloud. Lo standard adottato da quasi tutti i dispositivi wearable è il Bluetooth [18]. A questo vanno poi ad aggiungersi standard di nuova generazione come il Bluetooth Low Energy (BLE) [19], che permette di ridurre il consumo energetico e di aumentare la durata della batteria dei dispositivi e l'ANT+ [20].

2.3.2 OAuth 2.0 e autenticazione sicura

Per quanto riguarda l'autorizzazione per accedere alle API dei dispositivi wearable, lo standard più utilizzato è l'OAuth 2.0, il quale permette agli utenti di fornire l'accesso ai proprio dati ad un servizio richiedente senza dover condividere i propri dati di accesso. Lo standard si basa sul concetto di token; il servizio richiedente, dopo aver ricevuto l'autorizzazione da parte dell'utente, riceve un token di accesso con il quale può richiedere i dati all'API del dispositivo wearable. Questo tipo di autenticazione permette di garantire la protezione dei dati sensibili ed evitare che i dati di accesso possano venire compromessi.

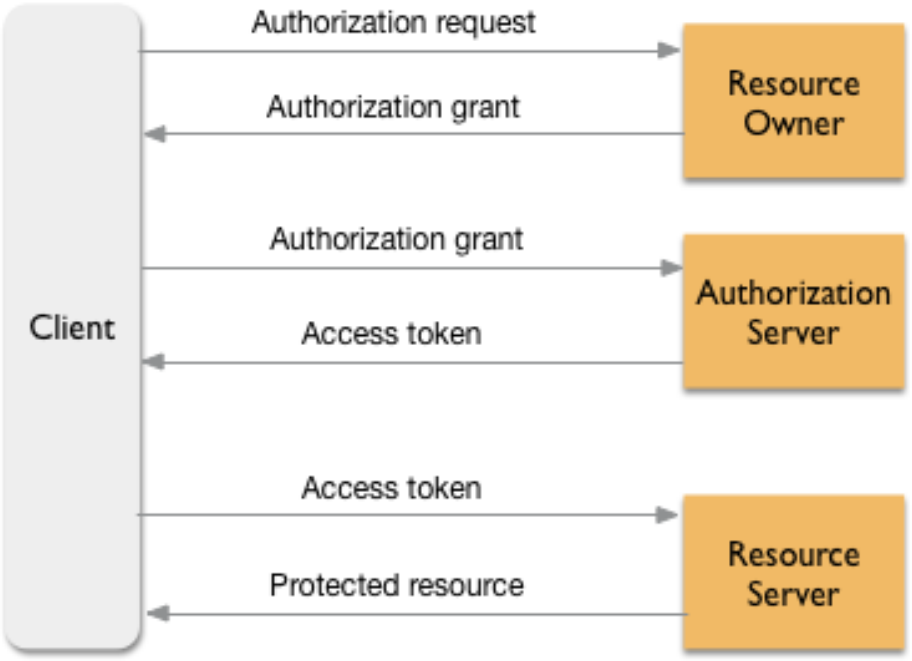

Figura 2.1: Flusso di autorizzazione OAuth 2.0 [28]

2.3.3 Piattaforme Fitbit e API disponibili

Fitbit è una marca di dispositivi wearable, aquisita nel 2019 da Google [21], che comprende una vasta gamma di prodotti i quali sono in grado di rilevare una grande quantità di dati. Una volta che l'utente ha connesso il proprio dispositivo alla piattaforma Fitbit, è possibile utilizzare una serie di API che permettono di accedere a tali dati. La piattaforma Fitbit adotta lo standard OAuth 2.0 per l'autenticazione. Tra i dati disponibili tramite le API Fitbit troviamo ad esempio:

- Battito cardiaco
- Calorie bruciate
- Calorie assunte
- Ossigenazione del sangue
- Attività fisica
- Sonno
- Temperatura corporea
- Temperatura della pelle

- Attività respiratoria

e molti altri.

Una nota da precisare è che quasi tutti i dati sono disponibili in formato intraday(ossia visibuli con una frequenza che va da qualche secondo a pochi minuti), ma per accedere a questo livello di dettaglio è necessario mandare una richiesta speciale a Fitbit per entrare in possesso dell'autorizzazione necessaria alla loro consultazione [22].

2.3.4 Altre piattaforme e loro integrazione

In commercio, oltre ai dispositivi Fitbit, esiste una vasta gamma di prodotti wearable per il monitoraggio dei parametri vitali con vari livelli di funzionalità: dai più economici ai più costosi. Tra questi possiamo trovare:

- **Garmin Smartwatch**: Dispositivi molto avanzati dotati delle tecnologie più evolute per il monitoraggio. Molto adottati dagli sportivi. [15]
- **Apple Watch**: Orologi smart progettati per un'integrazione perfetta con l'ecosistema Apple. L'accesso ai dati di tali dispositivi risulta più difficoltosa che su altre tipologie di dispositivi in quanto per ottenere l'accesso ai dati risulta necessario un dispositivo Apple. [24]
- **Amazfit**: Dispositivi la cui feature principale è il prezzo. Risultano infatti molto più competitivi (in linea generale) dal punto di vista economico rispetto ai dispositivi sopra citati. [25]

Capitolo 3

Analisi dei requisiti

Per l'analisi dei requisiti (funzionali e non funzionali) è stata utilizzata la metodologia MoSCoW [26], la quale permette di classificare i requisiti in quattro categorie: Must have (M), Should have (S), Could have (C) e Won't have (W).

3.1 Requisiti funzionali

3.1.1 Autenticazione e sicurezza - M

- Il sistema deve fornire un meccanismo sicuro di registrazione per i medici e gli operatori sanitari.
- Gli utenti devono potersi autenticare tramite email e password.
- Il sistema deve supportare l'autenticazione API tramite token JWT.
- Le password devono essere memorizzate con crittografia sicura (libreria Werkzeug [27]).

3.1.2 Gestione pazienti - M

- I medici devono poter creare nuovi record paziente con informazioni anagrafiche di base.
- I medici devono poter visualizzare la lista di tutti i loro pazienti.
- I medici devono poter visualizzare i dettagli completi di un paziente.
- I medici devono poter modificare le informazioni dei loro pazienti.
- I medici devono poter aggiungere note mediche ai record dei pazienti

3.1.3 Interfaccia Utente - M

- Il sistema deve fornire una dashboard per i medici che mostri statistiche rilevanti.
- L'interfaccia deve essere accessibile tramite browser web standard.
- L'interfaccia utente deve essere responsiva per supportare diversi dispositivi.
- La navigazione deve essere intuitiva e coerente in tutta l'applicazione.

3.1.4 API Base - M

- Il sistema deve fornire API RESTful per le operazioni CRUD sui pazienti.
- Le API devono supportare la ricerca e il filtraggio dei pazienti.
- Le API devono essere protette tramite autenticazione JWT.
- Le risposte API devono seguire standard coerenti e gestire gli errori in modo appropriato.

3.1.5 Integrazione con Piattaforme Sanitarie - S

- Il sistema dovrebbe integrarsi con Fitbit per recuperare dati sui parametri vitali.
- I medici dovrebbero poter visualizzare i parametri vitali del paziente in formato grafico.
- Il sistema deve supportare l'autenticazione API tramite token JWT.
- Il sistema dovrebbe supportare l'autenticazione OAuth con le piattaforme sanitarie.

3.1.6 Osservazioni sui Parametri Vitali - S

- I medici dovrebbero poter creare osservazioni sui parametri vitali dei pazienti.
- I medici dovrebbero poter modificare e eliminare le loro osservazioni.
- Il sistema dovrebbe supportare diversi tipi di parametri vitali (frequenza cardiaca, pressione sanguigna, ecc.).
- Le osservazioni dovrebbero essere visualizzabili in modo cronologico.

3.1.7 Internazionalizzazione - S

- L'interfaccia utente dovrebbe essere disponibile in italiano e inglese.
- Il sistema dovrebbe consentire agli utenti di cambiare facilmente lingua.
- Date e numeri dovrebbero essere formattati in base alle convenzioni locali.
- I messaggi di errore dovrebbero essere localizzati.

3.1.8 Reportistica - S

- Il sistema dovrebbe generare report base sui dati dei pazienti.
- I report dovrebbero essere esportabili in formati standard (PDF).
- I report dovrebbero essere inviabili al paziente tramite email.
- I medici dovrebbero poter personalizzare alcuni parametri dei report.
- I report generati dovrebbero essere archiviati per consultazioni future.

3.1.9 Integrazione con Ulteriori Piattaforme Sanitarie - C

- Il sistema potrebbe integrarsi con Apple Health.
- Il sistema potrebbe integrarsi con Google Fit.
- Il sistema potrebbe integrarsi con Garmin Connect.
- Il sistema potrebbe supportare dispositivi medici Bluetooth LE.

3.1.10 Collaborazione tra Medici - C

- I medici potrebbero condividere l'accesso ai pazienti con altri medici.
- Il sistema potrebbe supportare commenti collaborativi sulle note mediche.
- I medici potrebbero ricevere notifiche quando ci sono aggiornamenti sui pazienti condivisi.
- Il sistema potrebbe tenere traccia di chi ha effettuato modifiche ai record.

3.1.11 Autenticazione a Due Fattori - C

- Il sistema potrebbe supportare l'autenticazione a due fattori via SMS.
- Il sistema potrebbe supportare l'autenticazione a due fattori via app mobile.
- L'utente potrebbe configurare le preferenze di sicurezza del proprio account.
- Il sistema potrebbe richiedere 2FA per operazioni sensibili.

3.1.12 Analisi Avanzata dei Dati - C

- Il sistema potrebbe implementare algoritmi di rilevamento anomalie nei parametri vitali.
- Il sistema potrebbe offrire suggerimenti basati sui trend dei dati.
- Il sistema potrebbe generare report comparativi tra pazienti anonimi.
- Il sistema potrebbe supportare la visualizzazione di correlazioni tra diversi parametri.

3.1.13 Prescrizione Elettronica - W

- Il sistema non supporterà la generazione di prescrizioni elettroniche.
- Non ci sarà integrazione con farmacie o sistemi di prescrizione nazionali.
- Non sarà possibile tracciare l'aderenza ai farmaci prescritti.
- Non ci sarà un modulo di gestione inventario farmaci.

3.1.14 Cartella Clinica Elettronica Completa - W

- Il sistema non sostituirà una cartella clinica elettronica completa.
- Non ci saranno moduli per la gestione di esami di laboratorio.
- Non ci sarà integrazione con sistemi ospedalieri.
- Non ci sarà supporto per la gestione di immagini diagnostiche.

3.1.15 Telemedicina - W

- Il sistema non includerà funzionalità di videoconferenza.
- Non ci sarà supporto per consultazioni remote in tempo reale.
- Non ci saranno strumenti per la pianificazione di visite virtuali.
- Non ci sarà integrazione con sistemi di pagamento per visite telematiche.

3.1.16 App Mobile Dedicata - W

- Non verrà sviluppata un'app mobile dedicata nella prima fase.
- I pazienti non avranno accesso diretto al sistema.
- Non ci sarà supporto per notifiche push su dispositivi mobili.
- Non ci sarà funzionalità offline per l'app mobile.

3.2 Requisiti non funzionali

3.2.1 Sicurezza e Privacy - M

- Tutti i dati personali dei pazienti devono essere crittografati a riposo.
- Deve essere implementato un sistema completo di log per gli audit di sicurezza.

3.2.2 Performance - M

- Il tempo di risposta per le operazioni di base deve essere inferiore a 2 secondi.
- Il sistema deve supportare almeno 1000 utenti concorrenti.
- Il caricamento della dashboard non deve richiedere più di 3 secondi.
- Il sistema deve supportare la gestione di almeno 100.000 record paziente.

3.2.3 Disponibilità - M

- Il sistema deve avere un uptime del 99,9% durante le ore lavorative.
- I backup del database devono essere eseguiti quotidianamente.

3.2.4 Usabilità - M

- L'interfaccia utente deve essere utilizzabile senza formazione specifica ed essere user-friendly.
- I flussi di lavoro principali non devono richiedere più di 3 clic.
- I messaggi di errore devono essere chiari e fornire indicazioni per la risoluzione.

3.2.5 Scalabilità - S

- L'architettura dovrebbe supportare lo scaling orizzontale.
- Il database dovrebbe gestire efficacemente l'aumento di volume dei dati.
- Le prestazioni non dovrebbero degradarsi significativamente con l'aumentare degli utenti.
- Il sistema dovrebbe implementare tecniche di caching per migliorare la reattività.

3.2.6 Manutenibilità - S

- Il codice dovrebbe seguire standard di codifica e best practice.
- La documentazione del codice dovrebbe essere completa e aggiornata.
- L'architettura dovrebbe essere modulare per facilitare gli aggiornamenti.
- Il sistema dovrebbe supportare aggiornamenti con tempi di inattività minimi.

3.2.7 Portabilità - S

- Il sistema dovrebbe funzionare sui principali browser web (Chrome, Firefox, Edge, Safari).
- L'interfaccia utente dovrebbe adattarsi a diverse risoluzioni dello schermo.
- Il sistema dovrebbe essere containerizzato per facilitare il deployment.
- Il backend dovrebbe funzionare su diversi sistemi operativi server.

3.2.8 Interoperabilità - S

- Le API dovrebbero seguire standard RESTful.
- Il sistema dovrebbe supportare almeno il formato JSON per lo scambio di dati.
- Il sistema dovrebbe utilizzare formati standard per date, orari e dati medici.

3.2.9 Prestazioni Avanzate - C

- Il sistema potrebbe implementare tecniche di precaricamento dei dati.
- L'interfaccia utente potrebbe utilizzare rendering lato server per il caricamento iniziale.
- Il sistema potrebbe implementare la compressione delle risposte API.
- Il database potrebbe essere ottimizzato con indici avanzati e strategie di partizionamento.

3.2.10 Monitoraggio e Analytics - C

- Il sistema potrebbe implementare dashboard di monitoraggio in tempo reale.
- Le metriche di prestazione potrebbero essere raccolte e analizzate.
- Il sistema potrebbe implementare alerting automatico per problemi prestazionali.
- Gli errori utente potrebbero essere tracciati per identificare problemi di usabilità

3.2.11 Supporto Offline - C

- L'interfaccia utente potrebbe implementare funzionalità progressive web app.
- I dati critici potrebbero essere memorizzati nella cache del browser.
- Il sistema potrebbe supportare la sincronizzazione dei dati dopo la riconnessione.
- Le modifiche potrebbero essere accodate quando offline.

3.2.12 Testing Automatizzato - C

- Il sistema potrebbe avere una copertura di test unitari superiore all'85%.
- Il sistema potrebbe implementare test di carico programmati.
- Il processo di CI/CD potrebbe includere test di sicurezza automatizzati.

3.2.13 Supporto Legacy - W

- Non ci sarà compatibilità con sistemi operativi obsoleti.
- Non verranno fornite versioni desktop standalone.

3.2.14 Alta Disponibilità Geografica - W

- Il sistema non implementerà il deployment multi-regione nella prima fase.
- Non ci sarà failover automatico tra diversi data center.
- Non ci sarà ottimizzazione per utenti in regioni geografiche specifiche.
- Non ci sarà un sistema di content delivery network globale.

3.2.15 Integrazione Enterprise Completa - W

- Non ci sarà supporto per Single Sign-On aziendale completo.
- Non ci sarà integrazione con sistemi ERP legacy.
- Non ci saranno connettori personalizzati per ogni sistema clinico.

3.2.16 Conformità Internazionale Completa - W

- Il sistema non sarà inizialmente certificato per standard internazionali come HIPAA.
- Non ci sarà supporto completo per tutti i requisiti normativi regionali.
- Non ci saranno localizzazioni complete per tutti i paesi.
- Non ci sarà certificazione FDA come dispositivo medico.

3.3 Casi d'uso

3.3.1 Attori principali

I principali attori del sistema risultano essere i medici e gli operatori sanitari, i quali si interfacciano con il sistema per la gestione dei pazienti e dei dati clinici. I pazienti non hanno accesso diretto al sistema, ma possono comunque ricevere via email i report generati e sono necessari nel flusso di autenticazione necessario per concedere l'accesso dei proprio dati al sistema.

3.3.2 Gestione dei pazienti

I medici e gli operatori sanitari possono:

- Visualizzare i pazienti.
- Creare nuovi pazienti.
- Importare pazienti tramite UUID.
- Visualizzare i dettagli di un paziente.
- Modificare i dettagli di un paziente.
- Eliminare un paziente(dissociandolo dal proprio account).

3.3.3 Gestione dell'account

I medici e gli operatori sanitari possono:

- Registrare un nuovo account.
- Eseguire il login.
- Eseguire il logout.

3.3.4 Visualizzazione dei parametri vitali

I medici e gli operatori sanitari possono:

- Visualizzare grafici dei parametri vitali.
- Visualizzare i parametri vitali in formato di tabelle.
- Selezionare un livello di dettaglio per la visualizzazione dei dati (1g, 7g, 30g, 90g).

3.3.5 Gestione note

I medici e gli operatori sanitari possono:

- Creare nuove note cliniche.
- Visualizzare note esistenti.
- Eliminare note(solo quelle create da loro).
- Visualizzare i dettagli di un paziente.

- Modificare i dettagli di un paziente.
- Eliminare un paziente.

3.3.6 Gestione report

I medici e gli operatori sanitari possono:

- Generare report specifici relativi ad un parametro vitale.
- Generare un report generale relativo a tutti i parametri vitali.
- Selezionare gli elementi(note, osservazioni, grafici) da includere nel report.
- Scaricare il report in formato PDF.
- Inviare una copia per email al paziente del report in formato PDF.

3.3.7 Integrazione con piattaforme sanitarie

I pazienti possono:

- Autorizzare la connessione del proprio account alla piattaforma.

La piattaforma su cui sono salvati i dati può:

- Revocare l'autorizzazione di accesso al sistema.

I medici e gli operatori sanitari possono:

- Generare un link valido 24h per permettere all'utente di fornire l'accesso da parte della piattaforma ai suoi dati.
- Reimuovere il collegamento alla della piattaforma alla piattaforma su cui sono salvati i dati.

3.3.8 Gestione delle osservazioni cliniche

I medici e gli operatori sanitari possono:

- Creare un'osservazione clinica relativa ad un parametro vitale e ad un periodo temporale specifico.
- Visualizzare le note esistenti per ciascun paziente.
- Eliminare un'osservazione clinica(solo quelle create da loro).

3.4 User stories e scenari comuni principali

3.4.1 Registrazione e accesso

- Il medico o il personale sanitario accede al sistema collegandosi tramite un browser al link fornitogli.
- Si presentano due opzioni: registrazione e login.
- Per registrarsi l'utente clicca sul collegamento "Nuovo medico? Registrati Qui"(La lingua effettiva dipende dalla localizzazione selezionata).
- L'utente, seguendo le richieste specficate nella pagina, compila i campi con i propri dati, facendo attenzione a rispettare i requisiti di sicurezza della password.
- Se tutto è stato inserito correttamente, non esistono altri account nel sistema con la stessa email e il sistema non ha restituito errori, dopo aver premuto il pulsante "Crea Account" l'utente viene registrato e reindirizzato alla pagina di login.
- L'utente può accedere al sistema inserendo la propria email e password.
- L'utente, se i dati sono corretti e se il sistema non ha restituito errori, dopo aver premuto il pulsante "Accedi" viene reindirizzato alla Dashboard del sistema.

3.4.2 Importazione e gestione pazienti

- Il medico o il personale sanitario accede e si trova davanti alla Dashboard.
- Decide di aggiungere un paziente: clicca sulla shortcut nella Dashboard "Nuovo Paziente" nel Menù Azioni rapide(oppure va nella pagina "Visualizza tutti i pazienti" e clicca sul pulsante "Aggiungi nuovo paziente").
- Nel form che si apre, l'utente compila i campi obbligatori richiesti e, se desidera, quelli opzionali.
- Dopo aver compilato il form, se i dati inseriti sono validi e se il sistema non ha restituito errori, dopo aver cliccato il pulsante "Salva Paziente" verrà creato un nuovo account.
- Dalla schermata "Visualizza tutti i pazienti" l'utente può importare un paziente esistente tramite UUID cliccando sul pulsante "Importa Paziente tramite UUID".
- Nel modale che si aprirà, l'utente inserirà il codice di associazione richiesto.

- Se il codice di associazione inserito è valido e associato ad un account paziente esistente (e se il sistema non ha restituito errori), dopo il click sul pulsante "Importa Paziente" il paziente verrà aggiunto alla lista dei pazienti seguiti dall'operatore.

3.4.3 Collegamento dispositivi wearable

- Il medico o il personale sanitario accede e si trova davanti alla Dashboard.
- Decide di collegare un dispositivo wearable al profilo di un paziente: clicca sul pulsante "Visualizza tutti i pazienti" e, sulla lista dei pazienti, clicca sul pulsante Azione "Visualizza i parametri vitali".
- Nella pagina che si trova davanti, l'utente clicca sul pulsante "Health Sync".
- Viene aperto un modale che mostra un QR Code (per permettere ad un paziente in visita al medico di collegarsi scansionandolo) ed un link testuale che può essere condiviso con il paziente tramite email o messaggio.
- Il paziente, una volta ricevuto il link, cliccando su di esso viene reindirizzato alla pagina di autorizzazione all'accesso ai propri dati da parte del sistema, la quale può essere raggiungibile per 24h (o comunque fino a che la procedura di autorizzazione non sarà riuscita). Se il link non dovesse essere più valido il paziente verrà informato di questo.
- Il paziente clicca sul servizio relativo al dispositivo che possiede (inizialmente saranno supportati solo i dispositivi Fitvbit).
- Si apre la pagina di login della piattaforma scelta, la quale dopo aver effettuato l'accesso chiede all'utente di autorizzare una certa serie di permessi al sistema richiedente.
- Una volta concessi, l'utente viene reindirizzato alla pagina dei servizi disponibili e gli viene comunicato l'esito dell'operazione(fallita o riuscita).
- Il medico è ora collegato al dispositivo del paziente e può visualizzare i dati relativi ai parametri vitali nella schermata in cui ha cliccato "Health Sync".

3.4.4 Visualizzazione e interpretazione dei dati

- Il medico o il personale sanitario accede e si trova davanti alla Dashboard.
- Clicca su "Visualizza tutti i pazienti" e clicca sull'azione "Visualizza Paziente".

- Qui il medico può intragire con le note relative al paziente, oltre che a visualizzare i relativi dati di registrazione.
- Clicca sul pulsante "Visualizza i parametri vitali" e si apre la pagina con i dati relativi al parametri vitali del paziente.
- Il medico, dopo che è stato effettuato il collegamento al dispositivo wearable, può visualizzare i dati relativi ai parametri vitali in formato tabellare o grafico.
- Il medico può decidere il livello di dettaglio di visualizzazione dei dati andando a cliccare sul pulsante relativo al periodo scelto(1g, 7g, 30g, 90g).
- Il medico può cambiare il parametro vitale visualizzato cliccando sulla voce relativa al parametro vitale desiderato.

3.4.5 Generazione e invio report

- Il medico o il personale sanitario accede e si trova davanti alla Dashboard.
- Clicca su "Visualizza tutti i pazienti" e clicca sull'azione "Visualizza Parametri Vitali".
- Nella schermata che si apre, il medico visualizza i dati relativi ai parametri vitali del paziente.
- Nella sezione "Reports" il medico può scegliere di aprire la pagina di generazione del report relativa a tutti i parametri o a un singolo parametro vitale specifico.
- Se si sceglie su un parametro vitale specifico, la pagina verrà aperta con selezionato in automatico il grafico da includere nel report relativo al parametro vitale che era visualizzo nella pagina precedente(e con lo stesso livello di dettaglio).
- Se si sceglie su "Report Completo", la pagina verrà aperta con selezionati in automatico tutti i grafici relativi ai parametri vitali presenti(con il livello di dettaglio che era selezionato).
- In qualsiasi caso è possibile aggiungere o rimuovere parametri vitali e/o grafici. Inoltre è possibile aggiungere o rimuovere note cliniche e osservazioni cliniche.
- Il medico può inserire un messaggio opzionale riepilogativo con eventuali suggerimenti per il paziente.

- Il medico può selezionare di inoltrare una copia del report al paziente via email(solo se questa è stata inserita nel campo opzionale del paziente al momento della registrazione).
- Il medico cliccca su "Genera Report PDF" e una copia PDF del report verrà scaricata sul dispositivo.
- Se selezionata l'opzione di invio email, il sistema invierà una copia del report anche al paziente tramite email.

3.4.6 Visualizzazione degli audit

- Il medico o il personale sanitario accede e si trova davanti alla Dashboard.
- Clicca sul pulsante "Visualizza i Log delle Attività" e sistema si apre la pagina con gli audit.
- Il medico visualizza una pagina con tutte le operazioni eseguite sugli utenti che esso segue (generati o importati) organizzate per "Azione" ed "Entità".
- Il medico può filtrare i dati per utente, Data(inizio e fine), tipo di azione, tipo di entità, paziente.
- Il medico può visualizzare dei grafici riepilogativi delle azioni eseguite in fondo alla pagina.

3.4.7 Modifica dati personali

- Il medico o il personale sanitario accede e si trova davanti alla Dashboard.
- Clicca sul pulsante in alto a destra col proprio nome.
- Nella tendina che si apre clicca su "Profilo".
- Si apre una pagina con i suoi dati personali.
- Per modificare i propri dati, il medico aggiorna i campi contenenti i vecchi dati con quelli nuovi e, verificando che questi soddisfino la validazione, clicca su "Aggiorna Profilo" (per i dati normali) o "Aggiorna passato" (per la password).
- Se i dati inseriti sono validi e il sistema non ha restituito errori, il medico riceve un messaggio che lo informa dell'avvenuto aggiornamento dei suoi dati.

Capitolo 4

Progettazione

4.1 Architettura del sistema

4.1.1 Architettura generale e componenti principali

VitaLink è strutturata come un'applicazione web progettata per essere altamente scalabile e modulabile. Essa utilizza il Framework Flask (Python) per il backend, Bootstrap per il frontend e PostgreSQL come database relazionale(ma è possibile sostiturlo facilmente con un DBMS analogamente relazionale). La piattaforma è progettata per essere inizialmente accessibile solo tramite browser web, ma sono state inserite anche delle API per permettere l'integrazione futura con possibili app mobile o altri sistemi esterni.

4.1.2 Modularità e separazione delle responsabilità

La piattaforma è divisa in 15 moduli principali, ai quali vanno ad affiancarsi due moduli di supporto per la gestione delle migrazioni e per la compilazione delle traduzioni al momento dell'istanziazione di un un container. Vi sono poi le directory dei file static e dei template per quanto concerne il frontend, e la directory delle traduzioni per contenere le varie localizzazioni seguendo lo standard di Flask Babel. Infine nella root del progetto è presente la cartella dei test. La cartella contiene inoltre un file di configurazione per la creazione di un ambiente e la fornitura di funzioni per la gestione dei test.

```
VitaLink/
├── docs/ ...  (documentazione)
├── app/
│   ├── static/
│   │   ├── css/
│   │   │   ├── custom.css (stile dell'applicazione)
│   │   │   └── health_connect.css (stile pagina collegamento dispositivi)
│   │   └── img/
│   │       └── fitbit-logo.png (logo Fitbit)
```

```
│
│
│
│   │
│   │   │
│   │   ├── apple-health-logo.png (logo Apple Health)
│   │   └── health-connect-logo.png (logo Google Health Connect)
│   └── js/
│       ├── health_platforms.js
│       ├── main.js
│       ├── observations.js
│       ├── patients.js
│       ├── specific_report.js
│       ├── translations.js
│       ├── vital_charts.js
│       └── vitals.js
├── templates/
│   ├── audit_logs.html
│   ├── base-no-session.html
│   ├── base.html
│   ├── dashboard.html
│   ├── health_connect_result.html
│   ├── health_connect.html
│   ├── login.html
│   ├── patient_detail.html
│   ├── patients.html
│   ├── profile.html
│   ├── register.html
│   ├── specific_report_form.html
│   └── vitals.html
├── translations/
│   ├── babel.cfg (configurazione Babel)
│   ├── messages.pot (template traduzioni)
│   └── it/ (traduzioni in italiano)
├── __init__.py
├── app.py (configurazione applicazione e database)
├── audit.py (gestione log di audit)
├── auth.py (definizione autorizzazioni)
├── compile_translations.py (compilazione traduzioni)
├── email_utils.py (integrazione API invio email)
├── health_platforms_config.py (definizione API piattaforme
│   sanitarie)
├── health_platforms.py (recupero dati dalle API)
├── language.py (gestione lingue)
├── main.py (punto di ingresso dell'applicazione)
├── migrate.py (gestione migrazione database)
├── models.py (definizione modelli database)
├── observations.py (gestione osservazioni cliniche)
├── reports.py (generazione report)
├── utils.py (funzioni di utilità)
└── views.py (controller viste web)
```

```
    └── api.py (endpoint API RESTful)
├── tests/ (test unitari e di integrazione)
├── Dockerfile (configurazione immagine Docker)
├── docker-compose.yml (configurazione container Docker)
├── docker-entrypoint.sh (script avvio container)
├── .env.example (struttura variabili d'ambiente)
├── .env (configurazione locale)
├── db_migrate.yml (migrazione automatica in Docker)
├── pyproject.toml (dipendenze e configurazione test)
├── .dockerignore (esclusioni per Docker)
└── .gitignore (esclusioni per Git)
```

4.1.3 Flusso dei dati e interazioni tra componenti

Il frontend interagisce con il backend mediante richieste HTTP alle API RESTful della piattaforma, ricevendo risposte in formato JSON. Queste API sono protette mediante autenticazione JWT o sessioni, facendo in modo che solo gli utenti autorizzati possano accedere ai dati. Il backend, implementato con Flask, gestisce tali richieste attraverso moduli specializzati come `auth.py` per l'autenticazione, `views.py` per le viste web e `api.py` per gli endpoint REST. I dati rimangono immagazzinati nel database PostgreSQL, accessibile tramite l'ORM SQLAlchemy [29] che astrae le interazioni a livello SQL. Una componente chiave per l'obiettivo della piattaforma è il modulo `health_platforms.py` che gestisce l'integrazione con servizi esterni come Fitbit mediante OAuth 2.0: quando un paziente autorizza l'accesso ai propri dati, il sistema riceve token di accesso e refresh che vengono memorizzati e associati al profilo dell'utente interessato. Successivamente, il sistema può recuperare i dati sanitari chiamando le API esterne, elaborarli secondo le regole in `health_platforms_config.py` e memorizzarli temporaneamente nella cache per ottimizzare le prestazioni e ridurre le chiamate API (mitigando la possibilità di incorrere in blocchi).

4.2 Design pattern

Tra i pattern architetturali adottati nel progetto possiamo citare i più rilevanti.

4.2.1 Pattern architetturali

- MVC (Model-View-Controller): separa la logica di business dalla presentazione, dell'interazione con la base di dati e della logica di controllo.

- Function Organization Pattern : ogni modulo è organizzato in funzioni, ciascuna delle quali ha una responsabilità specifica e coerente con la natura del modulo stesso.

4.2.2 Pattern di progettazione

- Factory Pattern: utilizzato per creare oggetti in una superclasse, permettendo alle sottoclassi di alterare il tipo di oggetto creato.
- Strategy Pattern: permette di definire una famiglia di algoritmi incapsulati in classi separate.
- Decorator Pattern: permette di aggiungere responsabilità aggiuntive ad un oggetto senza modificarne la struttura.

4.3 Modello dei dati

Per il modello dei dati è stato scelto un'approccio relazionale in quanto non vi è una particolare necessità di un modello noSQL, essendo i dati sanitari di tipo strutturato e relazionabile.

4.3.1 Entità principali e relazioni - Schema ER

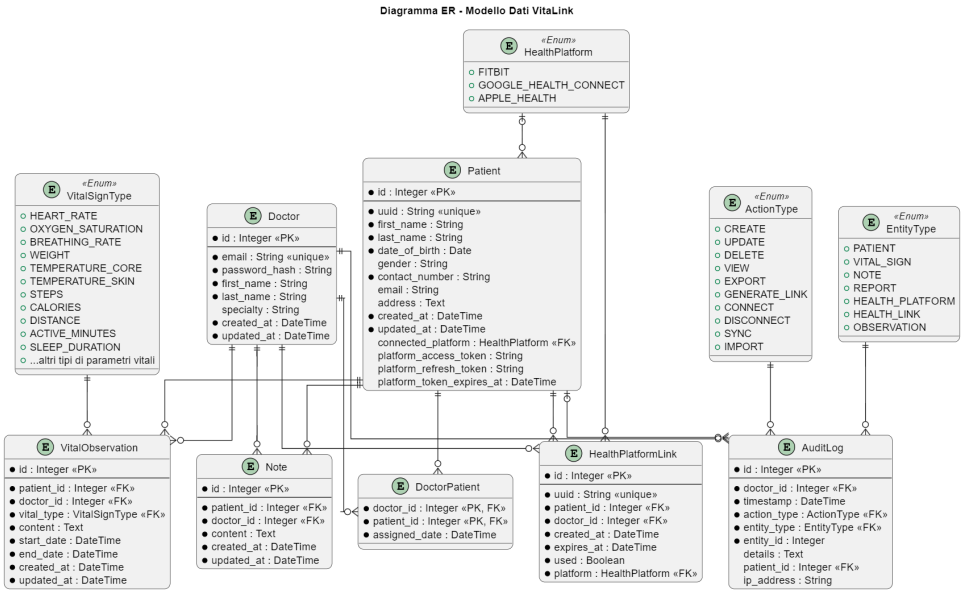

Figura 4.1: Schema ER completo del database

4.3.2 Enumerazioni e tipi di dati

È stato scelto di utilizzare i tipi di dati standard di PostgreSQL per la definizione dei campi del database, in quanto sono sufficienti a soddisfare gli obiettivi perseguiti. Sono presenti 4 Enum nel database utilizzati per casi specifici per garantire l'integrità dei dati e limitare i valori possibili ad un insieme predefinito di costanti.

- Tipi di parametri vitali
- Tipo di piattaforma sanitaria
- Tipo di attività per l'audit
- Tipo di entità per l'audit

4.4 Architettura del database

È stato scelto PostgreSQL come DBMS relazionale in quanto risulta essere uno dei più diffusi e supportati dalle piattaforme per il deployment. Non è comunque esclusa la possibilità di poter riconfigurare il sistema per l'utilizzo di un altro DBMS relazionale, come ad esempio MySQL o Microsoft SQL Server.

4.4.1 Strategia di migrazione

Sono presenti 2 file nel progetto per le migrazioni del database: Lo script db_migrate.sh è un file che viene utilizzato nelle situazioni di deployment (o comunque di istanziazione del container) per la creazione del database e delle tabelle necessarie e l'aggiornamento della struttura del database. Lo script migrate.py è un file che viene utilizzato per la migrazione del database in fase di sviluppo locale, per la creazione delle tabelle e per l'aggiornamento della struttura del database. Da notare che che per lo sviluppo locale si rende necessaria la creazione del database dall'istanza di PostgreSQL locale, mentre per il deployment è sufficiente, nel caso in cui non si sfrutti un database interno al container, specificare il nome del database nel file .env. Nel caso invece in cui in fase di deployment si decida di utilizzare un database esterno al container, valgono le stesse considerazioni fatte per lo sviluppo locale, ossia si rende necessaria la creazione del database sul provider del servizio database scelto.

4.5 Interfaccia utente

4.5.1 Design responsivo

L'interfaccia utente è stata progettata per essere quanto più intuitiva e semplice possibile seguendo un'approccio per formulare un design responsivo, utile per facilitare la consultazione dei dati sanitari su un'ampia gamma di dispositivi. La decisione di utilizzare Bootstrap per il frontend ha permesso di ridurre la mole di lavoro necessaria per questo tipo di task, consentendo di focalizzare gli sforzi di sviluppo su parti più cruciali del sistema. Le pagine sono state sviluppate con un design moderno e intuitivo, adatto per l'uso anche da parte di personale sanitario non addestrato all'uso dello strumento. Ciascuna pagina presenta un menù accessibile nella parte superiore dello schermo che permette un accesso rapido a:

- Dashboard
- Visualizza tutti i pazienti
- Visualizza i log di audit
- Menù a tendina con il nome dell'utente
- Selezione della lingua
  - Profilo
  - Logout

Nelle pagine di generazione dei report e di concessione dell'autorizzazione di accesso ai dati dei parametri vitali (per i pazienti) questo menù è stato rimosso per motivi di design in quanto queste pagine non sono state progettate come routes accessibili permanentemente, ma solo temporaneamente per determinate task.

4.5.2 Componenti UI per la visualizzazione dei dati sanitari

Per la visualizzazione dei dati sanitari sono stati utilizzati grafici generati con la libreria Chart.js [30], che permette di generare grafici personalizzabili e interattivi con un design accattivante. Inoltre i dati sanitari sono accessibili anche in formato tabellare, utile soprattutto per quei dati la cui rappresentazione grafica risulta poco utile, come ad esempio il cibo assunto o diari dell'utente.

4.6 API e integrazione con sistemi esterni

4.6.1 RESTful API design

L'architettura di VitaLink è stata progettata includendo i principi del paradigma REST (Representational State Transfer) [31], ossia un insieme di vincoli e proprietà basati sul protocollo HTTP per la creazione di servizi web. L'implementazione di un'API RESTful ha permesso di ottenere un'interfaccia standardizzata, scalabile e facilmente integrabile con sistemi esterni e possibili applicazioni di terze parti.

Principi di progettazione

Nella progettazione delle API di VitaLink sono stati seguiti i seguenti principi fondamentali REST:

- **Architettura client-server**: anche se frontend e backend sono inclusi nella stessa immagine Docker, mantengono una separazione logica e comunicano tramite richieste HTTP, rispettando il principio dell'architettura client-server.
- **Statelessness**: ogni richiesta dal client al server contiene tutte le informazioni necessarie per capire e processare la richiesta, senza dipendere da contesti memorizzati sul server.
- **Interfaccia uniforme**: tutte le risorse sono accessibili attraverso un'interfaccia standardizzata che utilizza i metodi HTTP (GET, POST, PUT, DELETE).
- **Sistema a livelli**: l'architettura è organizzata in livelli, con ogni componente che svolge un ruolo specifico.
- **Risorse identificabili**: ogni risorsa è identificata univocamente attraverso URI (Uniform Resource Identifier).

Struttura delle risorse

L'API è strutturata attorno a risorse chiaramente identificate. Tra le risorse possiamo trovare:

- `/patients`: gestione dei pazienti.
- `/patients/<uuid>/vitals`: parametri vitali di un paziente specifico.
- `/patients/<uuid>/notes`: note mediche associate a un paziente.
- `/observations`: osservazioni mediche.

Metodi HTTP semantici

I metodi HTTP sono utilizzati per le operazioni CRUD (Create, Read, Update, Delete):

- `GET`: recupero di risorse (es. lista pazienti, parametri vitali).
- `POST`: creazione di nuove risorse (es. aggiunta di una nota medica).
- `PUT`: aggiornamento di risorse esistenti (es. modifica di un'osservazione).
- `DELETE`: rimozione di risorse (es. eliminazione di una nota).

Rappresentazione delle risorse

Le risorse sono rappresentate in formato JSON [32]. Un esempio di rappresentazione per una richiesta di parametri vitali:

```
{
    "heart_rate": [
        {
            "timestamp": "2023-05-01T14:30:00Z",
            "value": 72,
            "unit": "bpm"
        },
        ...
    ]
}
```

Codici di stato HTTP

Le API utilizzano codici di stato HTTP standard per fornire il risultato delle operazioni:

- **200 OK**: operazione completata con successo (GET, PUT, DELETE)
- **201 Created**: risorsa creata con successo (POST)
- **400 Bad Request**: richiesta non valida (parametri mancanti o errati)
- **401 Unauthorized**: autenticazione mancante o non valida
- **403 Forbidden**: accesso non autorizzato alla risorsa richiesta
- **404 Not Found**: risorsa non trovata
- **500 Internal Server Error**: errore del server

Autenticazione e autorizzazione

L'accesso alle API è protetto tramite autenticazione JWT (JSON Web Token) per garantire:

- **Sicurezza**: comunicazioni crittografate con il server
- **Statelessness**: ogni richiesta contiene tutte le informazioni necessarie per l'autenticazione
- **Granularità dei permessi**: verifica che il medico abbia accesso al paziente richiesto

Documentazione delle API

Le API sono completamente documentate con docstring Python dettagliate (compilabili con il pacchetto mkdocs) che specificano:

- Scopo dell'endpoint
- Parametri di input richiesti e opzionali
- Formato della risposta
- Possibili codici di stato e loro significato
- Esempi di utilizzo

Gestione degli errori

È stato implementato un sistema di gestione degli errori che fornisce messaggi di errore chiari e dettagliati in formato JSON. Ad esempio:

```
{
    "error": "Patient not found",
    "details": "No patient with the provided UUID exists"
}
```

4.6.2 Integrazione OAuth con Fitbit

Per l'integrazione con i dispositivi Fitbit, l'applicazione utilizza il flusso di autorizzazione OAuth 2.0. Ciò consente agli utenti di autorizzare l'accesso ai propri dati sanitari senza condividere le proprie credenziali. Una volta che l'utente ha autorizzato l'accesso, l'applicazione riceve un token di accesso che può essere usato per effettuare richieste alle API

di Fitbit per recuperare i dati dei parametri vitali. Viene inoltre fornito un token refresh per ottenere un nuovo token di accesso una volta che quello precedente è scaduto (di base i token di accesso per fitbit scadono dopo 8 ore).

4.6.3 Sistema di caching temporaneo

Per ottimizzare le prestazioni e ridurre il numero di chiamate API con possibile conseguente superamento del rate limit specifico di una piattaforma, è stato implementato un sistema di caching temporaneo. Questo sistema memorizza nella cache i dati recentemente recuperati, in modo da poterli riutilizzare senza dover effettuare una nuova richiesta alle API. Questo approccio aiuta a rispettare i limiti di rate imposti da alcune API.

4.6.4 Estendibilità per future piattaforme

L'architettura delle API è stata progettata tenendo in considerazione l'estensione futura dei servizi offerti. Ciò significa che, in futuro, sarà possibile integrare facilmente nuove piattaforme e dispositivi semplicemente aggiungendo nuove implementazioni dei servizi e configurando le relative API, senza dover apportare modifiche all'architettura.

Capitolo 5

Implementazione

5.1 Stack tecnologico

5.1.1 Backend: Python e Flask

Il backend della piattaforma è stato sviluppato mediante l'uso di Python e del framework Flask relativo. Python è un linguaggio di programmazione molto flessibile e supoortato da una vasta gamma di librerie. La conoscenza pregressa e l'utilizzo Flask sono state le ragioni più rilevanti per la scelta del framwork.

5.1.2 ORM: SQLAlchemy

Per la gestione delle interazioni con il database, è stato utilizzato SQLAlchemy, un ORM (Object-Relational Mapping) [33] per Python. Esso permette di interagire con il database utilizzando oggetti Python invece di scrivere direttamente query SQL, semplificando così lo sviluppo e la manutenzione del codice.

5.1.3 Frontend: HTML5, CSS3, JavaScript, Bootstrap

Il frontend dell'applicazione è stato realizzato utilizzando tecnologie web standard come HTML5, CSS3 e JavaScript. L'utilizzo del framework Bootstrap ha semplificato e velocizzato lo sviluppo di un'interfaccia utente responsive e moderna. Anche le funzioni JavaScript sono state implementate in moduli separati per garantire una migliore organizzazione del codice e suddivisione delle responsabilità. Ciascuna funzione è stata documentata con docstring dettagliate.

5.1.4 Database: PostgreSQL

5.1.5 Containerizzazione: Docker

L'utilizzo di Docker ha permesso di creare un ambiente di sviluppo e produzione isolato facilmente replicabile. Un file Dockerfile è stato configurato per definire l'immagine del container, mentre un file docker-compose.yml è stato utilizzato per gestire il container nello sviluppo locale

Listing 5.1: Dockerfile

```
FROM python:3.11-slim
WORKDIR /app

RUN apt-get update && apt-get install -y --no-install-recommends
    \
    gcc libpq-dev postgresql-client iproute2 net-tools curl
       dos2unix \
    && rm -rf /var/lib/apt/lists/*

COPY . /app/

RUN pip install --no-cache-dir --upgrade pip \
 && pip install --no-cache-dir .

RUN mkdir -p /app/uploads && chmod 777 /app/uploads

RUN dos2unix /app/docker-entrypoint.sh \
 && chmod +x /app/docker-entrypoint.sh \
 && dos2unix /app/db_migrate.sh \
 && chmod +x /app/db_migrate.sh

EXPOSE $PORT

ENTRYPOINT ["/app/docker-entrypoint.sh"]
CMD ["sh", "-c", "/app/db_migrate.sh && gunicorn --bind $HOST:
   $PORT --workers 3 --access-logfile - --error-logfile - --log-
   level $(echo ${LOG_LEVEL} | tr '[:upper:]' '[:lower:]')
   $FLASK_APP"]
```

Listing 5.2: docker-compose.yml

```
services:
```

```
  web:
    build:
      context: .
      dockerfile: Dockerfile
    ports: ["${PORT}:${PORT}"]
    depends_on: [db]
    env_file: .env
    volumes:
      - ./uploads:/app/uploads
    healthcheck:
      test: ["CMD", "curl", "-f", "http://${HOST}:${PORT}/"]
      interval: 30s
      timeout: 10s
      retries: 3
      start_period: 40s

  db:
    image: postgres:15-alpine
    env_file: .env
    environment:
      POSTGRES_USER:  ${PGUSER}
      POSTGRES_PASSWORD: ${PGPASSWORD}
      POSTGRES_DB: ${PGDATABASE}
    volumes:
      - postgres_data:/var/lib/postgresql/data
    ports: ["${PGPORT}:${PGPORT}"]
    healthcheck:
      test: ["CMD-SHELL", "pg_isready -U ${PGUSER}"]
      interval: 10s
      timeout: 5s
      retries: 5
      start_period: 10s

volumes:
  postgres_data:
```

Si può notare come nel file Dockerfile vengono chiamati gli script di inizializzazione db_migrate.sh e docker-entrypoint.sh. Il loro utilizzo risulta fondamentale per la corretta configurazione del container e per la corretta inizializzazione del database. Non vengono invece chiamati nelle fasi di sviluppo locale.

5.2 Controllo di versione e integrazione continua

5.2.1 Repository GitHub

Lo sviluppo di VitaLink è stato gestito con Git come sistema di controllo di versione, con un repository ospitato su GitHub [34]. Lo strumento GitHub Desktop [35]ha permesso di gestire le operazioni di commit, push, pull e merge in maniera comoda.

5.2.2 GitHub Actions per CI/CD

- **Continuous Integration (CI)**: ogni pull request e commit nei rami principali innesca un workflow che:
  - Installa le dipendenze del progetto
  - Esegue i test automatizzati con pytest
  - Verifica la corretta istanziazione dell'immagine Docker
- **Continuous Deployment (CD)**: i commit nel ramo `main` attivano un workflow di deployment che:
  - Costruisce l'immagine Docker dell'applicazione
  - Carica l'immagine in un registro container (Docker Hub o GitHub Container Registry)
  - Aggiorna automaticamente l'ambiente di staging per i test di accettazione
- **Generazione documentazione**: aggiornamenti nella documentazione o nel codice attivano un workflow che genera automaticamente la documentazione aggiornata e la pubblica su GitHub Pages.

Questa automazione ha permesso di:

- Ridurre notevolmente gli errori dovuti a procedure manuali
- Accelerare il ciclo di feedback sulle modifiche al codice
- Garantire che solo codice di qualità verificata venga integrato nel progetto
- Mantenere un registro completo di tutte le build e i test eseguiti

5.2.3 Monitoraggio e miglioramento continuo

- **CodeQL**: per l'analisi automatica della sicurezza del codice, che identifica potenziali vulnerabilità di sicurezza
- **Dependabot**: per monitorare e aggiornare automaticamente le dipendenze del progetto, segnalando potenziali problemi di sicurezza
- **SonarCloud**: per l'analisi statica del codice, che fornisce metriche dettagliate sulla qualità e sulla complessità del codice

5.3 Implementazione del backend

5.3.1 Modelli di dati e ORM

I modelli di dati dell'applicazione sono stati definiti utilizzando SQLAlchemy, che consente di mappare le classi Python alle tabelle del database. Ogni modello rappresenta una entità del dominio dell'applicazione, come un paziente, un'osservazione clinica o un report. I modelli sono stati progettati per essere coerenti con le relazioni del database e per supportare le operazioni CRUD necessarie per l'applicazione.

5.3.2 Autenticazione e gestione degli utenti

Il sistema di autenticazione si basa su JWT (JSON Web Token), che consente di autenticare gli utenti in modo sicuro e stateless. Al momento della registrazione, gli utenti ricevono un token di accesso che deve essere incluso in ogni richiesta alle API protette. Il token viene verificato dal server per autenticare l'utente e autorizzarlo ad accedere alle risorse richieste.

5.3.3 Implementazione delle API RESTful

Le API RESTful sono state implementate utilizzando Flask-RESTful, un'estensione di Flask che semplifica la creazione di API REST. Ogni risorsa dell'API è rappresentata da una classe che estende `Resource` di Flask-RESTful e definisce i metodi per gestire le richieste HTTP (GET, POST, PUT, DELETE). Le risorse sono collegate agli URL tramite il router di Flask, che instrada le richieste agli handler appropriati.

5.3.4 Sistema di audit e logging

Il sistema di audit tiene traccia di tutte le azioni significative eseguite dagli utenti sull'applicazione, come la creazione, modifica ed eliminazione di record. Queste informazioni sono registrate in un'apposita tabella del database e possono essere consultate dai medici per monitorare le attività sui pazienti. Il registratore di audit è attivato da un middleware che intercetta le richieste e le risposte dell'API.

5.3.5 Integrazione con piattaforme sanitarie

L'integrazione con piattaforme sanitarie esterne, come Fitbit, è gestita tramite API RESTful che utilizzano OAuth 2.0 per l'autenticazione. Quando un paziente autorizza l'accesso ai propri dati, l'applicazione riceve un token di accesso che può essere utilizzato per recuperare i dati dai parametri vitali tramite le API di Fitbit. Questi dati vengono poi memorizzati nel database dell'applicazione per essere elaborati e visualizzati dai medici.

5.3.6 Gestione delle osservazioni cliniche

Le osservazioni cliniche sono gestite tramite un modulo dedicato che consente ai medici di creare, modificare ed eliminare osservazioni associate ai pazienti. Ogni osservazione è collegata a un paziente specifico e può includere informazioni come il tipo di parametro vitale, il valore misurato e la data e ora della misurazione. Le osservazioni possono essere visualizzate in modo cronologico per ciascun paziente, facilitando il monitoraggio dell'andamento dei parametri vitali nel tempo.

5.4 Implementazione del frontend

5.4.1 Struttura delle pagine e componenti

Il frontend è composto da una serie di pagine HTML collegate tra loro tramite un sistema di routing basato su Flask. Ogni pagina è progettata per essere responsiva e adattarsi a diverse dimensioni dello schermo. I componenti dell'interfaccia utente, come intestazioni, piè di pagina e barre di navigazione, sono stati realizzati come componenti riutilizzabili per garantire coerenza e facilitare la manutenzione.

5.4.2 Visualizzazione dei parametri vitali con grafici

I parametri vitali sono visualizzati utilizzando grafici interattivi creati con Chart.js. Questi grafici mostrano l'andamento dei parametri nel tempo e consentono ai medici di identifi-

care facilmente eventuali anomalie o tendenze significative. I grafici sono personalizzabili e possono essere esportati come immagini o PDF per essere inclusi nei report.

5.4.3 Form per la gestione dei dati

I form per la gestione dei dati, come l'aggiunta o la modifica di pazienti, osservazioni o note, sono stati realizzati utilizzando componenti di form di Bootstrap, che semplificano la creazione di form responsivi e accessibili. I form includono la validazione dei dati sia lato client che lato server per garantire l'integrità dei dati.

5.4.4 Interfaccia per la generazione di report

L'interfaccia per la generazione di report consente ai medici di selezionare i parametri vitali, le osservazioni e le note da includere nel report. I report possono essere generati in formato PDF e inviati via email ai pazienti. L'interfaccia guida l'utente attraverso il processo di selezione dei dati e personalizzazione del report, rendendo facile per i medici fornire ai pazienti informazioni dettagliate e consigli basati sui dati raccolti.

5.5 Integrazione con API esterne

5.5.1 Flusso OAuth 2.0 per Fitbit

Il flusso di autorizzazione OAuth 2.0 per Fitbit è stato implementato seguendo le linee guida ufficiali di Fitbit. Gli utenti possono collegare il proprio account Fitbit all'applicazione VitaLink autorizzando l'accesso ai propri dati tramite un processo di login sicuro. Una volta autorizzato, l'applicazione riceve un token di accesso che può essere utilizzato per recuperare i dati dei parametri vitali.

5.5.2 Gestione dei token e refresh

I token di accesso ricevuti da Fitbit hanno una durata limitata e devono essere rinnovati periodicamente utilizzando il token di refresh fornito durante il processo di autorizzazione. L'applicazione gestisce automaticamente il rinnovo dei token, garantendo che i dati dei pazienti siano sempre aggiornati senza richiedere interventi manuali da parte degli utenti.

5.5.3 Recupero e processamento dei dati

Il recupero dei dati dai dispositivi wearable avviene tramite chiamate API programmate che utilizzano i token di accesso per autenticarsi. I dati ricevuti sono in formato JSON e vengono elaborati e memorizzati nel database dell'applicazione per essere successivamente visualizzati dai medici. È stato implementato un sistema di caching per ridurre il numero di chiamate API e migliorare le prestazioni.

5.5.4 Gestione degli errori e rate limiting

È stata prestata particolare attenzione alla gestione degli errori e al rispetto dei limiti di rate limiting imposti da Fitbit. L'applicazione è in grado di gestire in modo elegante gli errori di rete, i timeout e le risposte di errore delle API, fornendo messaggi chiari e dettagliati agli utenti. Inoltre, sono state implementate strategie per evitare di superare i limiti di rate limiting, come l'implementazione di ritardi tra le richieste e la gestione dei token di accesso scaduti.

5.6 Sicurezza e autenticazione

5.6.1 Protezione degli endpoint

Tutti gli endpoint dell'API sono protetti da autenticazione JWT, che garantisce che solo gli utenti autenticati e autorizzati possano accedere alle risorse. Inoltre, sono stati implementati controlli di autorizzazione a livello di risorsa per garantire che gli utenti possano accedere solo ai dati per i quali hanno ricevuto esplicita autorizzazione.

5.6.2 Gestione delle sessioni

La gestione delle sessioni è stata implementata utilizzando le funzionalità di sessione di Flask, che consentono di memorizzare informazioni sulla sessione dell'utente sul server e di associarle a un identificatore di sessione memorizzato nel cookie del browser dell'utente. Questo approccio garantisce che le informazioni sensibili non siano mai esposte al client e che le sessioni siano sicure e difficili da manomettere.

5.6.3 JWT per autenticazione API

L'autenticazione delle API è basata su JSON Web Token (JWT), che consente di autenticare gli utenti in modo sicuro e stateless. Al momento della registrazione o del login, gli

utenti ricevono un token di accesso che deve essere incluso in ogni richiesta alle API protette. Il token viene verificato dal server per autenticare l'utente e autorizzarlo ad accedere alle risorse richieste.

5.6.4 Logging di sicurezza e audit

È stato implementato un sistema di logging di sicurezza che tiene traccia di tutte le operazioni di autenticazione e autorizzazione, inclusi i tentativi di accesso riusciti e falliti. Questi log sono utilizzati per monitorare attività sospette e per garantire la conformità alle politiche di sicurezza. Inoltre, tutte le azioni significative eseguite dagli utenti sono registrate in un sistema di audit, che consente di tracciare le modifiche ai dati e di identificare eventuali attività non autorizzate.

Capitolo 6

Testing e Validazione

6.1 Strategia di testing

6.1.1 Approccio al testing

L'approccio al testing per l'applicazione VitaLink si basa su una combinazione di test manuali e automatizzati, mirati a garantire la massima copertura dei requisiti e la rilevazione tempestiva di eventuali difetti o anomalie. I test sono stati progettati per verificare sia i requisiti funzionali che quelli non funzionali, assicurando che l'applicazione soddisfi gli standard di qualità attesi.

6.1.2 Ambienti di test

I test sono stati eseguiti in ambienti separati per lo sviluppo, il collaudo e la produzione, utilizzando dati di test realistici per simulare le diverse condizioni di utilizzo dell'applicazione. Gli ambienti di test sono stati configurati per rispecchiare il più possibile l'ambiente di produzione, al fine di garantire la massima validità dei risultati dei test.

6.1.3 Automazione dei test

È stata implementata un'infrastruttura di test automatizzati utilizzando strumenti come pytest e Selenium, che consentono di eseguire test automatici delle funzionalità dell'applicazione e di verificare la correttezza dei risultati. I test automatizzati vengono eseguiti regolarmente come parte del processo di integrazione continua, per garantire che eventuali modifiche al codice non introducano regressioni o nuovi difetti.

6.2 Unit testing

6.2.1 Test dei modelli

I test dei modelli verificano la correttezza delle definizioni dei modelli di dati e delle loro relazioni, assicurando che siano conformi allo schema del database e che supportino le operazioni CRUD come previsto.

6.2.2 Test delle API

I test delle API verificano il corretto funzionamento degli endpoint dell'API, inclusa l'autenticazione, l'autorizzazione e la gestione degli errori. Viene controllato che ogni endpoint restituisca i codici di stato HTTP appropriati e che i dati restituiti siano corretti e ben formati.

6.2.3 Test dei servizi

I test dei servizi verificano la logica di business dell'applicazione, assicurando che i servizi svolgano le loro funzioni come previsto e che interagiscano correttamente con i modelli e le API.

6.2.4 Mock e fixture

Per isolare i test e semplificare la gestione dei dati di test, sono stati utilizzati mock e fixture. I mock consentono di simulare il comportamento di oggetti complessi o di terze parti, mentre le fixture forniscono dati di test predefiniti e configurazioni per i test.

6.3 Integration testing

6.3.1 Test del flusso OAuth

I test del flusso OAuth verificano che il processo di autorizzazione e autenticazione tramite OAuth 2.0 funzioni correttamente, inclusa la gestione dei token di accesso e refresh.

6.3.2 Test del sistema di report

I test del sistema di report verificano che i report siano generati correttamente e contengano tutte le informazioni necessarie, formattate in modo appropriato per la visualizzazione e l'esportazione.

6.3.3 Test dell'integrazione con il database

I test dell'integrazione con il database verificano che l'applicazione possa interagire correttamente con il database, inclusa la creazione, lettura, aggiornamento e cancellazione di record.

6.4 User testing

6.4.1 Metodologia

La metodologia di user testing prevede il coinvolgimento di un gruppo di utenti rappresentativi che utilizzano l'applicazione in scenari reali o simulati. Durante il testing, gli utenti sono invitati a completare una serie di compiti mentre vengono osservati da un team di valutazione.

6.4.2 Raccolta feedback

Il feedback degli utenti viene raccolto tramite interviste, questionari e osservazioni dirette durante e dopo le sessioni di testing. Viene prestata particolare attenzione alle difficoltà incontrate dagli utenti, alle funzionalità mancanti e alle proposte di miglioramento.

6.4.3 Risultati e miglioramenti

I risultati dei test utente vengono analizzati per identificare modelli e tendenze nelle risposte degli utenti. Sulla base di questo feedback, vengono pianificati e implementati miglioramenti all'applicazione per risolvere i problemi identificati e ottimizzare l'esperienza dell'utente.

6.5 Validazione dei requisiti

6.5.1 Verifica dei requisiti funzionali

La verifica dei requisiti funzionali prevede il controllo sistematico di ogni requisito per garantire che sia stato implementato correttamente e soddisfi le aspettative degli stakeholder.

6.5.2 Analisi dei requisiti non funzionali

L'analisi dei requisiti non funzionali valuta le prestazioni dell'applicazione, la sicurezza, l'usabilità e altri attributi qualitativi per garantire che soddisfi gli standard richiesti.

6.5.3 Completezza della soluzione

La completezza della soluzione viene valutata verificando che tutte le funzionalità richieste siano state implementate e che l'applicazione soddisfi pienamente i requisiti degli utenti e degli stakeholder.

Capitolo 7

Deployment e Operations

7.1 Ambiente di deployment

7.1.1 Architettura dell'ambiente di produzione

L'architettura dell'ambiente di produzione è progettata per essere scalabile, sicura e altamente disponibile. Essa prevede l'uso di container Docker per l'implementazione dei vari componenti dell'applicazione, inclusi il backend, il frontend e il database.

7.1.2 Configurazione del server

Il server di produzione è configurato per eseguire Docker e Docker Compose, che consentono di gestire facilmente i container e le loro interazioni. Sono state configurate anche le variabili d'ambiente necessarie per il corretto funzionamento dell'applicazione.

7.1.3 Gestione delle variabili d'ambiente

Le variabili d'ambiente sono utilizzate per configurare l'applicazione in base all'ambiente in cui viene eseguita (sviluppo, test, produzione). Esse includono informazioni sensibili come le credenziali del database e i segreti per la crittografia, e sono gestite in modo sicuro per prevenire accessi non autorizzati.

7.2 Containerizzazione e orchestrazione

7.2.1 Docker e Docker Compose

Docker è stato utilizzato per containerizzare l'applicazione, creando immagini leggere e portabili che possono essere eseguite su qualsiasi sistema che supporti Docker. Docker

Compose è stato utilizzato per definire e gestire i servizi dell'applicazione, semplificando l'orchestrazione dei vari componenti.

7.2.2 Immagini e configurazione

Le immagini Docker sono state create per ciascun componente dell'applicazione, inclusi il backend, il frontend e il database. Ogni immagine include tutto il necessario per eseguire il componente, comprese le dipendenze e le configurazioni di sistema.

7.2.3 Persistenza dei dati

Per garantire la persistenza dei dati, è stato utilizzato un volume Docker per il database, che consente di mantenere i dati anche quando il container viene arrestato o ricreato. Inoltre, sono stati configurati meccanismi di backup e ripristino per proteggere i dati da perdite o danneggiamenti.

7.3 Continuous Integration e Continuous Deployment

7.3.1 Pipeline CI/CD

È stata implementata una pipeline CI/CD (Continuous Integration/Continuous Deployment) per automatizzare il processo di integrazione e distribuzione del codice. Questa pipeline include fasi per il build, il test e il deployment dell'applicazione, garantendo che ogni modifica al codice venga automaticamente testata e distribuita negli ambienti appropriati.

7.3.2 Automazione dei test

I test automatizzati sono eseguiti come parte della pipeline CI/CD per garantire che eventuali modifiche al codice non introducano regressioni o nuovi difetti. I test includono unit test, test di integrazione e test end-to-end.

7.3.3 Deployment automatizzato

Il deployment dell'applicazione è automatizzato tramite l'uso di script e strumenti di orchestrazione, che consentono di distribuire rapidamente e in modo affidabile nuove versioni dell'applicazione negli ambienti di test e produzione.

7.4 Monitoraggio e logging

7.4.1 Strategia di logging

È stata implementata una strategia di logging centralizzato che raccoglie e memorizza i log di tutti i componenti dell'applicazione in un'unica posizione. Questo approccio semplifica il monitoraggio delle attività dell'applicazione e la diagnosi di eventuali problemi.

7.4.2 Monitoraggio delle performance

Il monitoraggio delle performance dell'applicazione è effettuato tramite strumenti di monitoring che raccolgono metriche sulle prestazioni, come il tempo di risposta delle API, l'utilizzo della CPU e della memoria, e il numero di richieste gestite. Queste metriche sono utilizzate per identificare e risolvere eventuali colli di bottiglia o problemi di prestazioni.

7.4.3 Gestione degli errori in produzione

È stato implementato un sistema di gestione degli errori in produzione che rileva e notifica automaticamente gli errori critici o le anomalie nel funzionamento dell'applicazione. Questo sistema consente di intervenire rapidamente per risolvere i problemi e ripristinare il normale funzionamento dell'applicazione.

Capitolo 8

Risultati e valutazione

8.1 Obiettivi raggiunti

8.1.1 Funzionalità implementate

Tutte le funzionalità principali previste dal progetto sono state implementate con successo, inclusa la registrazione e gestione degli utenti, la gestione dei pazienti, il monitoraggio dei parametri vitali, la generazione di report e l'integrazione con dispositivi wearable.

8.1.2 Conformità ai requisiti

L'applicazione è conforme ai requisiti funzionali e non funzionali definiti all'inizio del progetto. Tutti i requisiti sono stati verificati e validati attraverso test sistematici e revisioni del codice.

8.1.3 Innovazioni apportate

Il progetto ha portato a diverse innovazioni, tra cui un sistema integrato di monitoraggio remoto dei parametri vitali, un'interfaccia utente intuitiva e un'architettura software modulare e scalabile.

8.2 Metriche di performance

8.2.1 Tempo di risposta

Il tempo di risposta medio per le operazioni di base è inferiore a 2 secondi, come richiesto. Le ottimizzazioni delle query e l'uso della cache hanno contribuito a raggiungere questo obiettivo.

8.2.2 Scalabilità e carico

L'applicazione è stata testata con carichi di lavoro simulati fino a 1000 utenti concorrenti, mostrando prestazioni stabili e tempi di risposta accettabili. La scalabilità orizzontale è stata verificata avviando più istanze dei container dell'applicazione.

8.2.3 Efficienza nell'uso delle risorse

L'applicazione mostra un'efficiente utilizzo delle risorse di sistema, con un consumo moderato di CPU e memoria anche sotto carico. Le tecniche di ottimizzazione, come il caricamento pigro e la memorizzazione nella cache, hanno contribuito a migliorare l'efficienza.

8.3 Feedback degli utenti

8.3.1 Metodologia di raccolta feedback

Il feedback degli utenti è stato raccolto tramite sondaggi, interviste e sessioni di testing. Gli utenti sono stati invitati a fornire feedback sulle funzionalità, sull'usabilità e sulle prestazioni dell'applicazione.

8.3.2 Analisi delle risposte

Le risposte degli utenti sono state analizzate per identificare aree di miglioramento e funzionalità aggiuntive richieste. La maggior parte degli utenti ha espresso soddisfazione per le funzionalità offerte e per l'interfaccia utente.

8.3.3 Aree di miglioramento identificate

Alcune aree di miglioramento identificate includono l'aggiunta di funzionalità di analisi predittiva, l'integrazione con ulteriori dispositivi wearable e miglioramenti alla reportistica.

8.4 Limiti e problemi riscontrati

8.4.1 Sfide tecniche

Sono state affrontate diverse sfide tecniche durante lo sviluppo del progetto, tra cui l'integrazione con le API di Fitbit, la gestione della sicurezza e dell'autenticazione, e l'otti-

mizzazione delle prestazioni.

8.4.2 Limitazioni delle API esterne

Le limitazioni imposte dalle API esterne, come i limiti di rate limiting e le restrizioni sui dati disponibili, hanno rappresentato una sfida significativa. È stato necessario implementare soluzioni creative per aggirare queste limitazioni e garantire un funzionamento fluido dell'applicazione.

8.4.3 Compromessi di design

Sono stati necessari alcuni compromessi di design per bilanciare le diverse esigenze e vincoli del progetto. Ad esempio, è stata data priorità alla semplicità d'uso e alla rapidità di sviluppo rispetto a funzionalità più avanzate che avrebbero richiesto tempi di sviluppo e complessità maggiori.

Capitolo 9

Conclusioni e sviluppi futuri

9.1 Conclusioni

9.1.1 Riepilogo del lavoro svolto

Il lavoro di sviluppo dell'applicazione VitaLink ha comportato la progettazione e implementazione di una piattaforma software innovativa per il monitoraggio remoto dei parametri vitali. L'applicazione consente ai medici di monitorare e gestire i pazienti in modo più efficace, migliorando la qualità delle cure e facilitando l'adozione di tecnologie digitali in ambito sanitario.

9.1.2 Contributi principali

I principali contributi del progetto includono lo sviluppo di un'architettura software modulare e scalabile, l'implementazione di un sistema sicuro di autenticazione e autorizzazione, e l'integrazione con dispositivi wearable per il monitoraggio dei parametri vitali.

9.1.3 Riflessione sul processo di sviluppo

Il processo di sviluppo ha seguito i principi dell'ingegneria del software, con particolare attenzione alla qualità, alla sicurezza e alla manutenibilità del codice. Sono state adottate pratiche di sviluppo agile, che hanno permesso di rispondere in modo flessibile e rapido ai cambiamenti e alle nuove esigenze emerse durante il progetto.

9.2 Sviluppi futuri

9.2.1 Integrazione con ulteriori piattaforme sanitarie

In futuro, si prevede di estendere l'integrazione dell'applicazione con ulteriori piattaforme sanitarie e dispositivi wearable, per offrire ai medici una visione ancora più completa e integrata dei parametri di salute dei pazienti.

9.2.2 Funzionalità avanzate di analisi predittiva

Si prevede di sviluppare e implementare funzionalità avanzate di analisi predittiva, che utilizzino algoritmi di machine learning per fornire ai medici informazioni e raccomandazioni basate sui dati storici e sulle tendenze dei parametri vitali dei pazienti.

9.2.3 Espansione del sistema di reportistica

Il sistema di reportistica sarà ulteriormente sviluppato per includere report più dettagliati e personalizzabili, che possano essere facilmente condivisi con altri professionisti della salute o con i pazienti stessi.

9.2.4 Applicazione mobile companion

È prevista anche lo sviluppo di un'applicazione mobile companion, che permetta ai pazienti di monitorare i propri parametri vitali e di ricevere feedback e raccomandazioni direttamente sul proprio dispositivo mobile.

9.3 Considerazioni personali

9.3.1 Apprendimenti chiave

Tra gli apprendimenti chiave vi è la comprensione approfondita delle sfide e delle opportunità offerte dalle tecnologie digitali in ambito sanitario, nonché lo sviluppo di competenze tecniche e professionali nella progettazione e implementazione di soluzioni software complesse.

9.3.2 Sfide personali

Le sfide personali hanno incluso la gestione del tempo e delle risorse, la risoluzione di problemi tecnici complessi e l'adattamento a nuove tecnologie e strumenti di sviluppo.

9.3.3 Valore aggiunto dell'esperienza

L'esperienza di sviluppo dell'applicazione VitaLink ha rappresentato un'importante opportunità di apprendimento e crescita professionale, offrendo la possibilità di contribuire in modo significativo a un progetto innovativo con un reale impatto positivo sulla salute e sul benessere delle persone.

Appendice A

Glossario

API Application Programming Interface

CI/CD Continuous Integration/Continuous Deployment

JWT JSON Web Token

MVC Model-View-Controller

OAuth 2.0 Protocollo di autorizzazione standard dell'industria

ORM Object-Relational Mapping

REST Representational State Transfer

UI User Interface

UML Unified Modeling Language

UUID Universally Unique Identifier

Appendice B

Codice sorgente significativo

B.1 Modelli di dati

B.2 Autenticazione e sicurezza

B.3 Integrazione OAuth

B.4 Sistema di osservazioni

B.5 Generazione report

Appendice C

Diagrammi UML

C.1 Diagrammi dei casi d'uso

C.2 Diagrammi delle classi

C.3 Diagrammi di sequenza

C.4 Diagrammi di stato

C.5 Diagrammi delle attività

C.6 Diagrammi di deployment

C.7 Diagrammi ER

Bibliografia

[1] MSD, Salute. (2025, January 23). *Telemedicina. Ocse: "Raddoppiato il suo uso dopo la pandemia"*. MSD Salute. https://msdsalute.it/approfondimenti/notizie/telemedicina-ocse-raddoppiato-il-suo-uso-dopo-la-pandemia/

[2] Anastasio, P. (2023, February 8). *Sanità digitale, Italia in ritardo. 'Serve collaboration e telemedicina.'*. Key4biz. https://www.key4biz.it/sanita-digitale-italia-ancora-indietro-puntare-su-collaboration-e-telemedicina/434361/

[3] *An EHR-integrated solution for remote patient care*. (n.d.). Validic. https://www.validic.com/

[4] *What is Human API?* (n.d.). Human API. https://reference.humanapi.co/docs/overview

[5] *Health Mate by Withings - La migliore app per monitorare la tua attività, peso e altro*. (n.d.). Withings. https://www.withings.com/it/it/health-mate

[6] SPA, T. (n.d.). *mywellness*. https://www.mywellness.com/cloud/

[7] Wellmo Mobile Wellness Solutions MWS Oy. (2025, May 6). *Platform and mobile app for personalised digital health services*. Wellmo. https://www.wellmo.com/

[8] *Improving lives through digital health coaching | LiVA Healthcare*. (n.d.). https://www.livahealthcare.com/

[9] *Doccla – Europe's leading virtual care solution*. (n.d.). https://www.doccla.com/

[10] GeeksforGeeks. (2024, December 10). *Top 7 Backend development Frameworks [2025 Updated]*. GeeksforGeeks. https://www.geeksforgeeks.org/frameworks-for-backend-development/

[11] Contributors, M. O. J. T. a. B. (n.d.). *Bootstrap*. https://getbootstrap.com/

[12] *Flutter - Build apps for any screen*. (n.d.) https://flutter.dev/

[13] *React*. (n.d.). https://react.dev/

[14] Altalex, R. (2019, February 22). *GDPR - Regolamento generale sulla protezione dei dati*. Altalex. https://www.altalex.com/documents/codici-altlex/2018/03/05/regolamento-generale-sulla-protezione-dei-dati-gdpr

[15] *MySQL*. (n.d.). https://www.mysql.com/it/

[16] PostgreSQL. (2025, May 7). *PostgreSQL*. https://www.postgresql.org/

[17] *MongoDB: the world's leading modern database*. (n.d.). MongoDB. https://www.mongodb.com/

[18] *Bluetooth App development: The role of Bluetooth in wearable technology | By Summer Swann | Connected Devices | Yeti LLC*. (n.d.). https://www.yeti.co/blog/bluetooths-role-in-wearable-technology

[19] contributori di Wikipedia. (2025, March 21). *Bluetooth Low energy*. Wikipedia. https://it.wikipedia.org/wiki/Bluetooth_Low_Energy

[20] contributori di Wikipedia. (2022, March 8). *ANT+*. Wikipedia. https://it.wikipedia.org/wiki/ANT%2B

[21] contributori di Wikipedia. (2024, April 15). *Fitbit*. Wikipedia. https://it.wikipedia.org/wiki/Fitbit

[22] *Fitbit Development: Intraday*. (n.d.). https://dev.fitbit.com/build/reference/web-api/intraday/

[23] *Smartwatch | Orologi per lo Sport | GARMIN*. (n.d.). https://www.garmin.com/it-IT/c/wearables-smartwatches/

[24] Apple. (n.d.). *Apple Watch*. Apple (Italia). https://www.apple.com/it/watch/

[25] amazfit-it. (2025, April 30). *Amazfit Italia | Negozio online ufficiale*. Amazfit-it. https://it.amazfit.com/

[26] contributori di Wikipedia. (2024, September 5). *Metodo MOSCOW*. Wikipedia. https://it.wikipedia.org/wiki/Metodo_MoSCoW

[27] *Werkzeug — Werkzeug Documentation (3.1.x)*. (n.d.). https://werkzeug.palletsprojects.com/en/stable/

[28] Google Cloud. (n.d.). *Flusso di autorizzazione OAuth 2.0*. Google Cloud Apigee. https://cloud.google.com/static/apigee/docs/api-platform/images/oauth-abstract.png

[29] SQLAlchemy. (n.d.). https://www.sqlalchemy.org/

[30] Chart.js. (n.d.). Open Source HTML5 Charts for Your Website. https://www.chartjs.org/

[31] contributori di Wikipedia. (2025, March 15). Representational state transfer. Wikipedia. https://it.wikipedia.org/wiki/Representational_state_transfer

[32] JSON. (n.d.). https://www.json.org/json-it.html

[33] Admin. (2024, February 25). Cosa è un ORM. Innovaformazione - Informatica Specialistica. https://innovaformazione.net/cosa-e-un-orm/

[34] andrearoggeri22/VitaLink: Piattaforma di monitoraggio dati vitali per pazienti in contesto di analisi e valutazione di terapie. (n.d.). GitHub. https://github.com/andrearoggeri22/VitaLink

[35] Download GitHub Desktop. (n.d.). GitHub Desktop. https://desktop.github.com/download/