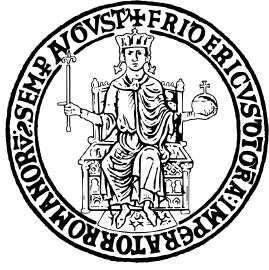


UNIVERSITÀ DEGLI STUDI DI NAPOLI FEDERICO II



SCUOLA POLITECNICA E DELLE SCIENZE DI BASE

DIPARTIMENTO DI INGEGNERIA ELETTRICA E TECNOLOGIE
DELL'INFORMAZIONE

CORSO DI LAUREA TRIENNALE IN INFORMATICA

**SYMBYOCARE: SVILUPPO DI UNA
PIATTAFORMA WEB E MOBILE PER LA
SANITA' DIGITALE**

Relatori

Prof. Rossi Alessandra RICCARDELLI

Correlatore

Candidato

Giuseppe SINDONI
N86004107

Anno Accademico 2024–2025

Indice

1	Introduzione	1
	Obiettivi del progetto	2
	Individuazione e Descrizione del problema	3
	Soluzioni Esistenti e Approcci	3
	Struttura della tesi	5
2	Analisi dei requisiti	7
2.1	Requisiti Funzionali	8
2.1.1	Gestione Utenti	8
2.1.2	Monitoraggio e Inserimento Dati	8
2.1.3	Dashboard e Visualizzazione Dati	8
2.1.4	Gestione Visite Mediche	9
2.1.5	Integrazione con Dispositivi Medici	9
2.2	Requisiti Non Funzionali	10
2.3	Target utenti e Personas	11
2.3.1	Medico dello sport	11
2.3.2	Atleta di running	13
2.3.3	Medico dello sport	14
2.3.4	Medico dello sport	16
2.3.5	Paziente cronico	18
2.3.6	Caregiver familiare	20

2.4	Casi d'uso	22
2.5	Mockup e modello Cockburn	24
2.5.1	Caso d'uso: Inserimento di un paziente	24
2.5.2	Mockup digitali – Inserimento paziente	26
2.5.3	Estensioni (Extension Flows)	29
2.6	Caso d'uso: Accettazione richiesta prenotazione	30
2.6.1	Mockup digitali – Accettazione richiesta prenotazione .	32
2.6.2	Extensioni (Extension Flows)	32
2.7	Caso d'uso: Inserimento rilevazione medica	34
2.7.1	Mockup digitali - Inserimento rilevazione medica . . .	36
2.8	Osservazioni fatte da stakeholder	37
3	Design del Sistema	41
3.1	Architettura del sistema	41
3.2	Class Diagram	42
3.2.1	Entità principali	43
3.2.2	Relazioni principali	44
4	Tecnologie Utilizzate	47
4.1	Tecnologie Frontend Web App	47
4.2	Tecnologie Frontend Mobile	51
4.3	Tecnologie Backend	52
4.4	Tecnologie di persistenza	54
5	Progettazione e codifica	57
5.1	Organizzazione del codice e Versioning	58
5.2	Funzionalità principali	61
5.2.1	Autenticazione e gestione delle sessioni	62
5.2.2	Visualizzazione elenco dei pazienti	65

5.2.3	Visualizzazione dati di un paziente	67
5.2.4	Visualizzazione delle rilevazioni mediche di un paziente	71
5.2.5	Gestione disponibilità per appuntamenti	74
5.2.6	Visualizzazione richieste di appuntamenti e appunta- menti confermati	79
5.2.7	Inserimento di un report di visita	83
5.2.8	Inserimento di un nuovo paziente	87
5.2.9	Registrazione paziente tramite app mobile	90
5.3	Visualizzare richieste di prenotazioni e appuntamenti confermati	96
5.3.1	Richiesta di prenotazione tramite app mobile	101
6	Testing	107
6.1	Obiettivi del testing	107
6.2	Tecnologie adottate	108
6.2.1	Jest	108
6.2.2	NestJS Testing Module	108
6.3	Struttura di un test con Jest	108
6.4	Esempi di test implementati	109
6.4.1	Caso nominale	109
6.4.2	Gestione degli errori	109
6.5	Vantaggi del testing con Jest e NestJS	109
6.6	Conclusioni	110
Bibliografia		111
Glossario		117

- 1 -

Introduzione

Il presente elaborato si inserisce nel crescente contesto della **medicina digitale e del monitoraggio della salute a distanza**, un ambito in forte espansione grazie alla sempre maggiore diffusione di dispositivi digitali capaci di raccogliere dati biometrici. Questi dispositivi hanno aperto nuove significative opportunità nel campo della **medicina preventiva e sportiva**.

In questo scenario, emerge la necessità cruciale di strumenti software avanzati che possano **integrare tali dati in modo strutturato, sicuro e facilmente accessibile**, sia per i professionisti della salute che per i pazienti.

In particolare, l'esigenza di disporre di strumenti software adeguati nasce da una serie di motivazioni:

- Continuità del monitoraggio: i dati raccolti dai dispositivi digitali hanno valore solo se possono essere integrati e consultati nel tempo, offrendo una visione longitudinale dello stato di salute del paziente e permettendo l'individuazione precoce di anomalie.
- Supporto decisionale per i professionisti: i medici necessitano di piattaforme che presentino i dati in maniera chiara e interpretabile, così da agevolare le decisioni cliniche, ridurre il margine di errore e personalizzare i

trattamenti.

- Empowerment del paziente: rendere le informazioni accessibili anche al paziente favorisce una maggiore consapevolezza del proprio stato di salute, incentivando comportamenti preventivi e una più stretta collaborazione con il medico.
- Interoperabilità e centralizzazione: spesso i dati biometrici provengono da dispositivi eterogenei (smartwatch, sensori dedicati, app mobili). Senza una piattaforma unificante, tali dati rischiano di rimanere frammentati e poco utilizzabili.
- Sicurezza e riservatezza: trattandosi di informazioni sensibili, è fondamentale che il sistema garantisca standard elevati di protezione e conformità normativa (es. GDPR, regolamenti sanitari), così da assicurare fiducia e affidabilità.
- Riduzione dei costi sanitari: una gestione più efficiente dei dati e il monitoraggio preventivo consentono di ridurre visite non necessarie, interventi tardivi e ospedalizzazioni evitabili.

Alla luce di queste esigenze, risulta evidente la necessità di soluzioni software capaci di integrare i dati biometrici in un ecosistema coerente, sicuro e facilmente consultabile, rispondendo sia alle aspettative cliniche sia a quelle organizzative e di usabilità.

Obiettivi del progetto

Lo scopo del presente elaborato è descrivere il processo completo di ideazione, progettazione e implementazione di **SymbioCare**, una piattaforma software dedicata al monitoraggio e alla gestione della salute in ambito medico-sportivo.

Il sistema e' stato sviluppato con l'obbiettivo di fornire ai professionisti del settore uno strumento completo per la gestione della loro clientela, fornendo funzionalita' quali il monitoraggio dei parametri clinici, la gestione delle disponibilita' e le prenotazioni degli appuntamenti. Allo stesso tempo, la piattaforma nasce con l'obiettivo di offrire ai pazienti un modo semplice e accessibile per tener traccia delle proprie rilevazioni mediche, mantenendo costantemente aggiornato il proprio medico-sportivo riguardo il proprio stato di salute, favorendo un rapporto medico-paziente moderno e digitale.

Individuazione e Descrizione del problema

Il problema centrale che questa tesi intende affrontare è la **lacuna nella disponibilità di piattaforme integrate ed efficienti** che permettano ai medici di gestire in maniera ottimale i dati provenienti dai dispositivi di monitoraggio dei pazienti a distanza. Sebbene la tecnologia offra un'ampia gamma di sensori e dispositivi wearable capaci di raccogliere dati biometrici e clinici, spesso questi dati rimangono frammentati o sono difficilmente accessibili e interpretabili in modo sistematico dai professionisti.

Attualmente, i medici possono trovarsi a gestire informazioni sanitarie distribuite su più fonti, con la conseguente difficoltà nel monitorare continuativamente lo stato di salute dei pazienti, specialmente quelli seguiti a distanza o con patologie croniche. Questa frammentazione impedisce un'analisi olistica e tempestiva, limitando la possibilità di interventi preventivi o aggiustamenti terapeutici basati su dati oggettivi e aggiornati. L'esigenza è quindi quella di consolidare questi flussi di dati in un'unica interfaccia che sia non solo uno strumento di raccolta, ma anche di analisi e visualizzazione intuitiva, facilitando il processo decisionale clinico e promuovendo un rapporto medico-paziente più moderno e digitale.

Soluzioni Esistenti e Approcci

Nel panorama attuale esistono diverse soluzioni digitali che affrontano, seppur parzialmente, le problematiche descritte. I principali competitor in ambito medico-digitale sono **MioDottore** [1] e **Doctolib** [2], due piattaforme largamente diffuse in Europa che offrono una gamma molto ampia di funzionalità, soprattutto per quanto riguarda la prenotazione di visite e la gestione del rapporto medico-paziente.

Entrambe le piattaforme fungono da **vetrina per i medici**, offrendo agli utenti la possibilità di cercare specialisti in base a parametri come la specializzazione, la città o le recensioni. Le funzionalità di prenotazione sono avanzate e ben progettate, mentre Doctolib include anche strumenti per la condivisione sicura di documenti clinici e una **chat integrata** per la comunicazione diretta con lo specialista. Tuttavia, queste piattaforme sono concepite con un focus principalmente amministrativo e comunicativo, piuttosto che clinico. [3] [4] [5]

Un altro insieme di applicazioni è quello legato alla **digital health personale**, spesso reperibile sugli store mobile. Queste app permettono all'utente di registrare manualmente o automaticamente (tramite Bluetooth) parametri vitali come frequenza cardiaca, pressione arteriosa, glicemia, temperatura corporea. Tuttavia, si tratta di soluzioni orientate all'uso individuale, **senza funzionalità di condivisione strutturata dei dati con il medico curante** [6]. In altre parole, l'approccio è centrato sull'auto-monitoraggio, ma manca un *ponte tecnologico* tra paziente e professionista sanitario.

Alla luce di queste osservazioni, **SymbioCare si propone come soluzione ibrida**, colmando il divario tra le due categorie sopra citate: da un lato offre strumenti per la gestione amministrativa del rapporto medico-paziente (pro-

filo dottore, prenotazioni, calendario), dall'altro consente un **monitoraggio clinico continuo e condiviso** grazie all'integrazione dei dati biometrici nella cartella personale del paziente.

Dal punto di vista metodologico, l'approccio adottato è stato quello di analizzare i bisogni reali di entrambi gli attori coinvolti: paziente e medico. In particolare, si è scelto di:

- Progettare un'architettura duale frontend (Web e Mobile), tenendo conto dei diversi contesti d'uso;
- Adottare uno stile di interfaccia semplice e intuitivo, ma modulare, per garantire un'esperienza utente fluida sia per medici esperti che per utenti meno digitalizzati;
- Sfruttare tecnologie moderne e sicure per garantire una comunicazione protetta tra le parti.

Questo approccio ha guidato ogni fase progettuale, con l'obiettivo di costruire una piattaforma centrata sull'**utilità pratica e sulla sinergia tra raccolta dati e supporto clinico**.

Struttura della tesi

TODO

-2-

Analisi dei requisiti

CONTENTS: **2.1 Requisiti Funzionali.** 2.1.1 Gestione Utenti – 2.1.2 Monitoraggio e Inserimento Dati – 2.1.3 Dashboard e Visualizzazione Dati – 2.1.4 Gestione Visite Mediche – 2.1.5 Integrazione con Dispositivi Medici. **2.2 Requisiti Non Funzionali.** **2.3 Target utenti e Personas.** 2.3.1 Medico dello sport – 2.3.2 Atleta di running – 2.3.3 Medico dello sport – 2.3.4 Medico dello sport – 2.3.5 Paziente cronico – 2.3.6 Caregiver familiare. **2.4 Casi d'uso.** **2.5 Mockup e modello Cockburn.** 2.5.1 Caso d'uso: Inserimento di un paziente – 2.5.2 Mockup digitali – Inserimento paziente – 2.5.3 Estensioni (Extension Flows). **2.6 Caso d'uso: Accettazione richiesta prenotazione.** 2.6.1 Mockup digitali – Accettazione richiesta prenotazione – 2.6.2 Extensioni (Extension Flows). **2.7 Caso d'uso: Inserimento rilevazione medica.** 2.7.1 Mockup digitali - Inserimento rilevazione medica. **2.8 Osservazioni fatte da stakeholder.**

Il presente capitolo è dedicato all'analisi dettagliata dei requisiti del sistema **SymbioCare**, la piattaforma di monitoraggio medico-sportivo digitale. Una chiara definizione dei requisiti è cruciale per guidare le fasi successive di progettazione e implementazione, assicurando che il prodotto finale soddisfi le esigenze degli utenti e gli obiettivi del progetto. I requisiti sono stati categorizzati in **funzionali** e **non funzionali**, per fornire una visione esaustiva delle capacità e delle caratteristiche attese del sistema.

2.1 Requisiti Funzionali

I requisiti funzionali descrivono le specifiche funzionalità che il sistema deve essere in grado di eseguire per soddisfare gli obiettivi degli utenti. Per **SymbioCare**, questi requisiti sono stati identificati per supportare sia il medico che gli sportivi/pazienti nel processo di monitoraggio e gestione della salute.

2.1.1 Gestione Utenti

Il sistema deve consentire la gestione di due tipi principali di utenti: **il medico** e gli **sportivi/pazienti**.

- **Registrazione e profilazione del medico**
- **Inserimento di un nuovo paziente da parte del medico.**
- **Generazione QR Code unico** per ogni nuovo paziente.
- **Registrazione del paziente tramite QR Code**

2.1.2 Monitoraggio e Inserimento Dati

Il cuore del sistema è la raccolta e l'analisi dei dati sanitari provenienti dai pazienti.

- **Inserimento manuale dei dati** (SpO_2 , frequenza cardiaca, temperatura corporea, peso corporeo):
- **Acquisizione automatica** da dispositivi Bluetooth.

2.1.3 Dashboard e Visualizzazione Dati

- **Dashboard medico :**
 - Grafici temporali per ogni parametro.
 - Monitoraggio di più sportivi.

- **Dashboard paziente** con interfaccia chiara e intuitiva.

2.1.4 Gestione Visite Mediche

- **Calendario disponibilità medico.**
- **Suddivisione automatica delle disponibilità in slot prenotabili.**
- **Richiesta di prenotazione da parte del paziente**
- **Gestione richieste di prenotazioni** da parte del medico.
- **Compilazione rapporto di visita** a cura del medico.

2.1.5 Integrazione con Dispositivi Medici

- **Supporto per dispositivi standard:** pulsossimetri, cardiofrequenzimetri, spirometri portatili.
- **Comunicazione via BLE.**

2.2 Requisiti Non Funzionali

I requisiti non funzionali descrivono le qualità del sistema e ne definiscono il comportamento generale.

- **Sicurezza e privacy dei dati:** il sistema deve proteggere le informazioni sensibili degli utenti, in particolare credenziali e dati personali
- **Scalabilità:** l'architettura deve essere in grado di sostenere un aumento progressivo del numero di utenti e della quantità di dati, senza degradare le prestazioni complessive del sistema.
- **Usabilità:** devono essere presenti interfacce utente semplici, coerenti e facilmente utilizzabili anche da utenti con bassa familiarità digitale.
- **Accessibilità:** il sistema deve essere ottimizzato per un'ampia gamma di dispositivi (smartphone, tablet, desktop), garantendo una corretta visualizzazione e un'interazione fluida su ciascuna piattaforma.
- **Prestazioni:** il sistema deve garantire tempi di risposta ridotti e fluidità dell'interfaccia durante le operazioni più comuni, come il caricamento dei dati o la navigazione nelle dashboard.
- **Interoperabilità:** è richiesta l'integrazione con dispositivi medici compatibili con tecnologia BLE (Bluetooth Low Energy), per consentire l'inserimento automatico delle rilevazioni mediche da parte dei pazienti.
- **Modularità:** l'architettura deve prevedere una chiara separazione tra i diversi moduli o microservizi, in modo da favorire la manutenibilità, il testing indipendente e l'evoluzione del sistema.
- **Affidabilità:** il sistema deve garantire un funzionamento continuo e stabile, riducendo al minimo i downtime e assicurando la persistenza e l'integrità dei dati in caso di errore o interruzione.

2.3 Target utenti e Personas

Per progettare in modo efficace la piattaforma **Symbiocare**, è stato condotto uno studio preliminare volto a identificare le principali categorie di utenti che ne usufruiranno. L'obiettivo di tale analisi è comprendere in profondità i bisogni, le aspettative e i comportamenti degli utenti finali, così da orientare correttamente le scelte progettuali e funzionali.

Le *personas* individuate rappresentano profili tipici che incarnano le caratteristiche, le esigenze e gli obiettivi degli utenti reali. Attraverso queste rappresentazioni, è stato possibile definire in maniera più chiara i requisiti pratici e funzionali, sia generali che specifici, associati a ciascun tipo di utente. Queste personas non costituiscono una rappresentazione esaustiva di tutti gli utenti possibili, ma offrono un valido riferimento per modellare l'esperienza utente e guidare le decisioni progettuali. L'approccio adottato consente inoltre di valutare con maggiore precisione l'impatto delle funzionalità implementate sulle diverse tipologie di utilizzatori. Le personas sono state definite tenendo conto sia delle esigenze esplicite rilevate durante l'analisi dei requisiti, sia di potenziali criticità emerse in fase di progettazione preliminare. Questo approccio orientato all'utente garantisce che la piattaforma sia accessibile, usabile ed efficace nel soddisfare i diversi scenari d'uso previsti. Nelle sezioni seguenti vengono presentate le categorie principali e i rispettivi bisogni.

2.3.1 Medico dello sport

Il profilo rappresentato in Figura 2.1 descrive una tipologia di utente professionale, ovvero un medico con esperienza clinica consolidata e una spiccata propensione all'adozione di soluzioni digitali per la gestione dei propri pazienti. Questo tipo di utente è fortemente orientato all'organizzazione e all'efficienza: lavora sia in presenza che da remoto e necessita di strumenti tecnologici avanzati per supportare le sue attività professionali.

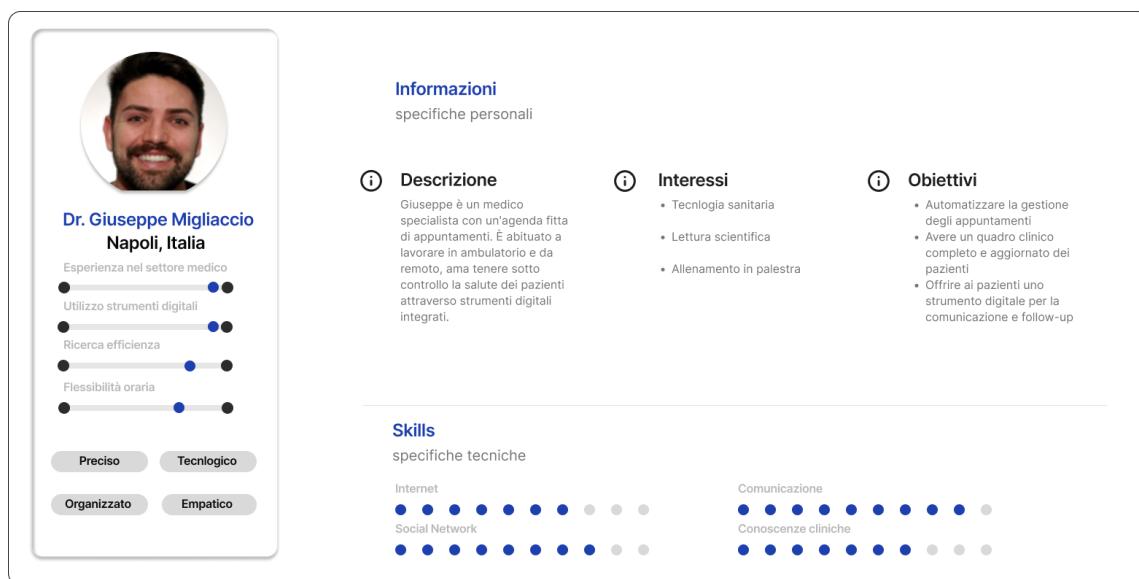


Figura 2.1: Medico Sportivo

logici che semplifichino il monitoraggio continuo della salute dei pazienti, la pianificazione delle visite e la comunicazione a distanza.

Tali utenti mostrano una spiccata flessibilità oraria, interesse per la tecnologia sanitaria e l'aggiornamento scientifico, e richiedono strumenti digitali che si integrino armoniosamente nel loro flusso di lavoro quotidiano.

Di seguito sono elencate le funzionalità più rilevanti per questa categoria di utenti:

- Gestione centralizzata e digitale dell'agenda clinica, con disponibilità oraria personalizzabile.
 - Creazione, modifica e consultazione dei profili pazienti in tempo reale.
 - Visualizzazione storica dei parametri biometrici dei pazienti mediante grafici interattivi.
 - Accesso rapido ai dati clinici anche in modalità remota.
 - Integrazione con dispositivi medici Bluetooth per il monitoraggio continuo dei parametri vitali.

2.3.2 Atleta di running

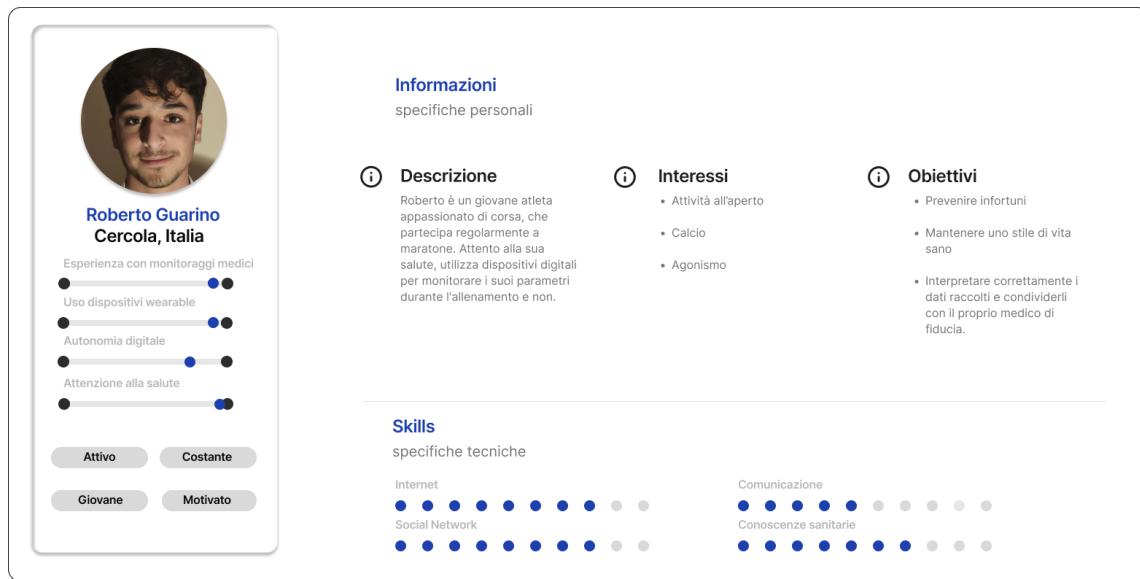


Figura 2.2: Atleta di Running

Il profilo rappresentato in Figura 2.2 descrive una tipologia di utente sportivo, giovane e motivato, che pratica regolarmente attività fisica di tipo agonistico. In particolare, l'utente è un maratoneta appassionato alla cura del proprio corpo e attento alla prevenzione degli infortuni, con un buon livello di alfabetizzazione digitale. È abituato a utilizzare dispositivi indossabili (wearable) per monitorare i parametri vitali durante l'allenamento e nel tempo libero.

Questi utenti richiedono strumenti tecnologici intuitivi che consentano un monitoraggio continuo della salute, la visualizzazione dei propri dati in modo comprensibile, e la possibilità di condividere informazioni cliniche con il proprio medico di riferimento.

Le funzionalità più ricercate da questa categoria di utenti includono:

- Integrazione fluida con dispositivi wearable (smartwatch, pulsossimetri, cardiofrequenzimetri).
- Visualizzazione giornaliera e storica dei propri dati.
- Possibilità di contattare il proprio medico di riferimento.
- Possibilità di prenotare una visita medica.

2.3.3 Medico dello sport

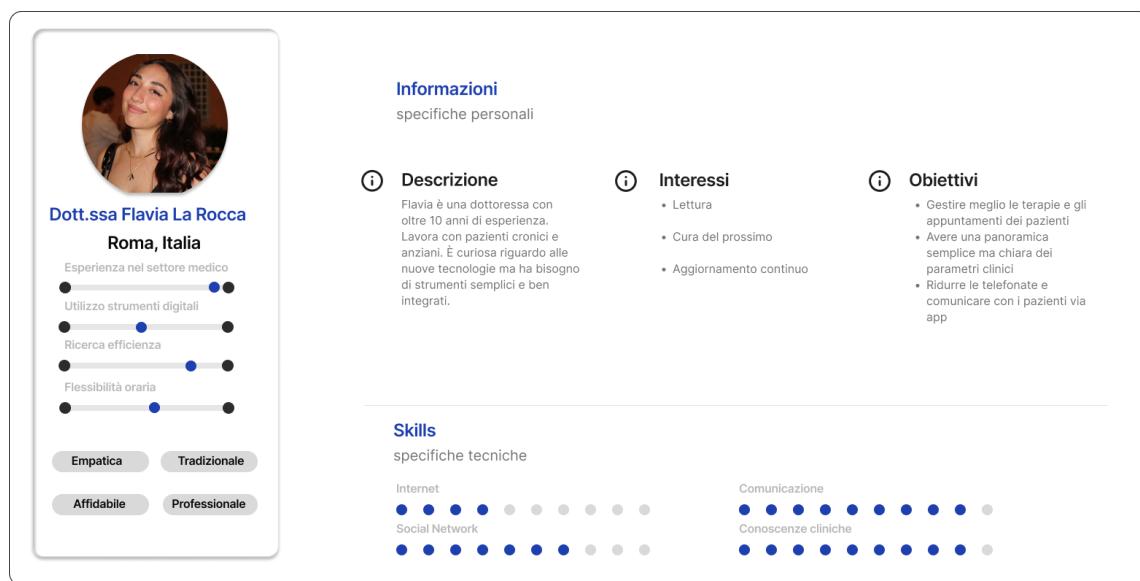


Figura 2.3: Medico dello sport

Il profilo rappresentato in Figura 2.3 descrive una tipologia di utente professionale con elevata esperienza clinica, orientata alla cura del paziente cronico e anziano. La dottoressa ha oltre dieci anni di pratica medica, dimostra un atteggiamento empatico e professionale, ed è interessata all'utilizzo di strumenti digitali purché siano semplici, ben integrati e non richiedano un'elevata curva di apprendimento.

Questa categoria di utenti richiede strumenti digitali che migliorino l'efficienza operativa nella gestione dei pazienti, riducano il carico telefonico e consentano una comunicazione strutturata con il paziente. L'usabilità e la chiarezza dell'interfaccia sono elementi fondamentali, poiché spesso il tempo a disposizione è limitato.

Le funzionalità maggiormente rilevanti per questo tipo di utente includono:

- Agenda medica semplificata con visualizzazione immediata degli appuntamenti.
- Accesso rapido e centralizzato alla scheda clinica del paziente, con parametri organizzati in modo chiaro.
- Visualizzazione intuitiva dell'andamento dei valori clinici nel tempo.

2.3.4 Medico dello sport

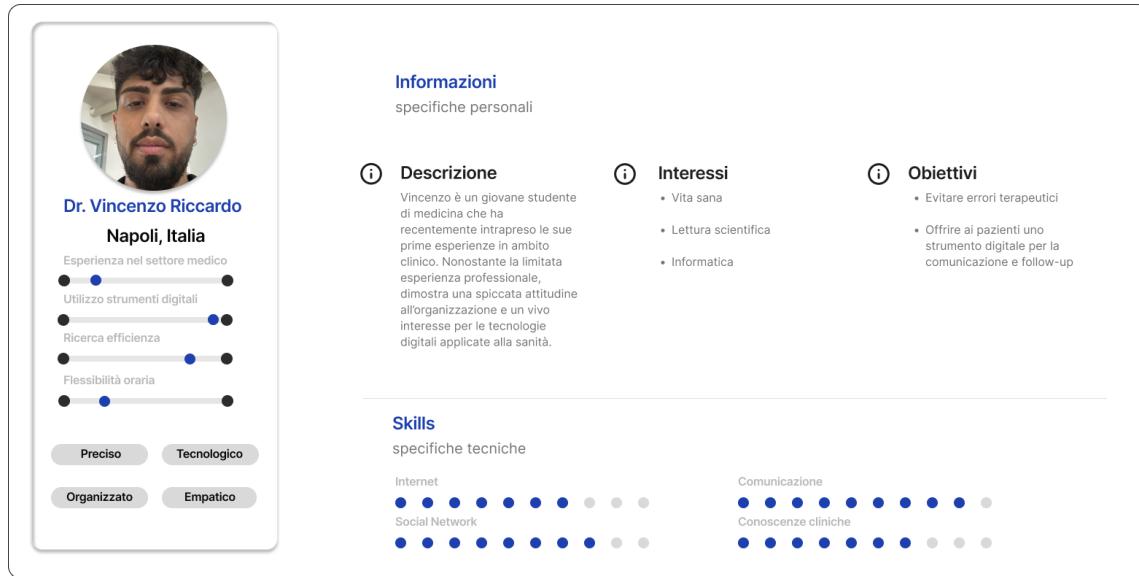


Figura 2.4: Profilo utente: giovane studente di medicina

Il profilo rappresentato in Figura 2.4 descrive un utente in formazione professionale, ovvero un giovane studente di medicina alle prime esperienze pratiche. Vincenzo ha iniziato da poco il suo percorso clinico, ma mostra una forte curiosità verso l'integrazione tra medicina e tecnologia. Pur avendo una limitata esperienza sul campo, si distingue per l'organizzazione personale, l'interesse verso uno stile di vita sano e la volontà di evitare errori terapeutici grazie all'uso di strumenti digitali.

Questa categoria di utenti è motivata, tecnologicamente ricettiva e desiderosa di strumenti che aiutino a gestire in modo ordinato i dati sanitari dei pazienti. In quanto giovane professionista, ricerca interfacce intuitive che possano aiutarlo a imparare e interagire con il paziente in maniera moderna, integrando follow-up e monitoraggio tramite app.

Le funzionalità più rilevanti per questo tipo di utente includono:

- Gestione semplificata delle cartelle cliniche e dei dati biometrici dei pazienti.
- Visualizzazione guidata e ben strutturata dei parametri rilevati (es. andamento SpO₂, BPM).
- Accesso centralizzato ai dati di follow-up per supportare la continuità terapeutica.
- Possibilità di accedere alla piattaforma in qualsiasi occasione, anche fuori dall'orario d'ufficio.

2.3.5 Paziente cronico

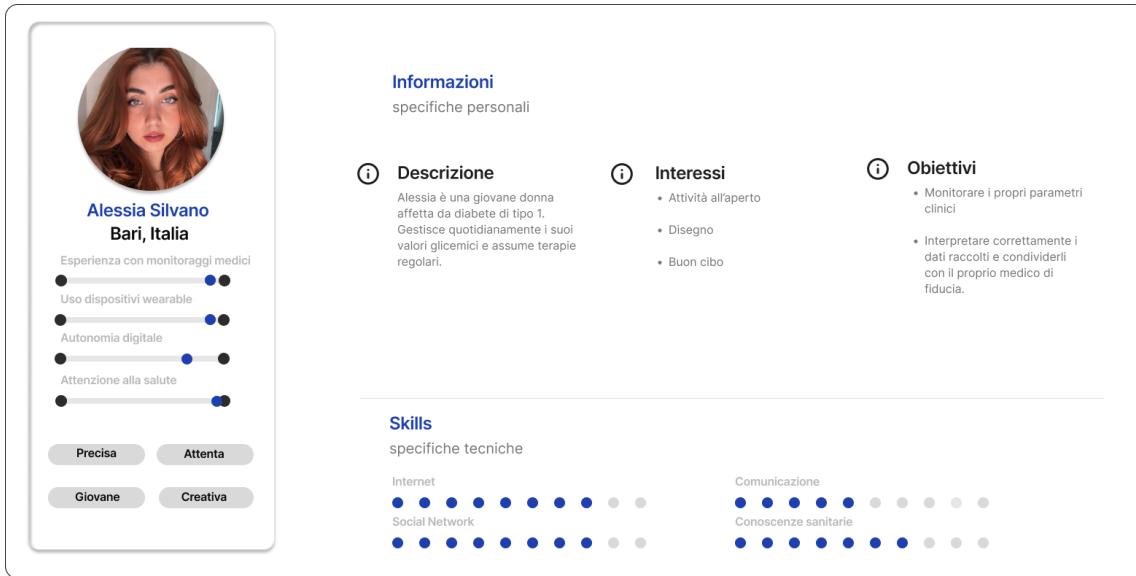


Figura 2.5: Profilo utente: giovane paziente cronico

Il profilo rappresentato in Figura 2.5 descrive una tipologia di utente paziente cronico, giovane e autonomo, che vive una condizione di salute da gestire quotidianamente, come il diabete di tipo 1. Alessia è attenta alla propria salute e ha acquisito una buona familiarità con strumenti digitali e dispositivi indossabili per il monitoraggio glicemico. È creativa, proattiva e desidera strumenti chiari e intuitivi per tenere sotto controllo i propri valori clinici e condividere i dati raccolti con il proprio medico.

Questi utenti richiedono interfacce semplici e affidabili, capaci di accompagnare la gestione quotidiana della malattia senza introdurre ulteriore complessità. L'accesso rapido a informazioni significative e la possibilità di interpretare autonomamente i dati sono elementi fondamentali per mantenere un buon equilibrio terapeutico.

Le funzionalità più utili per questa categoria di utenti includono:

- Acquisizione automatica dei dati glicemici tramite integrazione con sensori o inserimento manuale semplificato.
- Visualizzazione grafica chiara dei valori glicemici nel tempo, con evidenziazione di picchi, cali o deviazioni dalla norma.
- Condivisione sicura dei dati raccolti con il medico curante per facilitare il follow-up a distanza.

2.3.6 Caregiver familiare

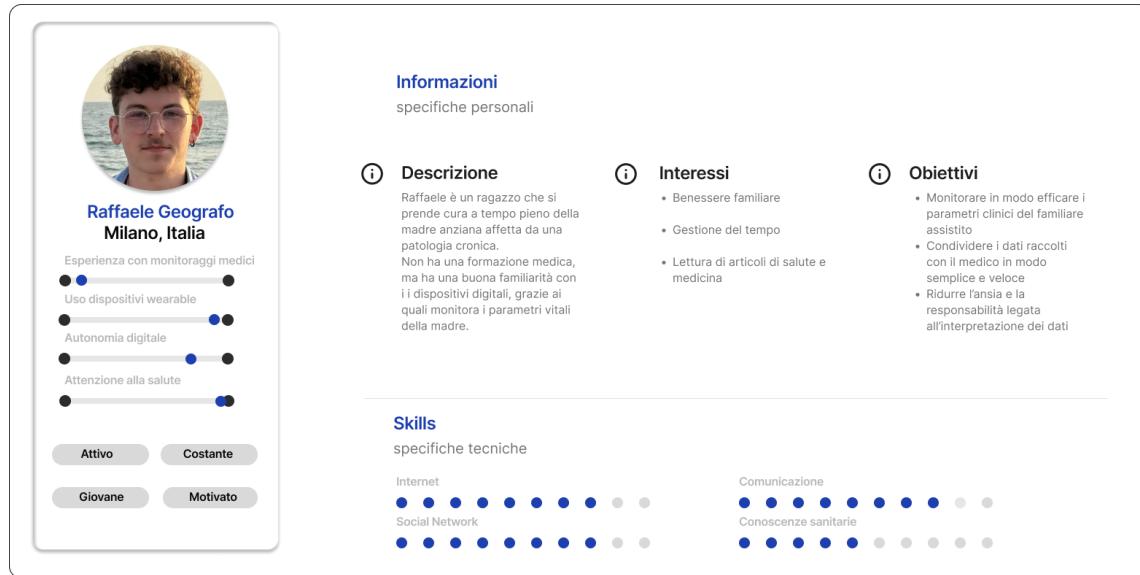


Figura 2.6: Profilo utente: caregiver familiare

Il profilo rappresentato in Figura 2.6 descrive una tipologia di utente caregiver, ovvero una persona che si occupa quotidianamente dell’assistenza sanitaria di un familiare non autosufficiente. Raffaele, in particolare, è una ragazzo di 29 anni che presta cure costanti alla madre affetta da una patologia cronica. Sebbene non possieda una formazione clinica, ha sviluppato un buon livello di alfabetizzazione digitale, imparando a utilizzare strumenti online e dispositivi wearable per il monitoraggio dei parametri vitali.

Questa categoria di utenti necessita di soluzioni semplici, affidabili e comprensibili, che la supportino nella gestione ordinaria della salute del proprio caro e nella comunicazione efficace con il personale medico.

Le funzionalità più utili per questa categoria di utenti includono:

- Visualizzazione semplificata e grafica dei parametri clinici (es. pressione, glicemia, saturazione).

- Sistema di notifiche intelligenti in caso di rilevazioni critiche o valori fuori soglia.
- Condivisione dei dati in tempo reale con il medico curante tramite piattaforma sicura.
- Assistenza guidata in app per interpretare i dati raccolti in modo autonomo.

2.4 Casi d'uso

In questa sezione vengono presentati i principali casi d'uso del sistema, modellati secondo la metodologia UML. I casi d'uso descrivono **le interazioni tra gli attori principali e il sistema**, evidenziando le funzionalità offerte e i flussi di comportamento attesi. L'analisi dei casi d'uso costituisce un passaggio fondamentale per comprendere i requisiti funzionali, facilitare la progettazione dei componenti software e garantire una corrispondenza chiara tra le aspettative degli utenti e le funzionalità implementate. Ogni caso d'uso è corredata da una descrizione testuale e da un diagramma che ne illustra visivamente gli attori coinvolti e le interazioni previste.

In particolare, ciascun caso d'uso consente di isolare uno scenario specifico di interazione e di rappresentarne il contesto operativo, le condizioni di attivazione, gli obiettivi da raggiungere e le eventuali estensioni o alternative.

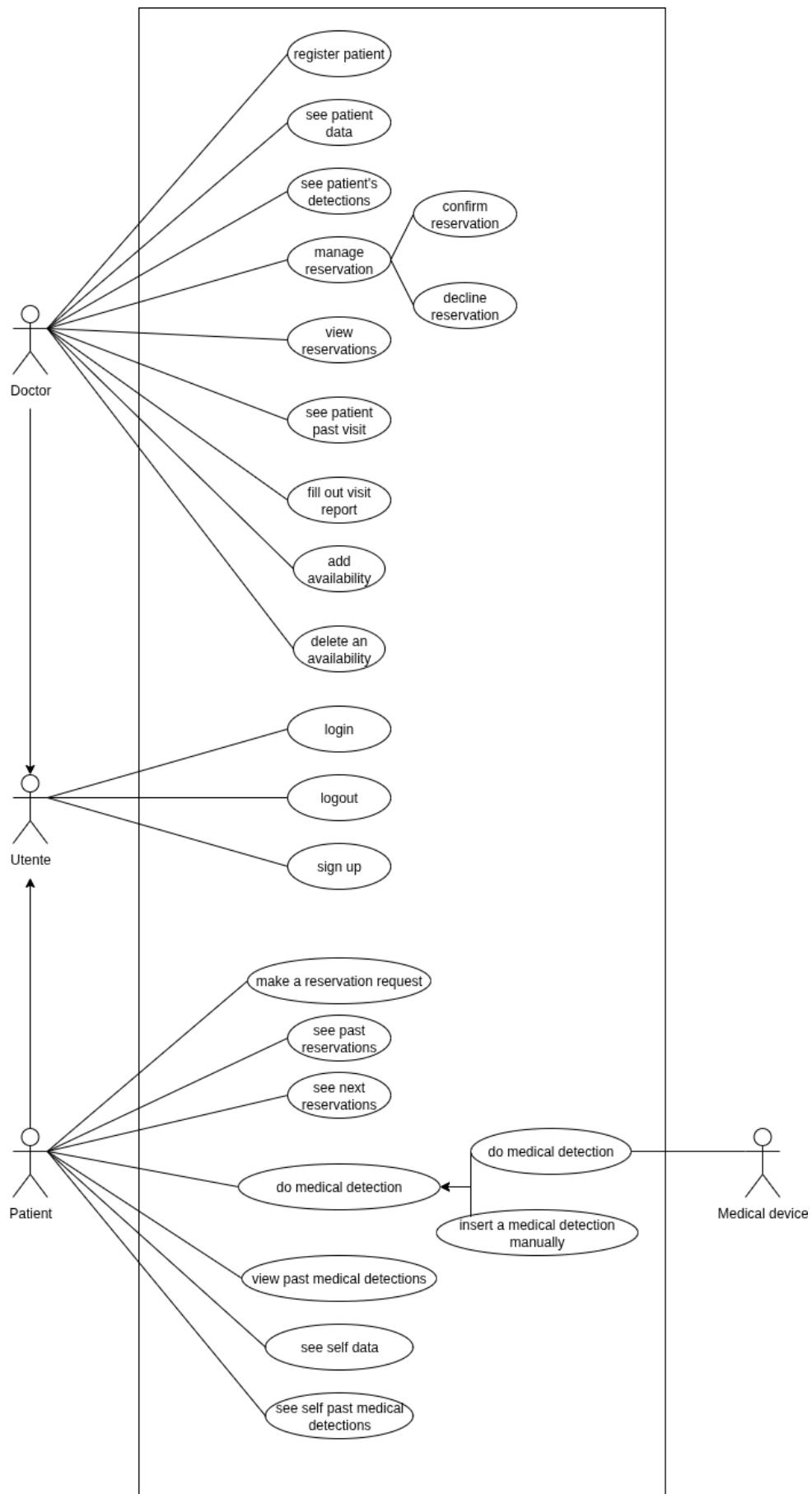


Figura 2.7: Diagramma UML dei casi d'uso principali

2.5 Mockup e modello Cockburn

In questa sezione vengono analizzati alcuni **casi d'uso rappresentativi**, descrivendo il modo in cui vengono gestiti dal sistema e l'interazione prevista con l'utente.

La stesura dei casi d'uso è stata supportata dalla realizzazione di **mockup cartacei**, utili per simulare l'interazione uomo-macchina. Questo approccio ha facilitato l'identificazione dei **flussi principali** e la definizione delle **funzionalità attese**, offrendo una visione preliminare degli scenari d'uso.

Successivamente, tali mockup sono stati digitalizzati tramite il software **Figma**, ottenendo schermate **ad alta fedeltà** utili alla validazione dei requisiti e allo sviluppo front-end. I prototipi così realizzati hanno rappresentato una **guida visiva** durante la progettazione dell'interfaccia utente.

2.5.1 Caso d'uso: Inserimento di un paziente

1. Il caso d'uso viene attivato quando il medico seleziona, dalla sidebar, il pulsante “**Aggiungi paziente**”.
2. Il sistema reindirizza il medico a una schermata contenente un **form a più step**, in cui inserire le informazioni del paziente.
3. Dopo aver compilato tutti i campi e premuto il pulsante “**Salva paziente**”, il sistema salva i dati e mostra la schermata del nuovo paziente.

Use Case		Registrazione di un paziente	
Obiettivo		Registrare un nuovo paziente all'interno della piattaforma.	
Precondizione		Il dottore dovrà essere loggato.	
Condizione Finale di Successo		Il dottore inserisce correttamente il proprio paziente e dispone di un codice QR per la sua registrazione.	
Condizione Finale di fallimento		Il dottore non inserisce il proprio paziente.	
Main Scenario	Step	Dottore	System
	1	Clicca su "Aggiungi paziente"	
	2		Mostra il primo step del form: Informazioni personali
	3	Inserisce le informazioni personali e clicca su 'Avanti'	
	4		Mostra il secondo step del form: Informazioni residenziali
	5	Inserisce le informazioni di residenza del paziente e clicca su 'Avanti'	
	6		Mostra il terzo step del form: Informazioni fisiche
	7	Inserisce le informazioni fisiche e clicca su 'Avanti'	
	8		Mostra il quarto step del form: Informazioni mediche
	9	Inserisce informazioni mediche e clicca su 'Salva paziente'	
	10		Mostra la pagina del paziente salvato

Figura 2.8: Diagramma Cockburn – Inserimento paziente

2.5.2 Mockup digitali – Inserimento paziente

1 Primo step
Informazioni personali 2 Secondo step
Informazioni di residenza 3 Terzo Step
Informazioni fisiche 4 Quarto Step
Informazioni mediche

Nome *
Inserisci il nome del paziente

Cognome *
Inserisci il cognome del paziente

Email *
Inserisci l'indirizzo mail del paziente

Telefono *
Inserisci il numero di telefono del paziente

Codice fiscale *
Inserisci il codice fiscale del paziente

Data di nascita *
gg/mm/aaaa

Genere *
Uomo

Step 1 – Informazioni personali

Primo step Informazioni personali 2 Secondo step Informazioni di residenza 3 Terzo Step Informazioni fisiche 4 Quarto Step Informazioni mediche

Indirizzo *

Inserisci l'indirizzo di residenza del paziente

Città *

Inserisci la città di residenza del paziente

CAP *

Inserisci il CAP di residenza del paziente

Provincia *

Inserisci la provincia

[Indietro](#) [Avanti](#)

Step 2 – Informazioni residenziali

Primo step Informazioni personali 2 Secondo step Informazioni di residenza 3 Terzo Step Informazioni fisiche 4 Quarto Step Informazioni mediche

Peso (kg) *

0

Altezza (cm) *

0

Sport *

Inserisci lo sport praticato del paziente

Livello *

INTERMEDIO

[Indietro](#) [Avanti](#)

Step 3 – Informazioni sportive

Primo step
Informazioni personali
 Secondo step
Informazioni di residenza
 Terzo Step
Informazioni fisiche
 Quarto Step
Informazioni mediche

Gruppo sanguigno *

Patologie

Farmaci

Infortuni

[Indietro](#)
[Salva Paziente](#)

Step 4 – Informazioni mediche

←



Andrea Leo

QR di invito

Informazioni personali

Codice Fiscale	Email	Telefono	Genere	Indirizzo
LEONDR01M04F839D	andrealeo19@gmail.com	3338965509	Uomo	Via Aldo Moro 11

Città
Sant'anastasia

CAP
80048

Informazioni mediche

Peso kg	Altezza cm	Gruppo sanguigno	Sport	Livello sportivo
---------	------------	------------------	-------	------------------

Schermata finale con dati paziente salvato

2.5.3 Estensioni (Extension Flows)

Estensione 1 – Annulla inserimento

Extension #1		Torna indietro
Step	Dottore	System
[3-5-7-9]/*a	Clicca sul tasto indietro	
		Torna allo step del Form precedente

Figura 2.9: Annullamento durante la compilazione del form

Il flusso alternativo si attiva quando il medico clicca su “Indietro” durante la compilazione. Il sistema ritorna allo step precedente senza perdere i dati inseriti.

Estensione 2 – Email già esistente

Extension #2		Email già presente nel sistema
Step	Dottore	System
4/4a		Mostra “Email già esistente” (Torna allo step 3 del Main Scenario)

Figura 2.10: Errore: email già registrata

Il sistema blocca la registrazione se l'email è già presente nella piattaforma, notificando l'errore all'utente.

Extension #3		
Step	Dottore	System
4/4a		Mostra "Codice fiscale già esistente" (Torna allo step 3 del Main Scenario)

Figura 2.11: Errore: codice fiscale già presente

Estensione 3 – Codice fiscale duplicato

Il sistema impedisce il salvataggio nel caso in cui venga inserito un codice fiscale già associato a un altro paziente.

Estensione 4 – Numero di telefono duplicato

Extension #4		
Step	Dottore	System
4/4a		Mostra "Telefono già esistente" (Torna allo step 3 del Main Scenario)

Figura 2.12: Errore: numero di telefono già presente

In caso di inserimento di un numero già registrato, il sistema notifica il conflitto e impedisce di procedere.

2.6 Caso d'uso: Accettazione richiesta prenotazione

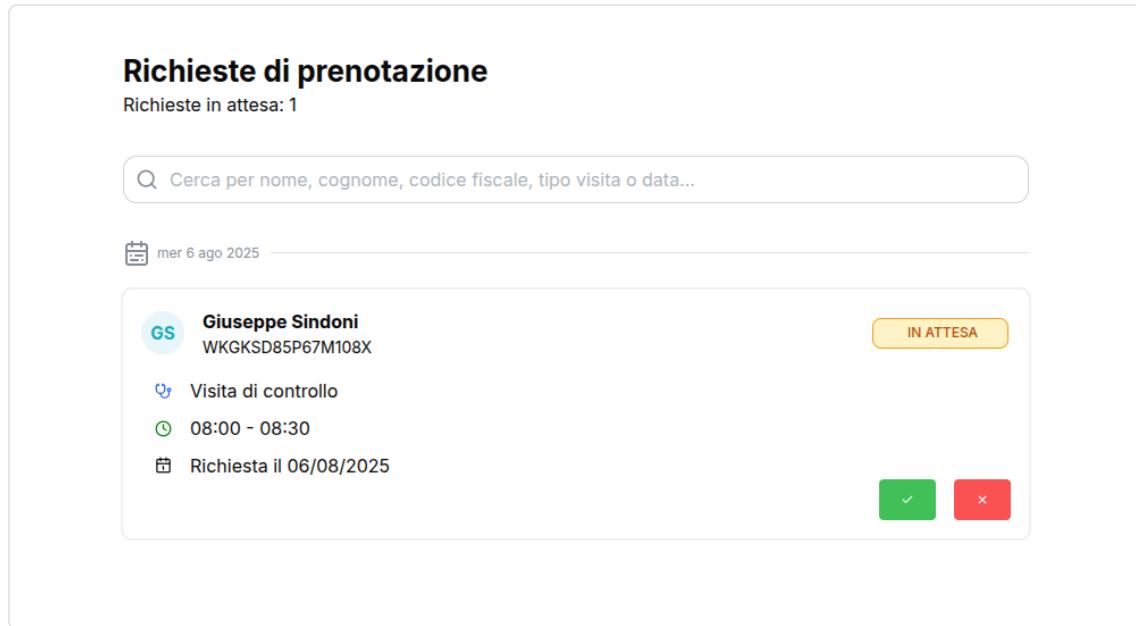
1. Il caso d'uso viene attivato quando il medico seleziona, dalla sidebar, il pulsante “Richieste prenotazioni”.

Use Case		Accettare una richiesta di prenotazione	
Obiettivo		Accettare una richiesta di prenotazione da parte di un paziente.	
Precondizione		Il dottore dovrà essere loggato.	
Condizione Finale di Successo		Il dottore accetta correttamente la prenotazione che comparirà nel calendario delle prenotazioni	
Condizione Finale di fallimento		Il dottore non accetta correttamente la richiesta di prenotazione del paziente.	
Attore principale		Dottore	
Trigger		Cliente clicca su "Richieste di prenotazioni"	
Main Scenario	Step	Dottore	System
	1	Clicca su "Richieste di prenotazione"	
	2		Mostra l'elenco di richieste di prenotazione
	3	Clicca sul tasto di conferma	
	4		Mostra "Prenotazione confermata con successo"

Figura 2.13: Diagramma Cockburn – Accettazione richiesta prenotazione

-
2. Il sistema reindirizza il medico a una schermata contenente una lista di richieste di prenotazioni raggruppate per giorni.
 3. Dopo aver individuato la richiesta di prenotazione che il dottore ha intenzione di accettare, una volta selezionato il tasto "Accetta" il sistema notifica la conferma di quell'appuntamento e sarà visibile nel calendario delle prenotazioni.

2.6.1 Mockup digitali – Accettazione richiesta prenotazione



2.6.2 Extensioni (Extension Flows)

Estensione 1 – Abbandono della schermata

Il flusso alternativo si attiva quando il medico clicca su un'altra sezione mentre è sulla pagina dedicata alle richieste.

Abbandono della schermata		
Step	Dottore	System
[2-3]/*a	Clicca su un'altra sezione della sidebar	

Estensione 2 – Impossibile accettare richiesta

Errore nell'accettazione		
Step	Dottore	System
4/4a		Mostra "Impossibile accettare offerta"

Il flusso alternativo si attiva quando vi è un problema nell'accettazione di tale richiesta.

2.7 Caso d'uso: Inserimento rilevazione medica

Use Case		Accettare una richiesta di prenotazione	
Obiettivo		Accettare una richiesta di prenotazione da parte di un paziente.	
Precondizione		Il dottore dovrà essere loggato.	
Condizione Finale di Successo		Il dottore accetta correttamente la prenotazione che comparirà nel calendario delle prenotazioni	
Condizione Finale di fallimento		Il dottore non accetta correttamente la richiesta di prenotazione del paziente.	
Attore principale		Dottore	
Trigger		Cliente clicca su "Richieste di prenotazioni"	
Main Scenario	Step	Dottore	System
	1	Clicca su "Richieste di prenotazione"	
	2		Mostra l'elenco di richieste di prenotazione
	3	Clicca sul tasto di conferma	
	4		Mostra "Prenotazione confermata con successo"

1. Il caso d'uso viene attivato quando il paziente seleziona, dalla navbar, il pulsante "**Rilevazioni mediche**".
2. Il sistema reindirizza il paziente a una schermata contenente lo storico delle rilevazioni mediche.

3. Il paziente seleziona il tasto per aggiungere una nuova rilevazione medica, sceglie come tipologia di inserimento "**Inserimento manuale**"
4. Il paziente compila un form per inserire la **tipologia di rilevazione** e il **valore** della rilevazione e clicca sul tasto "salva"

2.7.1 Mockup digitali - Inserimento rilevazione medica

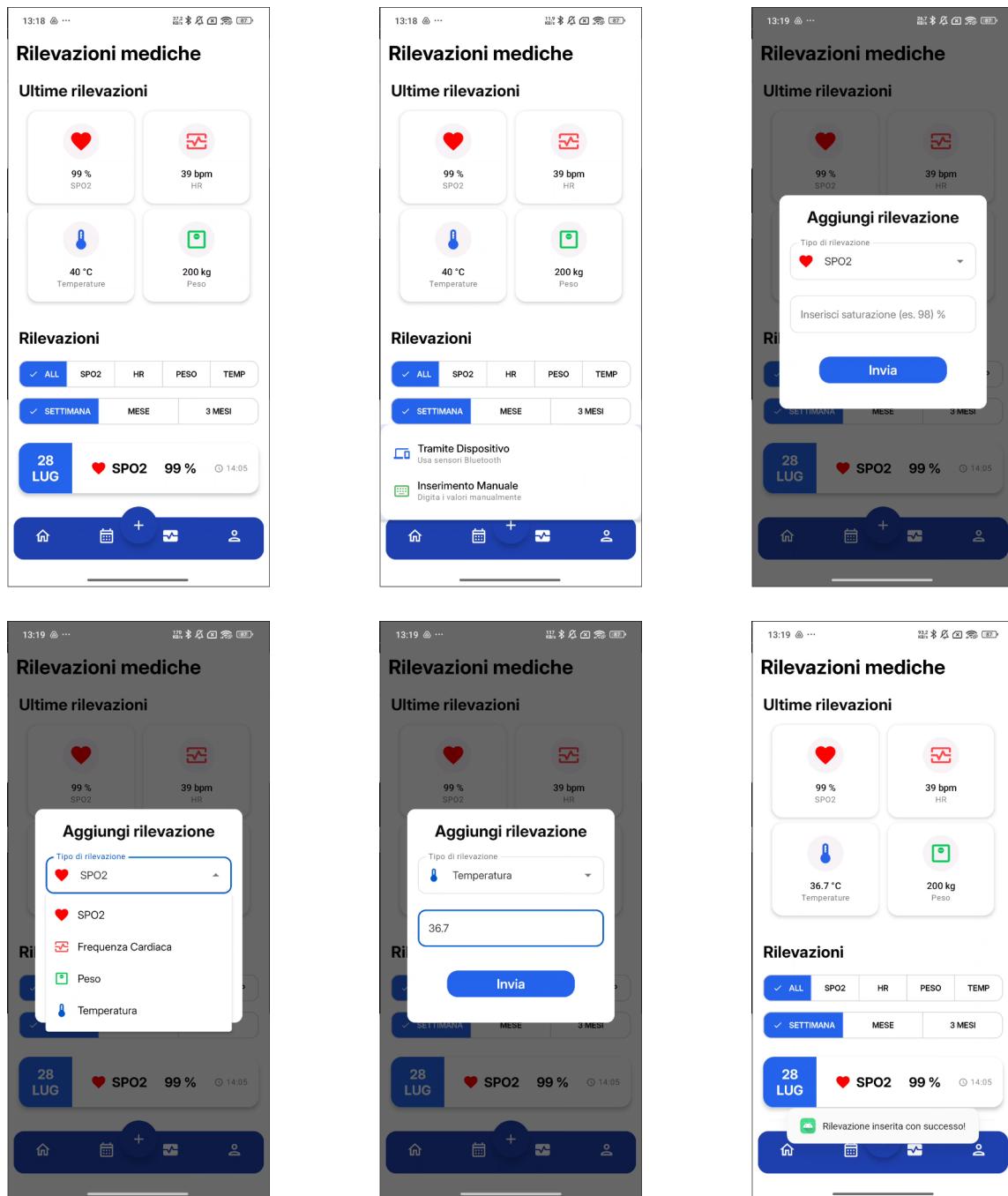


Figura 2.14: Schermate dell'inserimento di una rilevazione medica nell'app

2.8 Osservazioni fatte da stakeholder

Durante la fase di analisi dei requisiti, grazie al supporto e alla disponibilità dei colleghi dell'azienda presso cui è stato svolto il tirocinio curriculare, sono stati condotti numerosi colloqui e confronti diretti con stakeholder reali e potenziali utilizzatori della piattaforma, tra cui medici specialisti, pazienti cronici e personale sanitario coinvolto nella gestione delle visite. Queste interazioni si sono rivelate fondamentali per raccogliere spunti coerenti e osservazioni concrete, utili a orientare in modo consapevole le scelte progettuali.

Di seguito sono riportati i principali punti emersi durante tali confronti:

1. **Chiarezza e semplicità dell'interfaccia:** Molti professionisti sanitari hanno evidenziato l'importanza di un'interfaccia intuitiva, capace di ridurre al minimo il tempo di apprendimento. In particolare, è risultata apprezzata la presenza di dashboard riassuntive e viste grafiche per monitorare rapidamente lo stato clinico dei pazienti.
2. **Visualizzazione dei dati clinici mediante grafici:** È emersa una chiara preferenza per strumenti che consentano di visualizzare in modo interattivo l'andamento dei parametri rilevati (es. pressione, saturazione, temperatura). Questo approccio permette ai medici di individuare più facilmente pattern clinici o situazioni anomale.
3. **Gestione rapida delle prenotazioni:** I medici hanno richiesto un sistema che permetta una gestione agile delle proprie disponibilità e delle richieste di appuntamento da parte dei pazienti. La possibilità di visualizzare, approvare o riprogrammare una prenotazione in pochi clic è stata considerata essenziale.
4. **Inserimento flessibile delle rilevazioni:** Dal lato paziente, è stata sottolineata l'utilità di poter inserire manualmente le proprie misurazioni op-

pure sincronizzarle automaticamente con dispositivi compatibili via Bluetooth. Questa doppia modalità consente di coprire un ampio spettro di utenti, dai più esperti a quelli meno tecnologici.

5. Storico accessibile delle misurazioni: Numerosi utenti hanno espresso la necessità di poter consultare agevolmente lo storico delle proprie rilevazioni, possibilmente filtrandolo per tipo di parametro o intervallo temporale. Ciò facilita il confronto autonomo dei dati e ne aumenta la comprensibilità.

Le osservazioni raccolte sono state integrate nella fase di definizione dei requisiti funzionali, contribuendo a delineare una piattaforma più aderente alle esigenze reali degli utenti e più solida dal punto di vista dell'usabilità.

questa pagina è stata lasciata volutamente bianca

-3-

Design del Sistema

In questa sezione viene descritto il design del sistema **SymbioCare**, a partire dall’architettura generale, fino alle scelte tecnologiche e all’interfaccia utente. La progettazione è stata guidata da principi di modularità, riusabilità, semplicità ed efficienza, in linea con i requisiti funzionali e non funzionali precedentemente analizzati.

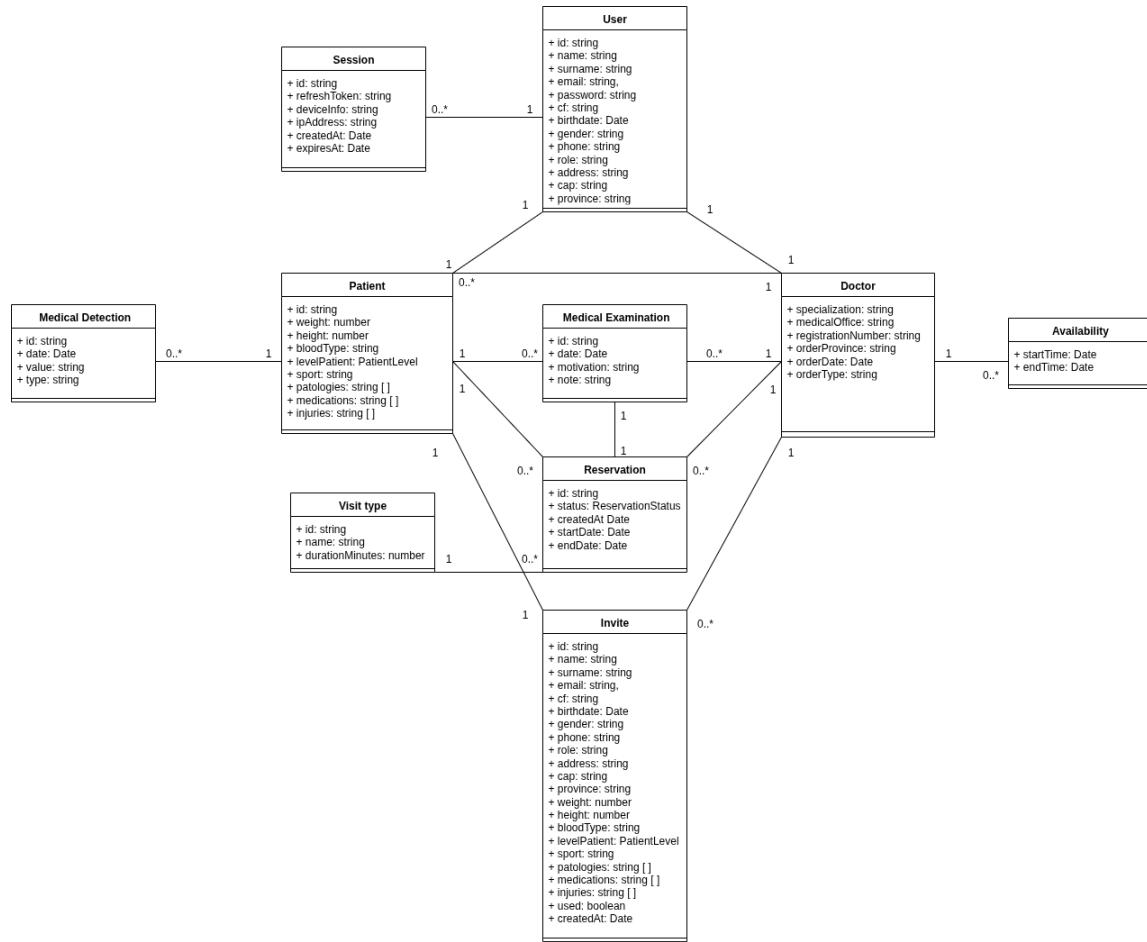
3.1 Architettura del sistema

Si è deciso di adottare un’architettura a **tre livelli** suddivisa in:

- **Livello di presentazione:** rappresentato da **due client distinti**: *una web application destinata ai medici* e *un’app mobile destinata ai pazienti*. Entrambe le interfacce comunicano tramite API REST con il backend.
- **Livello applicativo:** rappresentato da un server che espone un set di API REST per la gestione dei dati e la logica di business.
- **Livello di persistenza:** rappresentato da un sistema di persistenza dei dati relazionale.

Questa architettura garantisce scalabilità, facilità di manutenzione e separazione delle responsabilità.

3.2 Class Diagram



3.2.1 Entità principali

Nel modello concettuale del sistema sono state individuate le seguenti entità fondamentali, ciascuna con un preciso ruolo all'interno del dominio applicativo:

- **Utente (User)**: rappresenta un individuo registrato nella piattaforma. Ogni utente possiede attributi generici come nome, cognome, email, password (hashata), data di nascita e ruolo (es. paziente, dottore, amministratore).
- **Paziente (Patient)**: identifica un utente con ruolo di paziente. Possiede attributi medici specifici come peso, altezza, patologie, infortuni, e altro..
- **Dottore (Doctor)**: identifica un utente con ruolo di medico. Include attributi relativi alla professione (specializzazione, codice identificativo, ecc.)
- **Sessione (Session)**: rappresenta una sessione di accesso al sistema. Contiene informazioni come refresh token (hashato), data di creazione, data di scadenza, dispositivo utilizzato e indirizzo IP.
- **Invito (Invite)**: rappresenta un invito alla registrazione, generato da un medico per consentire a un paziente di registrarsi alla piattaforma in modo controllato e tracciabile.
- **Visita medica (MedicalExamination)**: rappresenta un evento clinico in cui il medico esegue una visita a un paziente. Include dati come data della visita, motivo clinico e note aggiuntive.
- **Prenotazione (Reservation)**: rappresenta la richiesta, da parte di un paziente, di una visita presso lo studio medico. È associata a uno slot temporale specifico, ha uno stato (es. confermata, annullata, in attesa).
- **Tipo di visita (VisitType)**: identifica la natura della visita medica (es. controllo periodico, prima visita).

-
- **Disponibilità (Availability)**: rappresenta un intervallo temporale durante il quale un medico è disponibile per ricevere prenotazioni. Contiene orario di inizio, orario di fine.
 - **Rilevazione medica (MedicalDetection)**: rappresenta un dato clinico raccolto da un paziente, ad esempio tramite dispositivo Bluetooth. Ogni rilevazione include valore, tipo (es. SpO₂, temperatura).

3.2.2 Relazioni principali

Di seguito sono descritte le principali relazioni tra le entità individuate nel sistema, con l'indicazione della cardinalità e del posizionamento delle chiavi esterne:

- Ogni **Utente** è associato esattamente a una delle entità **Paziente** oppure **Dottore**, attraverso una relazione 1:1. Il tipo di utente viene determinato dall'attributo `role`. Come scelta implementativa, la chiave esterna verso User è stata inserita all'interno delle tabelle Patient e Doctor.
- Ogni **Sessione** è riferita a uno specifico **Utente**, secondo una relazione 0:N. Un utente può quindi avere più sessioni attive (es. da dispositivi diversi). La chiave esterna è presente nella tabella Session.
- Ogni **Paziente** può essere associato a più **Rilevazioni mediche**, secondo una relazione 0:N. La chiave esterna verso Patient è contenuta nella tabella MedicalDetection.
- Ogni **Visita medica** è collegata a un unico **Paziente**, secondo una relazione 0:N. La chiave esterna si trova nella tabella MedicalExamination.
- Ogni **Visita medica** è anche riferita a uno specifico **Dottore**, secondo una relazione 0:N. Anche in questo caso, la chiave esterna è contenuta nella tabella MedicalExamination.

- Ogni **Visita medica** è associata a una sola **Prenotazione confermata**, secondo una relazione 1:1. La chiave esterna è presente nella tabella MedicalExamination.
- Ogni **Prenotazione** è associata a un **Tipo di visita**, tramite una relazione 1:N. La chiave esterna si trova nella tabella Reservation.
- Ogni **Prenotazione** è associata a un **Paziente**, tramite una relazione 1:N. La chiave esterna si trova nella tabella Reservation.
- Ogni **Prenotazione** è associata a un **Dottore**, tramite una relazione 1:N. La chiave esterna si trova nella tabella Reservation.
- Ogni **Disponibilità** è riferita a un unico **Dottore**, mentre un dottore può dichiarare più disponibilità. Si tratta quindi di una relazione 1:N, con chiave esterna nella tabella Availability.
- Ogni **Invito** è rivolto a un singolo **Paziente**, secondo una relazione 1:1. La chiave esterna è collocata nella tabella Invite.
- Ogni **Invito** è generato da un **Dottore**, che può crearne molteplici. La relazione è quindi 1:N, con chiave esterna nella tabella Invite.

-4-

Tecnologie Utilizzate

Nella fase di progettazione dell’ecosistema software, è stata condotta un’attenta analisi comparativa delle tecnologie moderne più diffuse nel contesto dello sviluppo full-stack. Le scelte finali sono state guidate da criteri di:

- Affidabilità e maturità delle tecnologie
- Scalabilità dell’architettura nel lungo termine
- Facilità d’integrazione tra i componenti
- Supporto della community e disponibilità di risorse/documentazione
- Curve di apprendimento compatibili con il team di sviluppo

4.1 Tecnologie Frontend Web App

Per lo sviluppo dell’interfaccia web rivolta ai medici, è stata adottata la libreria **React** [7]. Consente la creazione di interfacce utente dinamiche secondo un paradigma dichiarativo e component-based, ideale per applicazioni ad alta interattività come quella oggetto di questo progetto.

Le principali motivazioni alla base della scelta di React sono le seguenti:

- **Diffusione e adozione:** Come confermato dalla Stack Overflow Developer Survey 2024 [8], risulta il framework più utilizzato per lo sviluppo



Figura 4.1: Logo di React

frontend, ed è adottata da aziende leader del settore tecnologico (es. Meta, Airbnb, Netflix).

- **Modularità e riusabilità:** l'approccio a componenti consente di suddividere l'interfaccia in blocchi riutilizzabili, facilitando la manutenzione e l'estensione futura del sistema [9].
- **Gestione efficiente dello stato:** grazie a strumenti come useState¹, , useEffect² e a librerie avanzate (es. Redux³, React Query⁴),), è possibile sincronizzare lo stato dell'interfaccia con quello dei dati in modo reattivo ed efficiente.
- **Ecosistema ricco e community attiva:** esistono migliaia di pacchetti open-source e soluzioni già pronte per ogni esigenza, con una documentazione abbondante e costantemente aggiornata.
- **Supporto SPA (Single Page Application):** React si integra perfettamente con router e toolchain (es. Vite, React Router) per costruire interfacce

¹useState è un hook di React che consente di dichiarare e aggiornare variabili di stato all'interno di un componente funzionale.

²useEffect è un hook di React utilizzato per eseguire eggetti collaterali, come chiamate API o aggiornamenti del DOM, in risposta a cambiamenti di stato o proprietà.

³Redux è una libreria JavaScript per la gestione dello stato globale, basata sul concetto di store unico e immutabilità dei dati.

⁴React Query è una libreria che semplifica la gestione dello stato asincrono, in particolare per il fetching e la sincronizzazione dei dati da API.

fluide e a caricamento istantaneo.

Alternative considerate:

- **Vue.js** [10]: libreria reattiva e semplice da apprendere, molto adatta a progetti di piccola/media scala. Tuttavia, presenta una struttura meno rigorosa rispetto a React e una minore disponibilità di componenti enterprise-ready.
- **Angular** [11]: framework completo e potente, ideale per soluzioni enterprise di grandi dimensioni. Tuttavia, la sua curva di apprendimento più ripida, unita alla complessità architettonale, lo ha reso meno adatto alle tempistiche e alle esigenze progettuali di questo contesto.



Figura 4.2: Logo di Mantine UI

Per lo sviluppo dell’interfaccia utente è stata adottata la libreria **Mantine UI** [12], che si distingue per la sua leggerezza, flessibilità e supporto nativo a TypeScript.

L’integrazione di Mantine ha consentito di ottenere:

- un **design coerente e responsive** per tutti i dispositivi,
- un elevato livello di **accessibilità** (compatibilità con screen reader, focus trap, tastiera),
- un **time-to-market ridotto**, grazie alla disponibilità di oltre 100 componenti già pronti (modali, notifiche, datepicker, tooltip, stepper, ecc.),

-
- un'elevata **personalizzazione visiva**, tramite override di temi e classi, perfettamente integrata con CSS Modules.

In confronto a soluzioni come **Material UI** [13] (più rigido) o **Bootstrap** [14] (meno moderno), Mantine ha rappresentato un buon compromesso tra velocità di sviluppo, qualità visiva e flessibilità di layout.

4.2 Tecnologie Frontend Mobile



(a) Kotlin



(b) Jetpack Compose

Figura 4.3: Tecnologie utilizzate per il frontend mobile

Per lo sviluppo dell'app mobile destinata ai pazienti, è stata adottata una soluzione nativa Android basata su **Jetpack Compose** [15], lo standard più moderno nel panorama dello sviluppo Android [16].

Jetpack Compose è un moderno toolkit dichiarativo per la costruzione di UI Android, che consente di scrivere interfacce in modo reattivo, simile a React. **Le principali motivazioni alla base della scelta di JetpackCompose sono le seguenti:**

- Kotlin [17] è il linguaggio ufficiale supportato da Google per Android
- Jetpack Compose riduce drasticamente la complessità del codice UI
- Perfetta integrazione con le API Android per Bluetooth, fotocamera, sensori

Alternative considerate:

- **Flutter (Dart)** [18]: molto performante ma con meno supporto nativo a feature hardware avanzate (es. BLE)

-
- **React Native**[19]: non nativamente Android, meno adatto per integrazione stretta con API di sistema

4.3 Tecnologie Backend



Per lo sviluppo del backend è stato scelto il framework **NestJS** [20], una soluzione moderna e modulare costruita sopra **Node.js** [21] e interamente basata su **TypeScript**[22]. NestJS si distingue per la sua architettura ispirata a quella di Angular, offrendo un solido supporto per concetti avanzati come dependency injections, middlewares, guards, interceptors e validazione basata su *decoratorss*.

Le principali motivazioni alla base della scelta di NestJS sono le seguenti:

- **Modularità e scalabilità:** NestJS permette di organizzare il codice in moduli autonomi, facilitando la manutenzione e l'estendibilità del sistema.

- **Integrazione nativa con librerie chiave:** il supporto diretto a `class-validator`⁵, `class-transformer`⁶ e `TypeORM` semplifica la validazione dei dati e l'interazione con il database.
- **Allineamento con TypeScript:** l'utilizzo completo di TypeScript aumenta la sicurezza del codice tramite tipizzazione statica.
- **Community e documentazione:** NestJS è in rapida crescita ed è accompagnato da una documentazione ufficiale completa e aggiornata, che ne facilita l'adozione anche in contesti accademici o team non enterprise.

Alternative considerate:

- **Express.js** [23]: troppo minimale, avrebbe richiesto molta più configurazione manuale
- **Spring Boot (Java)** [24]: solido ma più verboso e con ciclo di sviluppo più lento
- **Django (Python)**[25]: ottimo per prototipi ma meno flessibile per strutture modulari complesse

⁵`class-validator` è una libreria per TypeScript/JavaScript che consente di applicare regole di validazione ai dati tramite decoratori, semplificando il controllo automatico degli input.

⁶`class-transformer` è una libreria per TypeScript/JavaScript che permette di trasformare oggetti plain in istanze di classi e viceversa, facilitando la serializzazione e la deserializzazione dei dati.

Object Relational Mapping (ORM)



Per la gestione della persistenza è stato utilizzato **TypeORM** [26], un Object Relational Mapper per TypeScript che consente di modellare le entità come classi e di eseguire operazioni CRUD su database relazionali tramite un’interfaccia ad alto livello.

4.4 Tecnologie di persistenza



Come sistema di gestione del database è stato adottato **PostgreSQL** [27], uno dei RDBMS open-source più affidabili, performanti e ampiamente utilizzati in ambito professionale.

Le principali motivazioni alla base della scelta di PostgreSQL sono le seguenti:

- **Ricchezza funzionale:** PostgreSQL supporta vincoli di integrità, chiavi esterne, transazioni ACID, funzioni personalizzate, e tipi avanzati come JSONB per la gestione di dati semi-strutturati.
- **Performance e ottimizzazione:** ottimizzato per gestire grandi volumi di dati e workload concorrenti in scenari ad alta intensità di lettura/scrittura.
- **Estendibilità:** possibilità di creare funzioni utente, estensioni (es. PostGIS) e personalizzare il comportamento del database in modo flessibile.

Alternative considerate:

- **MySQL/MariaDB** [28] [29]: simili ma con minore supporto a JSON e funzionalità avanzate
- **MongoDB:** [30] ottimo per dati semi-strutturati, ma meno adatto per modelli relazionali con forti vincoli di integrità

—5—

Progettazione e codifica

Questo capitolo descrive nel dettaglio il processo di realizzazione del sistema **SymbioCare**, a partire dalla progettazione concreta dei componenti software fino alla loro effettiva implementazione. Dopo aver definito l’architettura generale e le scelte tecnologiche nel capitolo precedente, si passa ora ad analizzare come queste siano state tradotte in codice, tenendo conto dei vincoli tecnici, delle esigenze degli utenti e degli obiettivi funzionali del progetto.

L’intero sistema è stato suddiviso in tre macro-componenti: **backend**, **web app per i medici** e **app mobile per i pazienti**. Ciascuna parte è stata progettata in modo modulare, per favorire la manutenibilità e l’estendibilità futura, e implementata seguendo best practice consolidate nei rispettivi ambiti (es. pattern MVVM per Android, architettura modulare per NestJS, approccio component-based in React).

Nel corso del capitolo verranno illustrati:

- l’organizzazione del codice e la struttura dei progetti;
- le modalità di interazione tra i diversi componenti software;
- le principali funzionalità implementate lato backend e frontend;
- le strategie adottate per l’acquisizione e la gestione dei dati medici;
- gli strumenti utilizzati per il testing.

Verranno inoltre evidenziate le principali difficoltà incontrate durante lo sviluppo e le soluzioni adottate per superarle, con l’obiettivo di fornire una visione completa e concreta del processo di sviluppo dell’intero ecosistema applicativo.

5.1 Organizzazione del codice e Versioning

Nel progetto **SymbioCare** è stata adottata una struttura **multi-repository**, con tre repository distinti per ciascuna componente del sistema (backend, frontend web, frontend mobile).

Come sistema di versionamento è stato utilizzato **Git**, con repository ospitati sulla piattaforma cloud **GitHub**, mostrata nella Figura 5.1.



Figura 5.1: Logo della piattaforma GitHub [31], utilizzata per il versionamento del codice

L’adozione di Git ha consentito di:

- tracciare puntualmente ogni modifica in modo reversibile;
- supportare lo sviluppo parallelo tramite branch dedicati;
- mantenere una cronologia chiara delle revisioni;
- gestire in modo strutturato eventuali conflitti.

La scelta di mantenere le codebase separate ha favorito la **separazione delle responsabilità**, l’**indipendenza dei workflow di sviluppo** e la possibilità di effettuare **rilasci modulari**.

Backend (NestJS)

Nel backend, sviluppato con NestJS, il codice è stato organizzato seguendo una

struttura modulare. Ogni funzionalità è incapsulata in un **modulo NestJS**, racchiuso in un proprio *package*, contenente:

- **controller.ts**: definisce gli endpoint REST e riceve le richieste HTTP;
- **service.ts**: contiene la logica applicativa e coordina le operazioni;
- **repository.ts**: gestisce l'accesso al database tramite TypeORM;
- **entity.ts** (se presente): definisce il modello di persistenza relazionale.

Il pattern adottato è **Controller → Service → Repository**, che garantisce una netta **separazione delle responsabilità**:

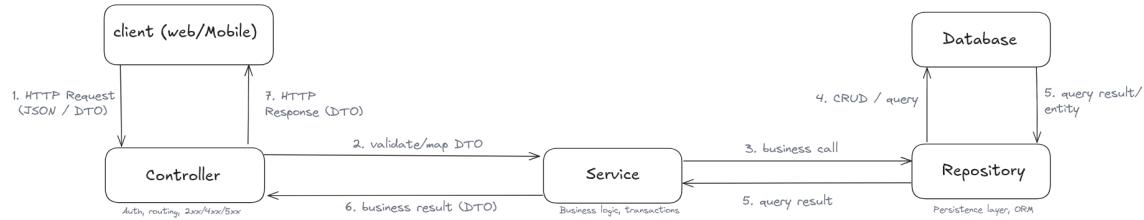


Figura 5.2: Pattern CSR

- Il **Controller** riceve la richiesta e la valida inizialmente, chiama i metodi dei service.
- Il **Service** implementa la logica del dominio, puo' effettuare operazioni sul database utilizzando le repository.
- Il **Repository** si occupa esclusivamente dell'accesso ai dati.

Frontend Web (React)

Nel progetto React non è stato adottato un pattern rigido, ma si è comunque mantenuta una chiara separazione logica dei compiti tramite cartelle tematiche:

- **pages/**: contiene le pagine principali del sito (routing);
- **components/**: raccoglie i componenti UI riutilizzabili;
- **api/**: definisce i servizi API, tramite istanze axios centralizzate;

-
- `context/`: ospita i React Context¹ per la gestione dello stato globale;
 - `hooks/`: include custom hook (es. `useQuery`, `useMutation`)².
 - `styles/`: contiene gli stili globali e i CSS module;
 - `types/` e `utils/`: rispettivamente per definizioni TypeScript e funzioni di utilità.

Questa struttura ha facilitato la scalabilità dell’interfaccia e la gestione collaborativa del codice, mantenendo le responsabilità ben distribuite.

App Mobile (Kotlin + Jetpack Compose)

Nel progetto Android è stato adottato il pattern **MVVM (Model-View-ViewModel)**, che consente di separare in maniera chiara la logica di presentazione dallo stato e dalla logica applicativa. La suddivisione in package è la seguente:

- `remote/`: contiene le interfacce Retrofit[32], i moduli di rete, gli interceptor e le repository;
- `viewModel/`: include tutti i `ViewModel`, responsabili della logica e dello stato dell’interfaccia;
- `views/`: raccoglie tutte le schermate e i componenti UI realizzati in Jetpack Compose;
- `models/`: definisce i modelli dati utilizzati nelle varie componenti.

Nel pattern MVVM:

- La **View (UI)** osserva lo stato esposto dal **ViewModel** e si aggiorna automaticamente;
- Il **ViewModel** gestisce la logica e dialoga con le repository;

¹Il Context API di React consente di condividere dati tra componenti senza dover passare le *props* manualmente lungo la gerarchia, risultando utile per gestire stato globale o configurazioni comuni.

²Gli custom hook `useQuery` e `useMutation` sono parte della libreria React Query (oggi TanStack Query) e servono rispettivamente per gestione automatica del fetching dati (con caching, refetching e gestione degli stati) e per operazioni di mutazione (creazione, aggiornamento o cancellazione), fornendo anche stati come `isLoading`, `isError`, `isSuccess`, callback per il successo o l’errore, e supporto per retry e aggiornamenti ottimistici.

- Le **Repository** accedono ai dati remoti (tramite Retrofit) o locali.

Questo approccio garantisce una maggiore testabilità del codice, una separazione netta delle responsabilità e una gestione reattiva dello stato dell’interfaccia.

5.2 Funzionalità principali

In questa sezione vengono illustrate le principali funzionalità implementate nel sistema, analizzandone il flusso di utilizzo end-to-end e la logica tecnica adottata. Ogni funzionalità viene descritta evidenziando sia gli aspetti di **backend** (API, sicurezza, persistenza) sia quelli di **frontend**, distinguendo tra l’interfaccia **web per il medico** e l’applicazione **mobile per il paziente**.

Le funzionalità che meritano un approfondimento sono le seguenti:

- **Autenticazione e gestione sessioni:** garantire l’accesso sicuro al sistema tramite JWT, refresh token e gestione delle sessioni utente.
- **Visualizzazione elenco dei pazienti:** permettere al medico di consultare in tempo reale la lista dei propri assistiti.
- **Visualizzazione dati di un paziente:** consentire la consultazione delle informazioni personali.
- **Visualizzazione delle rilevazioni mediche di un paziente:** consentire la consultazione delle rilevazioni inserite dal paziente tramite app mobile.
- **Visualizzazione dello storico delle visite mediche di un paziente:** consentire la consultazione dei report delle visite mediche passate.
- **Inserimento di un nuovo paziente:** consentire al medico la creazione di un invito digitale per l’onboarding del paziente.
- **Visualizzazione disponibilità per appuntamenti:** consentire al medico di visualizzare le sue disponibilità per gli appuntamenti.

-
- **Gestione disponibilità per appuntamenti:** consentire al medico di eliminare le sue disponibilità per gli appuntamenti ed inserirne di nuove.
 - **Visualizzazione degli appuntamenti confermati:** consentire al medico di visualizzare gli appuntamenti confermati
 - **Valutazione di una richiesta:** consentire al medico di accettare o rifiutare le richieste di prenotazione in base alle proprie necessità.
 - **Registrazione del paziente tramite app mobile:** permettere al paziente di completare la registrazione a partire dall'invito ricevuto tramite QR code.
 - **Richiesta di prenotazione:** permettere al paziente di inviare una richiesta di appuntamento selezionando uno slot libero.
 - **Inserimento di un report di visita:** consentire al medico di registrare un report associato al paziente al termine di un appuntamento.

5.2.1 Autenticazione e gestione delle sessioni

Uno degli aspetti fondamentali nello sviluppo di un sistema software che tratta dati sensibili, come nel caso di un'applicazione in ambito sanitario, è l'implementazione di un processo di autenticazione sicuro ed affidabile. La protezione delle credenziali e la gestione delle sessioni utente costituiscono infatti un requisito imprescindibile, sia per garantire la sicurezza dei dati personali, sia per assicurare la continuità operativa del servizio.

Per l'autenticazione è stato adottato lo standard **JSON Web Token (JWT)** [33], una tecnologia ormai consolidata e ampiamente diffusa, che permette di scambiare in maniera sicura informazioni tra client e server sotto forma di oggetti JSON firmati digitalmente. I JWT sono costituiti da tre parti principali (header, payload e signature) e risultano particolarmente adatti in scenari di

tipo *stateless*, in cui il server non mantiene informazioni di sessione ma delega al token la responsabilità di rappresentare l'identità dell'utente.

Nel contesto di questo progetto si è scelto di implementare una variante più sicura, basata su una combinazione di **access token** e **refresh token**. L'*access token*, con validità limitata (15 minuti), viene utilizzato per autenticare le richieste dell'utente; il *refresh token*, invece, ha una durata più lunga (7 giorni) ed è associato ad una specifica sessione utente. Ogni utente può quindi disporre di più sessioni attive, ciascuna identificata dal proprio refresh token, che viene salvato in maniera sicura e hashato nel database.

Il flusso di funzionamento è il seguente:

1. Quando il client effettua una richiesta, essa include l'*access token* (tramite cookie HTTP-only).
2. Se l'*access token* è valido e non scaduto, la richiesta viene elaborata normalmente.
3. In caso di scadenza, interviene il *refresh token*. Se questo risulta valido, il server genera un nuovo access token e prosegue l'elaborazione.
4. Se anche il *refresh token* è scaduto o non valido, l'utente riceve un errore di tipo 401 Unauthorized e deve rieseguire il login.

Questa architettura consente di bilanciare sicurezza e usabilità: da un lato l'uso di access token a breve durata riduce l'impatto di eventuali furti di credenziali, dall'altro l'impiego di refresh token evita la necessità di un'autenticazione manuale troppo frequente, garantendo una buona esperienza d'uso.

Middleware e controllo degli accessi

La logica di autenticazione è stata realizzata mediante un **middleware**³. Tale componente viene invocato automaticamente ad ogni richiesta diretta verso un

³Un middleware è un software che funge da strato intermedio tra le applicazioni e le componenti sottostanti

endpoint protetto e si occupa di:

- validare la presenza e la correttezza dei token;
- estrarre dal token i dati utente e associarli al contesto della richiesta;
- bloccare immediatamente le chiamate non autorizzate, restituendo un errore.

Oltre al semplice controllo di autenticazione, è necessario anche limitare l'accesso a specifiche risorse o operazioni in base al ruolo dell'utente. Nel sistema sono stati definiti tre ruoli principali: **Admin**, **Doctor**, **Patient**.

Guard e decorator per la protezione dei controller

Per la protezione dei controller è stata sviluppata una **Guard** denominata RolesGuard, che implementa l'interfaccia CanActivate. Le guard in NestJS rappresentano un meccanismo di sicurezza che intercetta la richiesta prima dell'esecuzione del metodo del controller, decidendo se proseguire o bloccare l'operazione.

La RolesGuard utilizza il servizio Reflector per leggere i metadati associati ai metodi tramite il decorator personalizzato @Roles. Questo approccio consente di dichiarare in maniera chiara e centralizzata i ruoli autorizzati per ogni handler del controller. Se l'utente non possiede il ruolo richiesto, la guard genera un'eccezione ForbiddenException, impedendo l'accesso.

Sintesi

Dal punto di vista implementativo, il modulo di autenticazione gestisce:

- **Registrazione e login:** l'utente riceve access token e refresh token, salvati in cookie protetti (`httpOnly`, `secure`, `sameSite=lax`);
- **Gestione delle sessioni:** ogni login genera un nuovo record nella tabella delle sessioni, associato all'utente e al dispositivo;

- **Refresh automatico:** un middleware intercetta le richieste e, in caso di scadenza dell'access token, utilizza il refresh token per generarne uno nuovo;
- **Logout:** l'invalidamento della sessione comporta la cancellazione del refresh token dal database e la rimozione dei cookie lato client.

5.2.2 Visualizzazione elenco dei pazienti

Una funzionalità centrale dell'applicazione è la possibilità, per il medico, di visualizzare in maniera strutturata l'elenco dei propri pazienti.

Lato backend

L'endpoint dedicato è GET /doctor/patients, accessibile esclusivamente agli utenti con ruolo **Doctor**. Il flusso è il seguente:

1. Il client effettua una richiesta HTTP al backend, includendo parametri di query opzionali (page, limit, search).
2. Il controller invoca il metodo del DoctorService, che si occupa di interfacciarsi con il database tramite **TypeORM**[26].
3. Viene effettuata una query che recupera i pazienti associati al medico autenticato. La query supporta:
 - **paginazione** (page, limit) per migliorare la scalabilità;
 - **ricerca testuale** (search) su nome, cognome, codice fiscale ed email.
4. I risultati vengono mappati in un DTO (PatientsResponse), filtrati e restituiti al client in formato JSON, comprensivo di dati e metadati (total, page, limit).

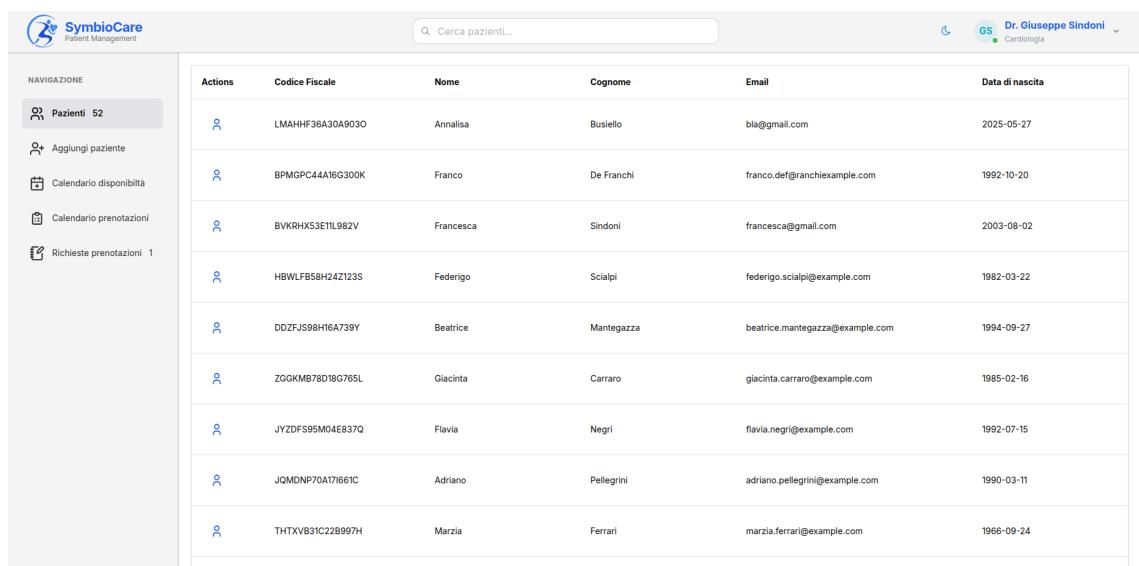
Lato frontend

Il frontend React consuma l'endpoint tramite uno specifico hook personalizzato (`usePatients`), realizzato con **React Query**[34], che gestisce:

- caching delle richieste,
- gestione automatica di retry in caso di errori,
- sincronizzazione dello stato dei dati con la UI.

I dati ricevuti vengono visualizzati tramite una tabella interattiva costruita con la libreria **Mantine React Table** [35]. Questa libreria consente di implementare rapidamente funzionalità di:

- visualizzazione tabellare dei pazienti con colonne configurabili (nome, cognome, codice fiscale, email, data di nascita, ecc.),
- paginazione lato server (*manual pagination*),
- azioni sulle righe, come l'apertura della scheda di dettaglio del paziente.



The screenshot shows the SymbioCare Patient Management interface. On the left, there's a sidebar with navigation links: 'Pazienti 52', 'Aggiungi paziente', 'Calendario disponibilità', 'Calendario prenotazioni', and 'Richieste prenotazioni 1'. The main area features a table titled 'Cerca pazienti...' with columns: Actions, Codice Fiscale, Nome, Cognome, Email, and Data di nascita. The table lists nine patient entries with sample data.

Actions	Codice Fiscale	Nome	Cognome	Email	Data di nascita
View	LMAHHF36A30A903O	Annalisa	Busiello	bla@gmail.com	2025-05-27
View	BPMGPC44A16G300K	Franco	De Franchi	franco.defranchi@example.com	1992-10-20
View	BVKRHX53E11L982V	Francesca	Sindoni	francesca@gmail.com	2003-08-02
View	HBWLFB58H24Z123S	Federigo	Scialpi	federigo.scialpi@example.com	1982-03-22
View	DDZFJS98H10A739Y	Beatrice	Mantegazza	beatrice.mantegazza@example.com	1994-09-27
View	ZGGKMB78D18G765L	Giacinta	Carraro	giacinta.carraro@example.com	1985-02-16
View	JY2DFS95M04E837Q	Flavia	Negri	flavia.negri@example.com	1992-07-15
View	JQMDNP70A17I661C	Adriano	Pellegrini	adriano.pellegrini@example.com	1990-03-11
View	THTXVB31C22B997H	Marzia	Ferrari	marzia.ferrari@example.com	1966-09-24

Figura 5.3: Schermata pazienti

5.2.3 Visualizzazione dati di un paziente

Questa funzionalità consente al medico di accedere a tutte le informazioni cliniche e anagrafiche di uno specifico paziente.

Lato Backend

L'endpoint dedicato è GET `doctor/patients/:patientId`, accessibile esclusivamente agli utenti con ruolo **Doctor**.

E' importante distinguere due tipologie di pazienti:

- **Pazienti non registrati all'app mobile:** in questo caso esiste solamente una tupla nella tabella Patient e una corrispondente nella tabella Invite, che rappresenta l'invito di registrazione generato dal medico. Tutti i dati anagrafici del paziente vengono quindi conservati nell'invito stesso.
- **Pazienti registrati all'app mobile:** oltre alla tupla in Patient, è presente anche un record nella tabella User, che contiene i dati confermati o aggiornati dal paziente in fase di registrazione.

In questa fase il paziente non ha la possibilità di modificare i dati identificativi (nome, cognome, codice fiscale, e-mail), che restano quelli definiti dal medico nell'invito, ma può correggere i dati di residenza e impostare la propria password di accesso.

Questa distinzione influenza direttamente il flusso logico dell'endpoint :

1. Il client invia una richiesta HTTP al backend specificando l'identificativo del paziente.
2. Il controller invoca il metodo corrispondente del DoctorService.
3. Il service interroga la tabella Patient per recuperare il paziente con l'id specificato, verificando che esso appartenga al medico richiedente (controllo di sicurezza ulteriore, utile a prevenire accessi non autorizzati).

4. A questo punto, il flusso si biforca:

- **Paziente registrato:** se è presente la relazione `patient.user`, vengono restituite le informazioni combinate di `Patient` e `User`. Il campo `inviteId` viene impostato a `null`.
- **Paziente non registrato:** viene effettuata una query aggiuntiva alla tabella `Invite`. Se l'invito è presente, i dati vengono estratti da quest'ultimo e restituiti insieme all'identificativo dell'invito. Se l'invito non esiste, viene sollevata un'eccezione.

5. Il risultato viene infine trasformato in un DTO (`PatientItem`) che garantisce un output tipizzato e sicuro da restituire al client.

Lato Frontend

Dal lato frontend, l'endpoint GET `doctor/patients/:patientId` viene consumato tramite un hook personalizzato (`usePatient`), sviluppato con **React Query**[34].

La navigazione verso la pagina di dettaglio del paziente avviene a partire dalla tabella dei pazienti: cliccando sulla *row action* con l'icona a forma di omino, il medico viene reindirizzato dalla schermata `/patients/` a `/patients/:patientId`.

La pagina in questione mostra le informazioni anagrafiche e cliniche del paziente in un layout strutturato. La logica di rendering si adatta a due scenari distinti, a seconda che il paziente risulti registrato o meno.

Paziente non registrato all'app mobile In questo scenario viene visualizzato un pulsante che permette al medico di mostrare il codice di invito.

Tale codice non è altro che l'`inviteId` restituito dal backend, convertito in **QR Code** tramite la libreria `react-qr-code`[36]. Il QR Code viene mostrato all'interno di un modal, così da essere scansionato dal paziente per completare la registrazione all'app mobile.

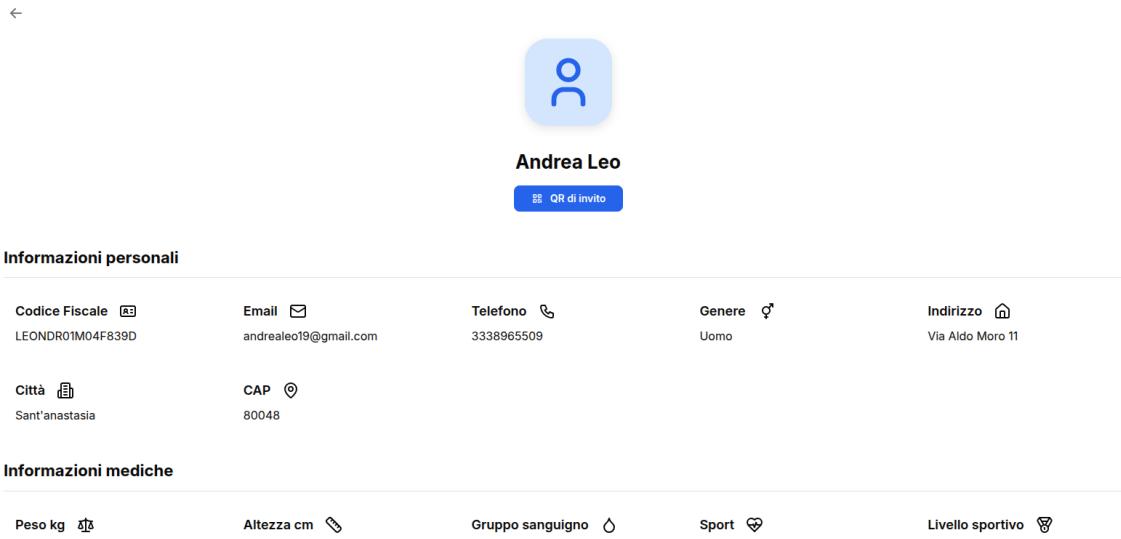


Figura 5.4: Paziente non registrato all'app mobile

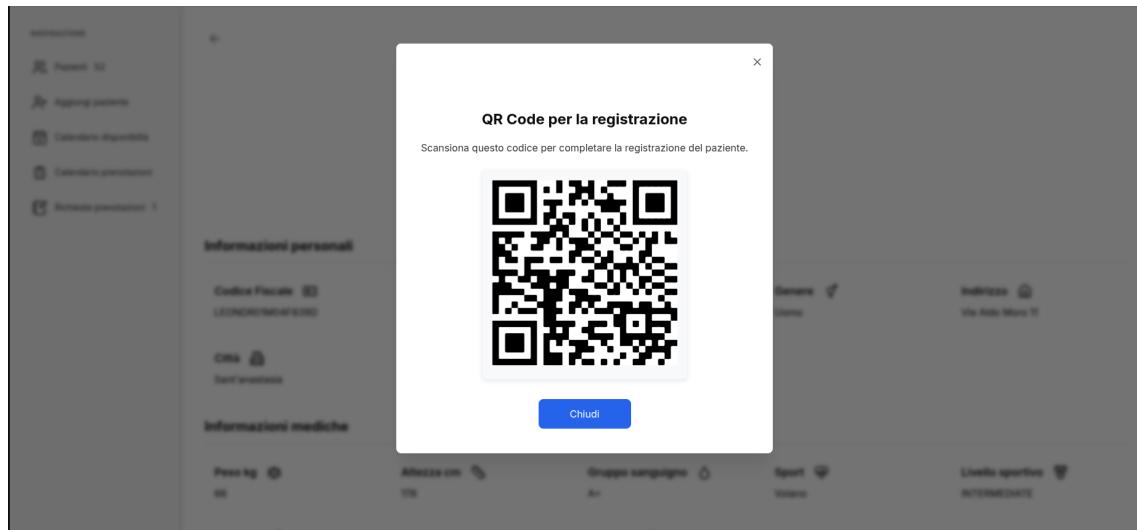


Figura 5.5: Codice Qr dell'invito

Paziente già registrato In questo caso non viene mostrato il pulsante *QR di invito*, ma compare invece un pulsante per inserire un nuovo report di visita medica.

The screenshot shows a patient profile for "Giuseppe Sindoni". At the top, there is a blue button labeled "Aggiungi visita". Below it, under "Informazioni personali", there are fields for Codice Fiscale (WKGKSD85P67M108X), Email (g@gmail.com), Telefono (781267632), Genere (Uomo), and Indirizzo (Via Dante Alighieri 12). Under "Informazioni mediche", there are fields for Peso kg (100), Altezza cm (200), Gruppo sanguigno (B-), Sport (Basket), and Livello sportivo (ADVANCED). A section titled "Altro" contains two expandable accordions: "Rilevazioni parametri" and "Visite mediche".

Figura 5.6: Schermata paziente registrato

Inoltre viene resa disponibile una sezione *Altro*, che attraverso un componente ad accordion permette di accedere allo storico delle visite mediche (Medical Examinations) e ai grafici delle rilevazioni cliniche (Medical Detections).

Questa logica di interfaccia, basata su *conditional rendering*, garantisce che il medico disponga sempre delle azioni coerenti con lo stato del paziente (registrato o meno).

5.2.4 Visualizzazione delle rilevazioni mediche di un paziente

Questa feature rientra tra le più fondamentali di questo progetto, lo scopo è quello di permettere al dottore di visualizzare le rilevazioni mediche inserite dal paziente mediante app mobile.

Lato Backend

L'endpoint dedicato è GET /medical-detection/patient/:patientId, accessibile esclusivamente a utenti con ruolo **Doctor**. Il flusso di esecuzione è il seguente:

1. Il client invia una richiesta HTTP al backend specificando l'ID del paziente come parametro di path. Facoltativamente, può includere i parametri di query type, startDate e endDate, che consentono rispettivamente di filtrare per tipologia di rilevazione, per data di inizio e per data di fine dell'intervallo temporale da analizzare.
2. Il **controller** riceve la richiesta e la inoltra al service
3. Il **service** esegue i seguenti passi:
 - verifica l'esistenza del paziente tramite il PatientRepository;
 - costruisce dinamicamente la query sulle rilevazioni mediche, filtrando le rilevazioni di quello specifico paziente
 - applica eventuali filtri aggiuntivi in base ai parametri di query (type, startDate, endDate);
 - ordina i risultati in ordine crescente di data;
 - recupera sia l'elenco delle rilevazioni sia il numero totale di elementi (getManyAndCount).

-
4. Se non vengono trovate rilevazioni compatibili con i criteri forniti, il servizio solleva un'eccezione `NotFoundException`.
 5. In caso positivo, i dati vengono trasformati in un DTO `MedicalDetectionsResponse`, contenente:
 - il numero totale di rilevazioni (`total`),
 - la lista delle rilevazioni con `value`, `type` e `date`.
 6. Infine, il DTO viene restituito al controller, che lo invia come risposta HTTP al client.

Lato Frontend

Dal lato frontend, l'endpoint GET `/medical-detection/patient/:patientId` viene consumato attraverso un hook personalizzato, che gestisce il recupero delle rilevazioni cliniche dal backend. Per ogni tipologia di dato supportata dalla piattaforma — **SpO₂, battito cardiaco, peso corporeo, temperatura corporea** — viene effettuata una chiamata API dedicata, così da fornire al medico una visione distinta e organizzata per singolo parametro.

La funzionalità è integrata all'interno della schermata del paziente (registrato tramite app mobile). Attraverso un menu ad accordion, il medico può espandere le diverse sezioni e accedere alle rilevazioni cliniche disponibili.

I dati ricevuti vengono rappresentati graficamente mediante semplici diagrammi cartesiani:

- sull'asse delle ascisse è riportato l'andamento temporale (date delle misurazioni),
- sull'asse delle ordinate sono mostrati i valori numerici associati alla rilevazione.

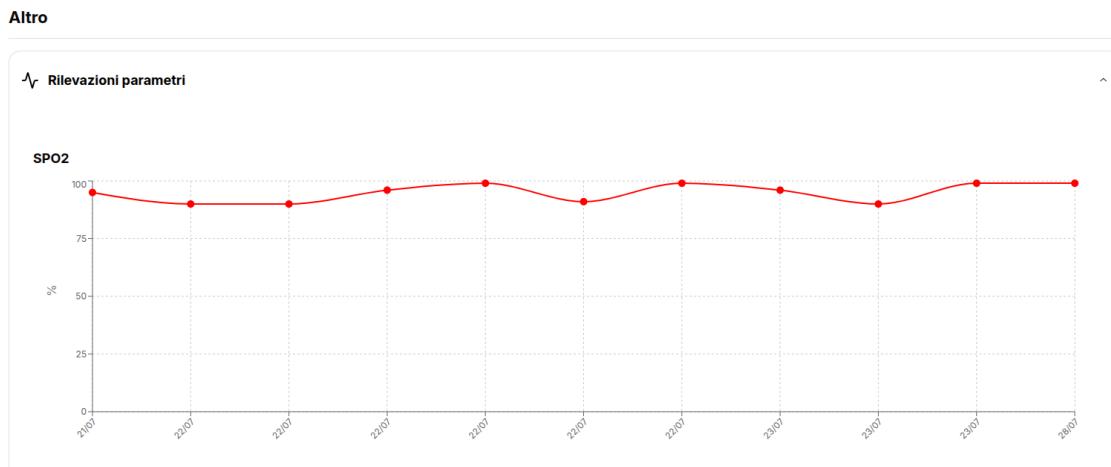


Figura 5.7: Visualizzazione grafica dei dati di SpO₂.

L’obiettivo di questa rappresentazione è fornire al medico un quadro chiaro e immediato sull’andamento delle condizioni del paziente, con la possibilità di confrontare visivamente i valori in periodi diversi. Elemento rilevante del design è l’**interattività**: i grafici consentono di esplorare i dati, evidenziando valori specifici al passaggio del cursore, in modo da rendere più intuitiva l’interpretazione clinica.

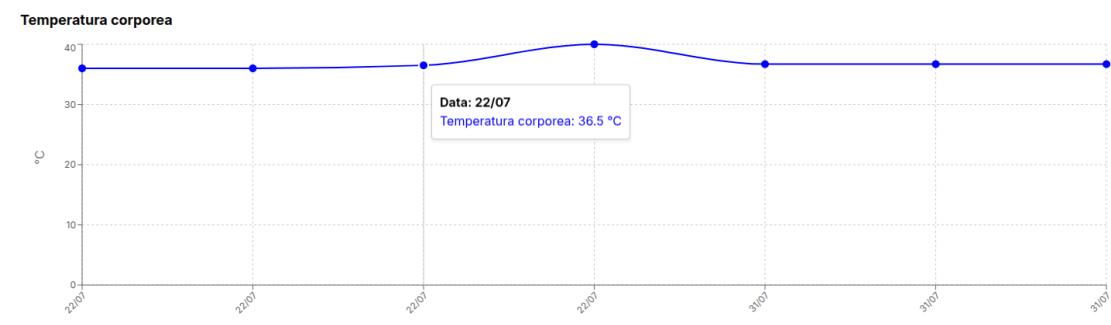


Figura 5.8: Grafico interattivo della temperatura corporea.

I grafici interattivi sono stati implementati mediante la libreria Recharts[37], in particolare utilizzando il componente LineChart, che consente di rappresentare l’andamento temporale delle rilevazioni mediche.

5.2.5 Gestione disponibilità per appuntamenti

Questa funzionalità consente al medico di organizzare la propria agenda, visualizzando le proprie disponibilità, inserendo nuove fasce orarie oppure eliminando quelle già esistenti. Il sistema è stato progettato per garantire sia la correttezza dei dati inseriti (controlli su durata e sovrapposizioni), sia un'interazione semplice e intuitiva lato utente.

Visualizzazione disponibilità lato backend

L'endpoint dedicato è GET /doctor/availability, accessibile esclusivamente agli utenti con ruolo **Doctor**. Il flusso di esecuzione prevede i seguenti passaggi:

1. Il client effettua una richiesta HTTP includendo, se presenti, i parametri di query start ed end, che delimitano l'intervallo temporale di interesse.
2. Il **controller** riceve la richiesta e delega l'elaborazione al **service**.
3. Il **service** costruisce una query dinamica tramite l'`AvailabilityRepository`, ricercando tutte le disponibilità del medico autenticato. Se vengono forniti i parametri start ed end, la query viene arricchita filtrando soltanto gli slot appartenenti a quell'intervallo, ottimizzando così la scalabilità del sistema.
4. L'output della query è strutturato come una mappa (`Record<date, AvailabilitySlot []>`), in cui ad ogni giorno è associata la lista degli slot disponibili. Tale struttura viene quindi restituita come risposta al frontend.

Visualizzazione disponibilità tramite frontend

Lato frontend, la gestione delle disponibilità è stata implementata tramite un componente *Calendar*, basato sulla libreria open-source `react-big-calendar`[38].

Questa soluzione ha consentito di ridurre i tempi di sviluppo, offrendo un calendario interattivo con viste multiple (giorno, settimana, mese) e funzionalità di selezione diretta.

La chiamata al backend viene gestita tramite un custom hook `useAvailability`, che riceve in input un oggetto `range` contenente la data di riferimento e la vista attuale del calendario. In questo modo si implementa una logica di **lazy loading**: vengono richiesti al server soltanto gli slot pertinenti alla porzione di calendario visualizzata, riducendo il carico e sfruttando la cache di React Query per evitare richieste ridondanti.

A partire dalla data e dalla vista selezionata (*day*, *week* o *month*), vengono calcolati dinamicamente i due parametri `startDate` ed `endDate`, che vengono poi passati come query parameter al backend. Ad esempio:

- nella vista giornaliera, i parametri corrispondono all'inizio e alla fine del giorno selezionato;
- nella vista settimanale, l'intervallo è compreso tra il primo e l'ultimo giorno della settimana di riferimento;
- nella vista mensile, l'intervallo coincide con l'intero mese visualizzato.

In questo modo il backend riceve sempre un intervallo temporale coerente con la vista del calendario, restituendo soltanto le disponibilità rilevanti per quella specifica porzione di agenda.

Gli slot restituiti dal backend vengono poi trasformati in eventi e passati come prop al componente del calendario, così da poterli visualizzare direttamente nell'interfaccia.

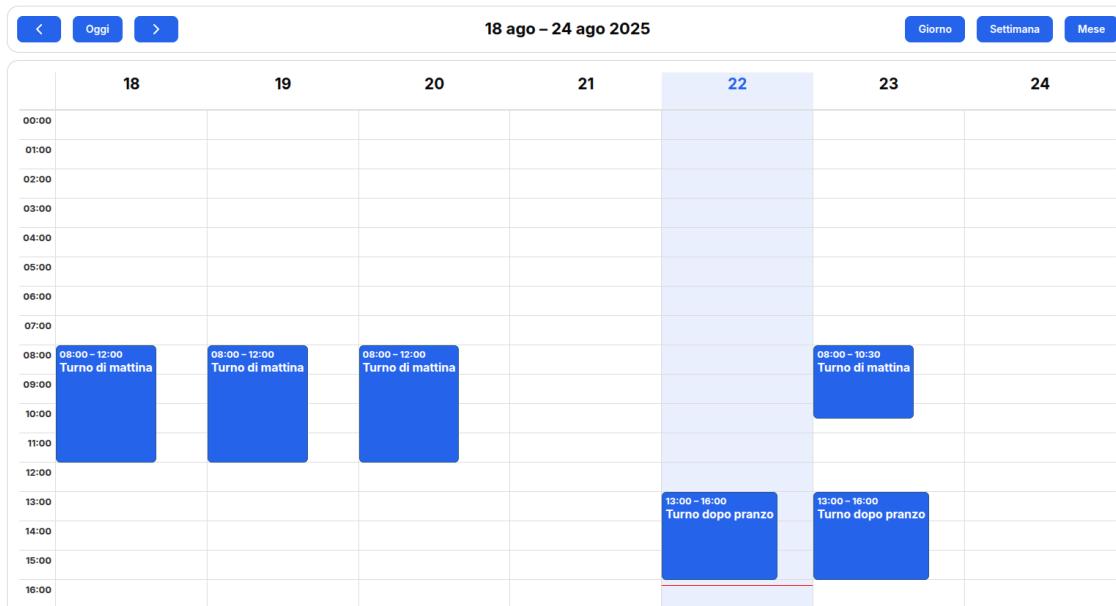


Figura 5.9: Calendario interattivo con le disponibilità del medico.

Creazione disponibilità lato backend

L'endpoint dedicato alla creazione è `POST /doctor/availability`, accessibile esclusivamente agli utenti con ruolo **Doctor**. Il flusso di esecuzione prevede i seguenti passaggi:

- Il client invia una richiesta HTTP con i dati della nuova disponibilità (`title`, `startTime`, `endTime`). Il payload è validato tramite il `CreateAvailabilityDto`, che applica controlli standard (`class-validator`) e custom (ad esempio `@IsSameDay` e `@HasMinimumDuration(30)`), così da assicurare che lo slot sia contenuto in una sola giornata e abbia durata minima di 30 minuti.
- Il **controller** riceve la richiesta validata e la inoltra al **service** dedicato.
- Il **service** esegue una serie di operazioni:
 - verifica l'esistenza del medico tramite il `DoctorRepository`;

- (b) converte gli orari in oggetti Date, effettuando controlli aggiuntivi (ad esempio, che l'orario di inizio preceda quello di fine);
 - (c) controlla l'assenza di sovrapposizioni con altre disponibilità già registrate, sollevando una ConflictException in caso contrario;
 - (d) in assenza di conflitti, crea una nuova entità di disponibilità nel database.
4. Il sistema restituisce infine al client la disponibilità creata, fornendo conferma del corretto inserimento.

Creazione disponibilità lato frontend

Per aggiungere una disponibilità, il medico seleziona con il cursore un intervallo di tempo (inizio–fine) sul calendario. Viene quindi mostrata una finestra modale dove è possibile inserire un titolo descrittivo. Alla conferma, i dati sono inviati al backend tramite una chiamata API.

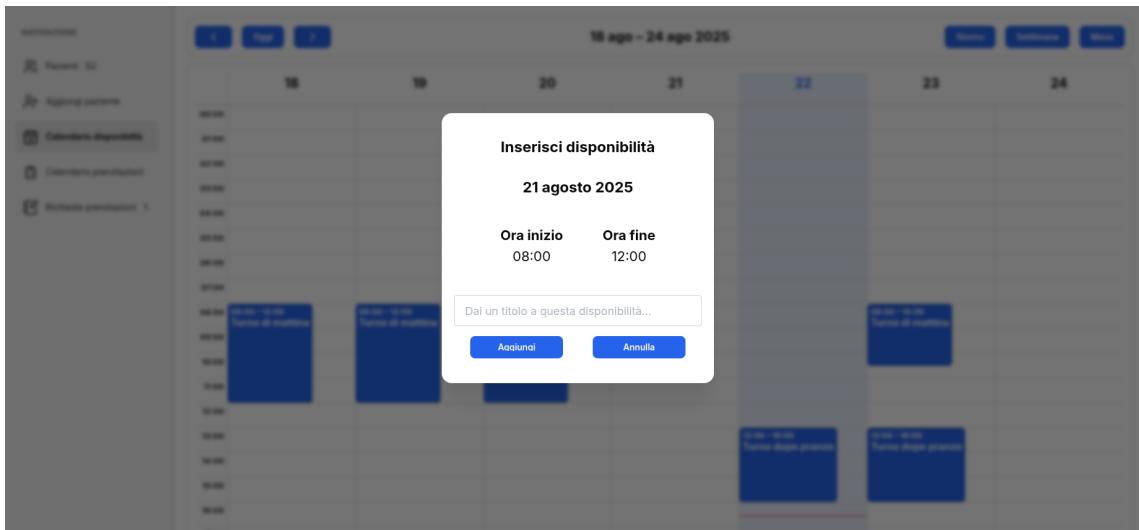


Figura 5.10: Modal per l'inserimento di una nuova disponibilità.

Eliminazione disponibilità lato backend

La rimozione delle disponibilità avviene tramite l'endpoint

`DELETE /doctor/availability/:availabilityId`, accessibile esclusivamente agli utenti con ruolo **Doctor**. Il flusso è il seguente:

1. Il client invia una richiesta HTTP indicando l'ID della disponibilità da eliminare.
2. Il **controller** delega la richiesta al **service**.
3. Il **service** verifica che la disponibilità esista e che appartenga effettivamente al medico autenticato:
 - se non esiste, viene sollevata una `BadRequestException`;
 - se trovata, viene rimossa dal database.

Eliminazione disponibilità lato frontend

Anche la cancellazione può essere effettuata direttamente dal calendario. Cliccando su un evento, compare un menù contestuale che permette di rimuovere lo slot selezionato.

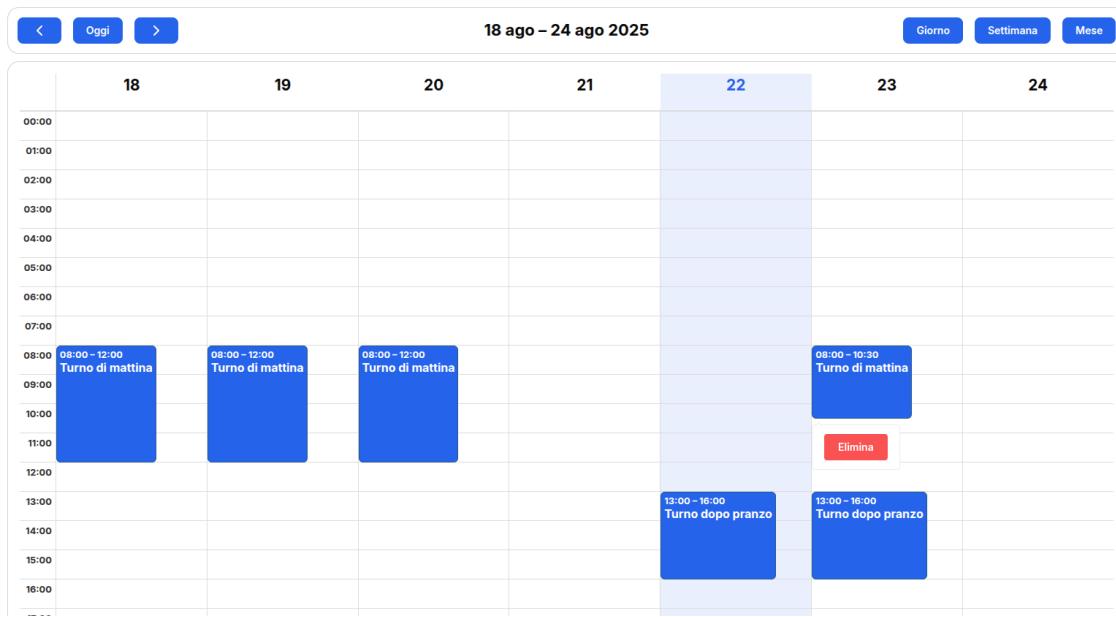


Figura 5.11: Eliminazione di una disponibilità tramite menù contestuale.

Questa funzionalità è stata realizzata tramite un *custom event wrapper* del calendario, che mostra un popover con l’opzione di eliminazione. L’azione scatena una chiamata API di tipo DELETE, gestita da un custom hook `useDeleteAvailabilityMutation`. Grazie all’integrazione con React Query, ogni operazione di inserimento o eliminazione invalida automaticamente la cache, mantenendo i dati del calendario costantemente sincronizzati con il backend.

In questo modo il medico dispone di uno strumento semplice, intuitivo e interattivo per organizzare e modificare la propria agenda direttamente all’interno dell’applicazione.

5.2.6 Visualizzazione richieste di appuntamenti e appuntamenti confermati

Questa funzionalità consente al medico di gestire in modo centralizzato le prenotazioni dei pazienti, distinguendo tra semplici richieste (pendenti) e appun-

tamenti effettivamente confermati. Per ciascuna prenotazione è possibile consultare i dettagli principali: orario di inizio e fine, tipologia di visita, data di creazione della richiesta e informazioni di base sul paziente.

Lato backend

L'endpoint dedicato è GET /reservations, accessibile esclusivamente agli utenti con ruolo **Doctor**. Il flusso di esecuzione prevede i seguenti passaggi:

1. Il client invia una richiesta HTTP includendo, se presenti, i parametri di query:
 - start ed end, che delimitano l'intervallo temporale di interesse;
 - status, che specifica lo stato delle prenotazioni da visualizzare (CONFIRMED, PENDING, DECLINED, ALL).
2. Il **controller** riceve la richiesta e delega l'elaborazione al service
3. Il **service** costruisce una query dinamica tramite il `ReservationRepository`, ricercando tutte le prenotazioni relative al medico autenticato. In base ai parametri di query, la ricerca viene arricchita:
 - filtrando per intervallo temporale, se start ed end sono presenti;
 - filtrando per stato, se status è valorizzato. Nel caso sia ALL, vengono incluse tutte le prenotazioni (eccetto quelle rifiutate).
4. I risultati vengono raggruppati per giorno e restituiti al frontend come una struttura del tipo `Record<date, ReservationsDTO[]>`.

Lato frontend

Dal lato frontend, la visualizzazione è stata implementata con lo stesso componente *Calendar* di `react-big-calendar`[38], ma in un'istanza separata rispetto a quella delle disponibilità, così da mantenere la UI più chiara e ridurre la complessità percepita dal medico.

La logica di chiamata API, lazy loading e caching è analoga a quella delle disponibilità: viene utilizzato un custom hook (`useReservations`) che calcola dinamicamente i parametri `startDate` ed `endDate` in base alla vista corrente del calendario (*day*, *week*, *month*) e li passa come query parameter al backend.

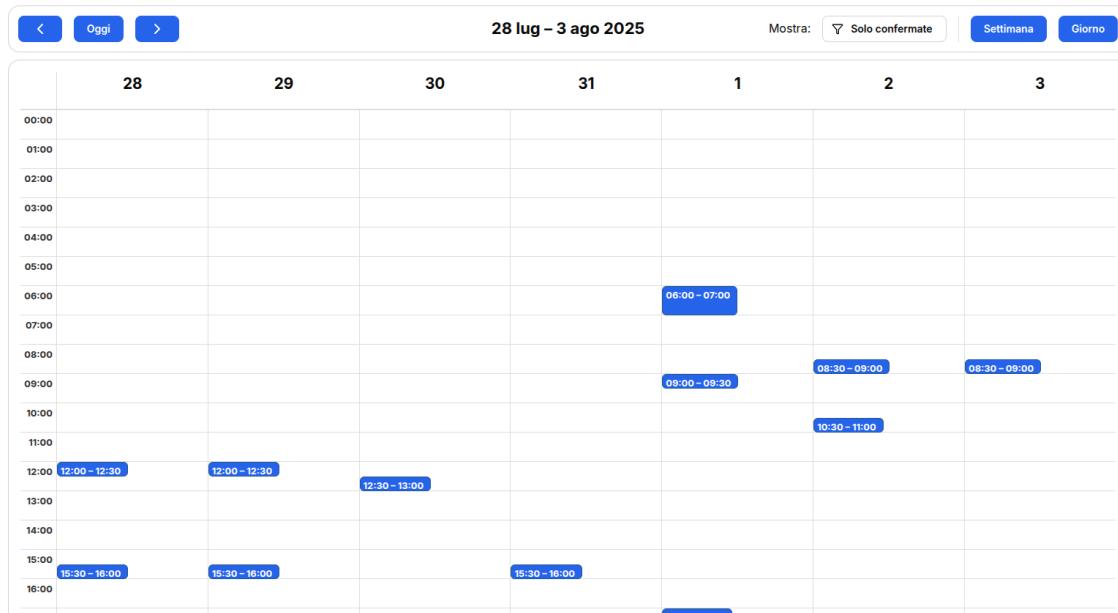


Figura 5.12: Calendario con richieste e prenotazioni del medico.

Gli eventi ricevuti vengono trasformati e visualizzati all'interno del calendario, con una differenziazione grafica in base allo stato: **blu**: prenotazioni confermate, **bianco**: richieste pendenti.

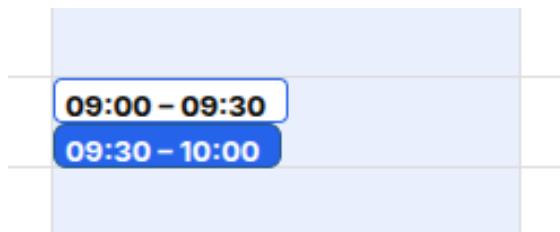


Figura 5.13: Visualizzazione prenotazioni.

Un menù a tendina consente inoltre di filtrare le prenotazioni da visualizzare (solo confermate oppure tutte).

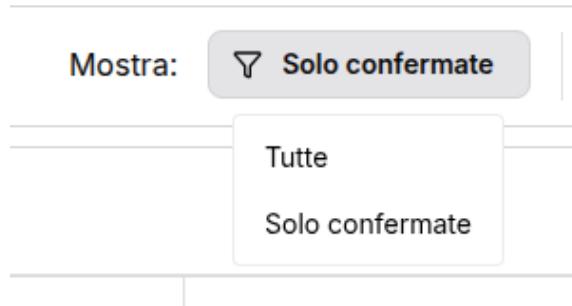


Figura 5.14: Filtro per lo stato delle prenotazioni.

Cliccando su una prenotazione viene aperto un popover con i dettagli. Nel caso di una richiesta pendente, oltre ai dati informativi, sono presenti i pulsanti per **accettare** o **rifiutare** la prenotazione, così da completare direttamente dal calendario l'intero flusso decisionale.



Figura 5.15: Dettagli di una prenotazione con azione di conferma/rifiuto.

Grazie a questa soluzione, il medico dispone di uno strumento unico per monitorare e gestire sia le richieste in arrivo sia gli appuntamenti già con-

fermati, con la stessa semplicità e immediatezza garantita dalla gestione delle disponibilità.

5.2.7 Inserimento di un report di visita

Questa funzionalità consente al medico di registrare un report relativo ad una specifica visita medica, documentando la motivazione della visita, le attività svolte e note utili per eventuali appuntamenti futuri. L'obiettivo è fornire uno strumento semplice ma strutturato per arricchire la cartella clinica del paziente con informazioni qualitative.

Lato backend

L'endpoint dedicato è POST `medical-examination/:reservationId`, accessibile esclusivamente agli utenti con ruolo **Doctor**. Il flusso di esecuzione prevede i seguenti passaggi:

1. Il client invia una richiesta HTTP includendo nel body la motivazione della visita medica e delle note
2. Il **controller** valida i dati ricevuti tramite il DTO `MedicalExaminationDTO`, sfruttando i decorator di `class-validator`. Successivamente, invoca il metodo del service passando il body e l'identificativo della prenotazione.
3. Il **service** esegue due operazioni principali:
 - (a) ricerca la prenotazione tramite l'id ricevuto, se non trovata solleva una `NotFoundException`
 - (b) crea un nuovo record di visita medica, collegandolo sia al paziente che alla prenotazione.
4. La visita appena salvata viene restituita come risposta al client, confermando l'avvenuta registrazione.

Lato frontend

Dal lato frontend, la chiamata al backend è gestita da un custom hook (`useAddMedicalExamination`) basato sulle *mutation* di React Query[34]. Sono stati implementati due flussi distinti per facilitare l'esperienza utente, entrambi convergenti nello stesso modal di inserimento.

Inserimento dal calendario delle prenotazioni

All'interno della schermata calendario, cliccando su una prenotazione è possibile visualizzarne i dettagli. Da qui il medico può aprire un modal dedicato per inserire la motivazione e le note della visita. In questo caso la prenotazione è già nota, in quanto il flusso parte direttamente dalla selezione dell'evento nel calendario, rendendo l'operazione immediata e veloce.

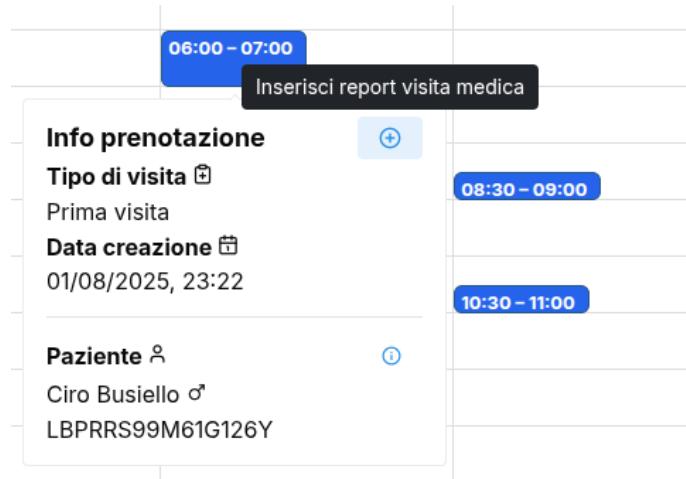


Figura 5.16: Inserimento di un report visita direttamente dal calendario.

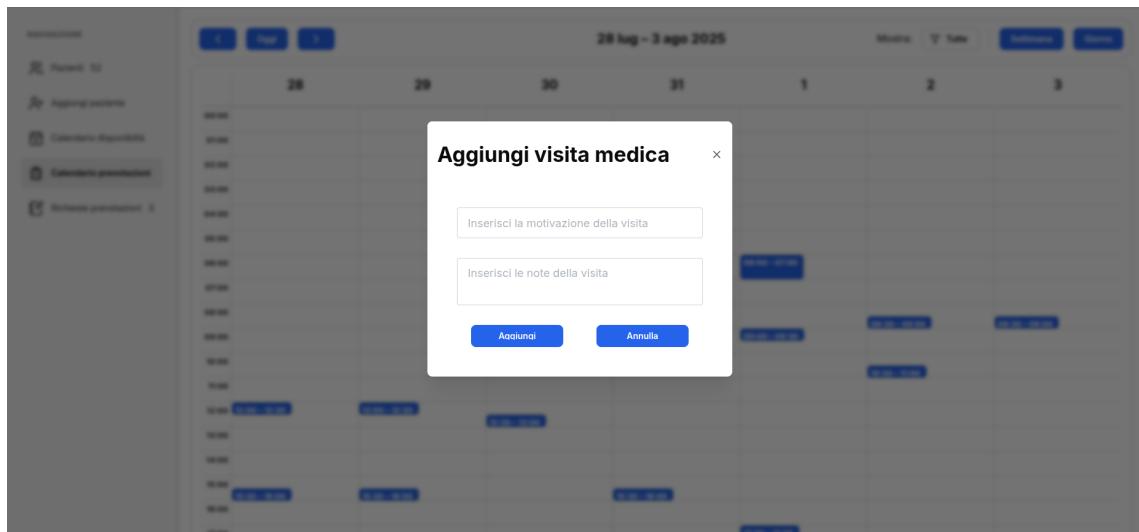


Figura 5.17: Inserimento di un report visita tramite modal.

Inserimento dalla schermata del paziente

Quando il medico si trova nella scheda di un paziente, è disponibile un pulsante "Aggiungi visita medica". In questo scenario, essendo noto il paziente ma non la prenotazione di riferimento, viene mostrata una tabella paginata con l'elenco delle prenotazioni confermate per quel paziente (GET /reservations/:patientId). Attraverso una row action, il medico seleziona la prenotazione desiderata e accede allo stesso modal per inserire motivazione e note della visita.

Selezione la prenotazione				
Actions	Giorno	Inizio	Fine	Tipo di visita
+	23/08/2025	09:30	10:00	Controllo
+	05/08/2025	07:30	08:00	Controllo
+	02/08/2025	10:30	11:00	Controllo
+	01/08/2025	17:00	17:30	Controllo
+	01/08/2025	09:00	09:30	Controllo
+	31/07/2025	15:30	16:00	Controllo
+	30/07/2025	12:30	13:00	Controllo
+	29/07/2025	15:30	16:00	Controllo
+	28/07/2025	15:30	16:00	Controllo
+	26/07/2025	13:30	14:00	Controllo
+	25/07/2025	09:30	10:00	Controllo

Figura 5.18: Selezione della prenotazione da cui creare un report di visita.

Considerazioni di design

La scelta di offrire due flussi paralleli risponde ad un'esigenza di usabilità:

- dal calendario, il medico può documentare rapidamente una visita appena conclusa, senza passaggi intermedi;
- dalla scheda paziente, invece, ha una visione aggregata delle prenotazioni e può decidere per quale di esse registrare un report.

Questa soluzione evita di vincolare l'utente ad un unico percorso, riducendo i click necessari e rendendo più fluida l'esperienza d'uso. In entrambi i casi, l'inserimento dei dati avviene attraverso lo stesso modal, garantendo coerenza visiva e uniformità di interazione.

5.2.8 Inserimento di un nuovo paziente

Questa funzionalità consente al medico di aggiungere un nuovo paziente all'interno della piattaforma. Dal punto di vista concettuale, l'operazione non è stata modellata come una creazione diretta di un utente, ma piuttosto come la generazione di un **invito di registrazione**. L'invito rappresenta un meccanismo di onboarding che permette al paziente di completare in autonomia la propria registrazione all'applicazione mobile, partendo dalle informazioni preliminari inserite dal medico.

Non si tratta quindi di un invito con scadenza temporale o vincoli particolari: il sistema ammette al massimo un invito attivo per ciascun paziente, identificato da parametri univoci come codice fiscale, numero di telefono o indirizzo email. Questo approccio evita duplicati e mantiene la coerenza dei dati, garantendo che ciascun paziente sia associato ad un solo medico tramite un unico invito attivo.

Lato backend

L'endpoint dedicato è `POST /invite`, accessibile esclusivamente agli utenti con ruolo **Doctor**. Il flusso di esecuzione è articolato nei seguenti passaggi:

1. Il client invia una richiesta HTTP contenente, nel body, i dati necessari alla creazione dell'invito.
2. Il **controller** valida i dati ricevuti tramite il DTO `CreateInviteDTO`, sfruttando i decoratori di `class-validator`. In seguito, inoltra la richiesta al service passando sia i dati dell'invito che l'identificativo del medico autenticato.
3. Il **service** esegue una serie di controlli e operazioni:
 - verifica l'esistenza del medico nel database, sollevando una `NotFoundException` se non trovato;

-
- controlla che non esista già un invito associato a codice fiscale, email o numero di telefono forniti, evitando duplicazioni e sollevando una `BadRequestException` in caso contrario;
 - crea un’istanza di paziente nella relativa tabella, popolando i campi clinici di base (peso, altezza, gruppo sanguigno, eventuali patologie, farmaci assunti, ecc.);
 - genera e salva l’invito nella tabella dedicata, collegandolo sia al paziente che al medico.
4. Come risposta viene restituito l’identificativo del paziente appena creato, confermando l’avvenuta registrazione dell’invito.

Lato frontend

Dal lato frontend, l’inserimento di un nuovo paziente è stato realizzato tramite un flusso guidato a **4 step**, già descritto in fase di analisi dei requisiti (Sezione (Sezione 2.5.1)), supportato da mockup realizzati in Figma. La chiamata al backend è gestita da un custom hook `useAddPatient`, basato sulle *mutation* di React Query [34].

La compilazione del form avviene tramite il custom hook `useForm`[39] di MantineUI[12], che semplifica la gestione dello stato e la validazione dei campi. È stato scelto di mantenere un unico form logico, suddividendolo in più step per migliorare l’esperienza utente e guidare il medico nell’inserimento progressivo delle informazioni.

Ogni step del form è validato secondo due modalità complementari:

- **Validazione sincrona** mediante schemi definiti con la libreria `Zod`[40], che garantiscono il rispetto dei vincoli strutturali e semantici (es. formato email, lunghezza minima dei campi, enumerazioni).

- **Validazione asincrona** tramite chiamate API, utilizzata per verificare in tempo reale l'assenza di duplicati su campi critici come codice fiscale, email o numero di telefono.

La logica degli step prevede una **navigazione sequenziale vincolata**: l'utente può procedere allo step successivo solo dopo aver completato e validato correttamente quello corrente, mentre la navigazione all'indietro è sempre consentita. In questo modo si riducono al minimo gli errori di compilazione e si mantiene un flusso coerente.

Una volta completati tutti gli step e confermata la compilazione, il sistema invia al backend la richiesta POST `/invite`, avviando così la creazione dell'invito e l'associazione del nuovo paziente alla piattaforma.

5.2.9 Registrazione paziente tramite app mobile

Questa funzionalità consente al paziente di registrarsi alla piattaforma in maniera guidata e sicura, partendo da un invito generato dal medico. Il flusso si basa sul concetto di **invito**: quando il dottore inserisce un nuovo paziente, il sistema crea una coppia di record costituita da:

- una tupla nella tabella Patient, contenente le informazioni cliniche e anagrafiche di base del paziente;
- una tupla corrispondente nella tabella Invite, che funge da “ponte” per la futura registrazione.

Il paziente riceve quindi un codice QR che rappresenta il riferimento univoco a quell’invito. La registrazione vera e propria avviene solo nel momento in cui il paziente scansiona il QR tramite l’app mobile ed **accetta l’invito**, portando così alla creazione del proprio account utente. A valle di questa operazione:

- viene creata una nuova tupla nella tabella User, contenente le credenziali e i dati identificativi;
- la tupla Patient generata in precedenza viene collegata all’utente appena creato;
- l’invito viene marcato come *utilizzato*, impedendone un uso successivo.

Lato backend

La registrazione del paziente si articola in due step principali, gestiti da altrettanti endpoint API: **ottenimento dell’invito** e **accettazione dell’invito**.

Ottenimento dell’invito

L’endpoint dedicato è GET /invite/:id. Il flusso di esecuzione è il seguente:

1. Il client invia una richiesta HTTP indicando come parametro l'identificativo dell'invito.
2. Il **controller** inoltra la richiesta al service
3. Il **service** verifica che l'invito esista e che non sia già stato utilizzato:
 - se non trovato, viene sollevata una `NotFoundException`;
 - se già usato, viene sollevata una `BadRequestException`.
4. In caso positivo, l'invito viene restituito come risposta al frontend.

Accettazione dell'invito

L'endpoint dedicato è `POST /invite/:id/accept`, accessibile esclusivamente agli utenti di ruolo **Patient**. Il flusso prevede i seguenti passaggi:

1. Il client invia una richiesta HTTP includendo l'identificativo dell'invito e, nel body, i dati necessari alla creazione dell'utente.
2. Il **controller** valida i dati ricevuti tramite l'`AcceptInviteDTO`, sfruttando i decoratori di `class-validator`, e delega la logica al service.
3. Il **service** esegue le seguenti operazioni:
 - (a) verifica l'esistenza e la validità dell'invito (non usato);
 - (b) cifra la password fornita e controlla che non esista già un utente con la stessa email, codice fiscale o telefono; in caso contrario viene sollevata una `ConflictException`;
 - (c) crea una nuova tupla nella tabella `User` con i dati forniti;
 - (d) associa la tupla `Patient` pre-esistente al nuovo utente, stabilendo così il legame definitivo tra paziente e user;
 - (e) marca l'invito come utilizzato.
4. Il sistema restituisce al client un messaggio di conferma, indicando l'avvenuta registrazione.

Lato frontend

Dal punto di vista frontend, la registrazione del paziente tramite app mobile si articola in due fasi principali: l'ottenimento dell'invito e la sua accettazione.

Ottenimento dell'invito

Il processo inizia dalla schermata di login dell'applicazione, dove è presente un pulsante dedicato alla registrazione mediante codice di invito.

Alla pressione del pulsante viene mostrato un *bottom sheet modal* che, previa autorizzazione all'uso della fotocamera, attiva il lettore QR code.

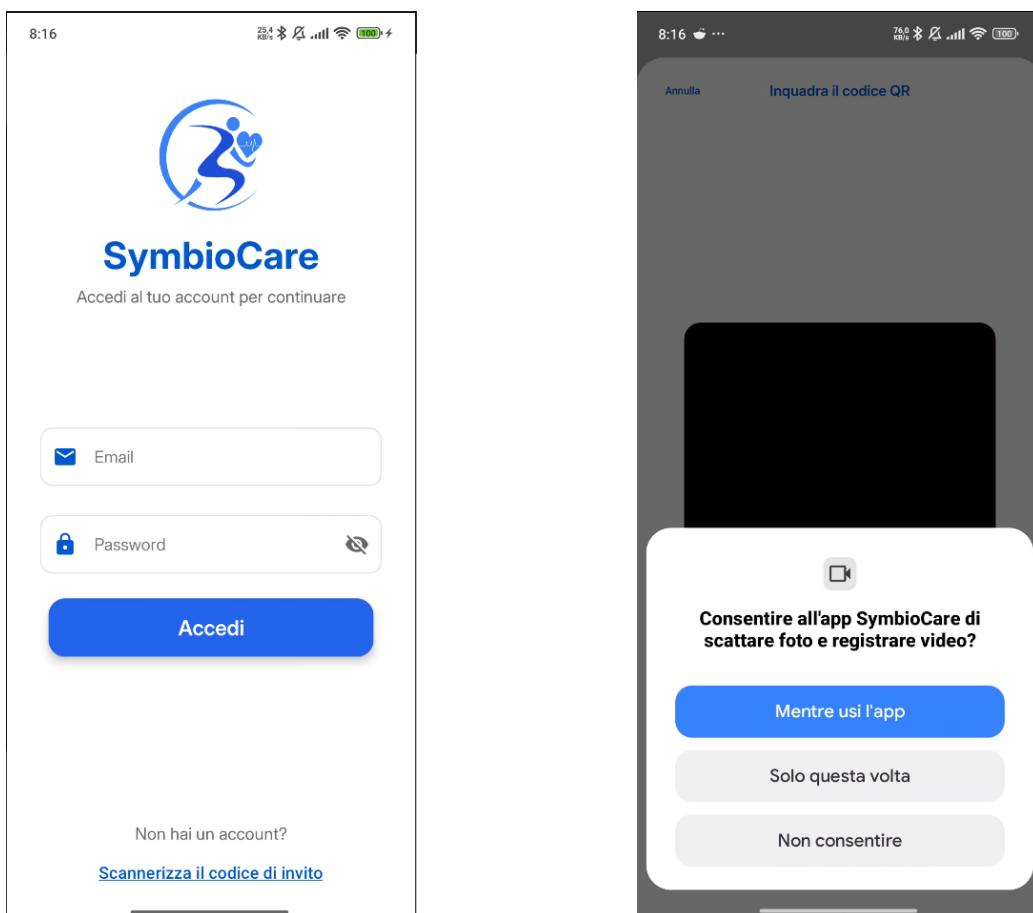
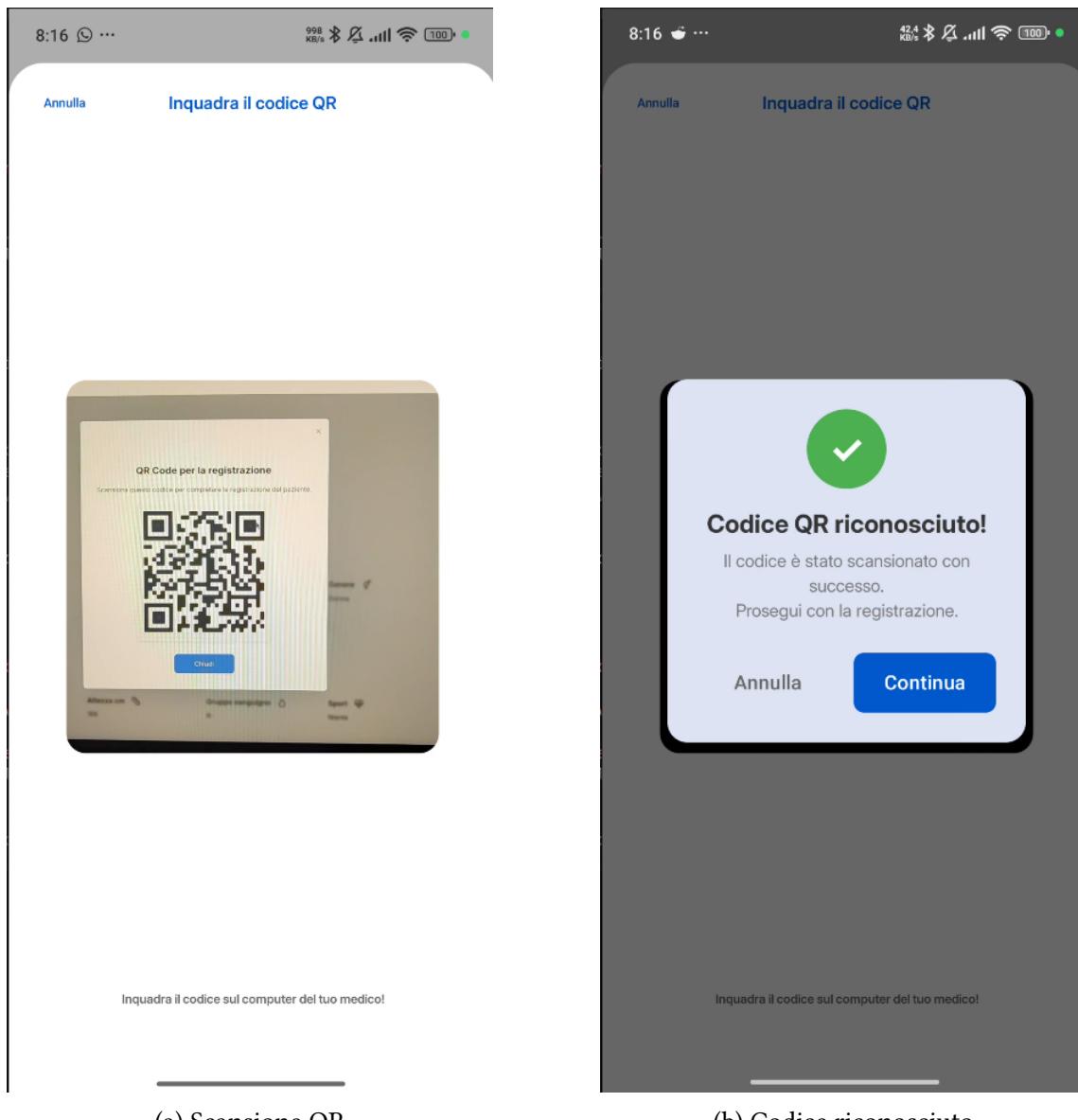


Figura 5.19: Pagina di login app mobile

Figura 5.20: BottomModalSheet scansione QR

Una volta inquadrato correttamente il codice QR fornito dal medico, l'app estrae l'identificativo dell'invito e invia una chiamata API all'endpoint GET `/invite/:id`.



(a) Scansione QR

(b) Codice riconosciuto

Figura 5.21: Fasi di scansione del codice QR

La risposta del backend contiene tutte le informazioni preliminari relative al paziente (dati anagrafici, medici e di contatto), che vengono utilizzate per precompilare automaticamente il form di registrazione.

Accettazione dell'invito

Nella schermata di registrazione, la maggior parte dei campi è presentata in sola lettura, così da offrire al paziente una conferma visiva dei dati precedentemente inseriti dal medico. Sono invece attivi:

- i campi relativi all'indirizzo di residenza, lasciati modificabili per consentire eventuali aggiornamenti;
- il campo per l'inserimento della password, necessaria per completare la creazione dell'account utente.

8:16 23,3 KB/s 100%
Completa la registrazione
Verifica i tuoi dati e imposta una password

Dati personali

Nome Giacinta Cognome Carraro
Codice Fiscale ZGGKMB78D18C Data di nascita 1985-02-16
Email giacinta.carraro@example.com

Inserisci una password

Password Conferma la password

Dati medici

Peso 52.4 Altezza 165.0
Gruppo sanguigno A+
Sport Corsa

8:16 180 KB/s 100%
Completa i tuoi dati

Indirizzo Piazza Giulietta 53
Città Sesto Marino CAP 61478
Provincia CR

Completa registrazione

Figura 5.22: Form di registrazione

Figura 5.23: Form di registrazione

Dopo aver scelto una password conforme ai requisiti di sicurezza e, se necessario, aggiornato i dati di residenza, il paziente conferma la registrazione.

Viene inviata una richiesta all'endpoint POST /invite/:id/accept, contenente sia le credenziali scelte che i dati aggiornati. Se l'operazione va a buon fine, l'invito viene marcato come utilizzato e il paziente viene reindirizzato alla schermata di login, potendo così accedere con le nuove credenziali appena create.

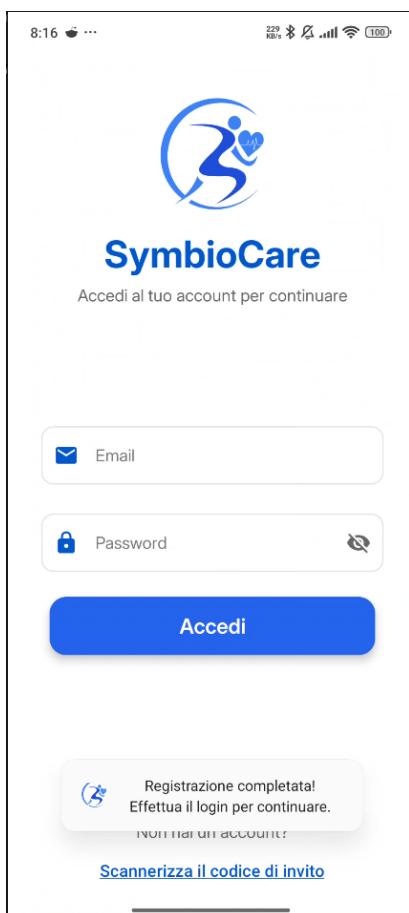


Figura 5.24: Registrazione completata

5.3 Visualizzare richieste di prenotazioni e appuntamenti confermati

Questa feature consente al paziente di avere una panoramica completa sia delle richieste di prenotazione effettuate, sia degli appuntamenti effettivamente confermati dal medico. La scelta progettuale è stata quella di distinguere due endpoint separati: uno dedicato alla visualizzazione dei soli appuntamenti futuri confermati, e uno volto a recuperare l'elenco generale delle prenotazioni (incluse quelle ancora in attesa o rifiutate). In questo modo si ottiene una migliore organizzazione dei dati e un recupero più efficiente a seconda del caso d'uso.

Lato backend

Prossimi appuntamenti confermati L'endpoint dedicato è GET /reservations/next-reservations, accessibile esclusivamente ad utenti con ruolo **Patient**.

Il flusso è il seguente:

1. Il client invia una richiesta HTTP senza parametri aggiuntivi.
2. Il **controller** intercetta la richiesta e la inoltra al service.
3. Il **service** esegue una query per ottenere tutte le prenotazioni associate al paziente autenticato che:
 - hanno stato CONFIRMED,
 - presentano una data di inizio successiva al momento corrente.

Le prenotazioni vengono poi ordinate in ordine cronologico crescente.

4. La lista degli appuntamenti futuri viene restituita come risposta al frontend.

Questo endpoint risponde a un'esigenza pratica: permettere al paziente di visualizzare rapidamente i prossimi appuntamenti a calendario, senza dover filtrare manualmente tutte le prenotazioni effettuate.

Prenotazioni e richieste di prenotazioni

L'endpoint dedicato è GET `/reservations/patient`, anch'esso accessibile solo ad utenti con ruolo **Patient**.

Il flusso è il seguente:

1. Il client invia una richiesta HTTP indicando, tramite query parameter, lo stato delle prenotazioni da recuperare. I valori ammessi sono:
 - ALL – tutte le richieste associate al paziente,
 - CONFIRMED – solo le prenotazioni confermate,
 - PENDING – richieste in attesa di conferma,
 - DECLINED – richieste rifiutate.
2. Il **controller** riceve la richiesta e, dopo aver validato i parametri, la inoltra al service insieme al riferimento del paziente autenticato e al medico associato.
3. Il **service** costruisce dinamicamente una query applicando i filtri sullo stato specificato e ordinando le prenotazioni in ordine decrescente rispetto alla data di creazione.
4. L'elenco delle prenotazioni filtrate viene restituito al frontend in forma di DTO.

Lato frontend

La visualizzazione delle prenotazioni è gestita in un unico composable `ReservationScreen`, che al caricamento iniziale invoca tramite **Retrofit**[32] i

due endpoint descritti lato backend: GET /reservations/next-reservation e GET /reservations/patient.

Prossimi appuntamenti

Se la chiamata API relativa ai prossimi appuntamenti confermati restituisce un risultato non vuoto, viene mostrata un elenco orizzonatale scrollabile. Ogni card contiene le informazioni essenziali dell'appuntamento:

- data e orario di inizio/fine,
- tipologia di visita,
- medico associato,
- studio presso cui si terrà la visita.

Questa sezione fornisce al paziente una panoramica rapida e compatta dei prossimi impegni confermati.



Figura 5.25: Prossimi appuntamenti

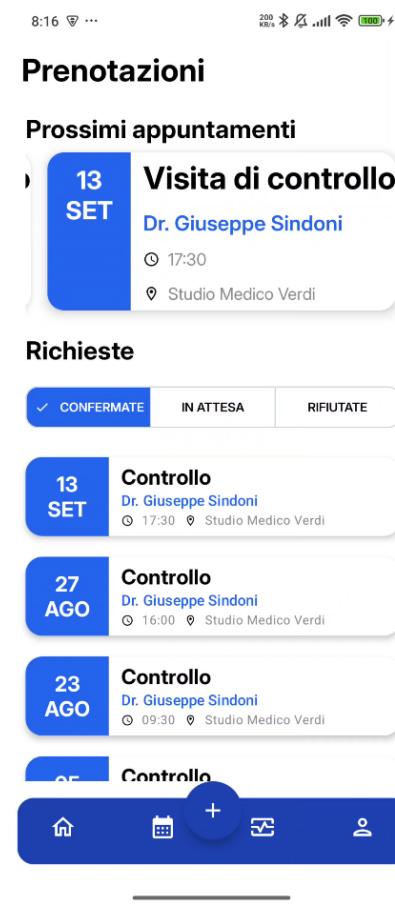


Figura 5.26: Prossimi appuntamenti

Storico prenotazioni

Al di sotto della sezione dei prossimi appuntamenti è presente la lista delle richieste di prenotazione. L'elenco è reso tramite una elenco verticale di card, e il filtro sullo stato delle prenotazioni è gestito tramite il componente `SegmentedButton` di Material Design, che consente di alternare facilmente tra:

- CONFIRMED – prenotazioni confermate,
- PENDING – prenotazioni ancora in attesa di risposta,
- DECLINED – richieste rifiutate.

Al cambio di stato, il ViewModel invoca la relativa API con il query parameter corretto e aggiorna lo StateFlow, garantendo un'esperienza fluida e reattiva.

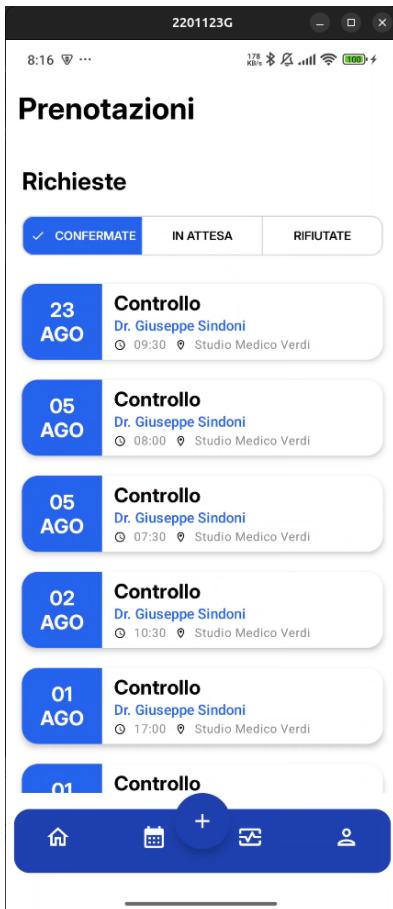


Figura 5.27: Storico richieste



Figura 5.28: Storico richieste

5.3.1 Richiesta di prenotazione tramite app mobile

Questa feature consente al paziente di inoltrare una richiesta di prenotazione direttamente dall'app mobile, selezionando uno slot orario tra quelli resi disponibili dal proprio medico. L'intera logica è stata modellata per rispettare le disponibilità definite dal dottore, evitando conflitti e garantendo la corretta tipologia e durata delle visite.

Lato backend

Per supportare questa feature sono stati previsti due endpoint distinti:

- GET `/reservations/slots` per ottenere gli slot disponibili in un determinato giorno,
- POST `/reservations` per inoltrare la richiesta di prenotazione.

Ottenimento degli slot disponibili

L'endpoint dedicato è GET `/reservations/slots`, accessibile solo agli utenti con ruolo **Patient**. Il flusso segue i seguenti passaggi:

1. Il client invia una richiesta HTTP indicando come query parameter la data desiderata e la tipologia di visita.
2. Il **controller** riceve la richiesta e la inoltra al service, passando l'utente autenticato, la data e la tipologia di visita.
3. Il **service** esegue le seguenti operazioni:
 - valida la tipologia di visita richiesta, sollevando una `BadRequestException` in caso di valori non ammessi,
 - recupera tutte le disponibilità del medico per il giorno selezionato, ordinandole cronologicamente,
 - recupera tutte le prenotazioni già confermate in quella data,

-
- per ciascuna disponibilità, genera iterativamente una lista di slot, frazionando l'intervallo in sotto-slot della durata prevista dal tipo di visita (30 o 60 minuti),
 - verifica per ogni slot generato se questo si sovrappone con prenotazioni già confermate, escludendolo in tal caso,
 - costruisce infine la lista degli slot liberi effettivamente prenotabili, che viene restituita al client.

Questa logica assicura che al paziente vengano mostrati unicamente gli slot ancora disponibili, coerenti con le disponibilità del medico e compatibili con la durata della visita richiesta. In questo modo si evitano conflitti e si garantisce un flusso di prenotazione sicuro e consistente.

Invio richiesta di prenotazione

L'endpoint dedicato è POST /reservations, accessibile esclusivamente ad utenti con ruolo **Patient**. Il flusso di esecuzione prevede i seguenti passaggi:

1. Il client invia una richiesta HTTP contenente nel body le informazioni necessarie: data e ora di inizio, data e ora di fine e tipologia di visita.
2. Il **controller** intercetta la richiesta, valida i dati ricevuti tramite il DTO CreateReservationDTO (sfruttando i decoratori di class-validator) e la inoltra al service applicativo.
3. Il **service** esegue una serie di controlli e validazioni:
 - verifica che la tipologia di visita esista e sia valida; in caso contrario solleva una BadRequestException;
 - nel caso la tipologia sia Prima visita, controlla che il paziente non abbia già effettuato altre visite; in caso contrario viene sollevata una BadRequestException;

- controlla che la durata dello slot richiesto sia coerente con quella prevista per la tipologia selezionata; in caso contrario viene restituita una `BadRequestException`;
 - verifica che non esistano altre prenotazioni confermate con lo stesso orario di inizio per quel medico; in caso contrario viene sollevata una `ConflictException`;
 - controlla che lo slot rientri effettivamente all'interno di una disponibilità registrata dal medico; se non viene trovata alcuna corrispondenza, viene sollevata una `BadRequestException`.
4. Superati tutti i controlli, viene creata una nuova prenotazione con stato iniziale `PENDING`, salvata a database e restituita al client sotto forma di DTO.

Lato frontend

La richiesta di prenotazione è implementata in un'unica schermata, `AddReservationScreen`, composta da:

- un *segmented button* **non interattivo**, che visualizza la tipologia di visita determinata automaticamente dal sistema;
- un elenco selezionabile dei **prossimi 21 giorni** (con giorno, mese e giorno della settimana), dal quale l'utente sceglie la data desiderata;
- una griglia di card che rappresentano gli **slot disponibili** per la data selezionata.

Dal punto di vista funzionale, il flusso utente prevede che il paziente:

- selezioni un giorno dal calendario,
- ottenga automaticamente la tipologia di visita,

-
- visualizzi la lista degli slot generati a partire dalle disponibilità del medico per quel giorno, suddivisi secondo la durata prevista per la tipologia selezionata,
 - scelga uno slot effettivamente libero (ovvero non occupato da altre prenotazioni confermate).

Selezione tipologia visita

La tipologia viene mostrata tramite un *segmented control* ma risulta **bloccata**: non è l'utente a selezionarla. Il valore viene calcolato all'avvio mediante una chiamata API a GET /reservations/isFirstVisit e rimane coerente con le regole di dominio (60 minuti per *Prima visita*, 30 minuti per *Visita di controllo*).

Selezione del giorno

La scelta della data avviene tramite un componente day-selector che mostra i prossimi 21 giorni; al primo render viene **preselezionata** la prima data utile (tipicamente il giorno corrente). Alla selezione o al cambio di data:

1. viene azzerata l'eventuale selezione di slot,
2. viene eseguita una chiamata API a GET /reservations/slots, passando la nuova data e la tipologia corrente.

Caricamento slot e rendering

Gli slot disponibili sono visualizzati in una griglia a tre colonne. La UI gestisce tre stati principali:

- **Loading**: visualizzazione di un indicatore durante il recupero dati,
- **Error**: messaggio contestuale in caso di problemi di rete o validazione,
- **Empty**: illustrazione e testo dedicati quando non sono presenti slot prenotabili per la combinazione giorno/tipologia.

Ogni card mostra l'orario di **inizio** già formattato nel fuso orario locale del dispositivo. Il tap su una card imposta lo **slot attivo**.

Conferma prenotazione

Il pulsante *Conferma*, collocato nella barra inferiore, è **abilitato** solo quando sono stati selezionati sia una data sia uno slot. Alla pressione, l'app effettua una chiamata API all'endpoint POST /reservations, passando l'orario di inizio e fine dello slot selezionato (convertiti in formato UTC, per garantire la coerenza temporale) e la tipologia della visita.

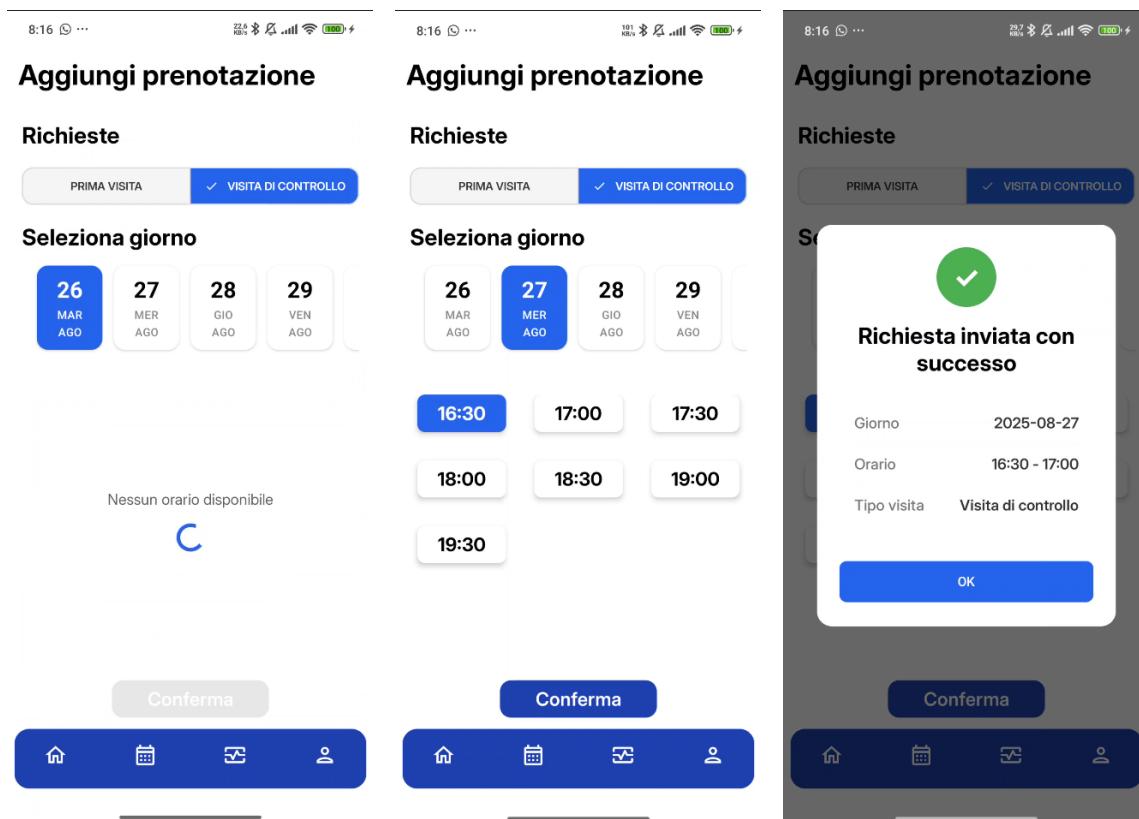


Figura 5.29: Schermata aggiungi prenotazione

Figura 5.30: Selezione giorno e slot

Figura 5.31: Conferma richiesta

—6—

Testing

Nel processo di sviluppo del sistema **SymbioCare**, una particolare attenzione è stata riservata alla fase di verifica e validazione, elementi fondamentali per assicurare l'affidabilità e la qualità del software. In questo capitolo vengono presentate le strategie adottate per definire, eseguire e documentare i test realizzati.

La definizione accurata dei casi di test, unita all'adozione di strumenti automatizzati per l'esecuzione di test unitari, ha permesso di individuare tempestivamente eventuali malfunzionamenti e incongruenze rispetto ai requisiti iniziali.

6.1 Obiettivi del testing

Lo scopo del testing è garantire che il sistema funzioni correttamente, sottoponendolo a scenari d'uso specifici, sia in condizioni nominali che anomale. In questo progetto l'attività di test si è concentrata principalmente sul **backend**, con i seguenti obiettivi concreti:

- verificare i **casi nominali** (happy path);
- coprire i **rami d'errore** (eccezioni, input incoerenti, condizioni di duplicato);

-
- mantenere i test **deterministici** tramite il mocking delle dipendenze;
 - fornire **documentazione eseguibile** del comportamento dei metodi principali.

6.2 Tecnologie adottate

6.2.1 Jest

Il framework di testing scelto è stato **Jest**[41], sviluppato originariamente da Facebook, che offre un ambiente moderno per la scrittura di test unitari in applicazioni JavaScript e TypeScript. Jest fornisce funzionalità complete come:

- mocking di funzioni, classi e moduli;
- supporto per test asincroni;
- generazione automatica di report di copertura del codice;
- snapshot testing per la validazione dell'output.

6.2.2 NestJS Testing Module

NestJS[20], il framework backend utilizzato per SymbioCare, integra nativamente un modulo dedicato al testing: `@nestjs/testing`. Questo strumento consente di costruire facilmente `TestingModule` isolati, nei quali mockare i provider e i repository tramite il costrutto `getRepositoryToken()`, evitando la necessità di un database reale durante i test. In combinazione con Jest, tale approccio permette di eseguire unit test rapidi, affidabili e indipendenti.

6.3 Struttura di un test con Jest

Un test in Jest segue tipicamente la sequenza:

1. **Setup** – viene creato l’ambiente di test, instanziando i moduli necessari e mockando le dipendenze.
2. **Act** – viene invocato il metodo da testare, con eventuali parametri di input.
3. **Assert** – si verificano i risultati attesi tramite le asserzioni (`expect(...)`).

6.4 Esempi di test implementati

6.4.1 Caso nominale

Il primo test sviluppato ha verificato la corretta creazione di un oggetto `Invite` a partire da un dottore e un paziente esistenti. In questo scenario, le dipendenze (repository) sono state mockate in modo da restituire entità simulate, evitando accessi al database reale. Il test ha avuto esito positivo verificando la presenza dell’ID e la corretta associazione con il dottore.

6.4.2 Gestione degli errori

Un secondo test si è concentrato sulla gestione di input errati. È stato simulato il caso in cui si tenti di creare un invito per un paziente già invitato in precedenza. Il servizio doveva sollevare un’eccezione `BadRequestException`, come previsto dai requisiti.

6.5 Vantaggi del testing con Jest e NestJS

L’approccio adottato ha consentito di ottenere diversi benefici:

- **Rapidità** – grazie al mocking, i test vengono eseguiti in pochi millisecondi.

-
- **Affidabilità** – ogni unità di codice è testata isolatamente, facilitando l'individuazione di errori.
 - **Copertura** – la generazione dei report di coverage ha permesso di quantificare la porzione di codice verificata.
 - **Integrazione** – compatibilità totale con TypeScript, NestJS e TypeORM.

6.6 Conclusioni

La fase di testing, pur non esaustiva al 100%, ha rappresentato un passaggio cruciale nel garantire l'affidabilità del sistema SymbioCare. Gli unit test realizzati hanno fornito una prima validazione delle logiche di business fondamentali, assicurando un comportamento coerente rispetto ai requisiti e riducendo il rischio di regressioni future.

Bibliografia

- [1] DocPlanner. *MioDottore – Prenota una visita.* 2025. URL: <https://www.miodottore.it/>.
- [2] DocPlanner. *Doctolib – Prenota una visita.* 2025. URL: <https://www.doctolib.it/>.
- [3] MioDottore. *Scenario della sanità digitale: funzionalità CRM e gestionali.* 2025. URL: <https://pro.miodottore.it/blog/centrimedici/topic/gestione/post/sanita-digitale-scenari-presenti-e-futuri>.
- [4] MioDottore. *Centro Prenotazioni per Medici di Famiglia e Pediatri.* 2025. URL: <https://pro.miodottore.it/funzionalita/centro-prenotazioni-mmgi-pediatri>.
- [5] Canvas Business Model. *Competitive Landscape of Doctolib Company.* 2025. URL: <https://canvasbusinessmodel.com/blogs/competitors/doctolib-competitive-landscape>.
- [6] Wikipedia. *Digital health interventions — over 100,000 mobile health applications in app stores.* Accesso: 17 agosto 2025. 2025. URL: https://en.wikipedia.org/wiki/Digital_health_interventions.
- [7] Meta. *React – A JavaScript library for building user interfaces.* 2025. URL: <https://react.dev/>.

-
- [8] Stack Overflow. *Stack Overflow Developer Survey 2024 – Most Popular Web Frameworks*. 2024. URL: <https://survey.stackoverflow.co/2024/technology#most-popular-technologies-webframe>.
 - [9] Domenico De Biase. «Component-based Software Engineering: A Case Study on Modular User Interfaces». In: *International Journal of Education and the Arts (IJE)* 2.1 (2021), pp. 45–56. URL: <https://www.gbspress.com/index.php/IJE/article/view/312>.
 - [10] Evan You e Vue.js Community. *Vue.js – The Progressive JavaScript Framework*. 2025. URL: <https://vuejs.org/>.
 - [11] Google. *Angular – The modern web developer's platform*. 2025. URL: <https://angular.io/>.
 - [12] Mantine. *Mantine UI – React components library*. 2025. URL: <https://mantine.dev/>.
 - [13] MUI. *Material UI – React component library based on Material Design*. 2025. URL: <https://mui.com/>.
 - [14] The Bootstrap Authors. *Bootstrap – The most popular HTML, CSS, and JS library*. 2025. URL: <https://getbootstrap.com/>.
 - [15] Google. *Jetpack Compose – Modern toolkit for building native Android UI*. 2025. URL: <https://developer.android.com/compose>.
 - [16] JetBrains. *Kotlin and Android | Android Developers Documentation*. Accessed: 2025-08-08. 2025. URL: <https://kotlinlang.org/docs/android-overview.html>.
 - [17] JetBrains. *Kotlin Programming Language*. 2025. URL: <https://kotlinlang.org/>.
 - [18] Google. *Flutter – Build apps for any screen*. 2025. URL: <https://flutter.dev/>.

- [19] Meta. *React Native – Create native apps for Android and iOS*. 2025. URL: <https://reactnative.dev/>.
- [20] NestJS. *NestJS – A progressive Node.js framework*. Ultimo accesso: 17 agosto 2025. 2025. URL: <https://nestjs.com/>.
- [21] OpenJS Foundation. *Node.js – JavaScript runtime built on Chrome's V8 engine*. 2025. URL: <https://nodejs.org/>.
- [22] Microsoft. *TypeScript – Typed JavaScript at Any Scale*. 2025. URL: <https://www.typescriptlang.org/>.
- [23] Express.js Contributors. *Express.js – Fast, unopinionated, minimalist web framework for Node.js*. 2025. URL: <https://expressjs.com/>.
- [24] Spring. *Spring Boot – Spring framework for rapid application development*. 2025. URL: <https://spring.io/projects/spring-boot>.
- [25] Django Software Foundation. *Django – The web framework for perfectionists with deadlines*. 2025. URL: <https://www.djangoproject.com/>.
- [26] TypeORM. *TypeORM – ORM for TypeScript and JavaScript*. 2025. URL: <https://typeorm.io/>.
- [27] PostgreSQL Global Development Group. *PostgreSQL – The world's most advanced open source database*. 2025. URL: <https://www.postgresql.org/>.
- [28] Oracle. *MySQL – The world's most popular open source database*. 2025. URL: <https://www.mysql.com/>.
- [29] MariaDB Foundation. *MariaDB – Open source relational database*. 2025. URL: <https://mariadb.org/>.
- [30] MongoDB Inc. *MongoDB – The developer data platform*. 2025. URL: <https://www.mongodb.com/>.

-
- [31] Inc. GitHub. *GitHub: Where the world builds software*. 2025. URL: <https://github.com/>.
 - [32] Retrofit. *Retrofit*. 2025. URL: <https://square.github.io/retrofit/>.
 - [33] JWT. *JWT*. 2025. URL: <https://www.jwt.io/introduction#what-is-json-web-token>.
 - [34] Tanner Linsley e Contributors. *TanStack Query (React Query)*. 2025. URL: <https://tanstack.com/query/v5>.
 - [35] Kevin Vandy. *Mantine React Table*. 2025. URL: <https://www.mantine-react-table.com/>.
 - [36] React-qr-code. *React-qr-code*. 2025. URL: <https://www.npmjs.com/package/react-qr-code>.
 - [37] Recharts Team. *Recharts: A composable charting library built on React components*. 2025. URL: <https://recharts.org/en-US/>.
 - [38] Jason Quense e contributors. *React Big Calendar*. 2025. URL: <https://www.npmjs.com/package/react-big-calendar>.
 - [39] Mantine. *useForm Hook*. 2025. URL: <https://mantine.dev/form/use-form/>.
 - [40] Colinhacks. *Zod: TypeScript-first schema validation with static type inference*. 2025. URL: <https://zod.dev/>.
 - [41] Meta Platforms, Inc. *Jest*. <https://jestjs.io/>. 2025.

Glossario

dependency injection Meccanismo che consente di gestire automaticamente le dipendenze tra componenti, migliorando modularità, riuso e testabilità.

guard (NestJS) Meccanismo che determina se una richiesta è autorizzata ad accedere a una risorsa o a un endpoint.

interceptor Componente che intercetta richieste o risposte per modificarle o arricchirle (es. logging, mapping della risposta, gestione errori).

middleware Funzioni eseguite durante il ciclo di gestione di una richiesta (es. log, autenticazione, parsing), prima o dopo l'esecuzione dei controller.

Node.js Ambiente di runtime open source per eseguire JavaScript lato server, basato su V8.

TypeScript Superset tipizzato di JavaScript che introduce tipizzazione statica e funzionalità per applicazioni su larga scala.

validazione basata su *decorators* Approccio dichiarativo in cui annotazioni (@decorator) su classi e proprietà definiscono regole di validazione dei dati.