Politecnico di Torino

Politecnico di Torino

Corso di Laurea Magistrale in Ingegneria Informatica

A.a. 2024/2025

Sessione di laurea Dicembre 2025

Design e sviluppo di una piattaforma Web a supporto della diagnosi linguistica nei bambini bilingui

Relatore:

Luigi De Russis

Candidato:

Giuseppe Bennardo

Sommario

Il presente lavoro descrive la progettazione e lo sviluppo di una piattaforma web ideata per supportare la diagnosi linguistica nei bambini bilingui, offrendo strumenti digitali per la raccolta, la gestione e l'analisi dei questionari e delle risposte. Il sistema si compone di due interfacce distinte: una dashboard per gli operatori sanitari, dedicata alla creazione, al monitoraggio e all'analisi dei questionari, e una web app per le famiglie, progettata per l'uso da dispositivi mobili, multilingue, accessibile e dotata di assistente vocale. Sono stati definiti i requisiti funzionali e non funzionali e progettata un'architettura software modulare, che ha consentito l'implementazione di funzionalità quali editor visuale dei questionari, autosalvataggio progressivo e monitoraggio in tempo reale. La fase di test e validazione ha permesso di valutare l'usabilità, la correttezza e la rispondenza del sistema ai requisiti prefissati. I risultati evidenziano come la soluzione proposta migliori l'efficienza del processo diagnostico e la qualità dei dati raccolti, aprendo la strada a futuri sviluppi quali analisi automatizzate, ampliamento del supporto a varianti linguistiche e dialettali, nuove funzionalità legate alla comunicazione con la famiglia, e infine interoperabilità con sistemi sanitari esistenti.

Ringraziamenti

Ai miei genitori, per il loro sostegno costante.
A mio fratello, per la sua presenza discreta ma importante.
Ai miei nonni e ai miei zii, per l'affetto e il supporto che non sono mai mancati.
Ai miei amici, per la leggerezza e la vicinanza in ogni fase di questo percorso.
Ad Agnese, per il suo supporto, la pazienza e l'amore di ogni giorno.

Indice

Elenco delle tabelle

Elenco delle figure

Capitolo 1

Introduzione

1.1 Motivazioni del progetto

Negli ultimi anni, la crescente presenza di bambini bilingui nel sistema educativo e sanitario italiano ha posto nuove sfide nella valutazione dello sviluppo linguistico in età evolutiva. Secondo i dati ISTAT 2023, gli stranieri residenti in Italia rappresentano circa l'8,6% della popolazione totale, con una netta concentrazione nel Centro-Nord (83,4%) [1]. I minori costituiscono circa un quinto (20,8%) della popolazione straniera residente, pari a oltre un milione di bambini e adolescenti, molti dei quali nati in Italia da genitori migranti di prima generazione. Nella Città Metropolitana di Torino, contesto territoriale di riferimento del presente lavoro, la quota di popolazione straniera ha raggiunto il 9,52% del totale, mentre i minori stranieri rappresentano il 14,2% della popolazione minorile complessiva [2].

Questo quadro demografico conferma come le situazioni di bi- o plurilinguismo siano ormai la norma più che l'eccezione, rendendo necessario un adeguamento degli strumenti clinici e didattici alla realtà multiculturale del territorio. Tuttavia, nella pratica quotidiana dei servizi sanitari e scolastici, il bilinguismo è spesso percepito come un fattore di rischio, piuttosto che come una risorsa cognitiva e comunicativa.

La letteratura scientifica contemporanea sottolinea invece che i bambini bilingui non presentano un ritardo patologico nello sviluppo linguistico, ma un diverso percorso evolutivo, fortemente influenzato dalla quantità e qualità dell'input linguistico ricevuto [3]. Durante la fase di acquisizione della seconda lingua (L2), il profilo linguistico di questi bambini può temporaneamente somigliare a quello dei coetanei monolingui con Disturbo Primario del Linguaggio (DPL), ma tali differenze riflettono una variazione tipica e non un deficit .

Come osservano Caselli e Rinaldi [4] nei bambini bilingui le differenze individuali risultano spesso più ampie rispetto ai monolingui, a causa della complessità dei fattori che influenzano lo sviluppo linguistico: età di prima esposizione, contesto d'uso

delle lingue, tempo dedicato a ciascuna lingua e livello di stimolazione linguistica familiare. Analogamente, Marini e Vicari [5] evidenziano che la corretta distinzione tra disturbo del linguaggio e variazione bilingue richiede strumenti di valutazione culturalmente sensibili e un'attenta analisi qualitativa del contesto comunicativo. Senza strumenti adeguati, il rischio è duplice: da un lato diagnosticare erroneamente un disturbo linguistico (over-identification), dall'altro non riconoscerne la presenza reale (under-identification), compromettendo l'intervento precoce e la presa in carico.

In molti centri di neuropsichiatria infantile italiani, la raccolta delle informazioni linguistiche avviene ancora tramite questionari cartacei o moduli digitali disomogenei, che comportano difficoltà nella gestione, archiviazione e analisi dei dati. Inoltre, la presenza di famiglie con competenze linguistiche limitate in italiano può ostacolare la compilazione autonoma dei questionari, riducendo la qualità delle informazioni raccolte e aumentando il carico di lavoro per gli operatori sanitari.

In risposta a tali criticità, è stato avviato lo sviluppo di una piattaforma web per la digitalizzazione e gestione dei questionari di valutazione linguistica nei bambini bilingui, ideata per essere utilizzata congiuntamente da operatori sanitari (neuropsichiatri, logopedisti, psicologi) e famiglie. L'obiettivo principale è digitalizzare e ottimizzare l'intero processo di valutazione, garantendo:

- la standardizzazione dei questionari e la raccolta strutturata dei dati;
- la riduzione degli errori e dei tempi di gestione;
- un'esperienza accessibile e multilingue (inizialmente italiano, spagnolo e arabo, con possibilità di estensione);
- l'integrazione di strumenti di accessibilità, come un assistente vocale;
- la conformità alle normative sulla privacy e sulla protezione dei dati sanitari (GDPR).

La piattaforma rappresenta non solo un'evoluzione tecnologica, ma anche un cambiamento di paradigma nella pratica clinica: consente una diagnosi linguistica più equa e inclusiva, fondata su dati digitali, dinamici e facilmente condivisibili. Essa si inserisce in una prospettiva di sanità digitale inclusiva, coerente con gli Obiettivi dell'Agenda 2030 delle Nazioni Unite, in particolare:

- **Obiettivo 3:** garantire salute e benessere per tutti e per tutte le età;
- **Obiettivo 4:** assicurare un'istruzione di qualità, equa e inclusiva;
- **Obiettivo 10:** ridurre le disuguaglianze tra e all'interno dei Paesi.

La promozione del multilinguismo è inoltre riconosciuta come una priorità a livello europeo. La politica linguistica dell'Unione Europea incoraggia ogni cittadino a padroneggiare almeno due lingue oltre alla propria lingua madre, considerando il multilinguismo un elemento chiave della competitività, della coesione sociale e della comprensione interculturale [6] In tale contesto, il progetto contribuisce concretamente alla realizzazione di questi obiettivi, promuovendo una diagnosi linguistica equa, accessibile e tecnologicamente avanzata, capace di valorizzare la diversità linguistica dei bambini e delle loro famiglie.

1.2 Contesto applicativo e obiettivi

Il progetto nasce dall'esigenza di supportare il lavoro clinico degli operatori sanitari che si occupano della valutazione linguistica di bambini bilingui in età prescolare e scolare. In ambito neuropsichiatrico e logopedico, la raccolta dei dati anamnestici e linguistici costituisce un passaggio essenziale per distinguere tra uno sviluppo linguistico tipico e la presenza di un Disturbo Primario del Linguaggio (DPL) o di altre difficoltà comunicative [7]. Tuttavia, nella pratica corrente, tale raccolta avviene prevalentemente attraverso questionari cartacei o file statici, la cui gestione comporta numerosi limiti: perdita o duplicazione dei dati, difficoltà di consultazione, tempi lunghi di analisi e mancanza di uniformità nelle informazioni raccolte.

Inoltre, la crescente presenza di famiglie con una lingua madre diversa dall'italiano rende la compilazione dei questionari spesso complessa o imprecisa. Gli operatori segnalano che le barriere linguistiche e tecnologiche possono compromettere-regr la completezza e l'attendibilità dei dati, rendendo più difficile la valutazione del profilo linguistico del bambino. Queste criticità si traducono non solo in inefficienze operative, ma anche in rischi clinici, come la possibilità di diagnosi non accurate o ritardate.

1.2.1 Digitalizzazione e ottimizzazione del processo

La piattaforma web fornisce un sistema centralizzato in cui gli operatori sanitari possono: creare, modificare e archiviare questionari digitali tramite un editor visuale interattivo; monitorare in tempo reale lo stato di compilazione dei questionari inviati alle famiglie; aggiungere note cliniche o osservazioni contestuali alle risposte ricevute; filtrare e esportare i dati in formati standard (CSV, Excel) per ulteriori analisi o integrazione con altri sistemi sanitari.

Per le famiglie, la piattaforma offre un'interfaccia intuitiva e accessibile, che consente di compilare i questionari in modo guidato e multilingue.

Obiettivi del progetto

Gli obiettivi progettuali sono stati definiti in stretta collaborazione con l'équipe clinica dell'ASL e possono essere articolati in tre macro-aree: clinica, tecnologica e sociale.

Obiettivi clinici

- Supportare gli operatori nella raccolta standardizzata di dati linguistici e anamnestici, riducendo la variabilità dovuta a compilazioni manuali.
- Migliorare l'accuratezza diagnostica nella distinzione tra disturbo del linguaggio e differenze bilingui, fornendo dati più completi e strutturati.
- Facilitare la continuità del percorso clinico, permettendo di consultare e aggiornare le compilazioni in modo tracciato e sicuro.

Obiettivi tecnologici

- Realizzare una *web application completa e responsive*, composta da due moduli principali:
  - una *dashboard gestionale* per gli operatori sanitari, che consenta la creazione, modifica e consultazione dei questionari, il monitoraggio delle compilazioni e la gestione delle segnalazioni o note cliniche;
  - un'interfaccia di *compilazione guidata per le famiglie*, ottimizzata per dispositivi mobili, che permetta un accesso rapido tramite link dedicato e autenticazione mediante codice fiscale del bambino.
- Garantire un'esperienza d'uso fluida e inclusiva, implementando *funzionalità di autosalvataggio progressivo* e una compilazione a più step, in modo da evitare la perdita di dati anche in caso di interruzione della sessione.
- Assicurare la compatibilità multipiattaforma (desktop, tablet e smartphone) e la piena accessibilità, nel rispetto delle linee guida WCAG 2.1, integrando meccanismi di lettura vocale e navigazione semplificata.
- Strutturare il sistema in modo che possa essere installato e mantenuto su infrastruttura locale o virtualizzata dell'ente sanitario, garantendo indipendenza da piattaforme cloud esterne e pieno controllo dei dati sensibili.

Obiettivi sociali e di accessibilità

- Promuovere l'inclusione linguistica delle famiglie migranti, offrendo interfacce multilingue e strumenti di assistenza vocale.
- Favorire la collaborazione tra operatori sanitari e genitori, rendendo la comunicazione più chiara e bidirezionale.
- Sostenere la riduzione delle disuguaglianze di accesso ai servizi diagnostici, in linea con i principi dell'Agenda 2030 (Obiettivi 3, 4 e 10).

1.3 Struttura della tesi

La presente tesi è articolata in sei capitoli, ciascuno dei quali affronta una fase specifica del percorso di analisi, progettazione e sviluppo della piattaforma web. Dopo il capitolo introduttivo, i capitoli successivi sono:

Il Capitolo 2 – *Stato dell'arte e soluzioni analoghe* analizza il panorama tecnologico di riferimento, esaminando le principali soluzioni esistenti per la creazione e gestione di questionari digitali. Il capitolo discute i limiti di tali sistemi e le motivazioni che hanno portato alla progettazione di una piattaforma dedicata, mettendo a confronto diverse tecnologie e modelli architetturali.

Il Capitolo 3 – *Progettazione e design del sistema* Vengono illustrati i criteri che hanno guidato la raccolta dei requisiti, i principali attori coinvolti e i casi d'uso rappresentativi, con il supporto di diagrammi e schemi descrittivi. Approfondisce l'architettura logica e fisica della piattaforma, descrivendo la struttura generale del sistema, il modello dei dati e i meccanismi fondamentali come autosalvataggio, gestione multilingua e autenticazione.

Il Capitolo 4 – *Implementazione e sviluppo* illustra nel dettaglio la realizzazione pratica della piattaforma, descrivendo l'organizzazione del progetto, i principali moduli software e le componenti di frontend e backend. Particolare attenzione è riservata alle funzionalità di accessibilità, tra cui il supporto per screen reader e la sintesi vocale tramite Web Speech API.

Il Capitolo 5 – *Test, validazione e risultati* presenta le strategie di verifica e validazione del sistema, cioè test API, test E2E e prove di usabilità condotte con utenti reali. I risultati ottenuti vengono analizzati in termini di correttezza, prestazioni e grado di soddisfazione degli utenti.

Il Capitolo 6 - *Conclusioni* Infine, il lavoro si conclude con una sezione di conclusioni, nella quale vengono riassunti i risultati raggiunti, discusse le principali difficoltà incontrate e delineate le possibili evoluzioni future della piattaforma.

Capitolo 2

Stato dell'arte e scelte tecnologiche

L'analisi dello stato dell'arte costituisce il punto di partenza per la definizione delle scelte tecnologiche e architetturali alla base della piattaforma sviluppata. In questa fase, è stato fondamentale valutare le soluzioni esistenti per la gestione e la digitalizzazione di questionari, al fine di comprenderne le potenzialità e i limiti e di individuare le aree in cui fosse necessario proporre un sistema alternativo.

Il progetto è nato su richiesta diretta del reparto di Neuropsichiatria Infantile dell'ASL CN2, che, in collaborazione con il Politecnico di Torino, ha espresso l'esigenza di una piattaforma web personalizzata per la gestione dei questionari diagnostici relativi alla valutazione linguistica dei bambini bilingui. Le soluzioni in uso – principalmente Microsoft Forms e Google Moduli – si erano rivelate inadeguate rispetto alle necessità operative del reparto.Le dottoresse del reparto hanno infatti segnalato numerose criticità, tra cui la mancanza di un controllo centralizzato dei dati, l'impossibilità di gestire in modo efficiente questionari multilingua, e la difficoltà di offrire ai genitori un'interfaccia realmente accessibile e intuitiva.

Questa insoddisfazione ha reso evidente la necessità di progettare un sistema ad hoc, più flessibile, moderno e indipendente dalle piattaforme cloud commerciali.

2.1 Soluzioni esistenti

Le principali piattaforme oggi disponibili per la creazione e gestione di questionari digitali sono *Google Moduli (Google Forms)*, *Microsoft Forms* e *Typeform*. Tutti e tre gli strumenti adottano un modello cloud *"Software as a Service" (SaaS)*, che consente di creare moduli interattivi, raccogliere automaticamente le risposte e visualizzare statistiche in tempo reale.

Google Moduli: parte della suite Google Workspace, è ampiamente utilizzato grazie alla sua semplicità e alla possibilità di esportare i risultati in Google Sheets. Tuttavia, il sistema offre funzionalità limitate di personalizzazione: il layout è rigido e la gestione multilingua non è nativa, ma richiede la duplicazione del questionario per ciascuna lingua. Inoltre, la conservazione dei dati avviene esclusivamente sui server Google, senza possibilità di archiviazione locale o gestione diretta delle informazioni.

Microsoft Forms: integrato nella suite Microsoft 365, è stato lo strumento effettivamente utilizzato dal reparto dell'ASL CN2 fino all'avvio di questo progetto. Le dottoresse hanno confermato che l'applicazione veniva impiegata per la distribuzione dei questionari ai genitori, apprezzandone l'interfaccia intuitiva e la facilità di condivisione. Tuttavia, l'esperienza d'uso si è rivelata limitante sotto diversi aspetti. In primo luogo, l'applicazione non offre un vero supporto per la gestione multilingua all'interno dello stesso questionario: ogni lingua richiede la creazione di un modulo separato, con conseguente aumento della complessità gestionale e del rischio di incongruenze tra versioni. In secondo luogo, la piattaforma non prevede una dashboard dedicata per gli operatori, rendendo difficile monitorare lo stato delle compilazioni o associare in modo strutturato le risposte ai singoli casi. Infine, la completa dipendenza dall'infrastruttura cloud Microsoft impedisce la conservazione locale dei dati e limita la possibilità di personalizzare interfaccia e flussi di compilazione. Per queste ragioni, nonostante la sua semplicità, Microsoft Forms è stata considerata una soluzione non adeguata per il contesto specifico del reparto.

Typeform: rappresenta una soluzione commerciale più evoluta, orientata all'esperienza utente. La piattaforma si distingue per un'interfaccia moderna e interattiva, basata su un modello "conversazionale" di compilazione: le domande vengono presentate una alla volta, rendendo l'esperienza più fluida e accattivante. Typeform consente un livello maggiore di personalizzazione grafica e supporta una logica condizionale più articolata rispetto ai concorrenti gratuiti. Tuttavia, anche questa soluzione presenta limiti sostanziali in contesti professionali che richiedano un controllo rigoroso dei dati: i dati vengono memorizzati su server esterni, le possibilità di esportazione automatizzata sono limitate ai formati supportati dal servizio e la versione gratuita non permette l'uso intensivo o l'integrazione diretta con database locali.

2.2 Analisi delle tecnologie e motivazioni delle scelte

Nel valutare le tecnologie possibili per il backend e il frontend della piattaforma, sono stati considerati diversi framework e modelli architetturali, con una particolare

attenzione alle reali esigenze operative del progetto: un numero moderato di utenti, flussi di compilazione relativamente leggeri, necessità di manutenzione sostenibile e installazione su infrastruttura interna. Tre alternative principali sono dunque state oggetto di analisi: *Spring Boot, Django, Express.js/Node.js*.

Spring Boot è ampiamente utilizzato in ambito enterprise e offre un ecosistema completo, ma la sua struttura modulare e la quantità di configurazioni lo rendono poco adatto a progetti di piccola scala. Secondo benchmark pubblicati da InfoQ [8] e JetBrains [9], le applicazioni Spring Boot hanno tempi medi di avvio di 5–6 secondi e un consumo di memoria superiore rispetto a equivalenti implementazioni Node.js. Un'analisi più ampia conferma che Node.js eccelle nelle operazioni I/O-bound e in scenari con elevata concorrenza, mentre Spring Boot risulta più adatta a carichi CPU-intensivi o applicazioni enterprise complesse [10].

Django, pur offrendo un solido ORM e un framework completo, adotta un paradigma più rigido e centralizzato, meno adatto a interfacce dinamiche e a componenti web altamente reattive. In considerazione della natura del sistema — questionari, flussi di compilazione, non elevati carichi di elaborazione— è stato dunque preferito un ambiente leggero e rapido da gestire. Express.js/Node.js ha offerto multipli vantaggi: linguaggio unico per frontend e backend (JavaScript/TypeScript), tempi di setup inferiori, minore sovraccarico di runtime e maggior flessibilità in fase di evoluzione. Queste condizioni hanno reso la scelta tecnologica coerente sia con le competenze già acquisite dallo sviluppatore, sia con la scalabilità contenuta prevista per l'applicazione.

Per il frontend, la selezione di React si è basata sulla sua diffusione (> 48 % degli sviluppatori frontend secondo un'indagine del 2025) [11] e sulla capacità di supportare interfacce modulari e accessibili. Per la persistenza dati, la combinazione di PostgreSQL e Prisma ORM ha garantito integrità relazionale, migrazioni automatizzate e manutenzione agevole, requisito rilevante in un contesto in cui le entità (operatori, questionari, risposte) presentano relazioni chiare e stabili.

2.3 Architettura generale e distribuzione

L'architettura della piattaforma è stata progettata con un approccio a tre livelli: livello di presentazione, logica applicativa e persistenza dei dati. Il frontend eroga l'interfaccia utente e le interazioni, il backend espone API RESTful e gestisce la logica applicativa, mentre il database memorizza le entità e le relazioni senza dipendere da infrastrutture esterne.

Dal punto di vista dell'infrastruttura, la scelta di installare il sistema su una macchina virtuale dedicata nella rete interna dell'ente è stata motivata da una valutazione costi-benefici che ha tenuto in conto il contesto normativo e operativo. In letteratura, il passaggio al cloud pubblico è stimato in molte organizzazioni al 41

%dei carichi di lavoro entro il 2020, con un conseguente declino di circa 10 punti percentuali per l'on-premise dal 37 % al 27 % [12]. Tuttavia, quando la priorità è il controllo dei dati e la conformità normativa, la soluzione on-premise o VM locale rimane preferita: ad esempio, oltre il 58 % delle aziende ha espresso la preferenza per mantenere carichi critici in ambienti locali.[12]

La decisione di evitare container e orchestratori deriva dal fatto che il carico operativo previsto non richiede scalabilità orizzontale massiva né gestione di microservizi complessi. Questo si traduce in una riduzione della complessità operativa, dei costi di manutenzione e dei rischi di failure legati a infrastrutture distribuite. Un'analisi sistematica tra cloud e on-premise evidenzia che, pur offrendo maggiore flessibilità, il cloud introduce overhead di gestione che in contesti piccoli può risultare controproducente. [13]

Per quanto riguarda lo scambio tra i livelli applicativi, l'adozione di API REST e formati JSON favorisce interoperabilità e future estensioni senza vincoli architetturali. L'isolamento tra componenti consente inoltre di sostituire o aggiornare ciascun modulo indipendentemente, riducendo il rischio di impatti sistemici. La scelta di una VM interna garantisce backup, snapshot e controllo completo sull'ambiente di esecuzione, elementi essenziali in un contesto che deve rispettare normative sanitarie e requisiti di riservatezza.

Capitolo 3

Analisi e progettazione del sistema

3.1 Introduzione e metodologia di analisi

La fase di analisi dei requisiti ha rappresentato il punto di partenza del processo di progettazione della piattaforma web, con l'obiettivo di tradurre in specifiche funzionali e tecniche le esigenze operative del reparto di Neuropsichiatria Infantile dell'ASL di Torino. Il progetto è nato su iniziativa di due dottoresse del reparto, che hanno espresso la necessità di disporre di uno strumento digitale dedicato alla somministrazione, raccolta e gestione dei questionari linguistici utilizzati durante le valutazioni dei bambini bilingui. In collaborazione con il Politecnico di Torino e sotto la supervisione del professor Luigi De Russis, è stato avviato un processo di co-progettazione che ha portato alla definizione delle funzionalità e dei requisiti della piattaforma.

L'attività di analisi è stata condotta attraverso una serie di incontri in presenza tra il team universitario e le professioniste del reparto. Durante tali riunioni, le dottoresse hanno illustrato le modalità operative attuali e le difficoltà riscontrate nell'utilizzo degli strumenti esistenti — in particolare Microsoft Forms, allora impiegato per la distribuzione dei questionari ai genitori. Dalle discussioni è emersa la necessità di una piattaforma centralizzata, personalizzabile e accessibile, in grado di superare i limiti di soluzioni cloud generiche e di ottimizzare la gestione del flusso di compilazione.

Tra le funzionalità ritenute prioritarie sono state individuate:

- la gestione multilingue dei questionari, per permettere la compilazione da parte di famiglie non italofone;

- l'integrazione di un assistente vocale a supporto dell'accessibilità e della comprensione delle domande;
- un'interfaccia responsive, utilizzabile da computer, tablet e smartphone;
- la presenza di un meccanismo di autosalvataggio progressivo con compilazione multistep, per evitare la perdita di dati e permettere di riprendere la compilazione in più momenti;
- un sistema per la tracciabilità e il monitoraggio delle compilazioni da parte degli operatori sanitari.

Nel corso degli incontri, le esigenze espresse sono state progressivamente formalizzate e organizzate in un insieme strutturato di requisiti. La definizione non è avvenuta in un'unica fase, ma attraverso *un processo iterativo e incrementale*: dopo una prima identificazione delle funzionalità chiave, è stata elaborata una proposta di requisiti preliminari, successivamente discussa e rivista insieme alle dottoresse e al relatore. Questo confronto ha permesso di valutare la *fattibilità tecnica* di ciascuna richiesta, di chiarire le priorità e di trasformare le esigenze cliniche e organizzative in specifiche tecniche realizzabili.

Il risultato di questa fase è consistito in un insieme coerente di *funzionalità, requisiti e casi d'uso principali*, che hanno guidato la progettazione dell'architettura del sistema e la successiva fase di implementazione. In particolare, l'analisi ha consentito di individuare le entità fondamentali del dominio applicativo, i ruoli degli utenti e i flussi di interazione che saranno descritti nei paragrafi seguenti.

3.1.1 Evoluzione e validazione dei requisiti

Durante il processo di analisi e di confronto con le dottoresse del reparto, alcune delle funzionalità inizialmente ipotizzate sono state oggetto di revisione, al fine di semplificare l'utilizzo del sistema e ottimizzare le risorse di sviluppo. Un esempio significativo riguarda il *meccanismo di autenticazione per le famiglie*.Nella prima versione concettuale del progetto, era stato previsto un sistema di accesso basato su *link o QR code univoco*, generato dall'operatore per ciascun caso clinico. La famiglia avrebbe ricevuto un collegamento personalizzato, valido per accedere direttamente al questionario senza necessità di registrazione. Questo modello garantiva un buon livello di anonimato e tracciabilità, ma richiedeva la gestione di codici temporanei, invii manuali e potenziali problemi di scadenza o smarrimento dei link. Durante la validazione con il reparto, si è deciso di semplificare l'approccio, adottando un sistema basato sull'inserimento del *codice fiscale del bambino* come chiave identificativa. Questa soluzione riduce la complessità gestionale, elimina la necessità di generare e distribuire link personalizzati e mantiene una sufficiente univocità, a scapito però di un leggero sacrificio in termini di anonimato. La scelta

è stata motivata da criteri di *praticità, rapidità d'uso e robustezza operativa*, più in linea con l'ambiente sanitario di destinazione.

Un'ulteriore modifica ha riguardato la funzionalità di *riconoscimento vocale (speech-to-text)*, inizialmente ipotizzata per consentire la compilazione tramite dettatura. Durante la fase di analisi è emerso che la maggior parte delle famiglie utilizza dispositivi mobili con tastiere già dotate di microfono integrato (Google Voice Typing o Siri Dictation). Implementare un riconoscimento vocale interno alla piattaforma avrebbe introdotto dipendenze da servizi esterni e possibili problemi di compatibilità tra sistemi operativi. La funzione che è stata mantenuta invece è quella di *lettura vocale (text-to-speech)* delle domande, più utile ai fini dell'accessibilità e già nativamente supportata dai browser moderni. Infine, alcune funzionalità considerate opzionali sono state eliminate in fase di revisione. Tra queste, il sistema di *notifiche e reminder automatici* per ricordare la compilazione dei questionari in sospeso. Le dottoresse hanno infatti osservato che la durata media della compilazione è breve e il tasso di abbandono minimo, rendendo tale funzione superflua rispetto ai costi di implementazione e manutenzione. Questi adattamenti dimostrano come l'intero processo di definizione dei requisiti sia stato condotto secondo un approccio *incrementale e partecipativo*, in cui le decisioni progettuali sono state progressivamente validate in base a criteri di efficacia, semplicità e coerenza con le esigenze del contesto clinico. Le versioni intermedie di alcuni workflow e diagrammi, come quello relativo al sistema di login inizialmente proposto, sono state mantenute a fini di documentazione interna e di confronto metodologico, ma non sono state incluse nel progetto finale. La progettazione successiva si è quindi basata sulle versioni consolidate dei flussi operativi, illustrate nei paragrafi seguenti.

3.2 Raccolta e definizione dei requisiti

3.2.1 Requisiti funzionali

I requisiti funzionali (RF) descrivono i servizi e le operazioni che il sistema deve essere in grado di eseguire per rispondere agli obiettivi del progetto. Essi definiscono *cosa* il sistema fa, indipendentemente dalle modalità implementative. Nella tabella seguente vengono elencati i principali requisiti funzionali individuati.

ID	Descrizione	Priorità
RF01	Il sistema deve permettere all'operatore sanitario di autenticarsi tramite credenziali e accedere alla dashboard di gestione.	Alta
RF02	Il sistema deve consentire all'operatore di creare, modificare e cancellare questionari digitali, partendo da template predefiniti o personalizzati.	Alta
RF03	Ogni questionario deve poter essere distribuito tramite un link univoco accessibile dalla famiglia destinataria.	Alta
RF04	All'apertura del questionario, la famiglia deve poter identificarsi tramite l'inserimento del codice fiscale del bambino.	Alta
RF05	Il sistema deve consentire la compilazione *multistep* del questionario, con *autosalvataggio progressivo* dei dati in caso di interruzione.	Alta
RF06	Le domande dei questionari devono supportare la traduzione in più lingue selezionabili all'avvio della compilazione.	Alta
RF07	La piattaforma deve integrare un assistente vocale per la lettura dei testi (text-to-speech) a supporto dell'accessibilità.	Media
RF08	L'operatore deve poter visualizzare, filtrare e scaricare le compilazioni ricevute, con possibilità di aggiungere note cliniche associate.	Alta
RF09	Il sistema deve consentire all'operatore di monitorare lo stato dei questionari (in corso e completati) e visualizzare in tempo reale le risposte già fornite, anche per questionari parzialmente completati..	Alta
RF10	L'utente familiare deve poter modificare e completare una compilazione interrotta senza perdere i dati inseriti in precedenza.	Alta
RF11	L'operatore deve poter eliminare questionari obsoleti o creati per errore. L'eliminazione potrebbe richiedere una conferma.	Media
RF12	L'operatore deve poter aggiungere annotazioni o osservazioni professionali alle risposte fornite dai genitori.	Media
RF13	Il sistema deve fornire un sistema di gestione dei feedback: durante la compilazione, l'utente famiglia può inviare, attraverso una modale, un feedback di segnalazione relativo all'intero questionario o a una domanda specifica. L'operatore può vedere la lista delle segnalazioni ed eventualmente risolverli.	Media
RF14	Nella definizione del template del questionario è possibile associare un testo di aiuto opzionale a ciascuna domanda. Se presente viene visualizzata un'icona ("?") accanto alla domanda. .	Bassa

Tabella 3.1: Requisiti funzionali del sistema

I requisiti funzionali sopra riportati coprono l'intero ciclo di vita del questionario, dalla creazione alla compilazione e alla gestione dei dati. Essi costituiscono la base operativa per la definizione dei casi d'uso e dei flussi applicativi descritti nelle sezioni successive.

3.2.2 Requisiti non funzionali

I requisiti non funzionali (RNF) descrivono le caratteristiche qualitative del sistema, ovvero *come* esso deve comportarsi per garantire sicurezza, affidabilità e usabilità. La tabella seguente riassume i principali requisiti non funzionali individuati.

ID	**Descrizione**	**Priorità**
RNF01	Il sistema deve essere conforme al Regolamento Europeo 2016/679 (GDPR) per la protezione dei dati personali.	Alta
RNF02	Tutti i dati devono essere memorizzati su un database locale o in ambiente virtuale interno all'ente, garantendo controllo e tracciabilità.	Alta
RNF03	L'interfaccia deve essere accessibile, in conformità alle linee guida WCAG 2.1, e supportare screen reader e assistenti vocali.	Alta
RNF04	L'applicazione deve essere *responsive*, garantendo un'esperienza ottimale su dispositivi desktop e mobile.	Alta
RNF05	I tempi medi di caricamento delle pagine devono essere inferiori a 1 secondo in condizioni di rete standard.	Media
RNF06	Il sistema deve essere altamente affidabile con tempi di inattività minimi, garantendo la disponibilità dei dati e dei servizi quando necessario..	Alta
RNF07	La piattaforma deve essere manutenibile e modulare, consentendo aggiornamenti e aggiunta di nuove lingue o funzionalità senza modifiche invasive.	Media
RNF08	L'interfaccia deve utilizzare un linguaggio visivo semplice e coerente, adatto a utenti con competenze digitali limitate.	Alta
RNF09	Tutte le operazioni devono essere tracciabili nel sistema di logging, con livelli di accesso differenziati per ruolo.	Media
RNF10	Il sistema deve essere in grado di gestire un numero crescente di utenti, questionari e dati senza degradazione delle prestazioni. Capacità di gestire almeno 10 questionari attivi contemporaneamente e supporto per almeno 2 operatori sanitri concorrenti.	Media

Tabella 3.2: Requisiti non funzionali del sistema

3.3 Casi d'uso principali

La modellazione dei casi d'uso è stata condotta successivamente alla definizione dei requisiti funzionali e non funzionali, con l'obiettivo di rappresentare in modo formale le interazioni tra gli attori e il sistema. Nel presente progetto, i requisiti sono stati elaborati e validati in una fase preliminare, poiché derivano direttamente dalle necessità espresse dagli operatori sanitari e dalle specifiche funzionalità richieste per la piattaforma. I casi d'uso illustrati di seguito traducono quindi tali requisiti in flussi operativi concreti, descrivendo il comportamento del sistema in relazione ai due principali attori identificati: *operatore sanitario* e *famiglia*. Questa scelta metodologica consente di mantenere una chiara tracciabilità tra requisiti e processi, garantendo coerenza tra la fase analitica e la progettazione successiva.

3.3.1 Descrizione dei casi d'uso

Il diagramma dei casi d'uso, riportato in Figura 3.1, rappresenta le principali interazioni tra i due attori identificati — *Genitore* e *Operatore sanitario* — e la piattaforma web. L'obiettivo della modellazione è descrivere, a livello concettuale, il comportamento del sistema in risposta alle azioni dell'utente, evidenziando le funzionalità chiave individuate durante la fase di analisi dei requisiti.

L'attore "Genitore" interagisce con il sistema principalmente nella fase di compilazione del questionario. Il flusso inizia con l'*accesso al questionario* tramite link o QR code fornito dall'operatore, seguito dalla fase di *identificazione mediante codice fiscale del bambino selezione della lingua* di compilazione. Fatto l'accesso l'utente può procedere attraverso una compilazione *multistep*, con *autosalvataggio automatico* dei progressi. Durante la compilazione, la famiglia può *segnalare eventuali domande ambigue* o difficili da comprendere, interagendo così indirettamente con l'operatore sanitario. In aggiunta, il caso d'uso *Usare assistente vocale* è definito come un'estensione opzionale della compilazione, permettendo la lettura automatica delle domande tramite sistema *text-to-speech*, al fine di migliorare l'accessibilità per utenti con difficoltà visive o linguistiche.

L'attore "Operatore sanitario" interagisce con la piattaforma in modo più ampio, gestendo l'intero ciclo di vita dei questionari. Le sue attività principali comprendono la *creazione di nuovi questionari*, la *gestione delle lingue disponibili*, il *monitoraggio delle risposte* e l'*aggiunta di note cliniche* associate a singole compilazioni. L'operatore può inoltre *risolvere le segnalazioni* inviate dalle famiglie, integrando le informazioni ricevute nel processo diagnostico, e infine *esportare i risultati* in formato Excel per successive analisi. Alcune relazioni di inclusione evidenziano dipendenze funzionali tra le attività, come l'inclusione obbligatoria della "Creazione del questionario" nel caso d'uso "Aggiungere lingua" o della "Visualizzazione risposte" in "Aggiungere note".

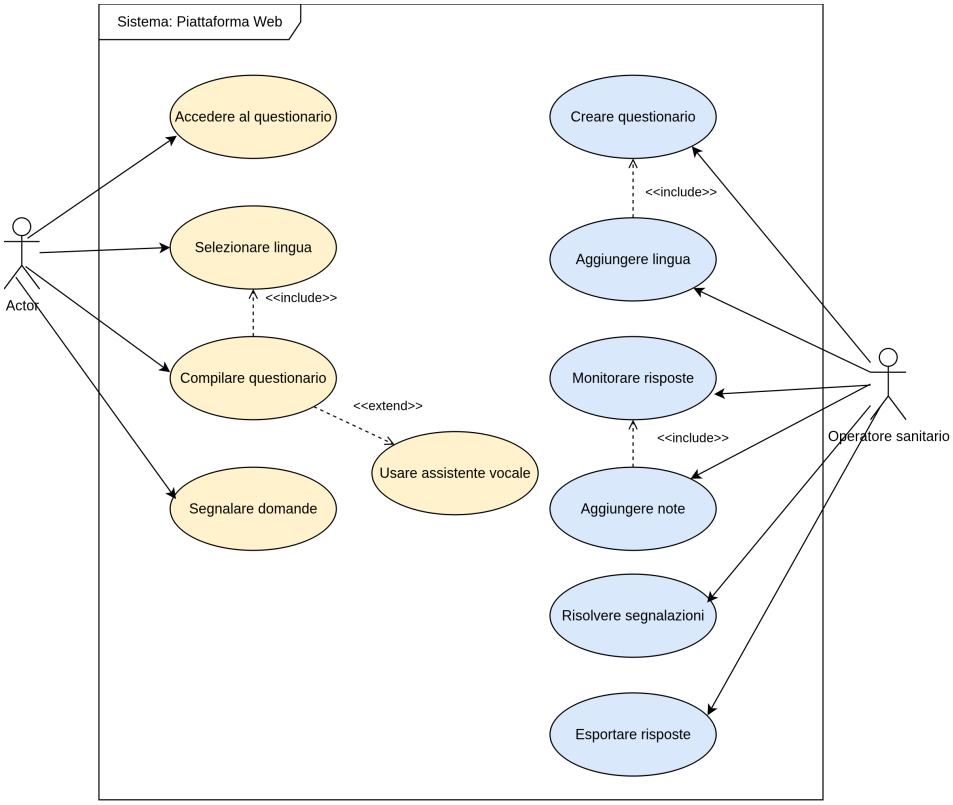


Figura 3.1: Diagramma casi d'uso

3.3.2 Workflow principale del ciclo di compilazione

Il diagramma in Figura 3.2 rappresenta il flusso complessivo di interazione tra i due attori principali della piattaforma — *operatore sanitario* e *famiglia* — durante l'intero ciclo di vita di un questionario, dalla sua creazione fino alla revisione delle risposte. Sono state adottate tre *swimlane* per distinguere chiaramente le attività di ciascun attore, riducendo al minimo le operazioni ridondanti svolte dal sistema.

Il flusso inizia con la *creazione del questionario* da parte dell'operatore, che accede alla dashboard, genera un nuovo template e ne ottiene il link univoco di compilazione. Tale link viene condiviso con la famiglia attraverso il canale preferito (email, messaggio o QR code).

La *famiglia*, aprendo il link ricevuto, accede alla pagina di ingresso in cui deve selezionare la lingua di compilazione e inserire il codice fiscale del bambino. Le lingue disponibili per la selezione non sono fisse, ma dipendono dalle traduzioni effettivamente presenti nel momento dell'accesso: il sistema mostra soltanto le versioni linguistiche già caricate per quel questionario. Una volta selezionata la lingua e inserito il codice fiscale, il sistema verifica la validità dei dati: se il codice

fiscale è valido, viene creata (o ripristinata, in caso di sessione esistente) una *submission* associata a quel questionario e a quella lingua, impostata nello stato *in progress*. È importante notare che, dopo l'avvio della compilazione, la lingua scelta diventa vincolante: se la famiglia tenta di riaccedere al questionario in una lingua diversa da quella iniziale, il sistema rileva l'incongruenza e blocca l'accesso, richiedendo di utilizzare la lingua originaria. Tale vincolo evita discrepanze tra le versioni linguistiche del questionario e garantisce la coerenza dei dati raccolti. Nella dashboard dell'operatore, la compilazione compare nella sezione dedicata, dove è possibile visualizzare le risposte ricevute, aggiungere note alle risposte e, se necessario, esportare i dati in formato Excel per ulteriori analisi. Il workflow termina quando l'operatore conclude la revisione, completando il ciclo logico del questionario.

3.3.3 Workflow di gestione delle segnalazioni

Il workflow illustrato in Figura 3.3 descrive il flusso di comunicazione tra famiglia e operatore sanitario in caso di segnalazione di un'anomalia o di una domanda poco chiara durante la compilazione del questionario. La famiglia, durante la compilazione, può attivare la funzione *Segnala domanda*, inserendo un breve messaggio descrittivo. Il sistema registra la segnalazione e la associa alla specifica domanda e al questionario compilato, impostandone lo stato iniziale a *nuova*. L'operatore visualizza nella dashboard la lista delle segnalazioni, con la possibilità di aprirle, impostare lo stato su *in esame* e, successivamente, su *risolta*. Al salvataggio, la segnalazione viene rimossa dall'elenco delle nuove e resta disponibile per consultazione archivistica.

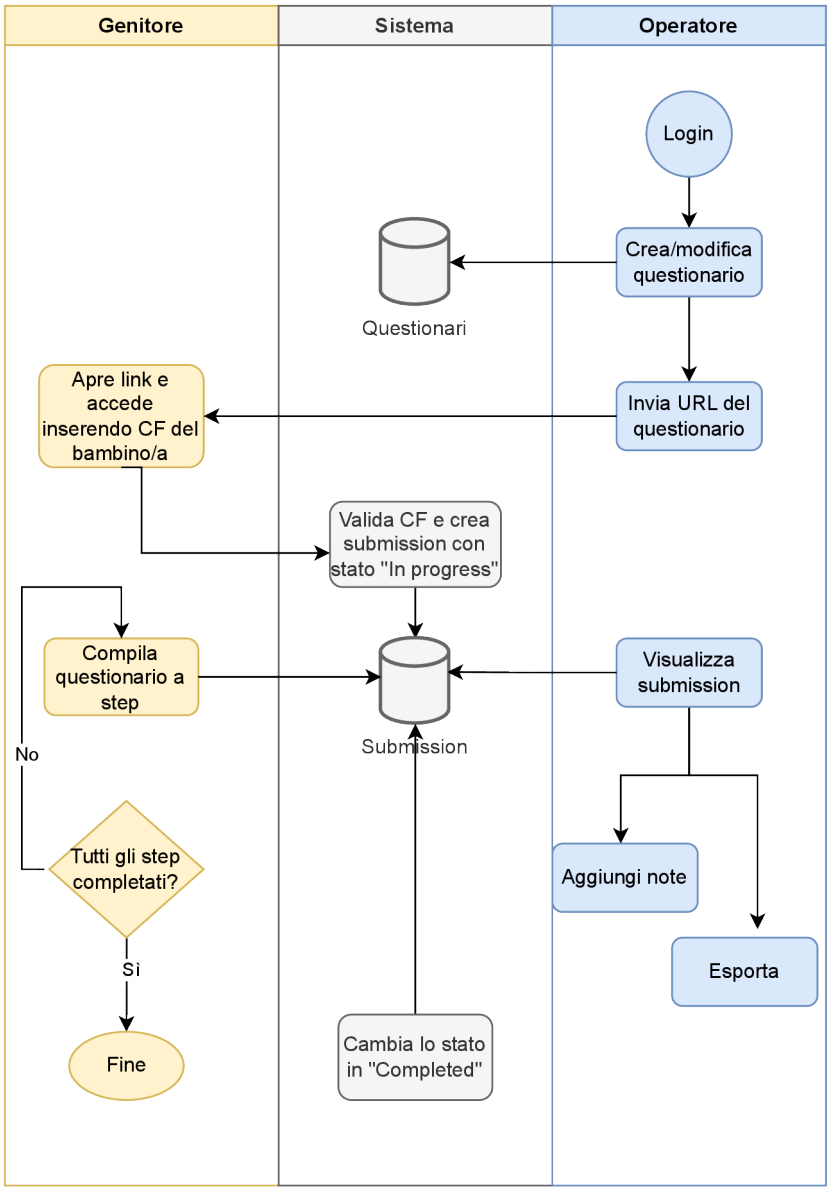


Figura 3.2: Workflow principale

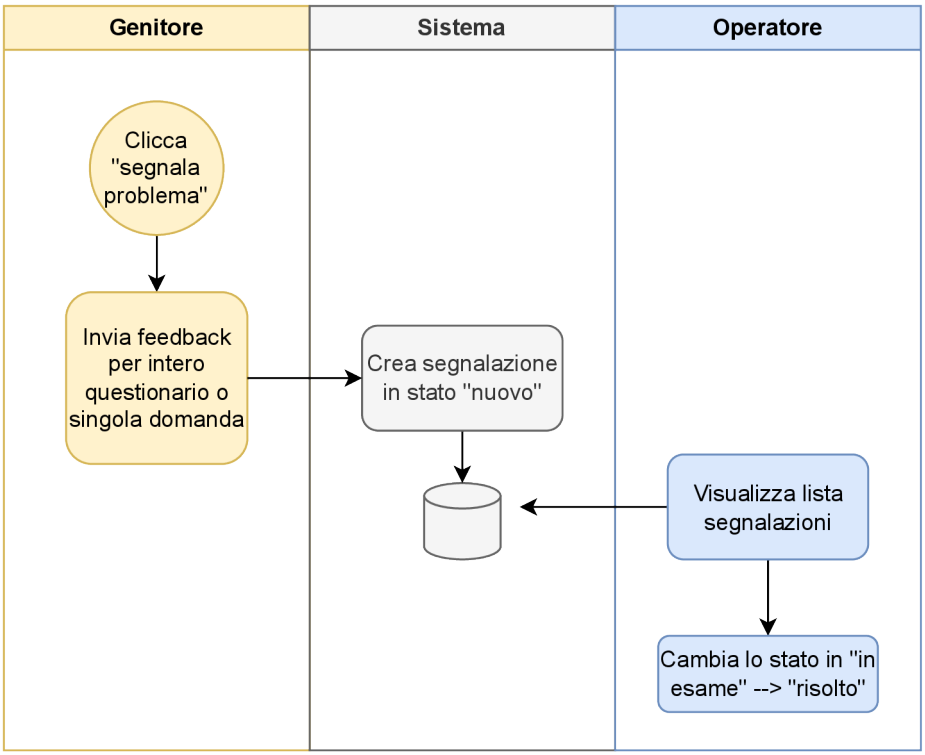


Figura 3.3: Workflow segnalazione

3.4 Software design

La progettazione software definisce la struttura logica del sistema, le relazioni tra i suoi componenti e i principi di design che ne guidano lo sviluppo. Nelle pagine seguenti viene presentato il diagramma dell'architettura generale della piattaforma e una panoramica delle principali tecnologie impiegate per la realizzazione dei vari moduli, con particolare attenzione al ruolo di ciascun componente all'interno del sistema.

3.4.1 Architettura del sistema

L'architettura generale, illustrato in Figura 3.4 della piattaforma è di tipo client–server, basata su un modello a tre livelli (presentation, application e data layer). Il sistema è composto da due interfacce distinte ma integrate: Frontend operatori sanitari, sviluppato in React e gestito tramite Vite, che fornisce una dashboard per la creazione, la gestione e l'analisi dei questionari. Frontend famiglie, anch'esso basato su React/Vite, dedicato alla compilazione dei questionari da parte dei genitori, con interfaccia semplificata e multilingue. Entrambi i frontend comunicano con

il backend tramite *API RESTful*, garantendo uno scambio dati standardizzato in formato JSON. Le API espongono endpoint per la gestione di autenticazione, questionari, compilazioni e note.

Il backend è sviluppato in *Node.js* con il *framework Express*, organizzato secondo una struttura modulare a livelli:

- *Route layer*, che definisce i percorsi delle API e applica i middleware di autenticazione e validazione;
- *Controller layer*, responsabile della logica di gestione delle richieste e della composizione delle risposte HTTP;
- *Service layer*, che incapsula la logica applicativa;
- *Repository layer*, che interagisce con il database tramite il Prisma ORM.

Il database relazionale scelto è PostgreSQL e Prisma funge da strato di astrazione per l'accesso ai dati, gestendo le migrazioni e fornendo un modello fortemente tipizzato in TypeScript. L'autenticazione avviene in modo differenziato per le due tipologie di utenti:

- gli operatori sanitari si autenticano tramite *JWT* (JSON Web Token), con credenziali gestite nel database;
- le famiglie accedono ai questionari tramite codice fiscale e link fornito dall'operatore, senza necessità di registrazione.

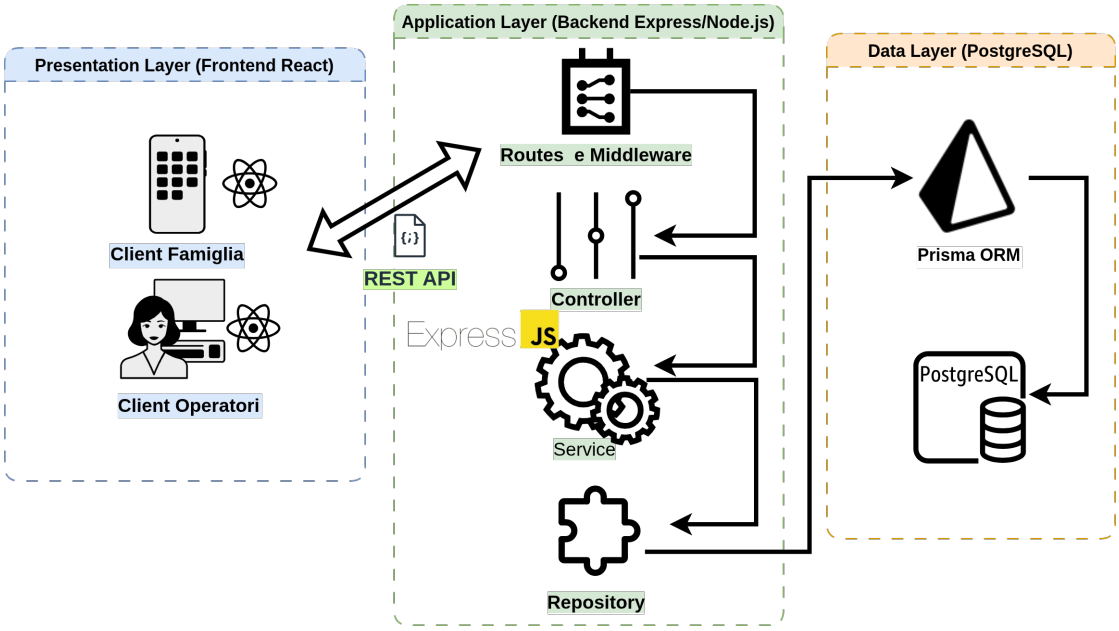


Figura 3.4: Architettura del sistema

3.4.2 Panoramica sulle tecnologie utilizzate

React

React è una libreria JavaScript open source sviluppata da Meta (in origine Facebook) per la costruzione di interfacce utente interattive e dinamiche. Introdotta nel 2013, si è rapidamente affermata come uno degli strumenti più utilizzati per lo sviluppo di applicazioni web moderne grazie al suo approccio dichiarativo, alla gestione efficiente del rendering e alla forte modularità basata sui componenti.

Alla base del modello di React vi è il concetto di componente, un'unità indipendente e riutilizzabile che rappresenta una parte dell'interfaccia utente. Ogni componente può possedere un proprio stato (state) e ricevere dati attraverso le props (proprietà), consentendo la costruzione di interfacce complesse mediante la composizione di elementi semplici. Questo paradigma component-based favorisce la separazione delle responsabilità e semplifica la manutenzione e l'estensione del codice.

Un aspetto distintivo di React è l'uso del Virtual DOM, una rappresentazione virtuale in memoria del Document Object Model reale. Ogni volta che lo stato dell'applicazione cambia, React aggiorna il Virtual DOM e calcola in modo efficiente le differenze rispetto alla versione precedente, applicando poi solo le modifiche necessarie al DOM effettivo. Tale meccanismo riduce i costi computazionali del

rendering e migliora notevolmente le prestazioni rispetto a un aggiornamento diretto del DOM tradizionale [14].

Il ciclo di vita di un componente React si articola in diverse fasi (montaggio, aggiornamento e smontaggio), che possono essere gestite mediante gli hook, introdotti a partire dalla versione 16.8. Gli hook sono funzioni speciali che permettono di utilizzare lo stato e altre funzionalità di React senza dover scrivere classi. Tra i più utilizzati vi sono useState per la gestione dello stato locale, useEffect per l'esecuzione di effetti collaterali e useContext per la condivisione di dati globali tra componenti senza ricorrere al passaggio esplicito di props. Gli hook favoriscono un modello di sviluppo più funzionale e leggibile, e costituiscono oggi la base di quasi tutte le applicazioni React moderne [15].

La libreria integra inoltre un sistema di routing client-side, generalmente implementato tramite pacchetti dedicati come React Router, che consente la gestione della navigazione tra pagine senza ricaricare l'intera applicazione. Questo approccio single-page application (SPA) offre una migliore continuità d'esperienza utente e riduce i tempi di caricamento percepiti.

React è progettato per essere agnostico rispetto al backend e si adatta facilmente a differenti contesti architetturali, comunicando con server o API REST tramite chiamate asincrone (fetch o librerie dedicate come Axios). Il flusso di dati unidirezionale – dai componenti genitore verso i figli – favorisce la prevedibilità dello stato e riduce il rischio di effetti collaterali indesiderati.

Ottimizzazione e architettura a componenti in React

Dal punto di vista della produttività, React si integra con un ampio ecosistema di strumenti: sistemi di bundling e sviluppo locale (come Vite, Webpack o Parcel), librerie di gestione dello stato globale (Redux, Zustand, Recoil), e framework fullstack come Next.js o Remix. L'ampia comunità open source garantisce un continuo aggiornamento, una documentazione estesa e il supporto di numerosi componenti di terze parti.

Un aspetto fondamentale dello sviluppo con React riguarda la gestione efficiente dei render e l'ottimizzazione delle prestazioni, specialmente in applicazioni complesse con numerosi componenti interattivi. Per questo motivo, la libreria mette a disposizione una serie di strumenti che permettono di controllare e ridurre i ricalcoli inutili, come gli hook useMemo e useCallback, insieme al componente di ordine superiore React.memo.

L'hook useMemo consente di memorizzare il risultato di un calcolo costoso finché le sue dipendenze non cambiano, evitando che l'operazione venga rieseguita a ogni render. È particolarmente utile per ottimizzare trasformazioni di dati o computazioni pesanti che dipendono da valori di input relativamente stabili. In modo analogo, useCallback permette di memorizzare la definizione di una

funzione, in modo che React la ricrei solo quando le variabili da cui dipende subiscono modifiche. Questo risulta utile nei casi in cui si passi una callback come prop a componenti ottimizzati o a hook che richiedono stabilità referenziale. Infine, React.memo può essere utilizzato per avvolgere componenti "puri", cioè che producono sempre lo stesso output a parità di input, impedendo il loro re-render se le props non sono cambiate.

Questi strumenti vanno però usati con consapevolezza. Un uso eccessivo della memoization può infatti introdurre complessità non necessaria e, paradossalmente, ridurre le prestazioni complessive dell'applicazione. Ogni funzione memoizzata comporta un costo di gestione della cache e un controllo delle dipendenze, e nei casi in cui il lavoro da evitare sia minimo, tale overhead può superare i benefici. Inoltre, la dichiarazione errata o incompleta delle dipendenze di useMemo o useCallback può produrre comportamenti inattesi, come valori obsoleti o mancati aggiornamenti. È quindi buona pratica applicare queste ottimizzazioni solo dopo aver individuato, tramite strumenti di profiling, le parti effettivamente critiche dal punto di vista delle performance.

Con l'introduzione del React Compiler (introdotto sperimentalmente nel 2025), molte di queste ottimizzazioni potranno essere automatizzate dal compilatore stesso, che sarà in grado di rilevare i calcoli ripetuti e stabilire autonomamente quando è opportuno applicare memoization. Tuttavia, comprendere il funzionamento di useMemo e useCallback rimane essenziale, sia per scrivere codice performante, sia per interpretare correttamente il comportamento dei componenti ottimizzati.

Sul piano architetturale, React promuove un modello basato su componenti: l'interfaccia utente è costruita come una composizione di elementi autonomi, ognuno dei quali incapsula logica, struttura e comportamento. Questo paradigma comporta numerosi vantaggi. In primo luogo, la riusabilità: un componente ben progettato può essere facilmente impiegato in diverse parti dell'applicazione o in progetti differenti, riducendo la duplicazione di codice e favorendo la coerenza visiva e funzionale. Inoltre, la modularità facilita la manutenzione: modificare o aggiornare una singola parte dell'interfaccia non influisce sul resto del sistema, migliorando la stabilità complessiva e semplificando l'attività di test.

Un ulteriore punto di forza risiede nel flusso unidirezionale dei dati, secondo cui le informazioni vengono trasmesse dai componenti genitore a quelli figli attraverso le props. Questo approccio rende il comportamento dell'applicazione più prevedibile e riduce la possibilità di effetti collaterali indesiderati, specialmente in applicazioni con un numero elevato di stati locali. In presenza di dati condivisi tra componenti distanti nella gerarchia, React mette a disposizione il Context API, che consente di evitare il cosiddetto prop drilling – il passaggio di props lungo catene di componenti intermedi – fornendo un meccanismo centralizzato e controllato per la distribuzione dei dati globali.

Tuttavia, l'architettura a componenti presenta anche alcuni limiti. In applicazioni

di grandi dimensioni, la frammentazione in troppi componenti può aumentare la complessità cognitiva del progetto, rendendo più difficile comprendere il flusso dei dati e le dipendenze tra i moduli. Inoltre, le variazioni di stato nei componenti di livello superiore possono propagarsi e causare ri-render a cascata, impattando le prestazioni. In questi casi, l'uso mirato di React.memo, useMemo e useCallback diventa fondamentale per isolare i componenti che non necessitano di aggiornamento e per mantenere l'interfaccia reattiva.

La gestione dello stato globale rappresenta un'altra sfida tipica: se tutto lo stato viene gestito esclusivamente tramite props, l'applicazione rischia di diventare rigida e difficile da estendere. Per questo motivo, la combinazione di stato locale, Context API e, quando necessario, librerie esterne di state management (come Redux o Zustand) consente di ottenere un equilibrio tra isolamento e condivisione dei dati.

In definitiva, React offre un modello architetturale fortemente modulare e scalabile, che incoraggia la costruzione di interfacce dinamiche attraverso componenti indipendenti e riutilizzabili. L'adozione consapevole dei suoi strumenti di ottimizzazione consente di realizzare applicazioni performanti e facilmente estendibili, mantenendo al tempo stesso chiarezza strutturale e coerenza progettuale.

In un contesto applicativo come quello della piattaforma per la valutazione linguistica, React consente di realizzare interfacce responsive, accessibili e facilmente internazionalizzabili. La struttura a componenti favorisce la separazione tra i moduli dedicati a operatori sanitari e famiglie, e l'adozione di hook come useState e useEffect rende possibile implementare funzionalità dinamiche quali l'autosalvataggio e l'aggiornamento in tempo reale delle risposte.

Vite

Vite è un build tool moderno e altamente performante progettato per offrire un'esperienza di sviluppo più veloce e fluida rispetto ai tradizionali strumenti di bundling come Webpack o Parcel. Il suo nome, derivato dal francese "vite" (che significa "veloce"), riflette la filosofia alla base del progetto: ridurre drasticamente i tempi di avvio e di ricompilazione durante lo sviluppo. Creato da Evan You, lo stesso autore di Vue.js, e mantenuto oggi come progetto open source indipendente, Vite è diventato uno degli strumenti di riferimento per applicazioni frontend basate su framework come React, Vue, Svelte o Preact [16].

Alla base del funzionamento di Vite vi è un approccio innovativo al processo di build e sviluppo locale. In ambiente di sviluppo, Vite non esegue un vero e proprio bundling iniziale del codice: invece, utilizza un dev server basato su moduli ECMAScript (ESM) che serve direttamente i file sorgente al browser. Grazie al supporto nativo di ESM nei browser moderni, il caricamento dei moduli avviene in modo dinamico e selettivo: viene eseguita e aggiornata solo la porzione di codice effettivamente necessaria. Ciò consente un tempo di avvio praticamente istantaneo,

indipendente dalle dimensioni complessive del progetto. Durante le modifiche, Vite sfrutta un meccanismo di Hot Module Replacement (HMR) estremamente efficiente, che aggiorna in tempo reale solo i moduli modificati senza ricaricare l'intera pagina. Questo garantisce un ciclo di sviluppo più reattivo e riduce significativamente il tempo di feedback per lo sviluppatore [17].

Nel momento in cui si passa alla fase di produzione, Vite cambia completamente modalità operativa e utilizza Rollup come motore di bundling. In questo modo, genera un output ottimizzato con tree shaking, code splitting e minimizzazione del codice JavaScript e CSS, garantendo dimensioni ridotte dei file finali e tempi di caricamento più rapidi in ambiente di deploy. La combinazione di un server di sviluppo ultrarapido e di un sistema di build solido e standardizzato rappresenta uno dei motivi principali per cui Vite è stato adottato in larga scala nelle applicazioni React moderne.

Uno dei punti di forza di Vite è la sua architettura modulare e configurabile, basata su un sistema di plugin compatibile con l'ecosistema Rollup. I plugin consentono di estendere il comportamento del tool, ad esempio per aggiungere il supporto a framework, preprocessori CSS o strumenti di analisi del bundle. Vite supporta nativamente TypeScript, JSX/TSX, PostCSS e CSS Modules, riducendo la necessità di configurazioni complesse. La configurazione è definita in un semplice file vite.config.ts o vite.config.js, dove possono essere personalizzati aspetti come alias di importazione, proxy verso API esterne, ottimizzazione delle dipendenze e parametri del server di sviluppo.

Nel contesto della piattaforma per la valutazione linguistica, Vite svolge un ruolo cruciale come strumento di build e ambiente di sviluppo del frontend React. Consente agli sviluppatori di lavorare con tempi di risposta immediati, facilitando la creazione di componenti interattivi e l'integrazione con le API del backend Express. Il supporto integrato per TypeScript permette di mantenere un codice fortemente tipizzato e ridurre gli errori in fase di compilazione, mentre l'integrazione nativa con Tailwind CSS semplifica la generazione dinamica degli stili e la gestione del design responsive. Inoltre, la possibilità di definire alias per i percorsi delle cartelle (@components, @hooks, ecc.) e di utilizzare variabili d'ambiente attraverso il file .env contribuisce a una struttura del progetto più ordinata e manutenibile.

Un ulteriore vantaggio di Vite risiede nel suo design trasparente e minimale: il tool non impone una particolare struttura o convenzione architetturale, ma fornisce un'infrastruttura leggera e coerente per la gestione del ciclo di vita del progetto. Questo approccio "non intrusivo" si adatta perfettamente a contesti accademici e professionali in cui la priorità è la sperimentazione e la rapidità di sviluppo, mantenendo al contempo standard industriali di efficienza e portabilità.

Infine, la documentazione ufficiale di Vite enfatizza l'obiettivo di "rendere il web development più semplice e immediato" attraverso un ecosistema basato su moduli moderni, un'interfaccia di configurazione minimale e un'integrazione naturale con

gli strumenti del frontend contemporaneo. In un progetto come quello descritto in questa tesi, in cui il frontend deve rimanere reattivo e leggero pur comunicando con un backend complesso, Vite si rivela una soluzione ideale per garantire velocità, modularità e stabilità lungo tutto il ciclo di sviluppo.

Tailwind CSS

Tailwind CSS è un framework CSS "utility-first" che si è affermato come una delle soluzioni più utilizzate per la progettazione di interfacce web moderne. A differenza dei framework tradizionali basati su componenti predefiniti (come Bootstrap o Materialize), Tailwind adotta un approccio più flessibile: invece di fornire set di componenti stilizzati, mette a disposizione una vasta collezione di classi atomiche che rappresentano singole proprietà CSS — come margini, colori, tipografia, layout, spaziature o ombre. Questo modello consente di costruire componenti e layout personalizzati direttamente nel markup, senza definire manualmente fogli di stile separati [1].

L'idea alla base di Tailwind è rendere il processo di sviluppo dell'interfaccia più rapido, prevedibile e scalabile. Il framework genera automaticamente classi standardizzate e coerenti, permettendo allo sviluppatore di combinare utility come blocchi Lego. Ad esempio, classi come flex, justify-between, p-4, text-gray-700 o rounded-xl descrivono in modo dichiarativo layout e stile, riducendo la necessità di scrivere CSS personalizzato. Questo approccio favorisce una maggiore coerenza visiva e limita la creazione di stili divergenti, uno dei problemi più comuni nei progetti di lunga durata [2].

Uno dei punti di forza più rilevanti è l'estrema configurabilità tramite il file tailwind.config.js. Questo file consente di definire palette di colori personalizzate, scale di spaziatura, breakpoint responsive e varianti avanzate, oltre a permettere l'estensione del tema con componenti riutilizzabili o classi personalizzate. Il sistema è pensato per adattarsi a qualunque design system, permettendo alle applicazioni di mantenere un'identità visiva consistente senza essere vincolate a uno stile predefinito. L'aggiunta di plugin ufficiali o di terze parti amplia ulteriormente le funzionalità del framework, ad esempio per gestire tipografia avanzata, forme, animazioni o accessibilità.

Tailwind è strettamente integrato con strumenti moderni come PostCSS e i build tool contemporanei. Nella pipeline di sviluppo, Tailwind analizza i file dell'applicazione per individuare le classi effettivamente utilizzate e applica una fase di purging per eliminare tutte quelle inutilizzate. Questo processo consente di mantenere il bundle CSS estremamente leggero in produzione, anche nei progetti con un elevato numero di componenti. In combinazione con strumenti come Vite, l'aggiornamento degli stili avviene quasi istantaneamente durante lo sviluppo,

grazie al supporto nativo per l'hot module replacement e per la ricompilazione incrementale [3].

Un altro elemento che ha contribuito alla popolarità di Tailwind è il suo eccellente supporto alla progettazione responsive. Il framework utilizza una sintassi intuitiva basata su prefissi (sm:, md:, lg:, xl:) che permettono di applicare stili specifici per ciascun breakpoint direttamente nella classe del markup. In questo modo, il comportamento responsive non richiede la scrittura di regole media query separate, riducendo la dispersione del codice e migliorando la leggibilità. Anche la gestione dei temi scuri (dark mode) è integrata nel sistema tramite varianti come dark: o strategie basate su classi CSS, permettendo una personalizzazione semplice e dichiarativa.

Tailwind presta particolare attenzione anche all'accessibilità: le sue utility incorporano best practice visive (spaziatura coerente, contrasti adeguati, uso corretto della tipografia) e possono essere facilmente combinate con attributi ARIA o con librerie specializzate per la gestione di componenti accessibili. Sebbene Tailwind non fornisca componenti ARIA-compliant pre-costruiti, la sua natura "non prescrittrice" consente di realizzarli in modo personalizzato, integrandosi bene con strumenti di design system e con framework che puntano all'accessibilità, come React in combinazione con librerie headless.

Dal punto di vista architetturale, l'adozione di un sistema utility-based riduce la dipendenza da fogli di stile monolitici e facilita la manutenzione del codice CSS in progetti su larga scala. Poiché ogni stile è dichiarato nel markup, diventa semplice individuare quali parti dell'interfaccia utilizzano specifiche proprietà grafiche, riducendo il fenomeno del "CSS morto" e semplificando il refactoring. Allo stesso tempo, la natura atomica delle classi permette di creare componenti ad alta coesione e facilmente riutilizzabili, aspetto fondamentale nelle applicazioni React con molti elementi interattivi.

In conclusione, Tailwind CSS rappresenta una soluzione moderna e performante per la progettazione di interfacce web, che si distingue per flessibilità, coerenza e velocità di sviluppo. La combinazione di classi utility, configurazione centralizzata, integrazione con strumenti di build contemporanei e supporto responsive lo rende particolarmente adatto a progetti modulari e dinamici, come la piattaforma descritta in questa tesi. Grazie a un ecosistema maturo e a una curva di apprendimento più rapida rispetto ai framework CSS tradizionali, Tailwind permette di costruire interfacce completamente personalizzate mantenendo un codice pulito, scalabile e semplice da mantenere nel lungo periodo.

Express JS

Express.js è uno dei framework web più diffusi nell'ecosistema Node.js e rappresenta oggi uno standard de facto per la realizzazione di API e applicazioni lato server.

Introdotto nel 2010 e mantenuto da una community molto ampia, Express si caratterizza per un approccio minimalista, non intrusivo e altamente estensibile. A differenza di framework più strutturati, Express non impone una particolare architettura applicativa, ma offre un insieme di primitive e astrazioni su cui gli sviluppatori possono costruire in libertà la propria organizzazione del codice. Questo equilibrio tra semplicità e flessibilità ha contribuito alla sua adozione in un'ampia varietà di contesti, dalle piccole API alle architetture enterprise [1].

Uno degli elementi centrali di Express è il sistema di routing, cioè il meccanismo che permette di definire quali funzioni debbano essere eseguite in risposta a specifiche richieste HTTP. Il framework fornisce metodi dedicati per ciascun verbo HTTP (app.get, app.post, app.patch, app.delete, ecc.), permettendo di associare a ogni rotta una o più funzioni middleware che implementano la logica di gestione. Il routing di Express è basato su pattern dichiarativi: percorsi statici, parametrici (ad esempio /users/:id) o annidati possono essere definiti in modo intuitivo, favorendo la modularità e la suddivisione del backend in file e router separati. Tale modello richiama la struttura REST tradizionale e si integra naturalmente con architetture basate su controller e servizi applicativi [2].

Un elemento distintivo del framework è il concetto di middleware, uno dei pilastri dell'architettura Express. Un middleware è una funzione che riceve l'oggetto della richiesta (req), della risposta (res) e un terzo parametro (next) che consente di delegare l'esecuzione al livello successivo della catena. L'uso di app.use() permette di applicare middleware a livello globale, mentre è possibile associare middleware specifici anche a singole rotte o a gruppi di rotte tramite gli oggetti Router(). Questo modello permette di separare responsabilità diverse in componenti autonomi, come la gestione dell'autenticazione, la validazione dei dati, il parsing del corpo della richiesta, la gestione degli errori o la registrazione dei log.

Un aspetto fondamentale del middleware è la loro composizione sequenziale: Express esegue i middleware nell'ordine in cui vengono dichiarati, con un flusso di controllo esplicito che attraversa ciascuna funzione fino alla generazione della risposta. La documentazione ufficiale sottolinea come questo modello favorisca l'estensibilità, poiché ogni middleware può trasformare la richiesta, interrompere il flusso o delegare ulteriori elaborazioni [1]. Questo paradigma è ideale per implementare pipeline come:

- parsing dei dati in ingresso
- validazioni basate su schema
- autorizzazione tramite token o sessioni
- gestione unificata degli errori
- logging strutturato delle richieste

Express integra inoltre alcuni middleware essenziali per la gestione dei formati più comuni. A partire dalla versione 4.16, il framework include nativamente express.json() e express.urlencoded(), che permettono di effettuare il parsing automatico del corpo delle richieste rispettivamente in formato JSON e URL-encoded. Questi middleware risultano fondamentali nella progettazione di API che accettano input strutturati o parametri di formulari, poiché garantiscono che i dati siano disponibili in req.body in modo semplice e coerente.

In contesti moderni, Express è spesso utilizzato come backend per applicazioni single-page (SPA) sviluppate con framework come React. Per questa ragione, è frequente l'uso di middleware dedicati alla gestione della Sicurezza (CORS) o di proxy locali che reindirizzano le richieste API provenienti dal frontend allo stesso dominio durante la fase di sviluppo. Sebbene Express non fornisca un middleware CORS integrato, la sua architettura modulare permette di includere facilmente pacchetti esterni come cors, raccomandato nella guida ufficiale per garantire compatibilità con applicazioni distribuite su origini diverse [3].

Un'altra componente essenziale nella progettazione di API è la gestione centralizzata degli errori, che Express supporta nativamente attraverso middleware specifici che accettano quattro parametri (err, req, res, next). Questo permette di intercettare eccezioni o fallimenti verificatisi in qualsiasi punto del flusso di middleware e di restituire risposte strutturate e uniformi, migliorando la robustezza dell'applicazione e semplificando l'identificazione dei problemi durante la fase di sviluppo e produzione.

Express si presta inoltre a essere utilizzato in combinazione con ORM e strumenti di accesso ai dati, integrandosi facilmente con Prisma, Sequelize o Mongoose. Il pattern più diffuso consiste nel delegare la logica applicativa a un livello di servizio separato, mentre i router di Express fungono da punto di ingresso che valida l'input e instrada la richiesta verso il servizio corretto. Questo approccio, ampiamente adottato nell'industria, permette di disaccoppiare il framework web dalla logica di dominio dell'applicazione, mantenendo il backend più flessibile e testabile.

Dal punto di vista progettuale, la leggerezza e l'assenza di vincoli strutturali forti hanno rappresentato uno dei principali motivi della diffusione di Express, soprattutto in applicazioni modulari o in contesti in cui si preferisce adottare una architettura esplicita e personalizzabile piuttosto che aderire rigidamente a un framework full-stack. La documentazione ufficiale invita gli sviluppatori a partire da una base minimale e ad arricchire progressivamente l'applicazione introducendo solo i middleware necessari, un approccio "opt-in" che riduce la complessità iniziale e facilita la manutenzione a lungo termine.

In sintesi, Express costituisce un framework maturo, stabile e versatile, capace di adattarsi a progetti di diversa scala grazie al suo modello basato sui middleware e alla sua filosofia minimalista. Le funzionalità cardine – routing dichiarativo, catena di middleware, parsing delle richieste, gestione centralizzata degli errori e ampia

estendibilità – lo rendono una scelta ideale per progettare API REST e sistemi server-side in Node.js, come nel caso della piattaforma descritta in questa tesi.

Prisma ORM

Prisma ORM è un Object-Relational Mapper moderno per ambienti Node.js e TypeScript, progettato per migliorare significativamente l'esperienza dello sviluppatore nel gestire la persistenza dei dati. Il cuore di Prisma è costituito dal file schema.prisma, che funge da unico punto di configurazione del modello dati: qui si definiscono la sorgente dati (datasource), il generatore del client (generator), e i modelli che rappresentano le tabelle/contenitori nel database.

```
datasource db {
  provider = "postgresql"
  url      = env("DATABASE_URL")
}

generator client {
  provider = "prisma-client-js"
}

model User {
  id    Int     @id @default(autoincrement())
  email String  @unique
  name  String?
  posts Post[]
}
```

Si tratta di una rappresentazione dichiarativa che viene poi trasformata, tramite migrazioni automatiche o manuali, in una struttura relazionale nel database. Una volta generato, il prisma client fornisce un'interfaccia tipizzata per eseguire query: ad esempio prisma.user.findMany() restituisce oggetti di tipo correlato al modello User, sicuri relativamente al tipo, con auto-completamento nel codice TypeScript. Questo riduce notevolmente gli errori a compile time e migliora la qualità delle interazioni con il database.

Una delle caratteristiche distintive di Prisma è il sistema di migrazione: definendo o modificando i modelli nel file schema.prisma, è possibile generare automaticamente file di migrazione SQL che evolvono la struttura del database in modo coerente. Tale meccanismo semplifica l'allineamento tra modello applicativo e struttura fisica del database, favorendo una manutenzione più agevole. In contesti in cui si lavora in team o in ambienti di produzione, questa funzionalità consente di tracciare l'evoluzione del modello dati, versionare le modifiche e garantire che tutti gli ambienti (sviluppo, test, produzione) condividano la stessa struttura. Nel

contesto di una piattaforma backend modulare, come quella progettata per questa tesi, Prisma svolge il ruolo di strato di persistenza che agisce tra il repository layer e il database relazionale. Grazie alla sua generazione del client tipato, i servizi business-logic possono invocare operazioni CRUD, join, filtri e relazioni senza incorrere in query SQL manuali, riducendo così la complessità del codice e il margine di errore.

Un aspetto spesso trascurato ma fondamentale nella gestione di un database durante le prime fasi di sviluppo è la possibilità di popolare automaticamente il sistema con un insieme di dati iniziali. Prisma supporta nativamente questo meccanismo attraverso il file di seed, una funzionalità che permette di definire script per creare utenti, record o configurazioni di base necessari per avviare l'applicazione. Il seed è utile sia per il lavoro locale dello sviluppatore, sia per ambienti di testing o staging, in cui è importante disporre di un set di dati coerente e riproducibile. La presenza di un file di seed è particolarmente vantaggiosa in applicazioni che richiedono la creazione di utenti amministratori, ruoli, permessi, template predefiniti o altre entità indispensabili per il funzionamento dell'applicazione. Nel caso di piattaforme che gestiscono dati strutturati — come questionari, lingue disponibili, profili utente o impostazioni della dashboard — un seed consente di automatizzare la configurazione iniziale, evitare errori legati a inserimenti manuali e garantire consistenza tra gli ambienti.

Data la natura della piattaforma — con entità multiple (operatori, famiglie, sottomissioni, risposte, note) e relazioni tra loro — scegliere Prisma consente di avere un modello dati chiaro, tipato e coerente con il backend in TypeScript. La generazione automatica del client tipato fornisce un grande vantaggio in termini di manutenibilità e sicurezza dei tipi, mentre l'uso delle migrazioni rende l'evoluzione del database più controllabile. In sintesi, Prisma ORM si configura come la soluzione di persistenza moderna e robusta per il progetto: offre l'equilibrio tra astrazione e controllo, produttività e correttezza strutturale, qualità essenziali per lo sviluppo di un sistema scalabile e mantenibile.

PostreSQL

PostgreSQL è un sistema di gestione di basi di dati relazionali open-source tra i più diffusi e affidabili nel panorama moderno. La sua architettura è pienamente conforme agli standard SQL e integra funzionalità avanzate che lo rendono particolarmente adatto a sistemi complessi, multi-utente e orientati alla consistenza. Per queste ragioni viene adottato in numerosi contesti sanitari, amministrativi e enterprise.

Nel caso della piattaforma descritta in questa tesi, PostgreSQL rappresenta una scelta ottimale principalmente per tre motivi: integrità dei dati, modellazione relazionale e robustezza nelle operazioni concorrenti.

Poiché la web app gestisce entità strettamente collegate fra loro—operatori sanitari, questionari, traduzioni, compilazioni, risposte, note cliniche—è essenziale disporre di un database che supporti nativamente relazioni uno-a-molti e molti-a-uno, garantendo vincoli di integrità tramite chiavi esterne, vincoli UNIQUE e NOT NULL. PostgreSQL eccelle in questo tipo di modellazione ed evita alla radice inconsistenze logiche nelle relazioni tra le tabelle, un aspetto delicato quando si conservano dati clinici o para-clinici.

Un altro punto di forza è la gestione delle transazioni ACID e del controllo della concorrenza tramite MVCC (Multiversion Concurrency Control). Questo meccanismo permette a più utenti di leggere e scrivere dati simultaneamente senza blocchi inutili, garantendo coerenza anche in momenti in cui più famiglie compilano questionari mentre gli operatori consultano o filtrano le risposte. Rispetto a database più semplici come MySQL in configurazione standard, PostgreSQL ha un sistema di concorrenza più avanzato e garantisce livelli di isolamento robusti senza compromessi sulle prestazioni.

PostgreSQL si distingue anche per l'affidabilità e la maturità del suo query planner, che genera piani di esecuzione ottimizzati sulla base di statistiche aggiornate. Per una web app che esegue frequenti interrogazioni filtrate—ad esempio recuperare compilazioni "in progress", ricercare per codice fiscale o estrarre tutte le risposte relative a un questionario—questo significa tempi di risposta stabili e prevedibili.

Infine, rispetto ad altre soluzioni SQL, PostgreSQL offre una maggiore estendibilità: supporta tipi di dato avanzati, indici specializzati, funzioni definite dall'utente e modalità operative ibridi (relazionali + JSONB). Queste estensioni, pur non indispensabili per la versione attuale della piattaforma, rendono PostgreSQL una scelta lungimirante in vista di possibili evoluzioni future, come log narrativi, metadati complessi o storicizzazioni avanzate delle revisioni dei questionari.

3.4.3 Modello dati e schema logico

Il modello dati della piattaforma è di tipo relazionale e organizza le informazioni in un insieme di entità strettamente collegate tra loro. Le tabelle principali sono:

- Operator
- Template
- Submission
- Answer
- OperatorNote
- FeedbackReport

Ciascuna entità è stata progettata per riflettere un'unità logica del dominio applicativo: gli operatori sanitari, i questionari, le compilazioni effettuate dalle famiglie, le singole risposte e le note/feedback associati.

Operator

L'entità Operator rappresenta gli utenti professionali del sistema, ovvero gli operatori sanitari che accedono alla dashboard, gestiscono i questionari e consultano le compilazioni. Ogni operatore è identificato da un identificativo univoco *operator_id* di tipo UUID, ha un indirizzo email univoco, un hash di password e un ruolo (ad esempio "operator" o "admin"). Sono inoltre tracciate le informazioni anagrafiche di base (nome completo) e alcuni flag applicativi, come l'obbligo di cambio password al primo accesso. La relazione con le altre entità è data principalmente dal legame uno-a-molti con OperatorNote, che rappresenta le note inserite dagli operatori sulle compilazioni.

Template

L'entità Template modella i questionari configurabili dalla piattaforma. Ogni template è identificato da un *template_id*, un nome univoco e un campo descrittivo opzionale. Un elemento centrale del modello è il campo *structure_definition*, di tipo JSON, che contiene la definizione strutturata del questionario (sezioni, domande, opzioni di risposta, lingue, ecc.). Il template include anche un array di *available_languages*, che elenca le lingue in cui il questionario è disponibile, e i timestamp di creazione e aggiornamento. Un flag booleano, *is_active*, permette di distinguere i questionari utilizzabili da logica di onDelete configurabile)quelli storicizzati o disattivati. La cancellazione di un template comporta l'eliminazione di tutti i feedback associati.

Submission

L'entità Submission rappresenta una compilazione di un questionario da parte di una famiglia. È identificata da submission_id ed è collegata a:

- un template (template_id),
- il codice fiscale del minore o della famiglia (fiscal_code),
- la lingua in cui è stata effettuata la compilazione (language_used).

La submission mantiene uno stato (status, ad esempio "InProgress" o "Completed") e traccia l'ultimo step compilato (current_step_identifier), utile per implementare il meccanismo di avvio/ripresa del questionario. Sono inoltre presenti

campi temporali e un campo metadata di tipo JSON per eventuali informazioni aggiuntive.

Answer

L'entità Answer contiene le singole risposte alle domande del questionario. Ogni record è identificato da un answer_id numerico autoincrementale ed è collegato a una submission tramite submission_id. Il campo question_identifier identifica in modo univoco la domanda all'interno del template, mentre answer_value è di tipo JSON e consente di rappresentare in modo flessibile diversi tipi di risposta (testuale, numerica, scelta multipla, ecc.). Il campo saved_at registra il momento in cui la risposta è stata salvata, a supporto dell'autosalvataggio e della ricostruzione della cronologia.

A livello di vincoli, è definita una unicità composta (submission_id, question_identifier), che garantisce l'esistenza di al massimo una risposta per ciascuna domanda all'interno di una data submission

OperatorNote

OperatorNote rappresenta le note inserite dagli operatori in relazione a una compilazione o a specifiche domande. Ogni nota è identificata da un note_id (UUID) ed è collegata sia a una submission sia a un operatore . In modo opzionale, può essere associata anche a una singola domanda tramite question_identifier, permettendo di agganciare un commento a una risposta specifica del questionario. La cancellazione di una submission comporta la cancellazione a cascata delle note associate, mentre la cancellazione o disattivazione di un operatore può comportare l'aggiornamento del riferimento, mantenendo comunque la nota nel sistema.

FeedbackReport

FeedbackReport modella i feedback o le segnalazioni raccolte sulla qualità del questionario o della compilazione. Ogni record contiene un feedback_id , la possibile associazione a una submission, il riferimento al template, un identificatore di domanda opzionale , il testo del feedback e un campo reporter_metadata di tipo JSON per memorizzare metadati sul segnalante (ad esempio canale, priorità, ruolo). Sono presenti anche lo status del feedback ("New", "Investigating", "Resolved", ecc.) e il timestamp di invio

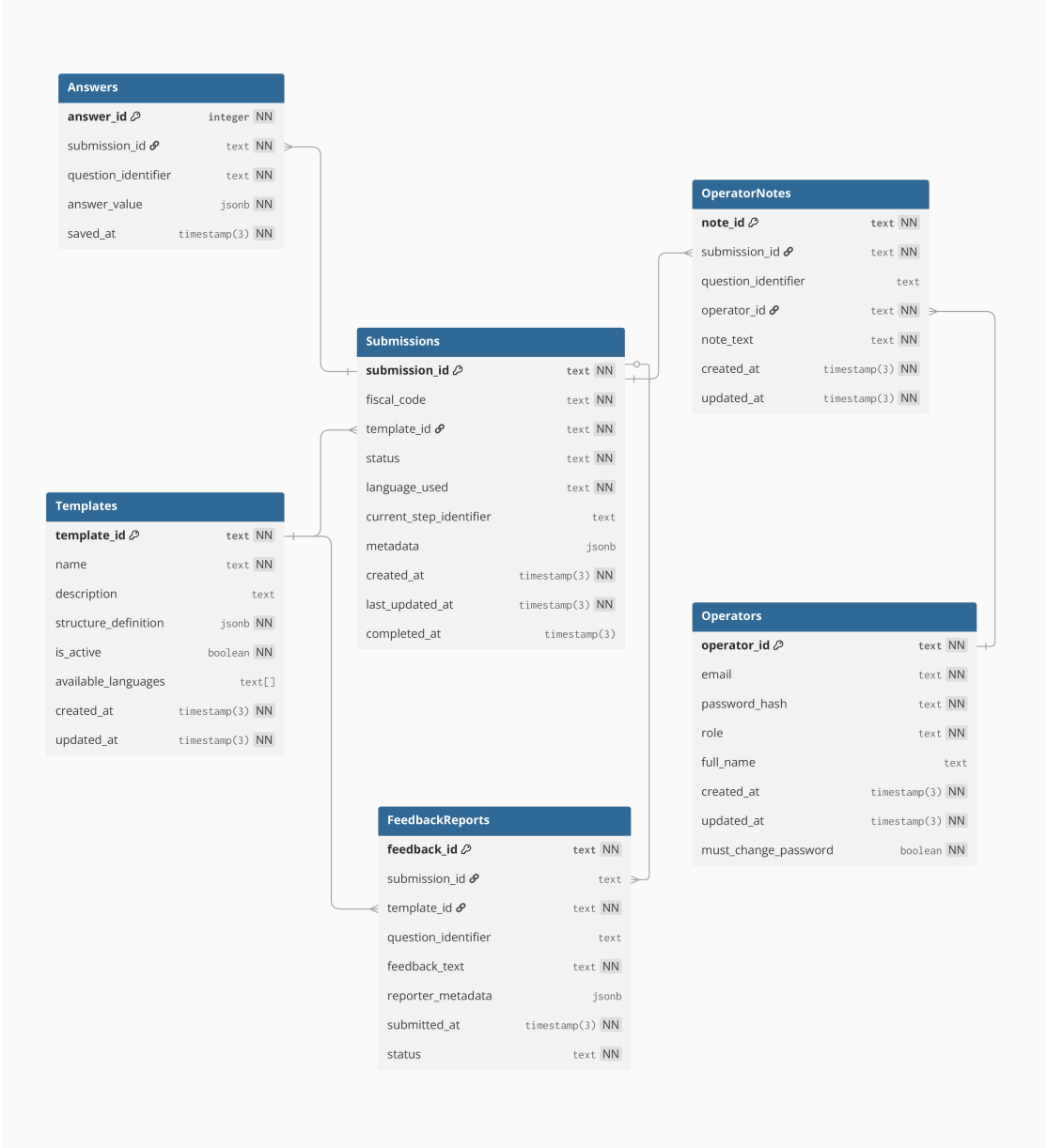


Figura 3.5: Schema database

3.4.4 Pattern architetturali: Controller-Service-Repository

Il pattern *Controller-Service-Repository* è una variante della classica architettura a livelli, progettata per promuovere la separazione delle responsabilità, migliorare la manutenibilità e rendere il codice più testabile. In questo schema, l'interazione dell'applicazione si articola in tre layer distinti:

- Controller: è il primo punto di contatto con le richieste esterne (tipicamente HTTP). Il controller riceve l'input, lo valida in parte, ed invoca uno o più servizi per soddisfare la richiesta. In questo modo, il controller si occupa della comunicazione con il mondo esterno piuttosto che della logica applicativa profonda.
- Service: layer in cui risiede la logica di business dell'applicazione, orchestrando le operazioni richieste dal controller, gestendo transazioni, invocando repository, combinando più operazioni, e applicando regole di dominio. Questo layer non dovrebbe occuparsi di accesso diretto al database o di logica di presentazione.
- Repository: strato dedicato all'accesso ai dati: contiene le query, le operazioni CRUD, l'interazione con l'ORM o il database. Il repository astrae il meccanismo di persistenza, offrendo al servizio una interfaccia chiara e indipendente dalla tecnologia di storage.

Confronto con altri pattern

Esaminare pattern alternativi aiuta a comprendere la scelta. Alcuni modelli simili sono:

- MVC (Model-View-Controller): pattern originario per interfacce utente, in cui il Controller gestisce input dell'utente, la View la presentazione e il Model i dati e le regole. Tuttavia, per applicazioni server-side che gestiscono logica complessa e persistenza, l'MVC può risultare troppo "piatto" e rischiare di concentrare troppa logica nel controller o nel modello.
- Service-Repository (senza controller separato): in alcuni contesti si uniscono Controller e Service, ma questo abbassa la separazione di responsabilità.
- Hexagonal Architecture: un'architettura più generale che enfatizza l'isolamento del core del dominio da infrastrutture esterne (come UI, DB, servizi esterni). In confronto, il pattern Controller-Service-Repository può essere visto come una declinazione più pragmatica e "livellata" di tale architettura, adatta a sistemi con vincoli moderati di complessità.

Nel contesto della piattaforma sviluppata, il pattern Controller-Service-Repository è stato scelto perché offre una struttura chiara e modulare, capace di separare in modo netto la gestione delle richieste HTTP, la logica di business e l'accesso ai dati. Questa suddivisione permette di mantenere il codice più leggibile e facilmente estendibile, riducendo il rischio che componenti diversi si mescolino tra loro. Allo stesso tempo, il pattern favorisce la testabilità: i servizi possono essere verificati in isolamento e i repository possono essere sostituiti o simulati senza modificare

il resto dell'applicazione. La scelta di Controller-Service-Repository rappresenta quindi un equilibrio efficace tra semplicità, scalabilità futura e manutenibilità del codice.

3.5 Meccanismi chiave del sistema

La piattaforma integra una serie di meccanismi progettati per garantire un'esperienza di compilazione affidabile, coerente e adatta al contesto d'uso. In questa sezione vengono descritti i principi alla base della gestione della compilazione dei questionari, della selezione linguistica e dell'autenticazione degli utenti. L'obiettivo è illustrare le scelte architetturali che guidano il funzionamento del sistema, senza entrare nei dettagli implementativi che verranno approfonditi nel Capitolo 4.

3.5.1 Avvio, ripresa e autosalvataggio della compilazione

La compilazione di un questionario è stata progettata come un processo progressivo, capace di adattarsi alle esigenze delle famiglie, che possono completare il modulo in più sessioni e con tempi variabili. Per evitare la perdita dei dati e garantire la continuità del flusso, il sistema introduce tre elementi concettuali: l'identificazione univoca della compilazione, la possibilità di ripresa e l'autosalvataggio incrementale.

Al momento dell'accesso, la famiglia indica il proprio codice fiscale e seleziona il questionario da compilare. Il sistema utilizza queste informazioni per determinare se esiste già una compilazione associata e non completata. In tal caso, l'utente viene ricondotto allo stato corrente della sessione, in base allo step precedentemente raggiunto e alle risposte già salvate. Se non esiste una compilazione attiva, viene creata una nuova istanza in stato "in corso", collegata al template scelto e caratterizzata da un identificatore univoco.

La ripresa del questionario è resa possibile dal fatto che ogni compilazione mantiene traccia sia del suo stato, sia dell'ultimo punto raggiunto. Le risposte non vengono accumulate al termine del processo, ma registrate man mano tramite aggiornamenti puntuali. Questo approccio consente di preservare i progressi anche in caso di chiusura accidentale della pagina, interruzioni prolungate o perdita temporanea della connessione. Per garantire l'affidabilità del salvataggio intermedio, la piattaforma adotta un modello incrementale e idempotente. Ogni modifica rilevante, come l'inserimento di una risposta o il passaggio allo step successivo, genera un aggiornamento della compilazione e della risposta corrispondente.

3.5.2 Gestione della lingua

La piattaforma prevede una gestione articolata della lingua, che coinvolge sia il contenuto dei questionari sia gli elementi dell'interfaccia utente. Le due dimensioni

sono complementari ma distinte, poiché rispondono a esigenze diverse: la prima riguarda la disponibilità del questionario in più lingue, la seconda la personalizzazione dell'esperienza utente.

Dal punto di vista dei contenuti, ogni questionario può essere reso disponibile in una o più lingue attraverso la definizione delle relative versioni nel template. Al momento dell'accesso, la famiglia seleziona la lingua in cui desidera compilare il questionario tra quelle offerte per quel template specifico. La scelta viene fissata al momento dell'avvio della compilazione, garantendo coerenza tra le domande e le risposte per tutta la durata del processo. Qualora l'utente tenti successivamente di accedere al questionario in una lingua diversa, il sistema impedisce la modifica, in modo da preservare l'uniformità dei dati raccolti.

Parallelamente alla lingua del contenuto, la piattaforma gestisce la lingua dell'interfaccia attraverso un sistema di traduzione centralizzato. I testi statici dell'applicazione — come etichette, messaggi di errore, pulsanti e indicazioni operative — non sono codificati direttamente nei componenti, ma vengono recuperati da un dizionario organizzato per chiavi e lingue. Un contesto globale nel frontend mantiene la lingua corrente e rende disponibili le traduzioni ai vari componenti, permettendo all'interfaccia di adattarsi dinamicamente alla lingua selezionata.

3.5.3 Modello di accesso e autenticazione

La piattaforma adotta due meccanismi distinti per l'accesso, riflettendo le diverse finalità e i diversi livelli di responsabilità previsti per famiglie e operatori sanitari. È importante sottolineare che, per le famiglie, il sistema non implementa un vero e proprio login, poiché non esiste un profilo utente, né una sessione autenticata nel senso tradizionale. L'accesso è invece strettamente legato al processo di compilazione di un singolo questionario.

Per le famiglie, l'ingresso nel sistema avviene attraverso un link associato al template del questionario, il cui identificativo univoco è presente nell'URI. Dopo aver aperto il link, viene richiesto esclusivamente il codice fiscale, utile a determinare se esiste una compilazione già avviata oppure se è necessario crearne una nuova. Il codice fiscale non funge da credenziale, ma da chiave di contesto che permette al backend di individuare la submission associata a quell'utente e a quel questionario. Una volta verificata la presenza di una compilazione in corso o la necessità di generarne una nuova, il sistema consente alla famiglia di procedere con la compilazione senza introdurre sessioni persistenti o token di autenticazione. Questo approccio riduce al minimo la complessità dell'esperienza utente e risponde alla necessità di eliminare la registrazione formale delle famiglie, mantenendo comunque un livello adeguato di coerenza logica nella gestione delle compilazioni. Gli operatori sanitari, al contrario, devono poter accedere a un'area riservata, consultare le compilazioni ricevute, gestire i questionari, inserire note e svolgere

attività operative. Per questa ragione il loro accesso richiede un meccanismo di autenticazione vero e proprio. Il sistema prevede account nominativi con credenziali e utilizza un modello basato su JSON Web Token per la gestione delle sessioni. Al momento dell'accesso, un token firmato viene rilasciato e inviato dal client in ogni richiesta successiva verso le API riservate. Il backend valida il token, identifica l'operatore e applica i controlli di autorizzazione necessari.

3.6 Progettazione UI e requisiti di accessibilità

La progettazione dell'interfaccia utente per le famiglie è stata guidata non solo da obiettivi estetici e di semplicità d'uso, ma soprattutto dai requisiti di accessibilità definiti durante l'analisi dei requisiti *(RNF3 e RNF8)*. In questa sezione vengono presentati i mockup preliminari realizzati in Figma e le principali scelte progettuali che hanno orientato la realizzazione di un'interfaccia adatta a famiglie con livelli differenti di competenza digitale, utilizzabile prevalentemente da smartphone e accessibile anche mediante screen reader e sintesi vocale. Particolare attenzione è stata dedicata all'uso di etichette ARIA, alla localizzazione degli annunci vocali, al supporto alle Web Speech API per la lettura assistita e alla scelta di colori ad alto contrasto finalizzati a migliorare la leggibilità.

3.6.1 Mockup e design preliminare

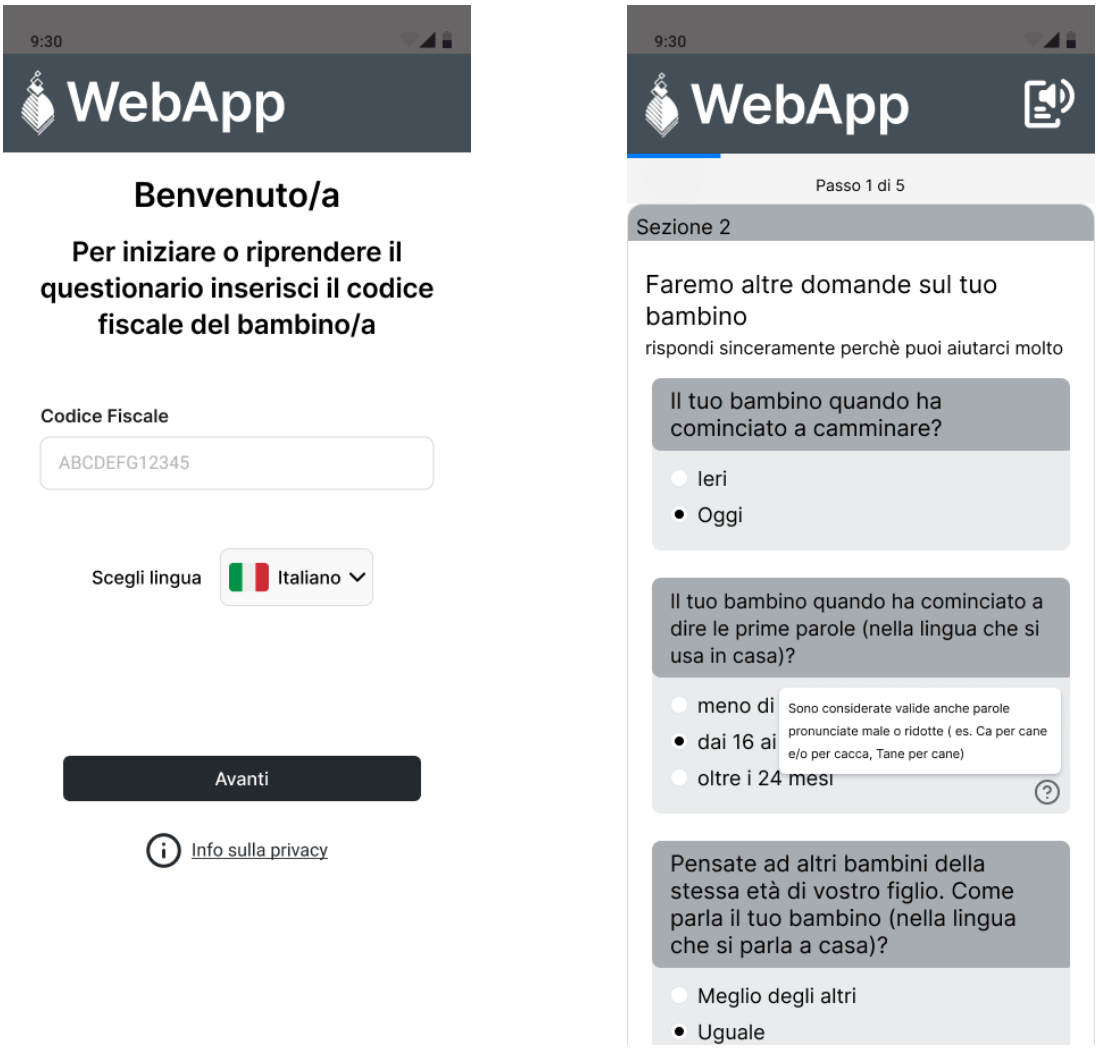
Nella fase di progettazione è stato realizzato un mockup dell'interfaccia dedicata alle famiglie, con lo scopo di definire il flusso di accesso e le principali componenti della schermata di compilazione. Poiché la compilazione avviene prevalentemente da smartphone, l'interfaccia è stata sviluppata secondo un approccio mobile-first, mantenendo comunque una piena responsività anche per tablet e desktop. Il design preliminare includeva anche un pulsante globale per l'attivazione della sintesi vocale (TTS), posizionato nella parte superiore della schermata. Tuttavia, durante l'evoluzione del progetto, questa soluzione è stata rivista a favore di un approccio più intuitivo, che prevede un pulsante di riproduzione direttamente accanto a ogni singola domanda, così da associare l'attivazione della lettura al contenuto specifico.

3.6.2 Supporto agli screen reader

In fase di design si è previsto l'utilizzo di attributi ARIA per descrivere semanticamente i componenti UI (pulsanti, campi di input, stati della risposta, avanzamento della compilazione). Un aspetto particolarmente rilevante riguarda la localizzazione degli annunci vocali: le etichette ARIA devono riflettere la lingua scelta dall'utente per la compilazione, evitando incongruenze tra lingua dell'interfaccia e lingua del lettore di schermo.

3.6.3 Lettura assistita tramite Text-to-Speech

Accanto al supporto per gli screen reader, la piattaforma integra un meccanismo di lettura assistita tramite la Web Speech API, progettato per aiutare utenti con difficoltà di lettura o che preferiscono un supporto audio durante la compilazione. La presenza di un sistema TTS rende inoltre l'applicazione più inclusiva non solo per utenti con disabilità, ma anche per persone con difficoltà temporanee, scarsa familiarità con l'italiano, o situazioni in cui la lettura del testo non è agevole.



(a) Schermata di accesso **(b)** Pagina di compilazione del questionario

Figura 3.6: Mockup preliminari dell'interfaccia utente dedicata alle famiglie

Capitolo 4

Implementazione

Il presente capitolo descrive nel dettaglio la fase di implementazione della piattaforma, illustrando come le scelte architetturali e i modelli progettuali definiti nel Capitolo 3 siano stati tradotti in componenti software concreti. L'obiettivo è mostrare in che modo la progettazione sia stata effettivamente realizzata all'interno del sistema, evidenziando le soluzioni adottate, le tecnologie utilizzate e il ruolo dei diversi moduli nello sviluppo complessivo dell'applicazione.

4.1 Struttura generale del progetto

L'implementazione è organizzata all'interno di una *monorepo*, scelta progettuale che consente di integrare in un unico spazio di lavoro i diversi moduli software che compongono la piattaforma. Tale approccio permette di mantenere un elevato livello di coerenza tipologica, semplificare la gestione delle dipendenze e favorire l'evoluzione congiunta dei componenti, rendendo il processo di sviluppo più lineare e controllabile.

La monorepo ospita quattro moduli principali:

- `server/`, che contiene il backend sviluppato in Node.js ed Express, responsabile dell'esposizione delle API REST, della logica applicativa e dell'accesso al database tramite Prisma;
- `client/`, il frontend dedicato agli operatori sanitari, realizzato in React e TypeScript, che include la dashboard, l'editor dei questionari e gli strumenti di gestione e consultazione delle compilazioni;
- `family-client/`, l'interfaccia di compilazione destinata alle famiglie, progettata per un utilizzo ottimale su smartphone e caratterizzata da un flusso lineare, multilingue e orientato all'accessibilità;

- `shared/`, che raccoglie gli schemi Zod, i tipi TypeScript, i DTO e le risorse comuni utilizzate trasversalmente dagli altri moduli, e che costituisce la base semantica condivisa dell'intera applicazione.

L'organizzazione interna della monorepo si avvale dei *workspace di npm*, che permettono una gestione centralizzata e non ridondante delle dipendenze, e di una serie di configurazioni condivise per TypeScript, Prisma, ESLint, Prettier e per i sistemi di build basati su Vite. Questa struttura modulare facilita lo sviluppo parallelo, la manutenzione del codice e la propagazione delle modifiche, garantendo che l'intera codebase evolva in modo coerente e controllato.

Il diagramma in figura 4.1 illustra la struttura delle directory e i rapporti tra i vari moduli, evidenziando la suddivisione logica e le dipendenze reciproche che caratterizzano l'architettura della piattaforma.

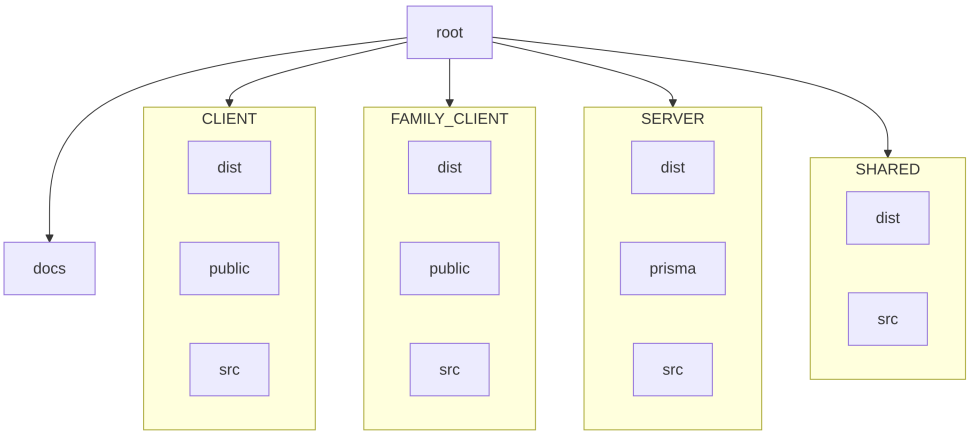


Figura 4.1: Struttura cartelle

4.2 Modulo shared: modelli, validazione e risorse comuni

Il modulo `shared/` costituisce uno dei componenti architetturalmente più significativi dell'intera piattaforma. Esso raccoglie le definizioni degli schemi dati, i modelli di validazione, i tipi TypeScript e le strutture ausiliarie che vengono utilizzati in modo trasversale dal backend e dai due frontend. L'obiettivo principale di questo modulo è garantire un punto di definizione univoco per tutta la semantica del sistema, preservando la coerenza tra le varie parti dell'applicazione e riducendo in modo significativo il rischio di disallineamenti tipici degli sviluppi multi-modulo.

La scelta di centralizzare la definizione dei modelli riflette un approccio *schema-driven*, nel quale la struttura e la forma dei dati sono definite in un solo luogo e successivamente importate dai diversi componenti della piattaforma. Questo principio permette di mantenere una corrispondenza esatta tra ciò che il backend si aspetta di ricevere e ciò che i frontend inviano o interpretano, costruendo così una pipeline dati omogenea, robusta e più semplice da mantenere nel tempo.

4.2.1 Definizione degli schemi con Zod

La definizione formale delle strutture dati all'interno del modulo shared è affidata alla libreria `Zod`, uno strumento che consente di descrivere oggetti, array, unioni e tipi complessi attraverso un sistema dichiarativo e pienamente tipizzato. A differenza di altri validatori, Zod è stato progettato per integrarsi in modo nativo con TypeScript, permettendo di ottenere non solo una validazione runtime, ma anche un modello statico perfettamente coerente con gli schemi definiti. Questa caratteristica lo rende particolarmente adatto in contesti full-stack in cui più componenti devono condividere le stesse strutture semantiche.

Il funzionamento di Zod si basa sulla definizione di oggetti schema che descrivono in modo rigoroso la forma dei dati ammessi. Ogni campo può essere annotato con vincoli specifici, con regole di optionalità, con predicati personalizzati e con trasformazioni applicate prima della validazione finale. Nel caso della piattaforma, questo approccio si rivela particolarmente utile poiché i questionari includono elementi eterogenei (testi, scelte multiple, numeri, date, testi di aiuto, etichette multilingua), che richiedono un controllo fine sulla correttezza formale delle compilazioni.

La validazione viene quindi applicata tanto nel backend, per garantire la conformità dei payload in ingresso, quanto nei frontend, dove svolge un ruolo preventivo intercettando errori prima dell'invio al server. In questo modo l'intero ciclo di vita dei dati è governato da un unico insieme di regole, mantenute e aggiornate all'interno del modulo shared.

4.2.2 Inferenza dei tipi TypeScript

Uno dei principali vantaggi offerti da Zod è la possibilità di inferire automaticamente, a partire dallo schema dichiarato, il corrispondente tipo TypeScript. Questo meccanismo elimina completamente la necessità di definire manualmente interfacce duplicate o strutture parallele, una situazione che nei sistemi distribuiti porta facilmente a incoerenze semantiche tra le parti.

Nel modulo shared, ogni schema è dunque accompagnato dalla derivazione esplicita del proprio tipo TypeScript. Ad esempio, lo schema di una domanda del questionario può essere definito in Zod nel modo seguente:

```
export const QuestionSchema = z.object({
  id: z.string(),
  type: z.enum(["text", "choice", "number"]),
  label: z.record(LanguageCodeSchema, z.string()),
  helpText: z.record(LanguageCodeSchema, z.string()).optional(),
  required: z.boolean().optional()
});

export type Question = z.infer<typeof QuestionSchema>;
```

Listing 4.1: Schema Question

In questo modo, ogni componente dell'applicazione che utilizza il tipo Question fa riferimento esattamente alla stessa definizione, senza margini di ambiguità. Qualunque modifica apportata allo schema viene immediatamente riflessa nei tipi generati, garantendo un allineamento automatico dell'intera codebase. Questo meccanismo si rivela particolarmente prezioso nelle interazioni tra client e server, dove la stabilità del contratto dati è essenziale per evitare errori difficili da diagnosticare durante la fase di esecuzione

4.2.3 Data Transfer Object e contratti di comunicazione

Oltre agli schemi dei modelli interni, il modulo shared contiene una serie di *Data Transfer Object (DTO)*, ossia strutture dati che definiscono in maniera esplicita il formato delle richieste e delle risposte che circolano tra frontend e backend. Un DTO non rappresenta il modello persistito nel database, ma la forma esatta con cui i dati devono essere serializzati durante la comunicazione attraverso le API. La distinzione è importante: mentre gli schemi applicativi modellano la struttura logica delle entità, i DTO definiscono il contratto che vincola client e server, assicurando che entrambi concordino sul formato dei dati scambiati.

Nel contesto della piattaforma, i DTO risultano fondamentali per descrivere formalmente operazioni delicate come il salvataggio progressivo delle risposte, l'avvio o la ripresa di una compilazione, l'invio di un feedback o la generazione dei dati per l'esportazione. Ognuna di queste operazioni richiede di definire con precisione quali

campi devono essere presenti, quali siano opzionali, quali trasformazioni vadano applicate e quali vincoli siano ammessi.

Il backend utilizza tali definizioni tramite un middleware di validazione che applica gli schemi Zod associati ai DTO, rifiutando automaticamente ogni richiesta che non sia conforme. I frontend, simmetricamente, ne sfruttano i tipi inferiti per costruire richieste sempre coerenti e per garantire che l'interazione con le API avvenga nel rispetto del contratto definito. Questo contribuisce a ridurre significativamente il rischio di errori runtime e rafforza la robustezza della pipeline di comunicazione.

Il ruolo degli schemi e dei DTO all'interno dell'architettura complessiva è sintetizzato nella Figura 4.2 che mostra come il modulo shared agisca da sorgente unificata di verità per frontend e backend.

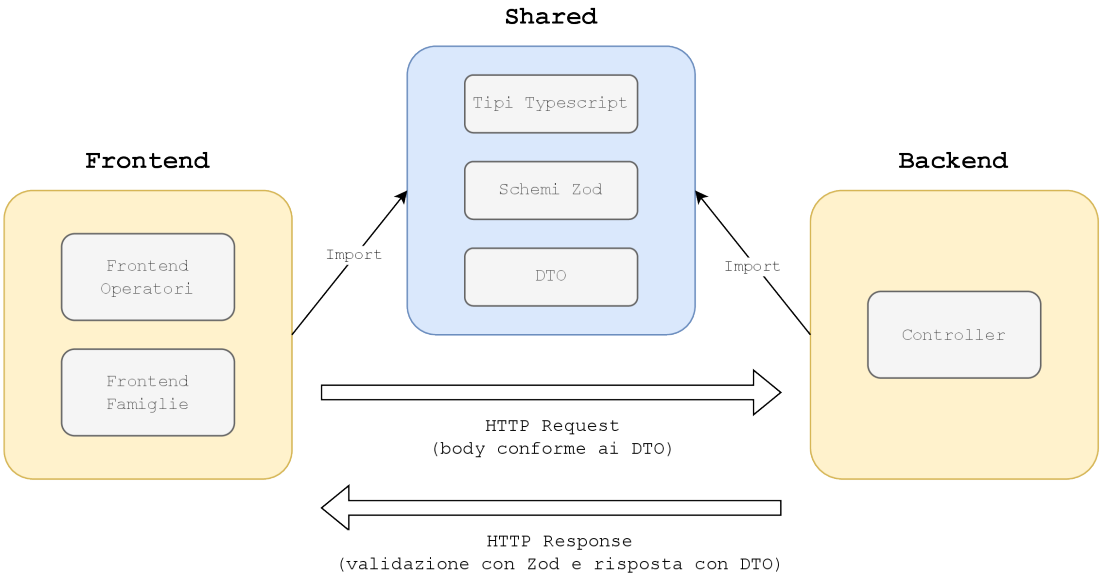


Figura 4.2: DTO e validazione con Zod

4.2.4 Gestione della localizzazione e catalogo delle lingue

Il modulo shared include anche un sottosistema dedicato alla gestione della localizzazione, che fornisce un punto centralizzato per la definizione delle lingue supportate e per l'accesso ai contenuti multilingue. Questo componente svolge un ruolo essenziale nell'assicurare coerenza tra i due frontend e il backend, evitando la duplicazione di logiche e garantendo un comportamento uniforme nella selezione e nel fallback dei testi.

La gestione delle lingue è articolata attorno a un catalogo linguistico, una struttura che descrive per ciascuna lingua il codice identificativo, il nome in inglese

e nella lingua nativa, eventuali metadati aggiuntivi e un indicatore che ne segnala l'appartenenza al set di lingue predefinite. Tale catalogo permette di mantenere in un unico punto l'elenco completo delle lingue disponibili e costituisce la base per le funzioni che ne regolano l'utilizzo all'interno dell'applicazione. Questo approccio consente, ad esempio, di presentare in modo uniforme la lista delle lingue selezionabili, di garantire etichette corrette nelle interfacce e di semplificare l'aggiunta di nuove lingue in futuro.

Il sistema di localizzazione gestisce anche i testi tradotti attraverso una struttura dati che associa a ciascuna lingua la relativa stringa. A partire da questa rappresentazione, funzioni dedicate determinano il testo da mostrare all'utente in base alla lingua scelta e applicano automaticamente regole di fallback nel caso in cui la traduzione specifica non sia disponibile. La presenza di un meccanismo centralizzato consente ai due frontend di ottenere un comportamento coerente, indipendentemente dal contesto in cui il testo viene utilizzato (etichette di interfaccia, domande del questionario, testi di aiuto, messaggi di sistema).

Oltre alla risoluzione dei testi, il modulo offre utilità per assicurare che gli oggetti contenenti traduzioni siano allineati alle lingue attualmente abilitate, aggiungendo le voci mancanti o rimuovendo quelle non più necessarie

4.3 Frontend dedicato agli operatori sanitari

Il frontend dedicato agli operatori sanitari costituisce la parte più articolata della piattaforma, poiché concentra le funzionalità necessarie alla creazione e modifica dei questionari, al monitoraggio delle compilazioni e alla consultazione delle risposte. Questa sezione descrive l'organizzazione interna dell'applicazione, le principali strutture architetturali e i meccanismi implementativi che ne regolano il funzionamento. Le sottosezioni successive approfondiscono gli aspetti relativi alla navigazione, ai componenti principali e alle interazioni con il backend.

4.3.1 Architettura

React Router L'applicazione per gli operatori è organizzata come un insieme di componenti React strutturati secondo un modello modulare, basato sulla separazione tra pagine, componenti riutilizzabili e servizi di comunicazione con il backend. La navigazione interna è gestita attraverso *React Router*, che permette di definire percorsi distinti per la dashboard, l'editor dei questionari, la gestione dei template, le compilazioni e le aree amministrative. Ogni pagina corrisponde a un componente principale che coordina i sotto-componenti necessari e richiama le funzioni API dedicate.

La comunicazione con il backend è implementata tramite un livello di servizi che incapsula tutte le chiamate HTTP. Questi servizi utilizzano i tipi e i DTO

definiti nel modulo shared per garantire che i dati scambiati siano coerenti con gli schemi della piattaforma. La gestione delle risposte, degli errori e delle eventuali trasformazioni dei dati è centralizzata, così da ridurre la duplicazione di logica nei componenti.

React Context Per lo stato globale dell'applicazione — in particolare autenticazione, informazioni sull'utente e gestione dei token — è stato implementato un *Context React* dedicato, che espone una serie di metodi per eseguire il login, verificare la sessione e gestire la sua scadenza. La scelta di utilizzare un Context consente a tutti i componenti dell'applicazione di accedere allo stato condiviso in maniera semplice e senza propagare esplicitamente proprietà attraverso la gerarchia dei componenti.

Tailwind CSS Lo stile dell'interfaccia è realizzato con *Tailwind CSS*, utilizzato in forma utility-first: ciò permette di definire layout e componenti visivi direttamente all'interno del markup dei componenti React, mantenendo lo stile dell'applicazione consistente e facilmente estendibile. Le strutture ricorrenti, come card, pulsanti e container, sono state organizzate in componenti UI riutilizzabili, così da facilitare l'omogeneità dell'interfaccia e la manutenzione del codice.

4.3.2 Dashboard principale e sistema di navigazione

La dashboard costituisce la pagina iniziale del frontend operatori e funge da punto di accesso principale alle funzionalità dell'applicazione. Essa organizza in modo sintetico l'insieme delle operazioni disponibili—consultazione delle compilazioni, gestione dei template, visualizzazione dei feedback e operazioni amministrative—offrendo una panoramica immediata dello stato del sistema e un accesso diretto alle sezioni più rilevanti.

L'interfaccia è costruita attraverso una combinazione di componenti dedicati e layout modulari. La struttura principale è gestita da `AppLayout` , che comprende la barra laterale di navigazione `Sidebar` e l'area di contenuto centrale. La sidebar contiene i collegamenti alle diverse rotte protette dell'applicazione, corrispondenti alle funzionalità operative effettivamente implementate nel routing: dashboard generale, lista delle compilazioni, gestione dei template, pagina dei feedback, pagina di registrazione utenti (solo per amministratori) e cambio password. Tale organizzazione deriva direttamente dalle rotte definite all'interno del componente `App.tsx`, dove ciascuna pagina è mappata a un percorso univoco e protetto da `ProtectedRoute`.

Il layout laterale è affiancato da una serie di componenti UI, come `StatsCard` e `TemplateCard`, che permettono di rappresentare indicatori sintetici, informazioni aggregate o elementi di elenco in modo coerente. Questi componenti, realizzati

con Tailwind CSS, favoriscono una presentazione visivamente uniforme e flessibile, mantenendo leggibilità e semplicità d'uso. La dashboard utilizza inoltre componenti di utilità come `LoadingSpinnger` e `Toast`, impiegati rispettivamente per segnalare lo stato di caricamento e per mostrare notifiche di esito.

All'interno della pagina `DashboardPage` vengono caricate e presentate le informazioni principali su cui l'operatore basa la propria attività: l'elenco delle compilazioni disponibili, filtrabile tramite il componente `Filters`, e il collegamento rapido alle funzionalità più utilizzate. La presenza della barra laterale garantisce una navigazione costante e stabile, mantenendo il contesto operativo dell'utente indipendentemente dalla sezione visitata.

La Figura 4.3 mostra la schermata principale della dashboard, evidenziando gli elementi organizzativi principali: la sidebar a sinistra, l'area centrale dedicata alla tabella delle compilazioni, filtri di ricerca e delle statistiche sulle compilazioni (sezione in alto).

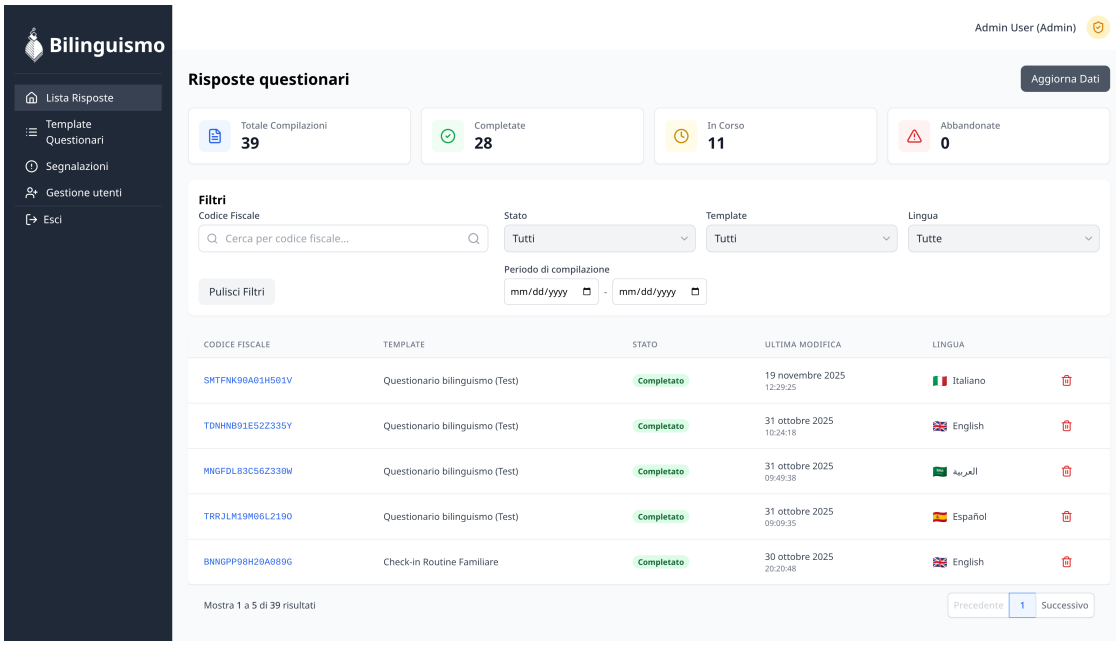


Figura 4.3: Schermata principale dashboard

4.3.3 Visualizzazione della compilazione

La visualizzazione del dettaglio di una compilazione consente agli operatori sanitari di analizzare in modo strutturato le risposte fornite dalle famiglie e di inserire eventuali annotazioni cliniche associate a specifiche domande. Questa funzionalità è accessibile dalla dashboard attraverso la selezione di una submission, che reindirizza alla rotta /submissions/:id gestita dal componente `SubmissionViewPage`.

Submission All'apertura della pagina, l'applicazione effettua il caricamento dei dati tramite il servizio `submissionApi`, recuperando sia le informazioni principali della compilazione (template utilizzato, stato, timestamp, codice fiscale), sia l'insieme delle risposte organizzate per sezione. La struttura interna della pagina è modulata attraverso due componenti dedicati: `SubmissionSectionView` e `SubmissionQuestionView`. Il primo si occupa di presentare il titolo della sezione e di iterare sulle domande in essa contenute, mentre il secondo gestisce il rendering del singolo item, mostrando testo della domanda, tipo di risposta e contenuto fornito dalla famiglia.

Data la natura variabile dei questionari e la possibilità che essi contengano domande di tipologie differenti, la componente SubmissionQuestionView effettua una renderizzazione condizionale basata sui tipi definiti negli schemi Zod del modulo shared, garantendo una rappresentazione coerente con la struttura del template originale. Le risposte vengono formattate in modo leggibile, con adattamenti specifici per tipologie come selezioni multiple, risposte numeriche o testo libero.

Note Una funzionalità rilevante della pagina riguarda l'inserimento delle *note*, che consentono all'operatore di aggiungere osservazioni contestuali legate alla singola domanda. Queste note vengono gestite tramite il servizio `notesApi` e sono rappresentate direttamente sotto ciascuna risposta tramite un campo di testo dedicato o un'area espandibile, a seconda della quantità di contenuto presente. L'operatore può aggiungere, eliminare la nota, con salvataggio immediato e notifica tramite il sistema di toast.

Il layout della pagina è progettato per favorire la leggibilità: le sezioni vengono presentate in sequenza verticale, ciascuna con una chiara separazione visiva, mentre le domande sono indentate e accompagnate da indicatori grafici che ne facilitano l'interpretazione. Le figure 4.4 e 4.5 mostrano un esempio della vista dettagliata di una sezione, illustrando la relazione tra il testo della domanda, la risposta fornita dalla famiglia e l'area destinata alle annotazioni.

Esportazione Oltre alla consultazione, dalla pagina è possibile eseguire azioni aggiuntive correlate alla compilazione, come l'esportazione del questionario in formato Excel, accessibile tramite il pulsante `Esporta`. Questa operazione consente agli operatori di ottenere una copia strutturata delle risposte per ulteriori analisi o per allegarla alla documentazione clinica. L'esportazione è implementata traimit il servizio `utilsApi` che invia una richiesta HTTP alla rotta dedicata del backend. Quest'ultimo elabora la richiesta e genera dinamicamente un file Excel a partire dalla submission. Nel frontend, la risposta del server viene gestita come un blob di tipo `application/vnd.openxmlformats-officedocument.spreadsheetml.sheet`, che viene convertito in un URL temporaneo e scaricato automaticamente dal browser.

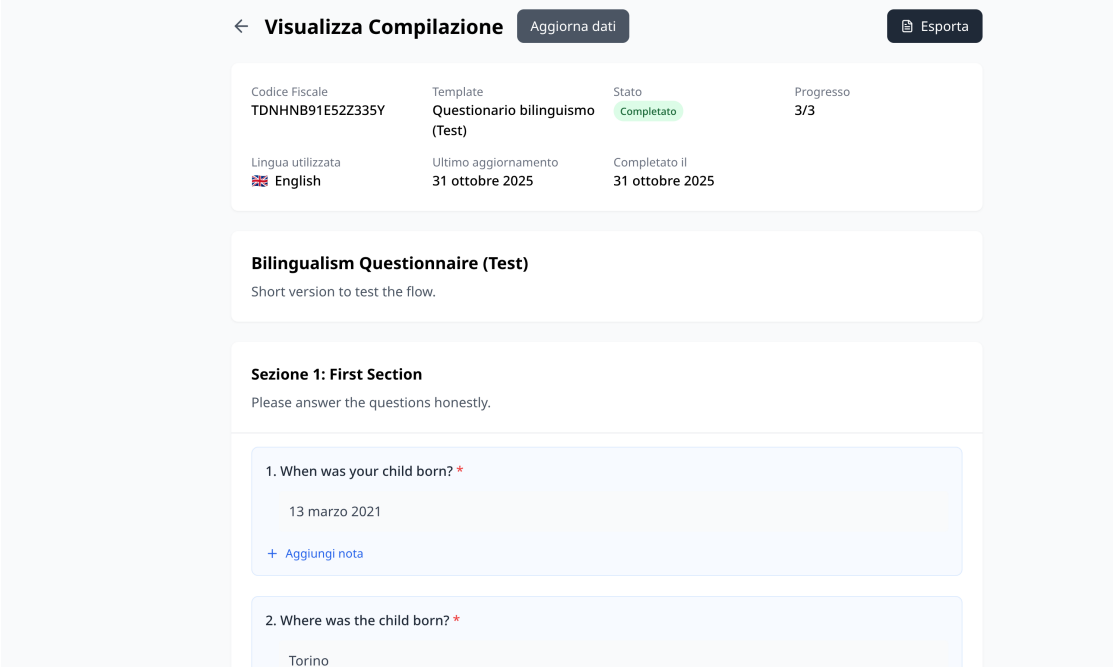


Figura 4.4: Dettagli singola compilazione

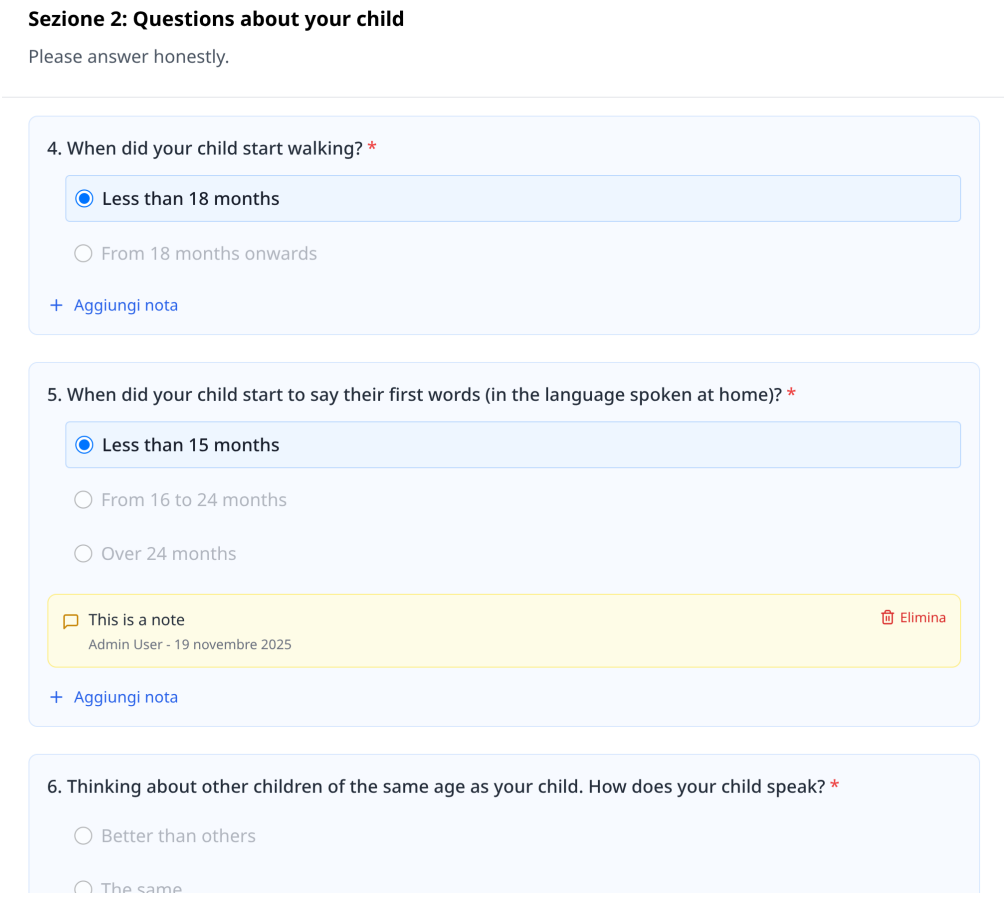


Figura 4.5: Risposte e note

4.3.4 Editor dei template per i questionari

L'editor visuale dei questionari è uno dei componenti centrali del frontend operatori e consente la creazione e la modifica dei template utilizzati dal frontend delle famiglie. L'interfaccia è raggiungibile attraverso le rotte `/templates/editor` e `/templates/editor/:id`, gestite dalla pagina QuestionnaireEditorPage, la quale carica i dati del template tramite il servizio `templateApi` e inizializza lo stato interno dell'editor con la struttura delle sezioni e delle domande definite nel backend.

L'editor è organizzato come una composizione di componenti specializzati, ciascuno dei quali gestisce una parte specifica della struttura del questionario. Il componente principale, `QuestionnaireEditorPage`, mantiene lo stato complessivo del template e si occupa di:

- recuperare il template esistente (se l'ID è presente),
- propagare i dati ai sotto-componenti,

- gestire il salvataggio tramite le API
- coordinare le modifiche effettuate nelle sezioni e nelle singole domande.

L'editor permette di operare sulle sezioni del template tramite il componente `SectionEditor`. Ogni sezione presenta un titolo multilingua, ottenuto dalla struttura `LocalizedText` definita nel modulo shared, ed è composta da un insieme ordinato di domande. Il SectionEditor espone funzionalità per:

- modificare il titolo della sezione nelle lingue attive,
- aggiungere una nuova domanda,
- eliminare una sezione,
- riordinare le domande tramite controlli dedicati (quando presenti nel layout).

Per la modifica delle singole domande viene utilizzato il componente `QuestionEditor`, che gestisce i campi principali dell'oggetto Question definito negli schemi Zod tra cui:

- il testo e il tipo della domanda,
- l'eventuale testo di aiuto,
- le opzioni, nel caso di domande a scelta multipla,
- la proprietà required, quando prevista.

La selezione del tipo di domanda avviene tramite `QuestionTypeSelector`, un componente dedicato che permette all'operatore di scegliere tra i tipi disponibili (testo, scelta, numerico, ecc.) come definiti nello schema condiviso. Il cambiamento del tipo aggiorna la struttura interna della domanda, mantenendo la validazione coerente rispetto allo schema Zod. Poiché ogni testo del questionario è definito come LocalizedText, l'editor integra anche un componente `LanguageSelector`, che consente di decidere quali lingue siano attive per il template in modifica. L'aggiunta o la rimozione di lingue avviene mediante funzioni del modulo shared, come `ensureLocalizedTextLanguages`, che garantiscono che ogni domanda e sezione dispongano delle chiavi necessarie, anche quando una nuova lingua viene attivata. Questo meccanismo centralizzato evita inconsistenze nella struttura interna del template e permette all'operatore di completare agevolmente tutte le traduzioni. Ogni modifica all'interno dell'editor aggiorna lo stato del template in memoria, e successivamente il salvataggio viene eseguito tramite un pulsante dedicato, che invia l'intera struttura al backend. L'editor non utilizza salvataggi

incrementali, ma invia il payload completo del template, sfruttando la validazione lato server basata sugli schemi Zod presenti in shared. Eventuali errori vengono restituiti al frontend e gestiti tramite il sistema di toast. Dal punto di vista visivo, l'interfaccia dell'editor presenta una struttura verticale, in cui sezioni e domande sono chiaramente separate e dotate di controlli localizzati. In Figura 4.6 è mostrata una sezione dell'editor dove è possibile selezionare la lingua del template che si sta creando o modificando, titolo e descrizione. Mentre la Figura 4.7 illustra il componente per scegliere la tipologia della domanda e la Figura 4.8 mostra invece l'editing di una domanda a risposta multipla.

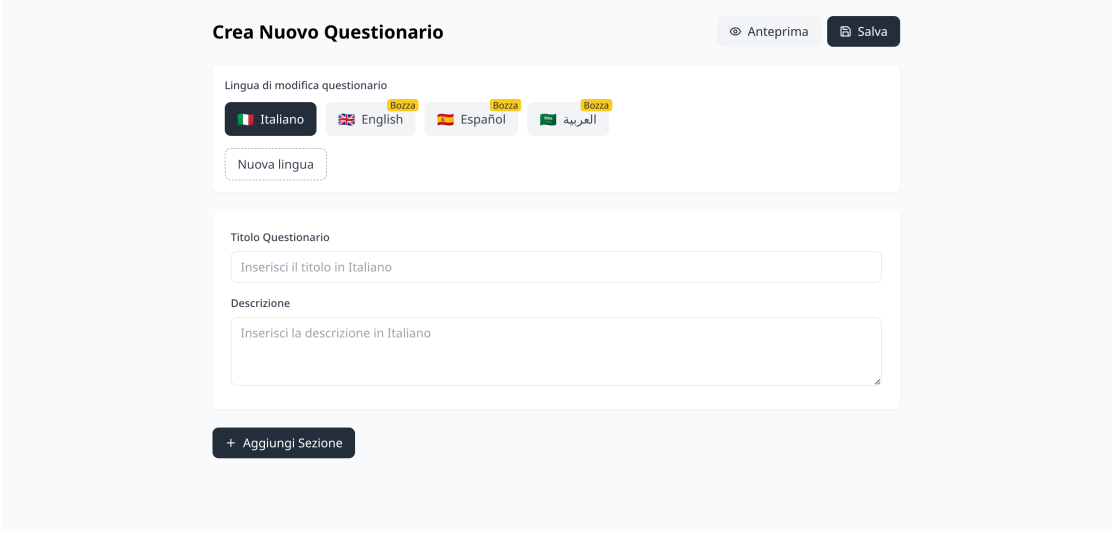


Figura 4.6: Creazione nuovo questionario

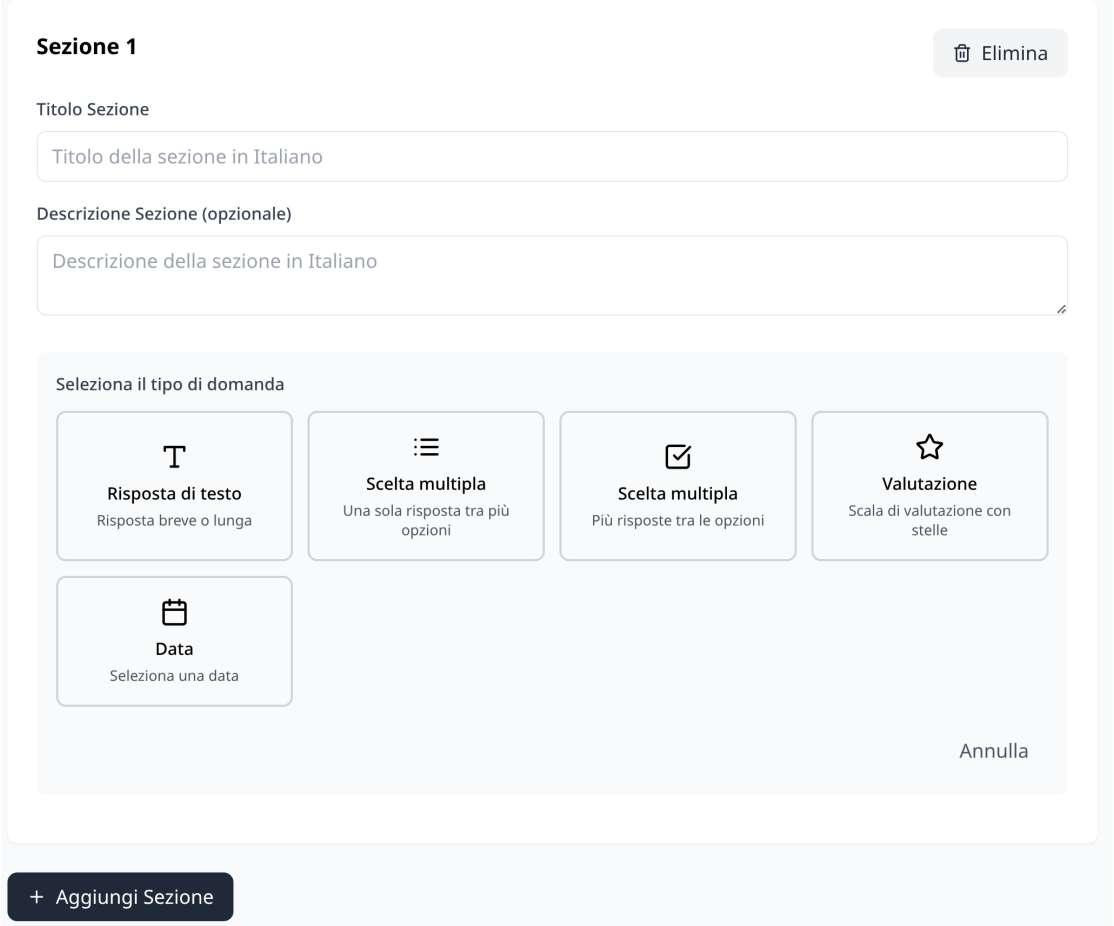


Figura 4.7: Selezione del tipo domanda

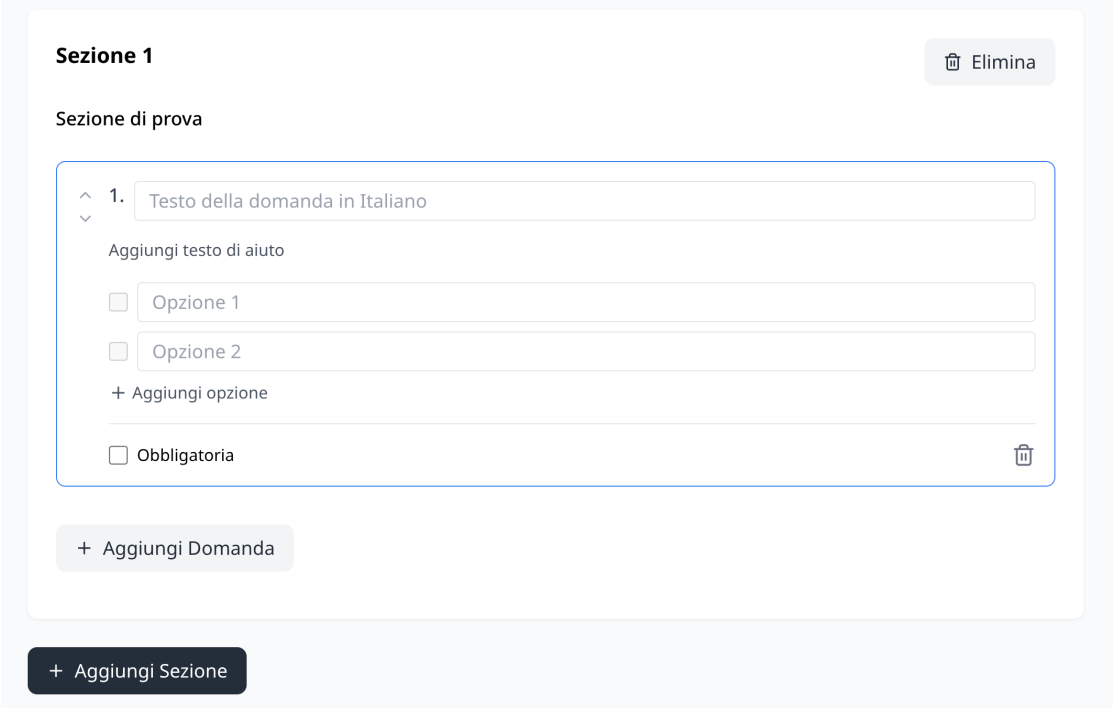


Figura 4.8: Editing domanda risposta multipla

4.3.5 Gestione dei feedback

La gestione dei feedback rappresenta uno strumento supplementare che consente agli operatori di ricevere segnalazioni, commenti o richieste di chiarimento da parte delle famiglie durante la compilazione del questionario. Questa funzionalità è accessibile tramite la rotta `/feedback`, gestita dal componente `FeedbackPage`, e permette agli operatori di consultare i messaggi ricevuti, applicare filtri e analizzare il contesto in cui il feedback è stato generato.

All'avvio della pagina, il frontend richiama il servizio `feedbackApi`, il quale interroga l'endpoint dedicato del backend per ottenere l'elenco dei feedback completi di metadati: stato della segnalazione *(Nuovo, In esame, Risolto)*, riferimento alla submission e al template associato, identificativo interno della domanda e testo del messaggio inserito dall'utente. Il backend elabora questi dati tramite il repository `feedback.repository.ts`, che costruisce una query arricchita con il nome del template, permettendo così all'interfaccia operatore di presentare un quadro più comprensibile di ogni segnalazione.

All'interno dell'interfaccia, i feedback vengono mostrati in forma tabellare o mediante card, con la possibilità di applicare filtri dinamici. Il componente `Filters` consente, ad esempio, di filtrare per stato o per template, mentre per la ricerca per domanda il sistema utilizza l'identificativo della domanda stessa

`question_identifier`. Quest'ultima modalità, pur essendo funzionale, risulta meno intuitiva, poiché richiede di conoscere o interpretare l'identificatore interno anziché visualizzare il testo completo della domanda. Tale limite è stato evidenziato come criticità durante le attività di validazione e rappresenta un punto di miglioramento previsto nelle revisioni future, in cui si prevede di sostituire il filtro basato sull'identificativo con un filtro testuale direttamente collegato al contenuto della domanda.

La pagina consente anche di accedere al dettaglio di ogni feedback, attraverso una modale che mostra il testo originale e i metadati correlati. L'operatore può modificare lo stato della segnalazione, permettendo una gestione strutturata delle comunicazioni ricevute. Le azioni effettuate vengono propagate al backend attraverso `feedbackApi`, che invia aggiornamenti utilizzando i DTO definiti nel modulo shared, garantendo la coerenza della struttura dati lungo l'intera pipeline.

In Figura 4.9 è riportato un esempio dell'interfaccia di consultazione dei feedback, con i principali elementi dell'interazione: elenco delle segnalazioni, filtri disponibili e stato di avanzamento delle attività di revisione. Tale rappresentazione evidenzia la funzione di raccordo svolta da questa sezione, che collega informazioni provenienti dalle compilazioni alle attività operative dell'operatore sanitario.

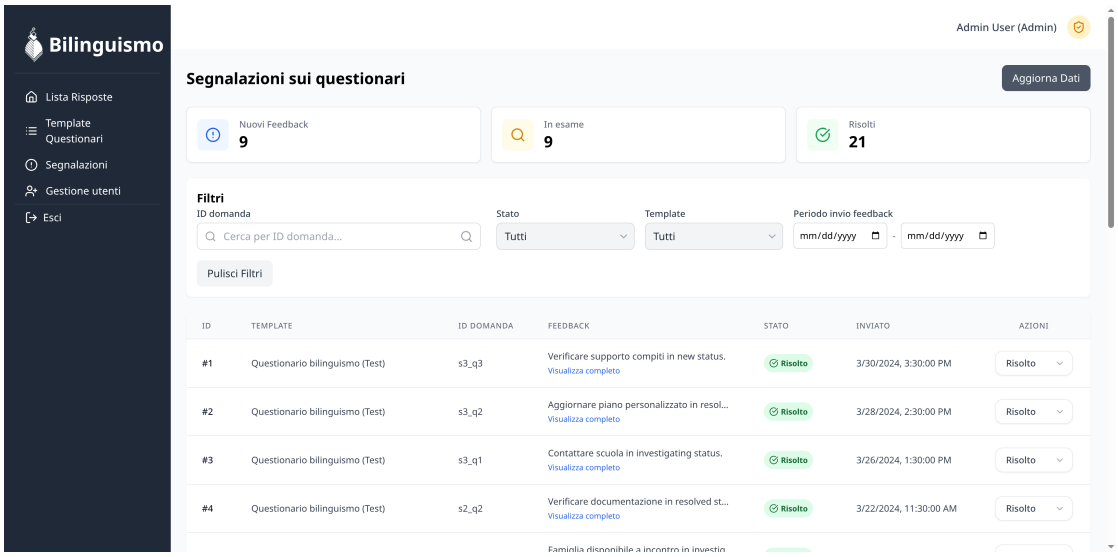


Figura 4.9: Pagina dei feedback

4.3.6 Gestione dell'autenticazione

L'accesso all'area operatori è regolato da un meccanismo di autenticazione basato su JSON Web Token (JWT), gestito interamente sul frontend tramite un contesto dedicato e un componente di protezione delle rotte. L'obiettivo è mantenere la logica di sessione in un unico punto, semplificando il controllo degli accessi e l'applicazione delle regole di autorizzazione ai diversi percorsi dell'applicazione.

Il nucleo della gestione dell'autenticazione è il contesto `AuthContext`. Questo componente mantiene lo stato dell'utente autenticato , il token JWT corrente e un indicatore di caricamento utilizzato durante le fasi di inizializzazione o di login. Al montaggio dell'applicazione, il contesto tenta di recuperare un eventuale token salvato in precedenza nello localStorage del browser; se presente, il token viene decodificato mediante la libreria `jwt-decode` e, in caso di validità, lo stato viene inizializzato di conseguenza. In questo modo l'utente non è costretto a effettuare il login a ogni ricaricamento della pagina, purché il token non sia scaduto.

Le operazioni di login e logout sono esposte dal contesto sotto forma di funzioni che i componenti di pagina possono richiamare. La pagina di accesso `(LoginPage)` raccoglie le credenziali dell'operatore e invia i dati al backend tramite il servizio `authApi`. In caso di successo, il token restituito viene memorizzato in `localStorage` e passato al contesto, che ne estrae le informazioni rilevanti (identificativo utente, ruolo, e un flag per cambio password al primo accesso). L'eventuale presenza di errori viene gestita tramite il `ErrorContext`, che visualizza messaggi all'utente tramite un sistema di notifiche. Il logout, al contrario, cancella il token dallo storage locale, azzera lo stato del contesto e reindirizza l'utente verso la pagina di login; il pulsante di uscita `(LogoutButton)` non fa altro che richiamare questa funzione centrale.

Il controllo degli accessi alle varie sezioni dell'applicazione è affidato al componente `ProtectedRoute`, che incapsula la logica di verifica dell'autenticazione e, quando necessario, dei permessi. Ogni rotta protetta in `App.tsx` è definita wrappando il componente di pagina all'interno di ProtectedRoute. Quest'ultimo legge lo stato del contesto di autenticazione e, se la sessione non è ancora stata inizializzata, mostra una schermata di caricamento; se l'utente non è autenticato, effettua un redirect verso /login, preservando nel proprio stato di navigazione la rotta di origine per consentire un eventuale ritorno dopo il login. Quando viene specificato l'elenco di ruoli richiesti, il componente confronta il ruolo dell'utente con quelli ammessi e, in caso di mancata corrispondenza, blocca l'accesso alla pagina.

Un aspetto particolare riguarda la gestione del flag `must_change_password`, utilizzato per imporre il cambio della password al primo accesso o dopo un reset effettuato da un amministratore. In questo caso, il ProtectedRoute può essere configurato con l'opzione `allowIfMustChange`, che consente l'accesso alla sola pagina /change-password anche quando il flag indica che l'utente deve aggiornare le

proprie credenziali. Per tutte le altre pagine, il componente intercetta la condizione e reindirizza automaticamente alla vista di cambio password, impedendo l'utilizzo dell'applicazione fino al completamento dell'operazione. Una volta cambiata la password, il backend restituisce un nuovo token senza il flag impostato, e il contesto aggiorna lo stato dell'utente di conseguenza.

Nel complesso, la combinazione di AuthContext, ProtectedRoute, pagina di login e pagina di cambio password realizza un sistema di autenticazione e autorizzazione pienamente integrato nel frontend, in grado di controllare l'accesso alle funzionalità sensibili (come la gestione dei questionari, delle compilazioni, dei feedback e degli utenti) e di riflettere correttamente le decisioni di sicurezza implementate nel backend.

4.3.7 Registrazione degli utenti e reset password

La piattaforma prevede un'area dedicata alla gestione degli utenti, accessibile esclusivamente agli operatori con ruolo di amministratore. Questa funzionalità è esposta attraverso la rotta protetta `/operators/register`, che verifica il ruolo dell'utente tramite il componente ProtectedRoute prima di consentire l'accesso alla pagina. All'interno della vista, implementata in `RegisterUserPage`, sono disponibili due operazioni principali: la registrazione di un nuovo operatore e il reset della password di un operatore esistente.

Il modulo di registrazione consente all'amministratore di inserire i dati principali dell'utente (nome, email e ruolo) e inviare la richiesta al backend mediante il servizio `authApi`. L'interfaccia gestisce in modo uniforme conferme ed eventuali errori tramite il sistema centralizzato di toast, così da garantire un feedback immediato. Una struttura simile è utilizzata per il pannello dedicato al reset della password, che richiede l'inserimento dell'email dell'operatore e mostra, in caso di successo, la nuova password temporanea generata dal server.

La pagina è integrata nel layout generale del frontend operatori e mantiene coerenza visiva con le altre viste amministrative. La Figura 4.10 illustra un esempio dell'interfaccia, mostrando i due moduli principali e l'organizzazione complessiva della pagina. In questa fase l'attenzione rimane concentrata sugli aspetti operativi della gestione degli utenti, mentre la logica interna di registrazione, validazione dei ruoli e aggiornamento delle credenziali sarà approfondita nel capitolo dedicato al backend.

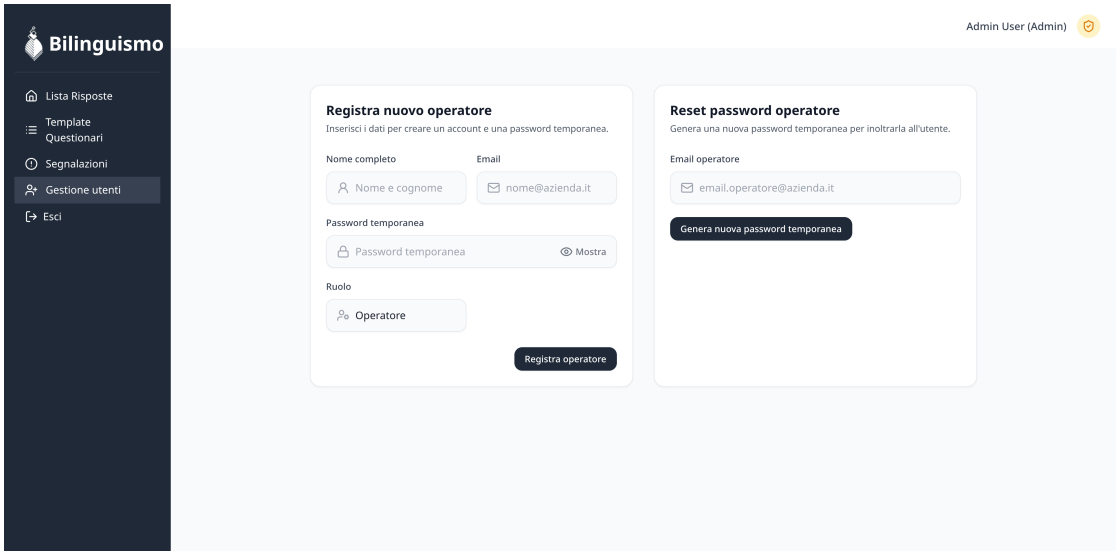


Figura 4.10: Registrazione e reset della password

4.4 Frontend dedicato alle famiglie

Il frontend destinato alle famiglie è progettato per offrire un'esperienza di compilazione semplice, lineare e accessibile, ottimizzata per l'utilizzo da dispositivi mobili. A differenza dell'area operatori, questa interfaccia è focalizzata esclusivamente sulla raccolta delle risposte e non richiede autenticazione tramite credenziali: l'utente accede attraverso un link dedicato e avvia o riprende la compilazione inserendo il codice fiscale del bambino

4.4.1 Architettura

Il frontend famiglie è organizzato in un insieme ridotto di pagine principali:

- `CFLoginPage` consente di inserire il codice fiscale e iniziare il flusso di compilazione.
- `QuestionnairePage` gestisce la visualizzazione dinamica del questionario, una domanda alla volta o per gruppi sequenziali, con autosalvataggio.
- `CompletionPage` mostra il messaggio di completamento al termine del questionario.

La logica di routing è gestita tramite React Router, come nel frontend operatori, ma in forma più semplice: non sono presenti rotte protette, mentre ogni pagina assume che sia già disponibile il contesto minimo necessario per procedere (template

associato e submission avviata). L'intero flusso è lineare e privo di biforcazioni complesse, favorendo una navigazione intuitiva su dispositivi mobili.

La comunicazione con il backend è regolata da due servizi API principali:

- `submissionApi`, che gestisce la creazione o il recupero dello stato di compilazione tramite l'endpoint `start_or_resume`, e l'autosalvataggio progressivo tramite `save_progress`;
- `feedbackApi` utilizzato per l'invio di eventuali segnalazioni da parte dell'utente durante la compilazione

Entrambi i servizi utilizzano gli stessi DTO e schemi TypeScript importati dal modulo shared, garantendo che la struttura dei payload di richiesta e risposta sia coerente con quella impiegata dal backend. Questo meccanismo permette di evitare duplicazioni nella definizione dei tipi e di assicurare che eventuali modifiche agli schemi Zod centrali siano propagate automaticamente a entrambe le applicazioni frontend.

4.4.2 Flusso di compilazione

La prima fase è gestita da `CFLoginPage`, che raccoglie il codice fiscale del bambino e permette di avviare o recuperare una compilazione già esistente. Alla conferma dell'input, la pagina richiama il servizio `submissionApi.startOrResume` il quale invia la richiesta al backend. In caso di risposta positiva, il frontend riceve dal server il contenuto del template e lo stato corrente della submission. Queste informazioni vengono memorizzate nello stato del componente tramite gli hook di React e passate come parametri alla fase successiva del flusso. Se la submission esiste già, l'interfaccia riprende automaticamente dal punto in cui la famiglia aveva interrotto la compilazione.

Render dinamico questionario Nella pagina di compilazione, l'intero questionario è costruito in modo dinamico a partire dalla definizione del template e dallo stato locale delle risposte, senza campi hard–coded.

All'avvio, il componente `QuestionnairePage` riceve tre informazioni fondamentali:

- il Template, che contiene in `structure_definition` la struttura del questionario serializzata in formato `QuestionnaireData` (sezioni, domande, opzioni, lingue);
- l'elenco delle risposte già salvate, come array di oggetti;
- l'eventuale identificativo dell'ultimo step completato, corrispondente a una sezione del questionario.

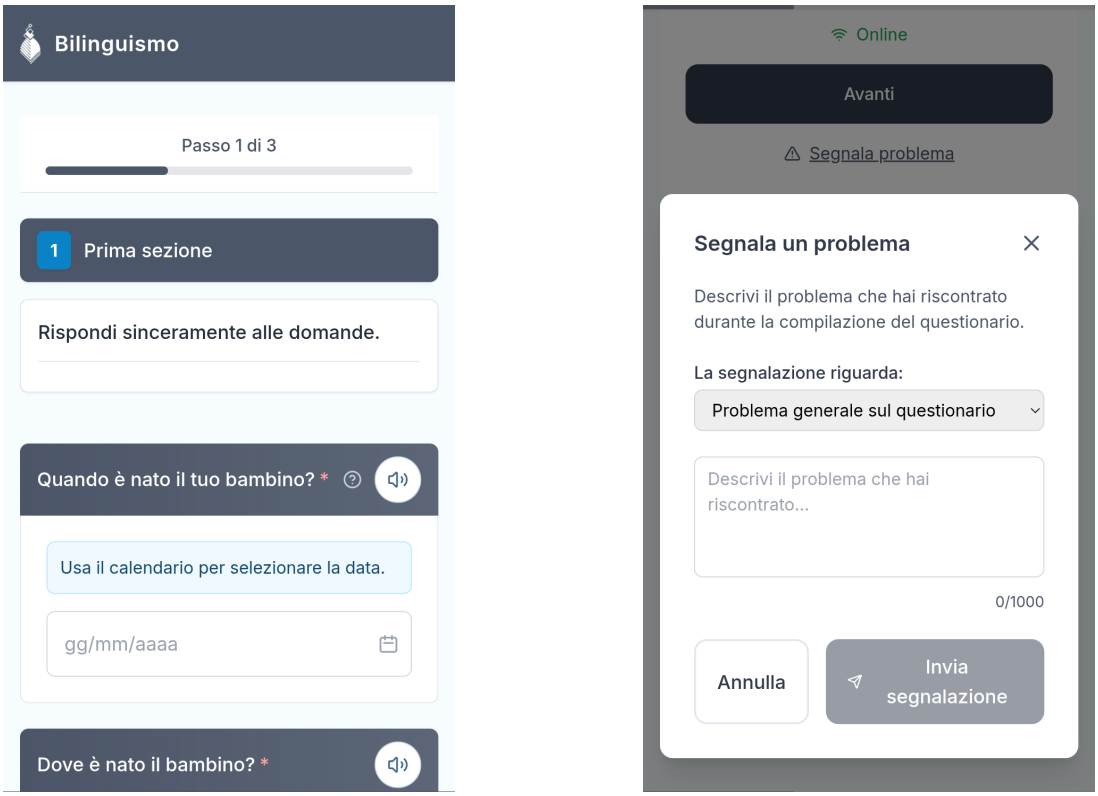
A partire da questi dati, QuestionnairePage imposta lo stato locale.

Le risposte restituite dal backend vengono immediatamente trasformate in una mappa `answersMap`, indicizzata per `question_identifier`. Per ogni elemento ricevuto, se sono presenti sia l'identificativo di domanda che il valore della risposta, la coppia viene copiata nello stato locale. Questa mappa costituisce l'unica fonte di verità delle risposte lato client: tutte le componenti di input leggono e aggiornano i valori partendo da `answers`. La vera costruzione dell'interfaccia avviene nella funzione `renderQuestion`, che, dato un oggetto `Question`, estrae il testo localizzato, recupera dal dizionario answers il valore corrente (se presente) e sceglie il componente di input appropriato.

Per ogni domanda della sezione corrente, `QuestionnairePage` invoca renderQuestion, che incapsula il testo, l'eventuale messaggio di errore, il valore proveniente da answers e i pulsanti di accessibilità (ad esempio per il TTS) dentro un `QuestionBlock`. In questo modo, l'interfaccia si adatta automaticamente a qualunque struttura sia definita nel template, senza logica specifica legata a singole domande.

Una volta completate tutte le sezioni, il sistema mostra la pagina CompletionPage, che conferma l'avvenuto invio delle risposte e conclude il flusso di compilazione. Durante la compilazione, l'utente ha inoltre la possibilità di inviare segnalazioni tramite il servizio `feedbackApi`.

Figura 4.11: Pagina di accesso al questionario



(a) Compilazione questionario **(b)** Modale feedback

Figura 4.12: Pagine compilazione

4.4.3 Gestione multilingua

Il frontend famiglie supporta la compilazione del questionario in più lingue e utilizza un sistema di localizzazione unificato che combina i testi definiti nel template con le traduzioni dell'interfaccia. La parte centrale di questo meccanismo è rappresentata dal modulo shared, che fornisce il tipo `LocalizedText` e funzioni come `getLocalizedText`, utilizzate per selezionare la traduzione corretta in base alla lingua scelta dall'utente. La funzione applica inoltre un semplice meccanismo di fallback: se la traduzione nella lingua selezionata non è disponibile, viene mostrato il testo nella lingua di default o, in ultima istanza, una delle traduzioni presenti.

L'applicazione dispone anche di un `TranslationContext`, che mantiene la lingua corrente e fornisce una funzione `t(key)` per tradurre le etichette dell'interfaccia. Tutte le pagine principali del flusso sono avvolte dal provider di questo contesto, che consente di cambiare lingua in qualsiasi momento e di aggiornare automaticamente i testi mostrati.

Il selettore di lingua recupera l'elenco delle lingue supportate dal catalogo condiviso e permette all'utente di passare dall'italiano ad altre lingue disponibili. La scelta viene memorizzata localmente, così da essere ripristinata a ogni accesso successivo. La stessa lingua viene poi utilizzata per risolvere i testi del questionario tramite le funzioni del modulo condiviso, garantendo coerenza tra interfaccia e contenuti.

In questo modo, sia i testi statici dell'applicazione sia quelli provenienti dal template risultano correttamente sincronizzati e aggiornati in base alla lingua selezionata dall'utente, offrendo un'esperienza di compilazione multilingue semplice e coerente.

4.4.4 Accessibilità

L'accessibilità rappresenta una componente essenziale del frontend destinato alle famiglie, in risposta ai requisiti non funzionali RNF3 e RNF8, che richiedono un'interfaccia utilizzabile anche da utenti con difficoltà linguistiche, cognitive o visive. L'implementazione combina tre assi principali: il supporto agli screen reader, la sintesi vocale delle domande tramite Web Speech API e un insieme di accorgimenti visivi conformi ai principi WCAG, come contrasto elevato, gerarchia tipografica chiara e pulsanti adeguati all'uso su dispositivi mobili.

Supporto agli screen reader

Il supporto è realizzato tramite il componente `AnnouncementProvider`, collocato alla radice del frontend famiglie. Questo componente espone un contesto React che permette a tutte le pagine e ai componenti del questionario di generare annunci vocali tramite una funzione `announce(message)`. Dal punto di vista tecnico, il provider crea una regione con `aria-live="polite"`, aggiornata dinamicamente quando viene richiesto un annuncio.

L'utilizzo della modalità *polite* garantisce che gli screen reader integrino i messaggi in modo non intrusivo, evitando interruzioni improvvise della lettura in corso. I messaggi vengono generati in situazioni chiave del flusso di compilazione:

- quando l'utente passa da una sezione del questionario alla successiva;
- quando una domanda obbligatoria non risulta compilata;
- quando si verifica un errore durante il caricamento o il salvataggio;
- quando viene aggiornato il contenuto di una domanda in base alla lingua selezionata.

Questa soluzione consente di migliorare l'orientamento dell'utente, soprattutto quando lo schermo non fornisce un riferimento visivo chiaro, come nel caso di schermate composte esclusivamente da elementi testuali. Poiché l'interfaccia è progettata per dispositivi mobili, il supporto allo screen reader diventa essenziale anche per utenti che utilizzano assistenti vocali integrati (es. VoiceOver su iOS o TalkBack su Android).

Sintesi vocale tramite Web Speech API

Alla funzionalità di screen reader si affianca un sistema di *Text-to-Speech (TTS)* integrato direttamente nel frontend e realizzato tramite *la Web Speech API*. La logica centrale è incapsulata nell'hook `useTextToSpeech`, che fornisce tre operazioni: avviare la lettura di un testo, interromperla e aggiornare automaticamente la lingua della voce.

Quando l'utente seleziona il pulsante "ascolta" all'interno di un QuestionBlock, il frontend genera un oggetto `SpeechSynthesisUtterance`, impostando come testo la versione localizzata della domanda e come lingua il codice scelto nel componente di selezione. La Web Speech API si occupa quindi di trasformare il testo in parlato, utilizzando la voce disponibile più compatibile con la lingua selezionata.

Il TTS è strettamente integrato con la navigazione del questionario:

- la lettura viene automaticamente interrotta quando l'utente passa alla domanda successiva;
- la lingua della voce viene aggiornata quando l'utente cambia lingua nel selettore multilingua;
- eventuali annunci generati dallo AnnouncementProvider non interrompono la sintesi vocale in corso, ma vengono accodati con priorità inferiore.

Questa soluzione supporta efficacemente utenti con difficoltà di lettura o alfabetizzazione limitata, consentendo loro di completare autonomamente il questionario.

4.5 Backend

4.5.1 Architettura a tre livelli

Controller Il backend della piattaforma è realizzato come API REST in Express e segue concretamente il pattern *Controller–Service–Repository* descritto nel Capitolo 3. In fase di implementazione, questa struttura è stata resa esplicita a partire dal livello di routing e dalla definizione dei middleware Express, in modo che il flusso di ogni richiesta sia chiaramente tracciabile.

I *controller* rappresentano il punto di ingresso delle richieste HTTP e sono raggruppati per area funzionale. Ognuno di essi riceve una richiesta già validata dai middleware, estrae i parametri tipizzati (ad esempio `LoginInput`, `StartOrResumeRequest`, `CreateTemplateInput` definiti nel modulo `shared` o nei validator locali) e delega l'esecuzione al servizio corrispondente, limitandosi a costruire la risposta HTTP sulla base del risultato. In questo modo il controller rimane sottile e privo di logica di dominio: si occupa principalmente di adattare la richiesta e la risposta al protocollo HTTP.

In caso di eccezioni, il controller non gestisce direttamente l'errore, ma si limita a invocare `next(error)`, demandando la gestione al middleware globale. Questo approccio consente di mantenere uniforme il formato delle risposte di errore e di centralizzare le decisioni relative ai codici di stato HTTP e ai messaggi restituiti al client.

Service Lo strato di *service* incapsula la logica applicativa vera e propria. Qui vengono orchestrate le operazioni che possono coinvolgere più repository, la gestione delle transazioni Prisma e l'applicazione delle regole di dominio (ad esempio, la distinzione tra nuova submission e ripresa in `startOrResume`, l'aggiornamento dello stato e dei timestamp in `save_progress`, la generazione di password temporanee lato admin, la preparazione dei dati per l'esportazione in Excel). Il service non conosce i dettagli HTTP: riceve parametri tipizzati e restituisce oggetti dominio o DTO, mantenendo il backend più testabile, riutilizzabile e indipendente dal framework web utilizzato.

Una parte rilevante della logica lato service riguarda la coerenza del flusso tra i diversi attori del sistema. Ad esempio, nei metodi legati alle submission, il service si occupa di verificare lo stato corrente (in corso o completata), applicare le regole di business che impediscono modifiche non consentite e mantenere allineati i metadati (step corrente, lingua di compilazione, timestamp).

Repository Il livello di *repository* è responsabile dell'accesso ai dati tramite Prisma. Ogni repository si occupa di una specifica entità o gruppo di entità, implementando operazioni CRUD, query con filtri e join necessarie a ricostruire le viste richieste dal service. In questo modo, lo strato di persistenza rimane isolato e può essere evoluto senza impattare i controller o i servizi, ad esempio nel caso di una ristrutturazione dello schema o di un cambio di tecnologia di storage.

I repository sfruttano le funzionalità avanzate di Prisma (come le *include* e le transazioni) per comporre query adatte ai casi d'uso reali, delegando comunque ai service la responsabilità di interpretare i dati recuperati in termini di logica di dominio.

Sopra questi tre livelli si colloca il *router* di Express. Ogni rotta dichiara in modo esplicito la sequenza di middleware da applicare (autenticazione, autorizzazione,

validazione) e il controller che dovrà elaborare la richiesta. Il file `app.ts` registra globalmente i middleware di infrastruttura (CORS, helmet, rate limiting, parsing JSON/URL-encoded), monta i router sul prefisso `/api/v1/...` e, infine, definisce un handler per le route non trovate e il middleware di gestione degli errori. In questo modo, la pipeline di una richiesta HTTP può essere riassunta come in Figura 4.13:

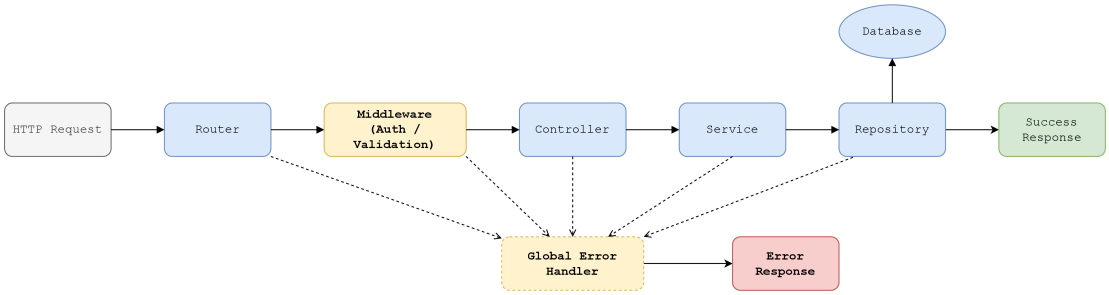


Figura 4.13: Pipeline della richiesta

Middleware di autenticazione

L'accesso alle API dedicate agli operatori è protetto da un middleware di autenticazione. La funzione `authMiddleware` verifica la presenza dell'header `Authorization`, controlla che sia nel formato `Bearer <token>` e delega a `verifyJwt` la validazione del token JWT firmato lato server. In caso di esito positivo, il payload decodificato viene aggiunto all'oggetto `Request` (campo `req.operator`), rendendo disponibili ai controller informazioni quali `operator_id`, `role` e il flag `must_change_password`. Se il token è assente, malformato o non valido, il middleware solleva un `ApiError` con codice `401`, che sarà poi gestito dal middleware globale.

Accanto al controllo di autenticazione, il backend implementa un middleware di autorizzazione basato sui ruoli, `requireRole`. Questa funzione di ordine superiore riceve l'elenco dei ruoli ammessi e restituisce un middleware che verifica che `req.operator` sia presente e che il ruolo contenuto nel token appartenga al set richiesto; in caso contrario viene sollevato un errore `403`. In questo modo, le route più sensibili (come la registrazione di nuovi operatori o l'eliminazione di template) possono essere protette aggiungendo semplicemente `requireRole(["admin"])` nella catena di middleware della rotta.

È importante notare che le famiglie non utilizzano alcun meccanismo di autenticazione basato su token o sessioni persistenti: l'accesso alle API lato famiglia è regolato dal codice fiscale e dall'identificativo della submission, come descritto nel Capitolo 3. L'assenza di un profilo utente tradizionale riduce la complessità dell'autenticazione per questo attore e concentra le verifiche di sicurezza sulle API di backoffice.

Middleware di validazione

La validazione degli input è centralizzata nel middleware generico `validate`. Questo middleware riceve in ingresso uno schema Zod e l'indicazione della parte della richiesta da validare (`body`, `query` o `params`); in fase di esecuzione tenta di eseguire `schema.parse` sulla porzione corrispondente della request. Se la validazione ha successo, la request viene sovrascritta con la versione tipizzata e normalizzata (ad esempio, trimming automatico delle stringhe o trasformazioni definite nello schema) e il flusso prosegue verso il controller.

In caso di `ZodError`, il middleware raccoglie in una lista i singoli errori (campo e messaggio) e li incapsula in un oggetto `ApiError` con codice `400`, delegando la risposta al gestore globale. Questa soluzione consente di riutilizzare lo stesso meccanismo per tutte le route, passando semplicemente lo schema appropriato, e garantisce che i controller ricevano sempre parametri già validati, riducendo il rischio di bug dovuti a input non controllati.

Il fatto che gli schemi Zod risiedano nel modulo `shared` permette inoltre di utilizzare gli stessi contratti sia nel backend, per la validazione runtime, sia nei frontend, per la tipizzazione statica e la validazione preventiva dei form. In questo modo, l'intero sistema si appoggia a una singola fonte di verità per la struttura dei dati.

Middleware di gestione degli errori

La catena dei middleware si chiude con un middleware di error handling globale, `globalErrorHandler`. Questo componente intercetta sia gli `ApiError` esplicitamente generati dall'applicazione (ad esempio per input non validi, token mancanti o permessi insufficienti), sia eventuali eccezioni non gestite, e restituisce al client una risposta JSON strutturata con campi `status_code`, `error`, `message`, `timestamp` e, facoltativamente, `details` quando sono disponibili informazioni puntuali sui campi in errore.

- In termini pratici: errori di validazione sui dati in ingresso generano tipicamente risposte con codice `400`,
- errori di autenticazione producono `401`, errori di autorizzazione `403`,
- tentativo di accedere a risorse inesistenti (ad esempio una submission non trovata) viene tradotto in un `404`.
- eventuali eccezioni non previste vengono invece mappate su un generico `500`, preservando i dettagli interni nei log del server ma evitando di esporli al client.

La presenza di un unico punto di raccolta degli errori semplifica sia il debugging sia la gestione dei messaggi lato frontend, che può contare su un formato uniforme e prevedibile per tutte le API.

4.5.2 Implementazione delle API principali

Avvio o ripresa della compilazione (`start_or_resume`)

L'endpoint `start_or_resume` rappresenta il punto di ingresso del flusso di compilazione dal lato delle famiglie. Il controller riceve un payload validato tramite Zod (codice fiscale, template e lingua) e delega l'intera logica al servizio dedicato. La prima operazione del service consiste nel verificare, tramite il repository delle submission, se esista già una compilazione attiva (cioè non completata) per la coppia *(codice fiscale, template)*. Questa ricerca viene effettuata tramite una query Prisma che include anche l'elenco delle risposte eventualmente già salvate.

Se non è presente alcuna submission attiva, il servizio ne crea una nuova, inizializzando i campi principali (`completed = false`, lingua selezionata, `current_step_identifier` relativo alla prima sezione del questionario). Se invece esiste una submission in corso, il servizio la riutilizza e, prima di restituire la risposta al client, ricava quale sezione deve essere mostrata successivamente. Tale informazione deriva dal campo `current_step_identifier`, aggiornato dal salvataggio progressivo e applicato nel frontend per riprendere la compilazione dal punto corretto.

Indipendentemente dal caso (creazione o ripresa), il service recupera la struttura del questionario dal repository dei template (campo `structure_definition`) e costruisce un DTO che contiene: i dati della submission, lo stato corrente, la struttura del questionario, la mappa delle risposte già fornite e lo step da cui riprendere. Il controller si limita infine a restituire questo oggetto al frontend, che potrà ricostruire dinamicamente l'interfaccia, mantenendo separati il formato interno dei dati e il layout di presentazione.

Autosalvataggio progressivo (`save_progress`)

L'endpoint `save_progress` implementa il meccanismo di salvataggio incrementale delle risposte, utilizzato dal frontend famiglie dopo ogni modifica. Il controller valida il payload (submission, risposte, step corrente ed eventuale flag di completamento) e passa la gestione al servizio. Il service recupera la submission tramite repository e verifica che non sia marcata come completata, prevenendo modifiche su questionari già conclusi.

La fase centrale dell'operazione è l'aggiornamento delle risposte nella tabella `Answers`. Tale operazione viene eseguita all'interno di una transazione Prisma: per ogni risposta ricevuta, il repository esegue un'operazione di *upsert* basata sulla chiave composta `(submission_id, question_identifier)`. In questo modo l'API

risulta idempotente: invii ripetuti dello stesso payload non generano duplicati ma aggiornano la risposta esistente. All'interno della stessa transazione il servizio aggiorna i metadati della submission, impostando il nuovo `current_step_identifier` e, nel caso il questionario sia stato completato, marcando la submission come `completed`.

Se tutte le operazioni della transazione si concludono con successo, Prisma effettua il commit; in caso di errore la transazione viene annullata e l'eccezione viene gestita dal middleware globale. Questo approccio garantisce che la submission e le risposte rimangano sempre in uno stato coerente, anche in presenza di richieste ripetute, connessioni instabili o interruzioni improvvise del flusso.

Template, feedback, note ed esportazione

Oltre alle API centrali di gestione delle submission, il backend espone un insieme di endpoint dedicati a operazioni di natura prevalentemente CRUD. Queste API seguono lo stesso schema architetturale basato su controller, service e repository, con una logica interna meno articolata rispetto a `start_or_resume` e `save_progress`.

Le API relative ai template consentono di creare un nuovo template, modificarne la struttura e recuperare l'elenco o un singolo template tramite il relativo `template_id`. Il repository dei template incapsula le operazioni di lettura e scrittura sul database e include funzioni per recuperare la struttura del questionario (`structure_definition`) e aggiornare le meta-informazioni associate. Poiché la struttura del questionario è memorizzata in formato JSON, le operazioni richieste si riducono principalmente alla serializzazione e deserializzazione del campo, più le normali logiche di validazione dei dati in ingresso.

La gestione dei feedback è implementata in modo analogo. Le API permettono alla famiglia di inviare una segnalazione durante la compilazione e agli operatori sanitari di consultare la lista dei feedback ricevuti applicando filtri sullo stato o sul template. La logica applicativa è principalmente focalizzata sull'applicazione dei filtri e sull'aggiornamento dello stato della segnalazione (ad esempio, "nuovo" o "revisionato"). Il repository corrispondente utilizza Prisma per comporre le condizioni di ricerca e includere le informazioni di contesto necessarie alla vista operatore. Nella versione attuale, il filtro sulla domanda si basa sull'identificativo interno, soluzione funzionale ma meno intuitiva rispetto a un filtraggio per testo o per sezione, come evidenziato nella fase di validazione.

Le note cliniche associate alle singole risposte vengono gestite tramite le API esposte in `notes.routes.ts`. Anche in questo caso, il servizio si limita a validare l'input, recuperare la risposta associata e aggiornare o creare la nota corrispondente nella tabella dedicata. Le note vengono restituite al frontend operatori integrandole nel dettaglio della compilazione, ma la loro gestione non richiede logiche particolarmente complesse.

Infine, il backend mette a disposizione un endpoint per l'esportazione delle compilazioni in formato Excel, implementato nel file `export.service.ts`. La generazione del file utilizza la libreria `exceljs`, che permette di creare un workbook e di aggiungere un foglio di lavoro per ciascuna sezione del questionario. Ogni foglio contiene le domande e le risposte corrispondenti, insieme alle eventuali note cliniche associate. Il risultato viene inviato al client come payload binario, consentendo al browser dell'operatore di avviare direttamente il download del file.

Nel complesso, queste API completano la funzionalità della piattaforma fornendo gli strumenti necessari per la gestione dei questionari, delle segnalazioni e dei dati clinici, mantenendo la coerenza con il modello architetturale e le convenzioni applicate alle API principali.

4.5.3 Gestione degli errori nel backend

La gestione degli errori del backend è stata implementata in modo centralizzato, con l'obiettivo di fornire risposte coerenti e strutturate a tutte le API della piattaforma. L'intero meccanismo si basa su una combinazione di middleware Express, una classe di errore applicativo dedicata e un formato uniforme di risposta. In questo modo, i controller e i service possono sollevare eccezioni senza occuparsi direttamente della costruzione della risposta HTTP, mantenendo il codice più leggibile e separando chiaramente la logica applicativa dalla gestione dei fallimenti.

L'elemento principale del sistema è la classe `ApiError`. Essa rappresenta un errore operativo previsto (ad esempio input non valido, permessi insufficienti o risorse non trovate) e incapsula tre informazioni fondamentali: il codice di stato HTTP, un messaggio descrittivo e, facoltativamente, un insieme di dettagli aggiuntivi da restituire al client. I service utilizzano questa classe per segnalare condizioni di errore applicative, come la presenza di una submission non compatibile con l'operazione richiesta o il tentativo di registrare un utente già esistente.

Tutti gli errori sollevati nel flusso — sia `ApiError` sia eccezioni non previste — vengono intercettati dal middleware globale `globalErrorHandler`, registrato come ultimo elemento della catena middleware in `app.ts`. Questo componente identifica il tipo di errore e costruisce una risposta JSON normalizzata che include almeno i campi `status_code`, `error`, `message` e `timestamp`. Se l'errore deriva da una validazione Zod, il middleware inserisce anche la lista dei campi non validi nel campo `details`, in modo da fornire al frontend informazioni utili per la visualizzazione dei messaggi di errore all'utente.

La validazione degli input è strettamente integrata con il meccanismo di gestione degli errori tramite il middleware `validate`, che applica lo schema Zod corrispondente alla parte della richiesta da validare (`body`, `query` o `params`). In caso di errore di validazione, il middleware non prosegue verso il controller ma solleva direttamente un `ApiError` con codice `400`, popolando automaticamente il campo

`details` con la lista degli errori Zod. Questo garantisce che nessun controller riceva mai input non validi, migliorando la robustezza e riducendo la complessità delle funzioni applicative.

4.5.4 Autenticazione e registrazione degli utenti

Il sistema di autenticazione degli operatori sanitari si basa su token JWT firmati dal server e verificati tramite middleware dedicati. L'intera logica è implementata nei file `auth.controller.ts`, `auth.service.ts` e nel repository degli operatori. Tutte le credenziali vengono gestite esclusivamente lato backend, con hashing tramite `bcrypt` e nessuna memorizzazione o restituzione della password in chiaro, ad eccezione delle password temporanee generate durante il reset amministrativo.

L'API di login valida le credenziali fornite dal frontend, recupera l'operatore dal database e confronta l'hash della password. In caso positivo, il servizio genera un JWT contenente l'identificativo, il ruolo e il flag `must_change_password`, che indica se l'utente è obbligato a modificare la password al primo accesso. Il token viene poi utilizzato dal frontend attraverso il contesto di autenticazione per controllare l'accesso alle sezioni riservate.

La registrazione di nuovi operatori è riservata agli amministratori e protetta tramite il middleware `requireRole(["admin"])`. Il servizio `register` verifica l'assenza di duplicati e crea un nuovo record nella tabella `Operators`, imponendo il flag `must_change_password = true`. In questo modo l'utente deve necessariamente impostare una password definitiva al primo login.

Il reset della password, anch'esso limitato agli amministratori, genera una nuova password temporanea mediante il modulo crittografico di Node.js. Il nuovo hash sostituisce quello precedente e il flag `must_change_password` viene nuovamente impostato a `true`. La password temporanea viene mostrata una sola volta all'amministratore nella risposta dell'API, in modo che possa essere comunicata all'operatore con canali separati.

Il cambio password è gestito dall'endpoint `change_password`, che non rappresenta una funzionalità di aggiornamento volontario della password, ma un passaggio obbligato quando `must_change_password` è attivo. Il middleware `authMiddleware` consente l'accesso solo agli utenti autenticati, mentre la logica del controller controlla esplicitamente che il flag sia effettivamente impostato a `true`. Solo in tal caso il servizio procede a validare la password corrente, a generare l'hash della nuova password e ad aggiornare il record nel database. Al termine, il servizio genera un nuovo JWT senza il flag di obbligo, consentendo all'utente di accedere al resto dell'applicazione.

Questo flusso garantisce che le password temporanee non possano essere utilizzate come credenziali permanenti e che il cambio venga effettuato in un punto ben

definito del processo, aumentando la sicurezza dell'intero sistema di autenticazione e riducendo al minimo l'esposizione delle credenziali.

4.6 Sintesi dei risultati implementativi

In questo capitolo è stata presentata l'implementazione completa della piattaforma, descrivendo nel dettaglio le soluzioni adottate nei due frontend, nel backend e nel modulo condiviso. Le sezioni hanno illustrato come le scelte architetturali definite in fase di progettazione siano state tradotte in componenti concreti: dalla struttura del frontend operatori con l'editor visuale dei questionari e gli strumenti di consultazione delle compilazioni, fino al frontend famiglie con il flusso di compilazione guidata, il salvataggio progressivo e il supporto multilingua. La parte server-side ha mostrato l'applicazione dello schema a tre livelli, l'integrazione con Prisma e l'uso di schemi condivisi per validazione e tipizzazione, insieme a un meccanismo uniforme di gestione degli errori e dell'autenticazione.

Il risultato è un sistema coerente e modulare, in cui i diversi componenti interagiscono attraverso contratti tipizzati e schemi condivisi, riducendo le possibilità di incoerenze tra client e server e migliorando la manutenibilità del codice. La gestione delle submission, l'autosalvataggio e la ripresa della compilazione sono stati implementati in modo da garantire robustezza anche in presenza di interruzioni, mentre la parte amministrativa offre agli operatori gli strumenti necessari per monitorare e gestire l'intero processo.

Il prossimo capitolo sarà dedicato alle attività di test e validazione, con l'obiettivo di verificare la correttezza funzionale delle API e dei principali flussi d'interazione, oltre a valutare l'usabilità e l'efficacia della piattaforma dal punto di vista degli utenti finali. I test riportati consentiranno di valutare in che misura l'implementazione soddisfa i requisiti individuati nella fase di analisi e se la piattaforma risponde adeguatamente alle esigenze cliniche che ne hanno motivato lo sviluppo.

Capitolo 5

Validazione e risultati

La fase di test e validazione costituisce un elemento fondamentale del processo di sviluppo della piattaforma, poiché permette di verificare il corretto funzionamento delle funzionalità implementate e di valutarne l'efficacia nel contesto operativo reale. Lo scopo di questo capitolo è presentare le attività svolte per garantire la qualità del sistema, analizzando sia gli aspetti tecnici sia l'esperienza d'uso degli utenti finali.

In questa fase del progetto non sono stati implementati test automatici di tipo *unit testing* o *integration testing*. Tale scelta è stata dettata dalla natura del sistema e dalla necessità di concentrare gli sforzi sulle modalità di verifica più rilevanti rispetto agli obiettivi del progetto. Trattandosi di una piattaforma orientata principalmente all'interazione utente e al supporto alla compilazione di questionari, la valutazione dell'usabilità e della chiarezza dei flussi riveste un ruolo centrale. Per questo motivo, è stato adottato un approccio focalizzato su test funzionali e su una validazione qualitativa approfondita con utenti reali, in grado di restituire indicazioni concrete sull'esperienza d'uso.

Le attività di testing si articolano quindi in tre aree principali:

- **Test tecnici delle API**, effettuati tramite strumenti dedicati per verificare la correttezza delle operazioni di creazione, lettura, aggiornamento e cancellazione dei dati, la gestione delle autorizzazioni e il comportamento del sistema in presenza di input non validi;
- **Test end-to-end manuali**, finalizzati a valutare il funzionamento complessivo dei principali flussi applicativi, come la compilazione del questionario, l'autosalvataggio delle risposte, la visualizzazione delle submission e la gestione delle note da parte dell'operatore;
- **Validazione con utenti reali**, la parte più rilevante dell'intero processo di verifica, condotta direttamente presso la struttura dell'ASL. Questa attività

ha coinvolto due gruppi distinti di partecipanti (famiglie e operatori) e ha permesso di raccogliere metriche oggettive, osservazioni qualitative e valutazioni soggettive tramite il questionario SUS.

L'insieme di queste attività consente di ottenere una valutazione completa del sistema, verificando non solo la correttezza del comportamento tecnico della piattaforma, ma anche la sua effettiva usabilità e il grado di supporto che offre agli utenti nei diversi scenari d'uso. Nelle sezioni seguenti vengono presentati nel dettaglio i test condotti e i risultati ottenuti.

5.1 Test tecnici

5.1.1 Test delle API

La prima fase di verifica tecnica ha riguardato il comportamento delle API del backend, sviluppate in Node.js ed esposte tramite architettura REST. L'obiettivo dei test era assicurare la correttezza delle operazioni principali del sistema, la gestione appropriata degli errori e la coerenza dei dati scambiati tra frontend e backend.

I test sono stati condotti utilizzando strumenti dedicati, tra cui *Postman* e *Thunder Client*, che hanno permesso di isolare e verificare singolarmente i singoli endpoint. Sono state testate tutte le principali categorie di operazioni:

- **Gestione dei template dei questionari** (creazione, aggiornamento, eliminazione, recupero dei questionari disponibili).
- **Gestione delle compilazioni (submission)** con particolare attenzione al meccanismo di autosalvataggio progressivo e al recupero dello stato corrente.
- **Gestione delle note** associate alle risposte fornite dalla famiglia.
- **Sistema di feedback** per l'invio, la consultazione e l'aggiornamento delle segnalazioni.
- **Autenticazione e autorizzazione** tramite token JWT per gli operatori sanitari.

Per ogni endpoint sono stati verificati:

- la correttezza del *payload* inviato e ricevuto;
- il rispetto del formato previsto dagli schemi di validazione lato backend;
- il comportamento in caso di input errati o dati mancanti;

- la gestione corretta dei codici di risposta HTTP;
- la coerenza dei vincoli del database tramite Prisma ORM.

I test hanno confermato la stabilità delle API e la correttezza del flusso dati, inclusi i casi limite come l'invio di risposte parziali, formati non validi o tentativi di accesso non autorizzati. Il sistema ha risposto sempre con messaggi chiari e codici di errore adeguati, garantendo un comportamento prevedibile e robusto per l'interfaccia frontend.

5.1.2 Test end-to-end manuali

Oltre ai test sulle API, è stata condotta una sessione sistematica di test end-to-end manuali finalizzata a verificare il comportamento dell'intera piattaforma nelle condizioni d'uso reali. A differenza dei test sulle singole funzionalità, i test E2E permettono di valutare la coerenza complessiva del sistema, il corretto funzionamento dei flussi di navigazione e l'integrazione tra frontend, backend e database.

I test sono stati eseguiti simulando i due principali attori della piattaforma: la famiglia, che compila il questionario, e l'operatore sanitario, che crea, gestisce e analizza i questionari. I flussi verificati includono:

- **Compilazione del questionario da parte della famiglia** apertura del link, inserimento del codice fiscale, scelta della lingua, compilazione step-by-step con autosalvataggio e conclusione della submission.
- **Gestione delle compilazioni lato operatore** visualizzazione in tempo reale dello stato (in corso, completato), apertura delle risposte e inserimento di note.
- **Ricerca e filtraggio avanzato** per data, stato, lingua e codice fiscale, verificando sia la correttezza dei risultati sia la responsività dell'interfaccia.
- **Creazione e modifica dei questionari tramite editor visuale** aggiunta e riordino delle sezioni, creazione di domande di tipologie diverse, gestione delle versioni multilingua.
- **Sistema di feedback** invio di una segnalazione da parte della famiglia, visualizzazione lato operatore, aggiornamento dello stato.
- **Esportazione dei dati** generazione del file Excel contenente tutte le risposte della submission selezionata.

I test sono stati condotti simulando scenari realistici, inclusi casi limite come l'interruzione della compilazione, l'inserimento di campi incompleti, la modifica di un questionario già pubblicato e la verifica dell'autosalvataggio in condizioni di rete instabile.

Tutti i flussi principali sono risultati correttamente eseguibili e coerenti. Le criticità emerse — principalmente relative alla chiarezza di alcuni pulsanti nell'editor e alla sezione dei feedback — sono state approfondite nella successiva validazione con utenti reali e hanno guidato le osservazioni qualitative presentate nelle sezioni seguenti.

5.2 Validazione con utenti reali

La fase di validazione con utenti reali rappresenta il momento centrale dell'intero processo di verifica della piattaforma. A differenza dei test tecnici preliminari, finalizzati principalmente a garantire la correttezza delle API, dei flussi di navigazione e dei meccanismi di autosalvataggio, la validazione in presenza permette di valutare l'usabilità effettiva del sistema e la sua capacità di supportare gli utenti finali nelle attività previste.

Per garantire una valutazione realistica e rappresentativa, la sessione di test è stata condotta direttamente presso la struttura dell'ASL, coinvolgendo due gruppi distinti di utenti:

- 4 operatori sanitari (fisioterapista, infermiere, amministrativa, segretaria), utilizzati come campione per testare il lato "operatore" della piattaforma;
- 4 famiglie/pazienti del reparto, incaricate di valutare l'esperienza d'uso lato compilazione.

Per ciascun partecipante e per ciascun task sono stati raccolti i seguenti parametri:

- Tempo di completamento, misurato dall'avvio del task al suo completamento o abbandono;
- *Numero di errori*, inteso come numero di volte in cui l'utente ha commesso degli errori che hanno compromesso il flusso e quindi ha dovuto ripetere il task;
- *Aiuti richiesti*, ovvero il numero di interventi espliciti da parte dell'osservatore (chiarimenti, suggerimenti, indicazioni sul prossimo passo);
- *Risultato*, codificato come completamento corretto del task, completamento con aiuto o fallimento.

Al termine delle attività, ad ogni partecipante è stato inoltre somministrato il questionario di valutazione *SUS (System Usability Scale)*, uno strumento standardizzato composto da dieci affermazioni finalizzato a misurare la percezione soggettiva dell'usabilità del sistema.

Le due sottosezioni seguenti riportano in dettaglio il processo di validazione per ciascun gruppo di utenti, con i relativi task, risultati quantitativi e osservazioni qualitative.

5.2.1 Validazone lato famiglia

La prima fase di validazione con utenti reali ha riguardato il lato famiglia della piattaforma. L'obiettivo principale era verificare se i genitori fossero in grado di completare in autonomia il flusso di compilazione del questionario e utilizzare le principali funzionalità messe a disposizione dal sistema.

Per questa attività è stato utilizzato un *template di prova*, ovvero una versione ridotta del questionario effettivamente impiegato dalle dottoresse per lo screening linguistico. Tale scelta ha permesso di mantenere la struttura e la logica del questionario reale, riducendo al contempo il carico cognitivo e la durata della compilazione durante il test.

Sono stati coinvolti quattro partecipanti, selezionati tra i pazienti del reparto e le relative famiglie. A ciascuno di essi è stato chiesto di svolgere individualmente due task specifici:

- **T1 – Completare il flusso di compilazione**: accedere alla piattaforma a partire dal link fornito, inserire il codice fiscale richiesto, selezionare la lingua (quando previsto) e completare interamente la compilazione del questionario di prova fino alla schermata di conferma finale.
- **T2 – Segnalare una domanda poco chiara**: individuare, all'interno del questionario, una domanda percepita come ambigua, difficile o poco comprensibile e utilizzare l'apposita funzionalità di segnalazione per inviare un feedback all'équipe.

Analisi quantitativa dei risultati

I risultati quantitativi raccolti per i due task sono riportati nelle tabelle 5.1 e 5.2, una per ciascun task.

Analisi qualitativa dei risultati

Dall'osservazione diretta delle sessioni di prova e dall'analisi dei parametri raccolti nelle Tabelle 5.1 e 5.2 emergono alcune considerazioni qualitative sull'esperienza d'uso della piattaforma da parte delle famiglie.

Utente	Tempo	Errori	Aiuti	Risultato
U1	2m20s	0	0	Successo
U2	5m43s	1	1	Successo parziale
U3	3m2s	0	0	Successo
U4	2m35s	0	0	Successo

Tabella 5.1: Risultati del task T1 - famiglia

Utente	Tempo	Errori	Aiuti	Risultato
U1	1m45s	0	0	Successo
U2	3m50s	1	2	Successo parziale
U3	1m31s	0	0	Successo
U4	2m07s	1	0	Successo parziale

Tabella 5.2: Risultati del task T2 - famiglia

Nel complesso, tre partecipanti su quattro hanno portato a termine entrambi i task senza incontrare difficoltà rilevanti. I tempi di completamento sono risultati contenuti e coerenti con le aspettative per un questionario di prova a complessità ridotta: la maggior parte degli utenti ha completato il flusso T1 in circa due o tre minuti e il task di segnalazione T2 in poco più di un minuto. L'assenza di errori e richieste di aiuto per questi tre utenti suggerisce che l'interfaccia utente sia sufficientemente chiara anche per persone con familiarità limitata con strumenti digitali.

L'utente U2 rappresenta un caso particolarmente interessante dal punto di vista qualitativo. È stato l'unico partecipante a commettere errori in entrambi i task e l'unico ad aver richiesto assistenza esplicita. Durante il task T1 ha inizialmente confuso il pulsante per avanzare allo step successivo con quello di ritorno alla schermata precedente, causando una ripetizione involontaria di una parte del questionario. Un errore simile si è verificato in T2, dove il partecipante ha tentato di inviare la segnalazione tramite la barra di ricerca del browser, confondendo tale interfaccia con il campo di inserimento previsto dalla piattaforma. Nonostante ciò, con un minimo supporto è riuscito a portare a termine entrambi i task.

Un aspetto rilevante emerso durante i test riguarda l'utilizzo delle funzionalità di accessibilità. Nessuno dei partecipanti ha avuto necessità di ricorrere allo *screen reader* o al sistema di sintesi vocale (TTS) durante la compilazione. Questo risultato è coerente con il profilo dei partecipanti, tutti dotati di buona autonomia nella lettura e nella comprensione del testo. Nonostante ciò, la presenza di tali funzionalità mantiene un ruolo fondamentale, in quanto pensate per supportare utenti con ridotte competenze alfabetiche, difficoltà linguistiche o condizioni che rendono difficoltosa la lettura del testo sullo schermo.

Complessivamente, il flusso di compilazione e il sistema di segnalazione sono stati percepiti come chiari e coerenti. La presenza di un solo utente in difficoltà – e per motivi legati principalmente alla scarsa dimestichezza con strumenti digitali – suggerisce che la piattaforma sia già sufficientemente robusta dal punto di vista dell'usabilità e adatta al target previsto, pur lasciando spazio a piccoli interventi migliorativi mirati a incrementare ulteriormente la chiarezza dell'interfaccia.

5.2.2 Validazione lato operatore

La validazione del lato operatore ha coinvolto quattro utenti appartenenti al personale dell'ASL, selezionati tra figure amministrative o sanitarie non direttamente coinvolte nella gestione dei questionari nella pratica clinica. L'obiettivo era simulare il comportamento di un utente che utilizza la piattaforma per la prima volta, valutando l'intuitività della dashboard, dei filtri, dell'editor dei questionari e delle funzionalità di condivisione ed esportazione.

A ciascun partecipante è stato chiesto di completare una serie di sette task, progettati per coprire in modo sistematico tutte le funzionalità principali dell'interfaccia operatore. Per ogni task sono stati definiti in anticipo il risultato atteso e la relativa motivazione, così da verificare in maniera oggettiva l'efficacia dei componenti dell'interfaccia e la chiarezza dei flussi.

Di seguito vengono riportati i task somministrati:

- **T1 – Filtrare compilazioni completate in italiano dopo l'1 marzo 2024 ed eliminare la prima dell'elenco** *Risultato atteso:* l'utente deve utilizzare i filtri della dashboard per selezionare le compilazioni con stato "Completato", impostare il filtro sulla lingua italiana e applicare il filtro sulla data. Una volta ottenuta la lista filtrata, deve individuare la prima compilazione ed eliminarla. *Motivazione:* questo task permette di valutare la comprensibilità e l'utilità dei filtri presenti nella dashboard, verificando se la loro posizione, etichettatura e comportamento risultano intuitivi.

- **T2 – Cercare la compilazione associata a un codice fiscale specifico, aggiungere una nota e esportare il questionario in formato Excel** *Risultato atteso:* l'utente deve utilizzare il filtro dedicato al codice fiscale per trovare la compilazione desiderata, aprirla e inserire una nota in una risposta a scelta. Successivamente deve individuare il pulsante per esportare l'intera compilazione in formato Excel.

  Motivazione: questo task valuta la visibilità del campo di ricerca per codice fiscale, la facilità nell'aprire una compilazione dalla lista filtrata e la chiarezza dei pulsanti dedicati alle note e all'esportazione.

- **T3 – Cercare un feedback in stato "In esame" per un questionario dal titolo specifico con identificativo domanda anch'esso specifico e portarlo nello stato "Risolto"** *Risultato atteso:* l'utente deve accedere alla sezione dei feedback, applicare i filtri per mostrare solo quelli in stato "In esame", selezionare il feedback corretto e modificarne lo stato impostandolo a "Risolto".

  Motivazione: il task consente di verificare l'efficacia dei filtri anche nella sezione dedicata alle segnalazioni e la chiarezza del meccanismo di aggiornamento dello stato di un feedback.

- **T4 – Selezionare un questionario dall'elenco dei template e condividerlo tramite link o QR code** *Risultato atteso:* l'utente deve navigare nella sidebar fino alla sezione dei template, selezionare un questionario e individuare il pulsante di condivisione che consente di generare link e codice QR. *Motivazione:* questo task valuta la capacità dell'utente di orientarsi tra le sezioni, riconoscere il contesto in cui si trova e individuare autonomamente la funzionalità di condivisione.

- **T5 – Creare un nuovo questionario in italiano con una sezione e una domanda a scelta multipla obbligatoria** *Risultato atteso:* dalla dashboard dei template l'utente deve selezionare la creazione di un nuovo questionario, inserire titolo e descrizione in italiano, aggiungere una sezione e inserire una domanda a scelta multipla con almeno due opzioni, contrassegnandola come obbligatoria. Infine deve salvare il questionario. *Motivazione:* questo task permette di testare la comprensibilità e la facilità d'uso dell'editor, verificando se l'utente è in grado di creare un questionario da zero senza supporto.

- **T6 – Modificare un questionario esistente aggiungendo una domanda aperta alla prima sezione e creando una seconda sezione con altre due domande** *Risultato atteso:* l'utente deve riaprire un questionario già creato (inizialmente visualizzato in modalità anteprima), individuare l'icona che consente di entrare in modalità modifica, aggiungere alla prima sezione una domanda a risposta aperta, creare una seconda sezione e inserire due domande aggiuntive, quindi salvare.

  Motivazione: il task verifica se il passaggio dalla modalità anteprima alla modalità modifica è chiaro e se la struttura dell'editor risulta intuitiva anche in caso di modifiche complesse.

- **T7 – Aggiungere una lingua extra (es. cinese) al questionario, compilare i campi necessari e salvare** *Risultato atteso:* l'utente deve utilizzare il pulsante dedicato alle lingue extra, selezionare una lingua dall'elenco esteso,

compilare almeno il titolo del questionario e i titoli delle sezioni nella nuova lingua, verificare lo switch linguistico e salvare.

Motivazione: questo task testa l'usabilità della funzionalità di gestione multilingua, in particolare la capacità dell'utente di individuare la sezione dedicata all'aggiunta delle lingue non predefinite e comprendere la struttura dei campi da compilare.

Analisi quantitativa dei risultati

I risultati quantitativi raccolti per i task sono riportati nelle tabelle elencate di seguito

Utente	Tempo	Errori	Aiuti	Risultato
U1	1m56s	1	1	Successo parziale
U2	1m17s	0	0	Successo
U3	2m7s	1	0	Successo parziale
U4	1m34s	0	0	Successo

Tabella 5.3: Risultati del task T1 - Operatori

Utente	Tempo	Errori	Aiuti	Risultato
U1	32s	0	0	Successo
U2	20s	0	0	Successo
U3	56s	1	0	Successo Parziale
U4	49s	0	0	Successo

Tabella 5.4: Risultati del task T2 - Operatori

Utente	Tempo	Errori	Aiuti	Risultato
U1	15s	0	0	Successo
U2	1m4s	0	1	Successo parziale
U3	1m17s	0	1	Successo parziale
U4	2m21s	0	1	Successo

Tabella 5.5: Risultati del task T3 - Operatori

Utente	Tempo	Errori	Aiuti	Risultato
U1	11s	0	0	Successo
U2	13s	0	0	Successo
U3	10s	0	0	Successo
U4	21s	0	0	Successo

Tabella 5.6: Risultati del task T4 - Operatori

Utente	Tempo	Errori	Aiuti	Risultato
U1	44s	0	0	Successo
U2	2m20s	0	1	Successo parziale
U3	1m56s	0	1	Successo
U4	1m35s	0	0	Successo

Tabella 5.7: Risultati del task T5 - Operatori

Utente	Tempo	Errori	Aiuti	Risultato
U1	17s	0	0	Successo
U2	1m	0	0	Successo
U3	1m17s	1	0	Successo parziale
U4	40s	0	0	Successo

Tabella 5.8: Risultati del task T6 - Operatori

Utente	Tempo	Errori	Aiuti	Risultato
U1	30s	0	0	Successo
U2	50s	0	0	Successo
U3	1m10s	0	1	Successo
U4	1m3s	0	0	Successo

Tabella 5.9: Risultati del task T7 - Operatori

Analisi qualitativa dei risultati

L'analisi qualitativa dei task T1–T7 svolti dagli operatori consente di evidenziare punti di forza e criticità dell'interfaccia gestionale della piattaforma. I risultati mostrano come tutte le funzionalità principali siano state comprese e utilizzate correttamente, pur con alcune difficoltà localizzate nei task più articolati o in quelli che richiedono un livello maggiore di precisione nell'interazione. Di seguito vengono presentate le osservazioni suddivise per categoria di task.

Task di ricerca e filtraggio (T1–T2). I task dedicati all'utilizzo dei filtri della dashboard (T1 e T2) sono stati completati correttamente da tutti i partecipanti. I filtri relativi allo stato della compilazione, alla lingua e alla data sono stati individuati rapidamente da tutti gli utenti, con tempi di completamento compresi tra 1 e 2 minuti nel caso del task T1. Gli errori registrati in T1 (da U1 e U3) derivano dalla selezione errata del filtro sulla data. In T2, tutti gli utenti hanno portato a termine il task con successo e senza necessità di aiuto, dimostrando che la ricerca tramite codice fiscale è ben visibile e compresa in modo uniforme. L'unica problematica è stata ritrovata nel T2 fatto da U3, la zona da cliccare per accedere alla submission non è stata individuata immediatamente. Questo può essere uno spunto da approfondire per migliorare la zona "cliccabile", potrebbe essere estesa a tutta la riga della tabella.

Task di gestione dei feedback (T3). Il task relativo alla gestione dei feedback è stato quello che ha generato il maggior numero di richieste di aiuto (tre su quattro utenti). I tempi di completamento sono stati significativamente più elevati rispetto alla media degli altri task, indicando una minore immediatezza dell'interazione. La causa principale della difficoltà è stata individuata nella modalità di filtraggio dei feedback: attualmente il filtro permette di ricercare una segnalazione attraverso il solo *identificativo della domanda* (es. `s3_q2`), una convenzione interna efficace per lo sviluppatore ma non immediatamente interpretabile dal personale sanitario o amministrativo. Durante il test è stato osservato che gli utenti tendevano a cercare la domanda attraverso il testo esplicito della domanda stessa, non trovando riscontro nell'interfaccia. Tale comportamento rende evidente che il filtro dovrà essere ampliato o riprogettato per consentire la ricerca tramite una descrizione più leggibile o attraverso parole chiave del testo della domanda.

Task di navigazione e condivisione (T4). Il task T4 è stato eseguito correttamente e con tempi estremamente ridotti (tra 10 e 21 secondi). Tutti gli utenti hanno dimostrato di sapersi orientare senza difficoltà nella sidebar e di individuare rapidamente il pulsante dedicato alla condivisione dei questionari. Questo risultato conferma che la struttura di navigazione viene percepita come chiara e coerente.

Task di creazione e modifica dei questionari (T5–T6). I task T5 e T6 hanno messo alla prova l'usabilità dell'editor dei questionari. In T5 gli utenti hanno completato con successo la procedura di creazione di un nuovo questionario, sebbene con tempi variabili (tra 44 secondi e 2 minuti e 20 secondi). Le due richieste di aiuto osservate (U2 e U3) sono state dovute alla necessità di individuare correttamente il pulsante per aggiungere una nuova sezione e la corretta tipologia di domanda da inserire, suggerendo che alcuni pulsanti dell'editor potrebbero beneficiare di icone più esplicative o di una maggiore evidenza visiva.

Il task T6, dedicato alla modifica di un questionario esistente, è risultato quello con la maggiore dispersione nei tempi e con la presenza di errori (U3) legati al mancato passaggio dalla modalità *anteprima* alla modalità *modifica*. Durante il test è emerso chiaramente che l'icona responsabile di questo passaggio non è sufficientemente evidente e rischia di essere confusa con elementi grafici non interattivi. Tale criticità suggerisce la necessità di rendere più visibile lo stato corrente del questionario e di distinguere maggiormente le due modalità.

Task di gestione multilingua (T7). Il task T7 è stato completato con successo da tutti gli operatori, mostrando tempi di esecuzione compresi tra 30 secondi e 1 minuto e 10 secondi. La procedura di aggiunta di una lingua extra è stata compresa, anche se un utente (U3) ha richiesto assistenza per individuare il pulsante che apre il menu delle lingue aggiuntive. L'osservazione suggerisce che la funzionalità è generalmente intuitiva, ma potrebbe trarre beneficio da un posizionamento più evidente o da un'etichettatura più esplicita, soprattutto considerando che l'operazione non è tra le più frequenti nella pratica quotidiana.

Nel complesso, l'analisi qualitativa indica che gli operatori hanno percepito la piattaforma come usabile e coerente, completando tutte le attività assegnate senza blocchi critici. Le principali aree di miglioramento riguardano la gestione dei feedback, dove il meccanismo di ricerca deve essere reso più leggibile, e alcune funzionalità dell'editor dei questionari, che richiedono una maggiore evidenza visiva per ridurre gli errori e il ricorso all'aiuto esterno.

5.2.3 Valutazione dell'usabilità tramite SUS

Per completare la valutazione dell'usabilità della piattaforma è stato utilizzato il **System Usability Scale (SUS)**, un questionario standardizzato composto da dieci affermazioni con risposta su scala Likert a cinque punti. Il SUS è ampiamente adottato nella letteratura scientifica e industriale grazie alla sua capacità di produrre un punteggio sintetico dell'usabilità percepita, compreso tra 0 e 100, interpretabile attraverso benchmark consolidati.

Il questionario è stato somministrato al termine della sessione di test a entrambi i gruppi di utenti coinvolti: le famiglie che hanno svolto il flusso di compilazione e gli operatori che hanno testato la dashboard gestionale. Ogni partecipante ha valutato la piattaforma sulla base della propria esperienza immediata, fornendo così una misura soggettiva della facilità d'uso, dell'intuitività e della soddisfazione generale.

I punteggi ottenuti nei due gruppi sono riportati nelle Tabelle 5.10 e 5.11. Per entrambe le categorie di utenti è stato inoltre calcolato il punteggio medio, utile per confrontare i risultati con le classificazioni proposte in letteratura.

Secondo i benchmark comunemente adottati , un punteggio *superiore a 68* indica un livello di usabilità considerato "Accettabile", mentre valori superiori a *80* rientrano nella fascia "Excellent" o "Grade A", tipica dei sistemi percepiti come estremamente facili da usare. Un valore compreso tra *70* e *80* è generalmente classificato come "Good" o "Usable".

L'analisi dei risultati raccolti mostra che entrambi i gruppi hanno espresso una valutazione complessivamente positiva della piattaforma, con punteggi stabilmente al di sopra della soglia di accettabilità e, in alcuni casi, prossimi alla fascia superiore di qualità. Ciò conferma che la piattaforma risulta intuitiva, semplice da utilizzare e adatta al contesto applicativo per cui è stata progettata.

Utente	**Punteggio SUS**
U1	90
U2	85
U3	87,5
U4	87,5
Media	87,5

Tabella 5.10: Punteggi SUS ottenuti dai partecipanti – lato famiglia

Utente	**Punteggio SUS**
U1	87,5
U2	90
U3	92,5
U4	87,5
Media	89,375

Tabella 5.11: Punteggi SUS ottenuti dai partecipanti – lato operatore

5.3 Interpretazione dei risultati

Le attività di test e validazione descritte in questo capitolo hanno permesso di verificare in modo approfondito il corretto funzionamento della piattaforma, valutandone sia gli aspetti tecnici sia l'effettiva usabilità da parte degli attori coinvolti nel processo di screening linguistico. I test delle API e i test end-to-end manuali hanno confermato la stabilità del backend, la coerenza dei flussi applicativi e l'affidabilità dei meccanismi di autosalvataggio, autenticazione, filtraggio ed esportazione dei dati.

La validazione con utenti reali ha costituito la parte più significativa dell'intero processo. Le famiglie hanno dimostrato di poter completare il questionario in autonomia, con tempi contenuti e senza difficoltà rilevanti, evidenziando la chiarezza dell'interfaccia e la linearità del flusso di compilazione. Anche gli operatori sanitari hanno portato a termine tutti i task assegnati, mostrando una buona comprensione delle funzionalità principali della dashboard, dell'editor dei questionari e del sistema di gestione dei feedback. Le difficoltà riscontrate hanno riguardato principalmente task complessi o meno frequenti, come l'aggiornamento dello stato dei feedback e il passaggio dalla modalità anteprima alla modalità modifica nei questionari esistenti. Tali osservazioni indicano aree specifiche in cui l'interfaccia può essere ulteriormente migliorata per incrementare la chiarezza e ridurre il carico cognitivo degli utenti.

I punteggi SUS ottenuti da entrambi i gruppi confermano la percezione positiva dell'usabilità della piattaforma, con valori superiori alla soglia di accettabilità e prossimi alle fasce di qualità "Good" o "Excellent". Questi risultati, insieme alle analisi quantitative e qualitative, indicano che il sistema risulta adeguato all'utilizzo nel contesto clinico per cui è stato progettato, offrendo un'esperienza d'uso intuitiva e coerente.

Nel complesso, il processo di validazione ha dimostrato che la piattaforma è già sufficientemente matura per un utilizzo pilota all'interno del reparto, pur evidenziando margini di miglioramento che potranno essere affrontati nelle future evoluzioni del sistema. Il capitolo successivo discuterà le conclusioni generali del progetto, le sue limitazioni e le possibili direzioni per lo sviluppo futuro.

Capitolo 6

Conclusioni

6.1 Sintesi del lavoro svolto

Il lavoro descritto in questa tesi ha portato alla progettazione e allo sviluppo di una piattaforma web dedicata alla raccolta strutturata di informazioni linguistiche e anamnestiche sui bambini bilingui, con l'obiettivo di supportare in modo più efficiente e inclusivo il processo clinico svolto presso il reparto di Neuropsichiatria Infantile dell'ASL CN2. La piattaforma è composta da due componenti principali: una dashboard per gli operatori sanitari, che consente la gestione dei questionari, il monitoraggio delle compilazioni e l'inserimento di annotazioni cliniche; e un frontend ottimizzato per smartphone destinato alle famiglie, che permette la compilazione guidata e multilingue dei questionari.

L'architettura full-stack, basata su React e TypeScript per i frontend, Node.js ed Express per il backend e PostgreSQL come database, è stata progettata per garantire modularità, coerenza dei dati e facilità di manutenzione. Particolare attenzione è stata dedicata agli aspetti di accessibilità, con l'adozione di etichette ARIA, una struttura visiva ad alto contrasto e il supporto alla lettura assistita tramite Web Speech API.

La fase di validazione ha coinvolto sia famiglie sia operatori, comprendendo test tecnici delle API, test end-to-end manuali e sessioni di usability testing basate su task specifici. I risultati sono stati complessivamente molto positivi: tutte le funzionalità critiche hanno mostrato stabilità e le interazioni principali sono state comprendibili anche per utenti con competenze digitali limitate. I punteggi SUS, con valori pari a 87,5 per le famiglie e 89,375 per gli operatori, indicano un livello di usabilità globalmente eccellente.

La piattaforma, nella sua versione attuale, risponde pienamente agli obiettivi iniziali di digitalizzazione, accessibilità e gestione multilingue, risultando idonea per un utilizzo pilota nel contesto clinico per cui è stata pensata.

6.2 Limiti del lavoro

Sebbene la piattaforma rispecchi gli obiettivi funzionali e non funzionali individuati nella fase di analisi, esistono alcune limitazioni che ne delineano i margini di miglioramento.

Un primo limite riguarda il processo di validazione, condotto su un campione ridotto di famiglie e operatori. Le osservazioni raccolte offrono indicazioni qualitative di grande valore, ma non permettono di generalizzare in modo completo l'efficacia del sistema a tutta la popolazione potenzialmente interessata. Una valutazione più ampia, condotta su un numero maggiore di utenti e in contesti diversi, permetterebbe di verificare la stabilità dell'esperienza d'uso in scenari più variabili.

Dal punto di vista funzionale, la piattaforma si concentra sulla raccolta e sulla consultazione dei dati, senza includere strumenti di analisi automatica, calcolo di punteggi o generazione strutturata di report clinici. L'interpretazione delle risposte resta quindi interamente affidata all'operatore, che deve ricorrere a strumenti esterni per valutare eventuali indicatori di rischio linguistico.

Per quanto concerne il supporto multilingue, il sistema consente già la gestione di numerose lingue attraverso template localizzati. Tuttavia, non tratta ancora in modo specifico le varianti dialettali, le varietà regionali o le forme miste di bilinguismo, molto frequenti nelle comunità migranti. Inoltre, il comportamento della sintesi vocale dipende dalle capacità del browser e dal sistema operativo: alcune lingue presentano voci di qualità inferiore o tempi di risposta meno regolari, con un potenziale impatto sull'esperienza d'uso.

Alcune scelte di interfaccia risultano migliorabili sulla base delle osservazioni emerse nei task di validazione. Il sistema di filtraggio dei feedback, ad esempio, si basa sull'identificativo interno della domanda (come `s3_q2`), che non è intuitivo per gli operatori, i quali farebbero riferimento più facilmente al testo della domanda o alla sezione del questionario. Allo stesso modo, alcune interazioni della dashboard — come la selezione della riga nelle tabelle o la distinzione tra anteprima e modifica nell'editor — richiedono un breve apprendimento iniziale.

Infine, la piattaforma è stata progettata e validata all'interno di un singolo contesto clinico, caratterizzato da flussi di lavoro specifici. Pur essendo questo un punto di forza in termini di aderenza alle pratiche reali, l'adozione in altri reparti o strutture sanitarie potrebbe richiedere adattamenti dei processi o dei template utilizzati.

Nel complesso, tali limiti non compromettono la solidità e l'efficacia del sistema, ma suggeriscono direzioni concrete per le evoluzioni future della piattaforma.

6.3 Sviluppi futuri

Le possibili evoluzioni del sistema riguardano sia l'arricchimento delle funzionalità, sia il consolidamento dell'esperienza d'uso sulla base dei risultati della validazione.

Una prima area di sviluppo riguarda l'introduzione di strumenti di analisi automatica dei dati, come il calcolo di punteggi o indicatori sintetici associati a specifiche aree del linguaggio. L'integrazione di tali strumenti permetterebbe agli operatori di ottenere rapidamente un quadro preliminare delle risposte fornite, riducendo il tempo necessario per la valutazione manuale. In parallelo, potrebbe essere utile implementare un sistema di generazione automatica di report clinici, sia in forma strutturata (PDF riepilogativi), sia in prospettiva mediante tecniche basate su modelli linguistici di grandi dimensioni (LLM) per produrre sintesi testuali più articolate.

Sul versante linguistico, l'ampliamento del supporto a nuove lingue potrebbe essere affiancato da una gestione più articolata delle varianti linguistiche e dialettali, eventualmente integrando strumenti di supporto alla traduzione o workflow dedicati per la revisione dei contenuti multilingua.

L'esperienza utente può essere affinata intervenendo su alcuni elementi emersi nei task di validazione. Il sistema di gestione dei feedback potrebbe essere riprogettato per permettere la ricerca tramite testo o tramite sezione di appartenenza della domanda, anziché tramite identificativi interni. L'editor dei questionari potrebbe includere meccanismi più espliciti per distinguere anteprima e modifica, mentre la dashboard potrebbe beneficiare di interazioni più ampie e di micro-feedback visivi per facilitare la selezione degli elementi.

Infine, la piattaforma potrebbe essere estesa con funzionalità opzionali legate alla comunicazione con le famiglie, come meccanismi di reminder sulle compilazioni incomplete o notifiche interne alla dashboard per segnalare nuove attività da parte degli utenti.

In sintesi, gli sviluppi futuri mirano a potenziare le capacità analitiche, ampliare la copertura linguistica e migliorare ulteriormente l'accessibilità e la chiarezza dell'esperienza utente. Queste direzioni evolutive, combinate con ulteriori cicli di validazione, potranno rendere la piattaforma uno strumento ancora più completo, flessibile e adattabile alle esigenze cliniche.

Bibliografia

[1] ISTAT. *Popolazione residente e stranieri in Italia – Dati demografici 2023*. Sito web ISTAT. 2023. URL: https://www.istat.it (cit. a p. 1).

[2] TorinoClick. *Osservatorio Stranieri 2022, Città Metropolitana di Torino: 9,52% stranieri; 14,2% minori stranieri*. Portale istituzionale TorinoClick. 2023. URL: https://www.torinoclick.it (cit. a p. 1).

[3] P. Bonifacci. *I bambini bilingui: Favorire gli apprendimenti nelle classi multiculturali*. Roma: Carocci, 2018 (cit. a p. 1).

[4] M. C. Caselli e P. Rinaldi. *Valutazione e presa in carico di bambini figli di migranti: Metodi e strumenti per l'identificazione di uno sviluppo atipico del linguaggio in età prescolare*. Trento: Erickson, 2024 (cit. a p. 1).

[5] A. Marini e S. Vicari. *I disturbi del linguaggio in età evolutiva: Caratteristiche, diagnosi e trattamento*. Bologna: Il Mulino, 2022 (cit. a p. 2).

[6] Parlamento Europeo. *La politica linguistica dell'Unione Europea*. Scheda tematica – Parlamento Europeo. 2020. URL: https://www.europarl.europa.eu/factsheets/it/sheet/142/la-politica-linguistica (cit. a p. 3).

[7] G. Gintoli. *Progetto formativo - Bilinguismo: definizioni, rcierca e stato dell'arte*. Slide Corso ECM - ASL CN2. 2015. URL: https://unric.org/it/agenda-2030/ (cit. a p. 3).

[8] InfoQ. *Java vs Node.js Performance Comparison*. Analisi comparativa delle prestazioni tra Java e Node.js pubblicata su InfoQ. 2024. URL: https://apuravchauhan.medium.com/node-js-vs-java-web-performance-benchmark-analysis-scaling-insights-de2ce3998d18 (cit. a p. 8).

[9] JetBrains s.r.o. *The State of Developer Ecosystem 2024*. Rapporto annuale sullo stato dell'ecosistema degli sviluppatori, JetBrains. 2024. URL: https://www.jetbrains.com/lp/devecosystem-2024/ (cit. a p. 8).

[10] Apurav Chauhan. *Node.js vs Java Web Performance Benchmark Analysis & Scaling Insights*. Articolo pubblicato su Medium – Analisi comparativa delle prestazioni tra Node.js e applicazioni Java Spring Boot. 2024. URL: `https://apuravchauhan.medium.com/node-js-vs-java-web-performance-benchmark-analysis-scaling-insights-de2ce3998d18` (cit. a p. 8).

[11] Stack Overflow. *Stack Overflow Developer Survey 2025*. Rapporto annuale sulla community degli sviluppatori – Stack Overflow. 2025. URL: `https://survey.stackoverflow.co/2025/` (cit. a p. 8).

[12] Microsoft Corporation. *Cloud storage vs. on-premises servers: 9 things to keep in mind*. Articolo tecnico Microsoft Business Insights. 2020. URL: `https://www.microsoft.com/en-us/microsoft-365/business-insights-ideas/resources/cloud-storage-vs-on-premises-servers` (cit. a p. 9).

[13] Asif Ali, Irfan Ahmed Kandhro et al. *Systematic Analysis of On-Premise and Cloud Services*. International Journal of Cloud Computing, Vol. 13, No. 3. 2024. URL: `https://www.researchgate.net/publication/379970310_Systematic_Analysis_of_On_Premise_and_Cloud_Services` (cit. a p. 9).

[14] React Documentation. *Introducing JSX and Virtual DOM*. Documentazione ufficiale di React. URL: `https://react.dev/learn` (cit. a p. 22).

[15] React Documentation. *Hook Reference*. Documentazione ufficiale di React. URL: `https://react.dev/reference/react` (cit. a p. 22).

[16] Vite Documentation. *Getting Started*. Vite official documentation website. URL: `https://vitejs.dev/guide/` (cit. a p. 24).

[17] Vite Documentation. *Development Server and Hot Module Replacement*. Vite official documentation website. URL: `https://vitejs.dev/guide/features.html` (cit. a p. 25).