UNIVERSITÀ DEGLI STUDI DI CATANIA

Dipartimento di Matematica e Informatica

Corso di Laurea Magistrale in Informatica

Petronio Petronio Davide

Smart Hospital

Relazione progetto Ingegneria dei Sistemi Distribuiti

Anno Accademico 2025 - 2026

Indice

1 Architettura del Sistema **3**
1.1 Server di Autenticazione 4
1.2 Server Dati e Logica di Business 5
1.2.1 PazienteService 6
1.2.2 StaffService 7
1.2.3 AdminService 7
1.3 Server Web 8
1.3.1 Comunicazione e Data Transfer Object (DTO) 9
1.3.2 Proxy Pattern e Resilienza 9
1.3.3 Gestione delle Sessioni 10

2 Interfaccia Utente e Casi d'Uso **11**
2.1 Accesso al Sistema 11
2.2 Dashboard Amministratore (Admin) 12
2.3 Dashboard Personale Sanitario (Staff) 12
2.4 Portale Paziente 14

Introduzione

L'idea alla base di questo progetto è sviluppare "Smart Hospital", un sistema distribuito pensato per gestire in modo semplice ed efficiente un ente ospedaliero. L'obiettivo è fornire a tutte le entità coinvolte (pazienti, personale sanitario e amministrativo) un'interfaccia comoda per svolgere le proprie attività quotidiane.

Nella pratica, il sistema offre allo staff una piattaforma per snellire il lavoro di tutti i giorni: dai ricoveri all'assegnazione dei turni, fino alla prescrizione degli esami clinici. Per i pazienti, invece, è stato pensato un portale che permette di consultare in tempo reale la propria cartella clinica e i referti.

Dal punto di vista architetturale, il sistema è stato strutturato secondo un modello N-Tier per garantire un elevato disaccoppiamento tra l'interfaccia utente, la logica di business e la persistenza dei dati. Il Presentation Layer è stato sviluppato utilizzando il framework Spring Boot MVC, mentre la comunicazione tra i nodi distribuiti del backend si basa su Java RMI. Infine, per soddisfare i requisiti di sicurezza e manutenibilità, l'implementazione fa uso di specifici Design Pattern per la gestione delle sessioni e adotta una logica di controllo degli accessi basata sui ruoli.

Capitolo 1

Architettura del Sistema

Il sistema è stato progettato con l' intento di disaccoppiare la persistenza dei dati dalla logica di business e dall'interfaccia utente. Si è optato per una separazione tra il Web Server e i moduli applicativi di backend. Questa scelta è dettata dalla volontà di rendere il sistema quanto più scalabile possibile poiché, in contesti reali, si può voler successivamente aggiungere una nuova tipologia di client come, ad esempio, un'applicazione per smartphone. Così facendo, i server che si occupano di interagire con il database non necessiteranno di alcuna modifica rilevante all'aggiunta di una nuova tipologia di client.

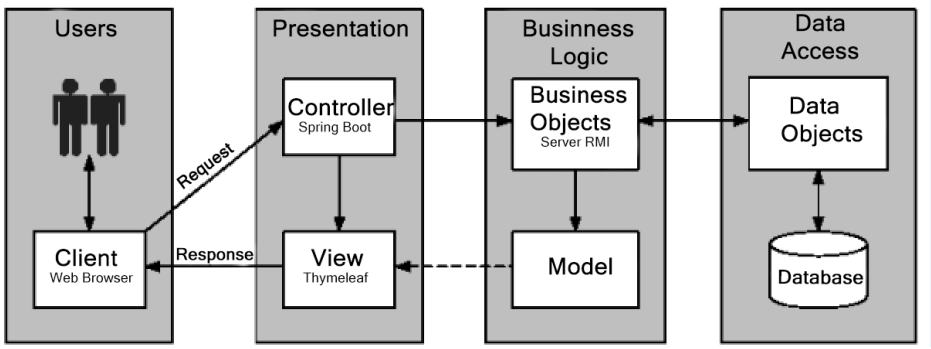
Per soddisfare questi requisiti, l'architettura generale è stata strutturata secondo il modello N-Tier a 3 livelli:

1. **Data Layer:** Costituisce il livello di persistenza affidato a un DBMS relazionale. Al suo interno risiedono tutte le informazioni anagrafiche, cliniche e di sistema necessarie per lo svolgimento delle funzionalità dell'infrastruttura.

2. **Business Logic Layer:** Rappresenta il fulcro del sistema in cui risiede l'intera logica di dominio. È suddiviso in due moduli server principali, le cui interfacce sono esposte tramite Java RMI: il servizio di autenticazione e il servizio di gestione dei dati. Il primo si occupa di validare le identità e rilasciare i token di sessione. Il secondo, invece, fornisce ai client le operazioni che interagiscono con il Data Layer. Quest'ultimo modulo opera con il servizio di autenticazione per garantire che qualsiasi operazione invocata sia autorizzata, implementando un rigoroso controllo degli accessi basato sui permessi.

3. **Presentation Layer:** Sviluppato utilizzando il framework Spring Boot, fornisce agli utenti finali un'interfaccia grafica per effettuare l'accesso

e interagire con le funzionalità a loro consentite. Questo livello è progettato per essere agnostico rispetto alla logica di business: comunica con i server applicativi agendo da client RMI e scambia le informazioni esclusivamente tramite appositi oggetti di trasporto (Pattern DTO - Data Transfer Object), garantendo un disaccoppiamento ottimale tra frontend e backend.



1.1 Server di Autenticazione

Il Server di Autenticazione è il componente implementato come nodo remoto tramite la tecnologia **Java RMI**, che fornisce e gestisce l'accesso all'interno del sistema. Esso espone un'interfaccia pubblica `IAuthService` che permette ai client di invocare le procedure di login a distanza in modo del tutto trasparente. Solamente dopo essersi autenticati con successo tramite questa interfaccia remota è possibile interagire con il Server Dati per effettuare le varie operazioni. Questa scelta è funzionale alla gestione degli accessi basata sui ruoli (RBAC) poiché, all'interno del database, a ogni utente è associato uno specifico ruolo tra: "doctor", "nurse", "admin" e "patient". Questi ruoli sono fondamentali al fine della sicurezza del sistema, poiché solo possedendo un certo ruolo è possibile effettuare determinate operazioni.

Dunque, il Server di Autenticazione fornisce ai client le seguenti operazioni principali:

- **tryLogin**: Operazione tramite la quale, fornendo username e password, è possibile effettuare l'accesso. In caso di successo, viene generato e restituito un token di sessione, salvato in maniera persistente sul database per i riconoscimenti futuri.
- **invalidateSession**: Operazione tramite la quale è possibile invalidare una sessione autenticata. Risulta utile non solo per il semplice logout dell'utente ma, anche nei casi in cui, per una qualunque ragione

di sicurezza, si renda necessario revocare forzatamente l'accesso a un utente.

- **getValidSessionDTO**: Operazione che, a partire da un token, ne verifica l'autenticità e la validità temporale restituendo le informazioni e i permessi a esso associati.
- **creaUtenteAuth**: Operazione che permette la creazione di un nuovo utente all'interno del database contenente le credenziali.
- **eliminaUtenteAuth**: Operazione che permette l'eliminazione definitiva di un utente e, a cascata, invalida tutte le sue sessioni attive per impedirne futuri accessi.
- **getListaUtentiAuth**: Operazione tramite la quale è possibile ottenere la lista di tutti gli utenti registrati nel sistema con il rispettivo ruolo associato.

Questa serie di operazioni risulta indispensabile ai fini del corretto utilizzo del sistema. Senza di queste non sarebbe possibile effettuare un controllo di autenticità e, di conseguenza, chiunque potrebbe eludere la sicurezza e inviare richieste non autorizzate al server di gestione dei dati.

1.2 Server Dati e Logica di Business

Il Server Dati rappresenta il fulcro operativo del sistema "Smart Hospital". Mentre il server di autenticazione si limita a riconoscere le identità, questo componente contiene tutta la logica necessaria per interagire con il database dell'ospedale.

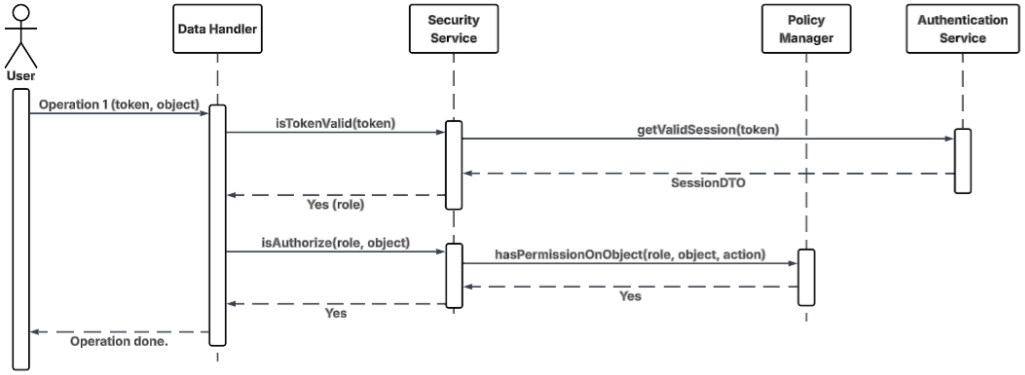
Per ottimizzare le comunicazioni di rete e nascondere la complessa ramificazione dei servizi interni al client, il server implementa il Design Pattern **Remote Facade**. Sfruttando la tecnologia Java RMI, la classe `DataHandler` espone verso l'esterno un'unica interfaccia remota `IDataService` che funge da punto d'accesso centralizzato. Il client non ha bisogno di conoscere l'architettura interna del backend poiché si limita a invocare semplicemente le operazioni sul `DataHandler`, il quale, si occuperà di smistare le chiamate in modo trasparente e sicuro ai vari servizi specializzati interni quali: `AdminService`, `StaffService` e `PazienteService`.

Ogni richiesta in arrivo ai servizi viene prima validata tramite `SecurityService`. Questo componente esegue due passaggi cruciali:

1. **Validazione del Token:** Interroga il Server di Autenticazione per verificare che il token di sessione fornito sia ancora valido e non scaduto.

2. **Verifica dell'Autorizzazione (RBAC):** Estrae il ruolo dell'utente (es. "doctor" o "patient") e consulta le policy di sicurezza nella classe `PolicyManager` per determinare se possiede il permesso specifico (es. Permesso di scrittura) sull'oggetto protetto richiesto (es. Cartelle cliniche o esami).

Solo se entrambi i controlli hanno esito positivo, il server procede con l'esecuzione dell'operazione sul database o altrimenti, l'azione viene bloccata e viene restituita un'eccezione di accesso negato. Questo garantisce che, ad esempio, un paziente non possa mai modificare l'esito di un esame o che un infermiere non possa eliminare utenti dal sistema.



1.2.1 PazienteService

Il servizio `PazienteService` si occupa principalmente di tutte quelle operazioni che riguardano il paziente. Tra queste, troviamo:

- **ccPaziente**: Restituisce i dati generali della cartella clinica di uno specifico paziente.
- **esamiPaziente**: Restituisce la lista di tutti gli esami prescritti e associati al paziente.
- **PazientiReparto**: Fornisce la lista di tutti i pazienti attualmente ricoverati in un determinato reparto.
- **esamePaziente**: Restituisce i dettagli e l'esito di un singolo esame specifico.
- **aggiornaSoddisfazione**: Permette di aggiornare l'indice di gradimento del paziente relativo alle cure ricevute.

Tutte queste funzionalità sono liberamente accessibili al personale medico verso qualunque paziente. Al contrario, qualora il richiedente sia un paziente stesso, l'accesso è limitato ai soli dati personali, implementando un vincolo fondamentale per non violare la privacy degli altri utenti del sistema.

1.2.2 StaffService

Il servizio `StaffService` espone le operazioni cliniche e organizzative dedicate al personale sanitario, ovvero agli utenti con ruolo "doctor" o "nurse". Tra queste, abbiamo:

- **dimettiPaziente**: Operazione che permette di registrare la dimissione di un paziente, aggiornandone lo stato all'interno del sistema.
- **prenotaEsame**: Consente di prenotare un nuovo esame clinico per un paziente specifico.
- **aggiornaEsito**: Permette di inserire o modificare il referto di un esame una volta che questo è stato eseguito.
- **turniStaff**: Funzionalità che permette ai membri dello staff di consultare la propria turnazione relativa alla settimana corrente e alle tre settimane successive.

È bene notare come il sistema operi un controllo degli accessi anche all'interno del personale sanitario stesso poiché operazioni critiche come `dimettiPaziente`, `prenotaEsame` e `aggiornaEsito` sono rigorosamente accessibili solo in qualità di medico. In questo modo, il sistema impedisce al personale infermieristico di compiere azioni al di fuori delle proprie competenze, rispettando le policy del suo ruolo.

1.2.3 AdminService

Infine, il servizio `AdminService` si occupa di fornire agli amministratori del sistema tutte le operazioni necessarie per il monitoraggio dei reparti e per la gestione del personale e delle anagrafiche. Un compito cruciale di questo servizio è garantire la perfetta sincronizzazione tra il server di autenticazione (che detiene le credenziali) e il server dati (che contiene le tabelle operative di staff e pazienti).

Le operazioni fornite, accessibili esclusivamente con il ruolo "admin", sono:

- **getStatisticheReparti**: Permette di recuperare i dati statistici storici su base settimanale relativi alle performance dei vari reparti (es. richieste totali, letti disponibili, soddisfazione media).
- **creaPaziente**: Gestisce la registrazione in ingresso di un nuovo paziente. Crea contestualmente sia le credenziali di accesso nel sistema Auth (assegnando il ruolo "patient"), sia la cartella clinica iniziale nel database operativo locale.
- **getListaUtenti**: Interroga il server di autenticazione per fornire all'amministratore l'elenco completo degli account registrati e dei relativi ruoli.
- **creaUtente**: Operazione adibita all'assunzione e registrazione di nuovo personale ospedaliero (medici, infermieri o altri admin). Si occupa di creare il profilo di login e di allinearlo istantaneamente con l'anagrafica della tabella `staff`.
- **eliminaUtente**: Permette la rimozione sicura e completa di un profilo dal sistema. Per preservare l'integrità referenziale, l'operazione elimina prima i dati anagrafici dal database locale (cancellando a cascata i record collegati) e successivamente revoca in via definitiva le credenziali dal server di autenticazione.
- **getLiveStats**: Calcola e restituisce in tempo reale le statistiche operative attuali dell'ospedale, interrogando dinamicamente il database per estrapolare il numero di posti letto occupati al momento e la media del livello di gradimento dei pazienti correntemente ricoverati.

Grazie a questo servizio, gli amministratori dispongono di un pannello di controllo completo che centralizza la gestione degli accessi, delegando al sistema la complessità di mantenere coerenti i dati distribuiti sui vari nodi del sistema.

1.3 Server Web

Il Server Web sviluppato utilizzando il framework Spring Boot, funge a tutti gli effetti da client per i servizi di backend. Questo si basa sul pattern architetturale **MVC (Model-View-Controller)**. I vari *Controller* (come `LoginController`, `AdminController`, `StaffController` e `PazienteController`) si occupano di intercettare le richieste HTTP degli utenti, estrapolare i parametri dai form web e delegare l'esecuzione effettiva

della logica di business ai server remoti. Ottenuta la risposta dai backend, i controller popolano il Model e viene successivamente restituita la View appropriata che sarà composta da pagine HTML che sfruttano Thymeleaf.

1.3.1 Comunicazione e Data Transfer Object (DTO)

Poiché il Web Server e i server applicativi risiedono su nodi separati, la comunicazione è soggetta allo scambio per mezzo della rete e, per ottimizzare lo scambio di informazioni ed evitare di trasferire oggetti pesanti o strettamente legati al database, è stato adottato il pattern **Data Transfer Object (DTO)**.

I dati viaggiano incapsulati in classi serializzabili progettate ad-hoc (come `SessionDTO`, `PazienteDTO`, `EsameDTO`, `LiveStatsDTO`), che contengono esclusivamente gli attributi necessari alla presentazione. Questo approccio garantisce un totale disaccoppiamento poiché il Web Server ignora la struttura relazionale del database e lavora solo con oggetti "piatti". In questo modo, il traffico di rete generato da RMI viene ridotto grazie anche alla semplicità dei DTO.

1.3.2 Proxy Pattern e Resilienza

Il collegamento tra il Web Server e i backend avviene tramite protocollo Java RMI. In questo contesto, il Web Server sfrutta il **Proxy Pattern** poiché, attraverso l'interfaccia di configurazione `RmiConfig`, il client ottiene uno "Stub" locale che implementa le interfacce `IAuthService` e `IDataService`. Il Web Server successivamente invoca i metodi su questi Proxy come se fossero oggetti locali, mascherando del tutto la complessità della comunicazione di rete ai Controller.

Un aspetto cruciale di questa implementazione è la gestione della connessione ai registry RMI, progettati secondo una logica di **Lazy Initialization** all'interno della classe `RmiConfig`. All'avvio del Web Server, le connessioni verso i server di backend non vengono stabilite immediatamente, al contrario, la ricerca del Proxy avviene solo nel momento in cui un Controller richiede effettivamente l'accesso al servizio.

Questa scelta è stata effettuata principalmente per ottenere:

1. **Avvio indipendente:** Il Web Server può avviarsi con successo anche se i server RMI non sono ancora online al momento del boot.

2. **Fault Tolerance:** In caso di disconnessione o caduta momentanea di un server di backend, la logica di inizializzazione ritardata permette al sistema di tentare un nuovo collegamento non appena viene invocato

nuovamente il servizio. Qualora un server andasse giù, basterà resettare il riferimento al Proxy per forzare una riconnessione automatica alla richiesta successiva garantendo, così, che il collegamento ai server possa essere ristabilito senza riavviare il client web.

1.3.3 Gestione delle Sessioni

Un'ulteriore implementazione rilevante all'interno del Web Server riguarda la gestione delle sessioni utente e la loro resilienza. Inizialmente, lo stato di autenticazione era legato esclusivamente alla memoria volatile (RAM) del web server, comportando la disconnessione di tutti gli utenti in caso di riavvio del servizio.

Per risolvere questo problema e rendere il sistema più robusto, è stato implementato un meccanismo di controllo preventivo sulle richieste HTTP in ingresso. Qualora la sessione in memoria risulti assente (ad esempio, subito dopo un riavvio del Web Server), il sistema effettua un tentativo di recupero: va a leggere un apposito cookie salvato lato client contenente l'ultimo token rilasciato all'utente. Questo token viene immediatamente inviato al Server di Autenticazione per essere convalidato tramite RMI e, se risulta ancora attivo, il Web Server ricostruisce la sessione locale in modo del tutto trasparente. Questa logica evita disconnessioni e rende il frontend tollerante ai riavvii temporanei.

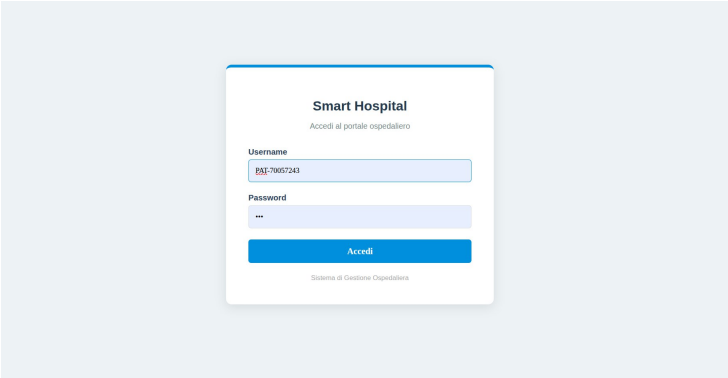
Capitolo 2

Interfaccia Utente e Casi d'Uso

Per validare il sistema di backend e le scelte descritte nei capitoli precedenti, è stato sviluppato un client web completo e responsivo tramite il framework Spring Boot. Di seguito vengono presentati i principali casi d'uso del sistema attraverso le interfacce utente dedicate ai vari ruoli.

2.1 Accesso al Sistema

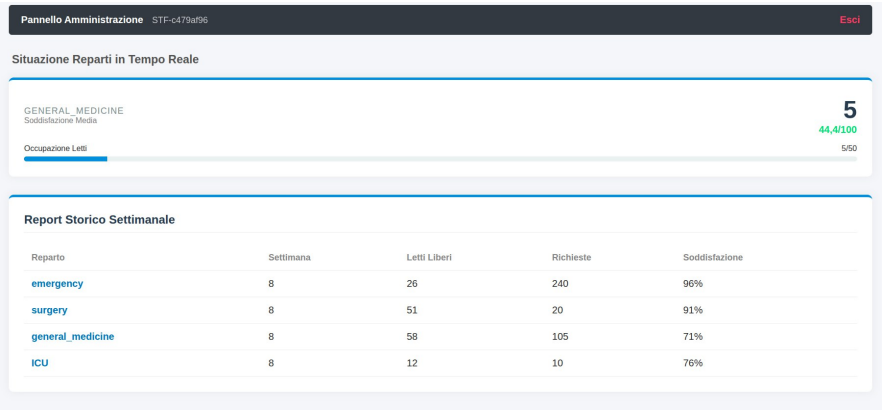
La schermata di login rappresenta il singolo punto di ingresso per tutti gli utenti del sistema. Inserendo le proprie credenziali, la richiesta viene inviata al Server di Autenticazione. Una volta generato e validato il token di sessione, il sistema reindirizza automaticamente l'utente verso la dashboard dedicata al proprio ruolo.



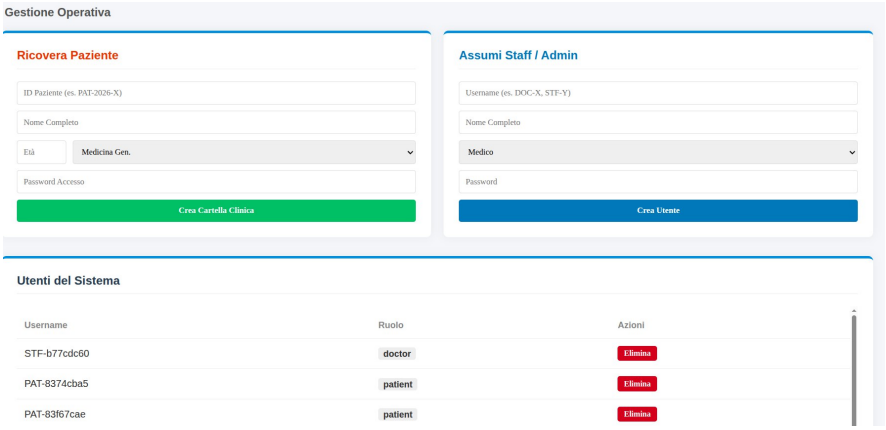
Si segnala il fatto che, se l'utente prova ad andare alla schermata di login quando possiede ancora un token valido, allora questo viene indirizzato automaticamente alla dashboard bypassando il login.

2.2 Dashboard Amministratore (Admin)

L'interfaccia dedicata agli amministratori fornisce un pannello di controllo. Come si evince dall'interfaccia, l'amministratore ha a disposizione gli strumenti per monitorare in tempo reale le statistiche di occupazione dei reparti e analizzare i dati storici. Inoltre, la vista integra i form necessari per l'assun-

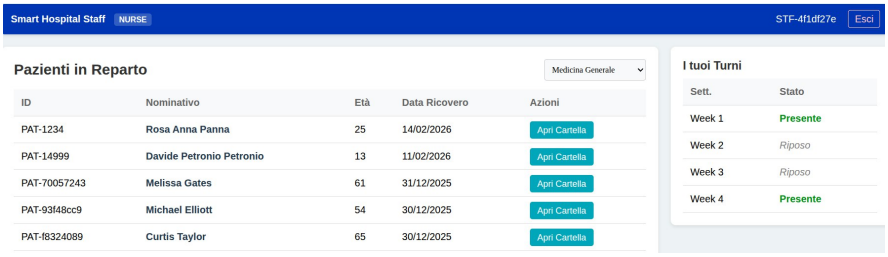


zione di nuovo personale e la registrazione di nuovi pazienti, interfacciandosi, in background, con l'`AdminService`.

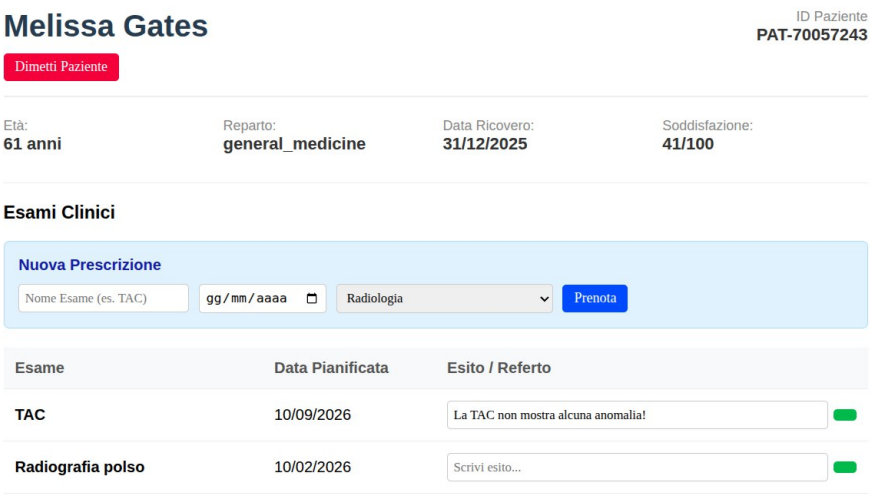


2.3 Dashboard Personale Sanitario (Staff)

L'interfaccia per il personale medico (Doctor e Nurse) è progettata per ottimizzare la gestione del reparto. L'utente può visualizzare la lista dei pazienti attualmente assegnati al proprio reparto e consultare la propria turnazione lavorativa.



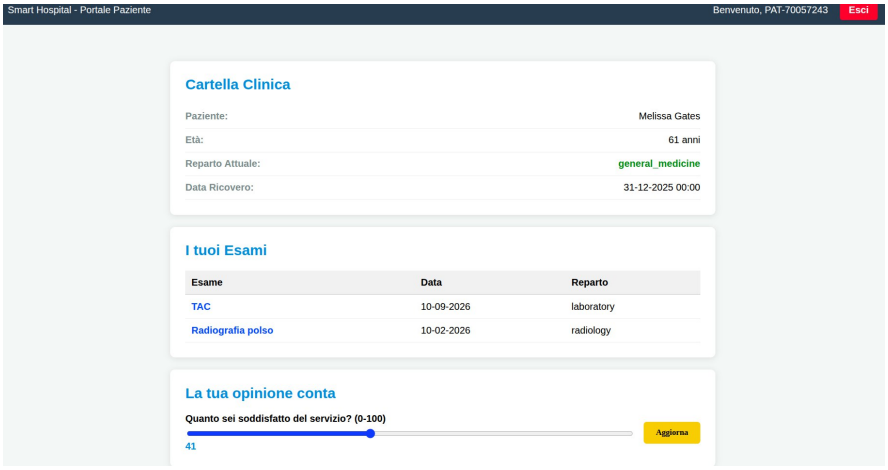
Entrando nel dettaglio di una cartella clinica, l'interfaccia si adatta dinamicamente alle policy RBAC descritte in precedenza: se l'utente è autenticato come medico , il frontend sblocca e mostra le azioni critiche come il form per prescrivere esami, aggiornare i referti e dimettere il paziente.



Qualora l'accesso avvenga come infermiere, tali funzionalità non risultano presenti.

2.4 Portale Paziente

Il paziente, una volta effettuato l'accesso, è in grado di visualizzare esclusivamente il riepilogo della propria cartella clinica personale e lo storico degli esami prescritti. Inoltre, l'interfaccia fornisce un modulo per inviare un feedback sul livello di soddisfazione delle cure ricevute, dato che andrà ad alimentare le statistiche di reparto gestite dal sistema.



Infine, è possibile per il paziente visualizzare tutti i riepiloghi dei suoi esami nel dettaglio.