POLITECNICO DI TORINO
Corso di Laurea in Ingegneria Informatica

Tesi di Laurea Magistrale

# Cartella clinica informatizzata

Sviluppo applicazione web

**Relatore**
Prof. Maurizio Morisio

**Studente**
Carmelo Metallo
matricola: 177895

**Supervisore aziendale**
**SAN s.r.l.**
Dott. Davide Trombini

Anno accademico 2020-2021

# Sommario

La cartella clinica informatizzata si presenta come un raccoglitore di informazioni sicuro, real-time, incentrato sul paziente fruibile da medici, infermieri, terapisti e tutti gli operatori sanitari e amministrativi che ne possono fare accesso [1].

È uno strumento che aiuta il processo decisionale dei medici, fornendo accesso a tutte le informazioni del paziente dove e quando ne hanno bisogno. La cartella informatizzata automatizza e semplifica il flusso di lavoro degli operatori clinici, limitando i ritardi e le interruzioni nella cura.

Tale strumento supporta la raccolta di dati diversi dall'assistenza clinica diretta, come la fatturazione, la gestione della qualità, la segnalazione dei risultati, la pianificazione delle risorse.

In Italia si ha una buona diffusione nella digitalizzazione della cartella clinica se ci si ferma a guardare gli esami specialistici e di diagnostica. Come evidenziato da Chiara Sgarbossa (direttore dell'Osservatorio Innovazione Digitale in Sanità del Politecnico di Milano), quando si va più nel dettaglio, ci sono degli aspetti ancora poco affrontati. Tali aspetti riguardano soprattutto la prescrizione e la somministrazione di farmaci [2] oltre ad aspetti più specifici come le anamnesi, le condizioni sociali, le relazioni dei vari operatori sanitari.

Lo scopo di questa tesi è quello di contribuire allo sviluppo di una cartella informatizzata completa ed è strutturata nel seguente modo:

- il primo capitolo introduce l'argomento e presenta i soggetti partecipanti al progetto;
- il secondo capitolo presenta l'analisi di mercato svolta nella fase embrionale del progetto;
- il terzo capitolo raccoglie gli attributi e i requisiti funzionali che deve avere una cartella clinica informatizzata;
- il quarto capitolo pone l'attenzione sull'infrastruttura scelta e sul framework utilizzato per lo sviluppo della piattaforma;
- il quinto capitolo approfondisce la composozione del nucleo della cartella informatizzata sviluppata;
- il sesto capitolo presenta i plugin che permettono alla cartella informatizzata di avere sempre più dati, con un approfondimento sul plugin che riguarda le valutazioni mediche;

- l'ultimo capitolo espone le considerzioni finali e i possibili sviluppi futuri del presente progetto.

# Indice

# Elenco delle tabelle

# Elenco delle figure

# Capitolo 1

# Introduzione

## 1.1 Che cos'è la cartella clinica informatizzata

La *cartella clinica* si può definire come il fascicolo nel quale si raccolgono i dati anamnestici e gli obiettivi riguardanti il paziente ricoverato, quelli giornalieri sul decorso della malattia, i risultati delle ricerche e delle analisi effettuate, quelli delle terapie praticate e infine le diagnosi della malattia che ha condotto il paziente in ospedale. Questo strumento è finalizzato a:

- fornire una base informativa documentando il quadro clinico, il processo diagnostico-terapeutico e i risultati conseguiti;
- consentire la tracciabilità delle attività svolte, rispetto alle modalità della loro esecuzione, della responsabilità delle azioni e della cronologia delle stesse;
- facilitare l'integrazione di competenze multiprofessionali nel processo diagnostico-terapeutico;
- costituire una fonte informativa per ricerche clinico-scientifiche, per la formazione degli operatori, per esigenze amministrative e gestionali;
- permettere l'esericizio di diritti e la tutela di legittimi interessi della persona assistita, dell'ospedale e dei professionisti che agiscono in suo nome.

Con il termine di *cartella clinica informatizzata* si individuano tutte le soluzioni di Information e Communication Technology (ICT) a supporto della relazione tra il personale sanitario e il paziente, in ambito di reparto ospedaliero, ambulatoriale o domiciliare.

Adottare una soluzione completamente digitale permette di cambiare le abitudini delle persone nella redazione della cartella clinica, la standardizzazione documentale (i dati clinici vengono condivisi da tutto l'ospedale e quindi devono essere capiti da tutti) e la reingegnerizzazione dei processi (cambia la modalità di inserimento e di utilizzo dei dati).

La cartella clinica informatizzata non è semplicemente la trasformazione della documentazione cartacea in digitale, ma deve poter abilitare la gestione evoluta dell'informazione e ogni potenzialità dei sistemi di supporto alla decisione [3].

## 1.2 Cartella clinica cartacea e cartella clinica informatizzata a confronto

Il passaggio da un cartella clinica cartacea a una cartella clinica informatizzata porta con sé una serie di cambiamenti in termini di organizzazione, condivisione dell'informazione e accessibilità (tabella 1.1).

L'utilizzo della cartella clinica informatizzata consente di ottenere benefici in termini di tempo, di salute, ambientali e finanziari [4].

**Vantaggi in termini di tempo:** gli specialisti trascorrono 50 o più ore alla settimana nella cura diretta del paziente, mentre i medici di base trascorrono dalle 30 alle 40 ore a causa della quantità elevata di pratiche burocratiche e di doveri amministrativi che devono assolvere.

Gli infermieri che utilizzano la cartella clinica informatizzata hanno notato una riduzione del tempo di documentazione pari al 45%. L'utilizzo di computer, tablet e smartphone per inserire i dati del paziente aumenta la completezza delle informazioni, quindi il tempo dedicato dal personale alla ricerca di dati mancanti diminuisce sensibilmente.

**Benefici alla salute:** negli Stati Uniti, circa 4,5 milioni di visite ambulatoriali sono relative a eventi avversi da farmaci errati; 400.000 di questi casi sfociano in ricoveri ordinari.

Tali eventi potrebbero essere evitati utilizzando la cartella clinica informatizzata per tenere traccia della storia dei farmaci prescritti al paziente con relative allergie, anamnesi e cure ricevute.

**Benefici sull'ambiente:** ogni visita a un paiznete richiede in media 10 - 13 fogli di carta; una buona parte di medici visita da 50 a 100 pazienti a settimana. Questo vuol dire che ogni medico consuma in media 975 fogli di carta ogni settimana.

Ogni anno il settore sanitario consuma migliaia di tonnellate di carta, causando problemi di conservazione e danni ambientali.

**Vantaggi finanziari:** le strutture ambulatoriali che adottano e utilizzano la cartella clinica informatizzata in 15 anni potrebbero avere un risparmio pari a 142 miliardi di dollari; le strutture ospedaliere che adottano e utilizzano la cartella clinica informatizzata nello stesso periodo avrebbero un risparmio pari a 371 miliardi di dollari.

Nel 2009, gli studi medici indipendenti che utilizzavano una cartella clinica informatizzata hanno registrato un incremento pari a circa 50.000 dollari in più rispetto agli studi che utilizzavano sistemi cartacei tradizionali.

Dopo 5 anni di utilizzo della cartella clinica informatizzata, si può registrare un margine operativo del 10,1% superiore rispetto alle pratiche nel loro primo anno di utilizzo.

| | **Cartella clinica cartacea** | **Cartella clinica informatizzata** |
|---|---|---|
| **Organizzazione** | Le registrazioni cartacee vengono disperse tra le varie strutture mediche, sono spesso incomplete e contribuiscono alla ripetizione di trattamenti e test che altrimenti non sarebbero necessari. Tali registrazioni risultano inefficienti perché chi prende in carico il paziente deve recuperare le informazioni sullo stesso mettendosi in contatto con diversi uffici | La digitalizzazione riduce la ridondanza tra gli operatori e consente di avere l'informazione completa della storia del paziente in un'unica procedura facilmente accessibile. Una cartella clinica informatizzata completa semplifica la generazione di documentazione longitudinale che può migliorare l'assistenza a lungo termine |
| **Condivisione dei dati** | Nei sistemi con registrazione cartacea è presente un consistente spreco di tempo dovuto al fatto che le informazioni devono essere trasferite via fax o via posta. Tali informazioni non sono archiviate in una posizione centralizzata facilmente accessibile, risulta, così, spesso difficile mettere insieme una storia completa del paziente | Lo scambio di informazioni è più veloce perché il personale degli uffici non deve recuperare le informazioni da diversi posti di stoccaggio e non deve inviarle via posta o via fax, può trasferirle elettronicamente. La cartella clinica informatizzata fornisce l'accesso a informazioni complete sul paziente e in un unico posto |
| **Accessibilità** | L'accesso degli operatori sanitari alle cartelle cliniche è limitato alla posizione e agli orari degli uffici. Questo può avere un impatto devastante sulla salute del paziente in circostanze critiche, come in una procedura di emergenza | Una cartella clinica informatizzata web-based fornisce accesso 24 ore su 24 e 7 giorni su 7 alle cartelle dei pazienti a ai risultati degli esami svolti, da qualsiasi luogo con accesso a Internet. L'uso di dispositivi mobili per accedere alla cartella informatizzata permette di visualizzare e inserire informazioni mentre si è in movimento, durante gli appuntamenti oppure durante le visite in reparto |

Tabella 1.1. Cartella clinica cartacea VS Cartella clinica informatizzata [4]

## 1.3 Il progetto

Il progetto nasce dalla necessità di due clienti di informatizzare i numerosi processi che si attivano al momento del ricovero di un paziente: dall'arrivo dello stesso in struttura con l'accettazione amministrativa alla generazione del foglio unico di terapia, dal consenso

informato alla stesura delle valutazioni in ingresso, dalle consegne in reparto alle rilevazioni dei parametri come temperatura, peso, pressione.

La prima parte del progetto si è articolata nello studio di tali processi. Il team di sviluppo di SAN in collaborazione con i referenti dei due clienti, hanno analizzato tutte le operazioni eseguite dai vari operatori nelle diverse fasi del ricovero del paziente. L'analisi dei processi è stata svolta intervistando gli operatori (clinici e amministrativi), prendendo in considerazione i documenti cartacei compilati e le procedure informatiche già esistenti. Tale studio si è reso necessario anche per ottimizzare i processi attivi.

Tenendo conto dello studio svolto e delle esigenze avanzate dai clienti, si è svolta l'analisi di mercato dei prodotti già esistenti, di cui sarà presentato un approfondimento nel prossimo capitolo.

Concluse le operazioni di analisi dei punti precedenti è iniziata la fase di ricerca della migliore tecnologia a disposizione in grado di soddisfare i criteri stabiliti con i clienti e di rendere tale progetto un prodotto per SAN. Tra i criteri principali, di cui si parlerà più approfonditamente nel terzo capitolo, ci sono la possibilità di utilizzo dello strumento su qualsiasi piattaforma, la portabilità e la semplicitià nell'uso.

L'ultimo step, quello attualmente in corso, tratta lo sviluppo vero e proprio della Cartella informatizzata. Come verrà spiegato nel capitolo successivo si punta sulla costante interazione con i clienti per avere un prodotto il più possibile aderente alle operazioni svolte quotidianamente dai vari operatori.

In qualità di tesista sono stato coinvolto in tutte le fasi del progetto, in particolare nella scelta della tecnologia e nello sviluppo dei vari plugin necessari.

## 1.4 L'azienda

SAN s.r.l. è un'azienda che da circa 25 anni si occupa di sviluppo software e di consulenza nell'ambito ICT.

La prinicipale area di competenza è basata sulla progettazione e lo sviluppo di software per le imprese.

Nello svolgimento della propria attività SAN punta all'ottimizzazione dei processi aziendali tramite l'analisi delle condizioni esistenti, la mappatura del percorso delle informazioni, l'adozione della tecnologia per l'informatizzazione.

Pur avendo le necessarie certificazioni da parte di alcune aziende nel settore (Apple, IBM, Microsoft, etc.) SAN afferma di scegliere sempre la migliore, applicabile allo specifico caso in esame, prendendo ogni volta in considerazione anche le soluzioni offerte dal mondo opensource che negli ultimi anni sta occupando un ruolo crescente nei diversi progetti, sia come strumento di sviluppo sia come insieme di oggetti funzionali già pronti e integrabili.

Negli anni di attività, SAN si è specializzata soprattutto negli ambiti della sanità, delle utilities e dell'industria aeronautica. L'acquisizione di competenza nella gestione dei processi in questi settori ha portato SAN a ricevere incarichi di IT management presso aziende clienti, con responsabilità gestionali che vanno oltre l'offerta di soluzioni proprie ma anche di analisi, confronto e integrazione con prodotti e servizi di altre aziende presenti sul mercato.


Figura 1.1. SAN s.r.l. - Systems Applications Networking

## 1.5 I clienti

Fautori di questo progetto sono stati, in separata sede, due clienti molto importanti per SAN: la Casa di Cura - CRRF "Mons. Luigi Novarese" di Moncrivello (VC) e il Presidio Sanitario San Camillo di Torino.

La Casa di Cura - CRRF "Mons. Luigi Novarese" fa parte dell' associazione Silenziosi Operai della Croce.

L'Associazione opera quale ONLUS e si propone come obiettivo la cura e l'assistenza dei malati che necessitano di terapia riabilitativa con una particolare attenzione agli aspetti dell'umanizzazione dell'assistenza e alla qualità delle cure offerte. La Casa di Cura pone al centro del suo operato il paziente, che viene assistito e seguito nella complessità totale della persona, considerata negli aspetti fisici, psicologici, affettivi, relazionali e spirituali. Il credo fondamentale della Casa di Cura è che non basta agire solo sul sintomo specifico della malattia, ma la guarigione passa anche attraverso l'attenzione che si dà alla persona considerata nella totalità del suo essere.

Grande attenzione è posta all'educazione professionale e alla formazione del personale sanitario: dall'istruzione universitaria a quella specialistica, fino alla formazione continua di tutti gli operatori. Inoltre, presso il centro, è attivo da anni il Corso di Laurea in Terapia Occupazionale dell'Università Cattolica di Roma, unico Corso universitario del genere in Piemonte [5]

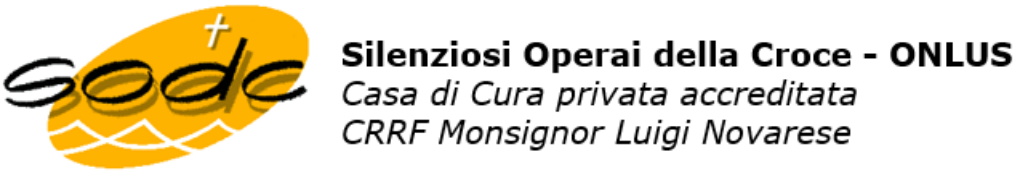


Figura 1.2. Silenziosi Operai della Croce - CRRF "Mons. Luigi Novarese"

Il Presidio Sanitario San Camillo di Torino appartiene all'Ordine dei Chierici Regolari Ministri degli Infermi (camilliani) fondato da San Camillo de Lellis nel 1584 e fa parte delle strutture gestite dalla Fondazione Opera San Camillo istituta nel dicembre del 2008. È un centro ospedaliero specializzato in recupero e rieducazione funzionale, dispone di 100 posti letto ordinari e 20 di day hospital, è dotato di un poliambulatorio e di un servizio di diagnostica per immagini.

I fini istituzionali prioritari del Presidio Sanitario San Camillo sono l'assistenza ospedaliera, l'attività ambulatoriale, la formazione professionale e didattica in ambito universitario e la ricerca scientifica.

I principi fondamentali ai quali questa struttura si richiama nell'erogazione dei servizi sono: *eguaglianza* - i servizi devono essere erogati in modo uguale per tutti nel rispetto dell'articolo 3 della costituzione; *imparzialità* - tutti gli operatori devono tenere un comportamento corretto, obiettivo e imparziale; *continuità* - il servizio deve essere erogato in maniera regolare, integrata e continua; *diritto di scelta* - il cittadino può scegliere di essere curato ovunque sul territorio nazionale; *partecipazione* - al cittadino è garantita l'informazione, la personalizzazione del servizio, la tutela dei suoi diritti; *umanizzazione* - ogni servizio reso al cittadino dev'essere contraddistinto da rispetto, cortesia e disponibilità; *efficienza ed efficacia* - il servizio dev'essere garantito attraverso la migliore utilizzazione delle risorse per il raggiungimento massimo dei risultati in termini di salute [6].


Figura 1.3. Fondazione Opera San Camillo - Presidio Sanitario San Camillo

# Capitolo 2

# Ricerca di mercato

## 2.1 Soggetti coinvolti e analisi delle necessità

L'analisi di mercato per questo progetto è stata portata avanti da addetti della SAN srl in collaborazione con i referenti del progetto delle due strutture ospedaliere, coinvolte in sedi separate. In questa fase iniziale, sono state raccolte informazioni sulle caratteristiche fondamentali che avrebbe dovuto avere il prodotto.

Tali caratteristiche sono:

- Accesso unico a tutte le informazioni del ricovero
- Possibilità di utilizzo su diverse tipologie di dispositivi (smartphone e/o tablet Android o iOS, PC Windows, Linux o iOS)
- Ruoli per accesso controllato alle risorse
- Accesso in modalità esclusiva sul singolo record in fase di modifica
- Versioning di tutte le informazioni, con possibilità di recupero del record e logging
- Semplicità d'uso, l'operatore deve rivolgere la propria attenzione al paziente e non allo strumento utilzzato per gestirlo
- Ricerca del farmaco per nome commerciale e per principio attivo
- Diari giornalieri
- Firma elettronica

## 2.2 Analisi di mercato

A fronte delle caratteristiche fondamentali espresse nel paragrafo precedente, sono stati creati degli indicatori di diversa tipologia (economici, ICT e clinici) per poter procedere con l'analisi di mercato.

Vengono di seguito approfonditi gli indicatori ICT e gli indicatori clinici.

**Indicatori ICT** - Prendono in considerazione i seguenti aspetti tecnici:

- Server locale - è necessario che la struttura fornisca un server locale? L'applicativo è raggiungibile anche dal cloud esterno alla struttura?
- Database - quale database viene utilizzato per la memorizzazione dei dati?
- Dettagli - con che tecnologia o framework è stata sviluppata?
- WEB - può essere utilizzata via WEB?
- Mobile - è stata sviluppata anche una versione per dispositivi mobili quali smartphone o tablet?
- Sicurezza - sono implementati degli accorgimenti nell'ambito della sicurezza del dato?
- Backup e disaster recovery - viene gestita una procedura di Backup? La struttura è ridondata? In caso di disaster recovery, esiste una procedura per il recupero dei dati?

**Indicatori clinici** - Prendono in considerazione le richieste fondamentali avanzate dai clienti:

- FSE - Fascicolo Sanitario Elettronico - si integra con il FSE?
- FUT - Foglio Unico di Terapia - viene gestito?
- Prontuario Farmaceutico - come viene eseguita la ricerca del farmaco?
- Diari - la compilazione dei diari è suddivisa per professione? Le varie figure professionali possono leggere i diari delle altre figure?
- Firma digitale - i documenti essenziali vengono firmati digitalmente? Alla dimissione del paziente, i record vengono consolidati?

A partire da questi indicatori sono state ricercate una serie di aziende che potessero soddisfare il maggior numero di richieste.

Nella tabella 2.1 vengono presentate le aziende con alcune informazioni base. Nella colonna ID viene indicato l'identificativo del prodotto, utilizzato nelle successive tabelle.

| **Azienda** | **Nome del prodotto** | **N. addetti** | **N. clienti** |
|---|---|---|---|
| InSoft srl | Senex | 8 | circa 400 |
| NetPolaris Srl | TuttiXTe | 8 | |
| Sysdat | SysRiposo | 5 | circa 150 |
| Gruppo Margotta | ABC Web | 10 | |
| Advenias | Epersonam | 15 | circa 500 |
| GPI | Riposo | 4 | 12 |
| CBA | SipCar Plus | 107 | circa 350 |
| Zuccheti Group | SoftwareUNO | 35 | circa 200 |

Tabella 2.1. Aziende analisi di mercato

Nelle tabelle 2.2 e 2.3 troviamo il dettaglio degli indicatori ICT mentre nella tabella 2.4 troviamo il dettaglio degli indicatori Clinici.

| **Nome del prodotto** | **Server locale** | **Database** | **Dettagli** |
|---|---|---|---|
| Senex | No | MySql | Sviluppato in JAVA |
| TuttiXTe | Su richiesta | SQL Server | Sviluppato in JAVA e TomCat lato server |
| SysRiposo | Sì | DB Maker | Sviluppato in ambiente Microsoft |
| ABC Web | Sì | SQL Server | |
| Epersonam | No, risiede su cloud Amazon Europa | Postgre-SQL | Sviluppato con framework Ruby on Rails |
| Riposo | Sì e su cloud | | |
| SipCar Plus | Sì | | Sviluppato in ambiente Microsoft |
| SoftwareUNO | Sì e su cloud | SQL Server | Sviluppato su framework Angular |

Tabella 2.2. Indicatori ICT, parte 1

| **Nome del prodotto** | **WEB** | **Mobile** | **Sicurezza** | **Backup e Disaster recovery** |
|---|---|---|---|---|
| Senex | Sì, HTTPS su piattaforma Windows | Sì | Standard Cloud | Sì |
| TuttiXTe | Sì, HTTPS su piattaforma Windows | No | Standard Cloud | Sì |
| SysRiposo | No | No | Client Server | No |
| ABC Web | Sì, con tecnologia Microsoft IIS | No | | |
| Epersonam | Supportato Chrome | Su browser e APP per operatore a domicilio | Standard Cloud | Sì |
| Riposo | Sì, HTTPS | In fase di sviluppo | Standard WEB | Sì |
| SipCar Plus | Sì | Sì, solo data entry | | |
| SoftwareUNO | Sì, con tecnologia Microsoft IIS | Sì | | |

Tabella 2.3. Indicatori ICT, parte 2

| **Nome del prodotto** | **FSE** | **FUT** | **Prontuario Farmaceutico** | **Diari** | **Firma digitale** |
|---|---|---|---|---|---|
| Senex | No | Sì | Ricerca per nome commerciale | Sì | No |
| TuttiXTe | No | Sì | Ricerca per nome commerciale | Sì | No |
| SysRiposo | No | Sì | Ricerca per nome commerciale | Sì | No |
| ABC Web | No | Sì | Ricerca per nome commerciale | Sì | Sì con OTP |
| Epersonam | No | Sì | Ricerca per nome commerciale e per principio attivo | Sì | No |
| Riposo | No | Sì | | Sì | No |
| SipCar Plus | No | Sì | Ricerca per nome commerciale e per principio attivo | Sì | In fase di sviluppo |
| SoftwareUNO | No | Sì | Ricerca per nome commerciale e per principio attivo | Sì | In fase di sviluppo |

Tabella 2.4. Indicatori Clinici

Per comprendere la scelta fatta nello sviluppo di un nuovo applicativo, è necessario approfondire alcuni aspetti che riguardano gli applicativi presi in esame:

- Senex (InSoft srl) presenta problematiche durante l'utilizzo offline dall'applicazione mobile: fonte di possibili conflitti/duplicati
- SysRiposo (Sysdat) è una soluzione client/server classica, richiede un cliente locale su sistema operativo Windows oppure l'utilizzo di un terminal server
- ABCWeb (Gruppo Margiotta) è una reingegnerizzazione di un applicativo originariamente sviluppato su Microsoft Access
- Epersonam (Advenias) è una soluzione modulare ma è necessario verificare gli sviluppi per la firma digitale
- Riposo (GPI) restyling in corso dell'applicativo, propone una separazione sostanziale tra parte gestionale/amministrativa e quella clinica
- SipCar Plus (CBA) alcuni elementi in fase di sviluppo tra cui la gestione della firma digitale
- SoftwareUNO (Zucchetti Group) integrato con Zucchetti per la parte amministrativa, in sviluppo l'utilizzo di sensori NFC per l'autenticazione e la firma digitale

## 2.3 Strada intrapresa

Tutti i prodotti visionati sono in fase di sviluppo e progettazione e tutti presentano delle peculiarità che non si adattano alle richieste e alle aspettative dei clienti.

Per questi motivi si è scelto di procedere all'analisi, alla progettazione e allo sviluppo di una cartella informatizzata costruita ad hoc.

Si è scelto di procedere con lo sviluppo, prediligendo l'approccio agile: rilasciare rapidamente modifiche al software in piccole porzioni con l'obiettivo di migliorare la soddisfazione dei clienti [7].

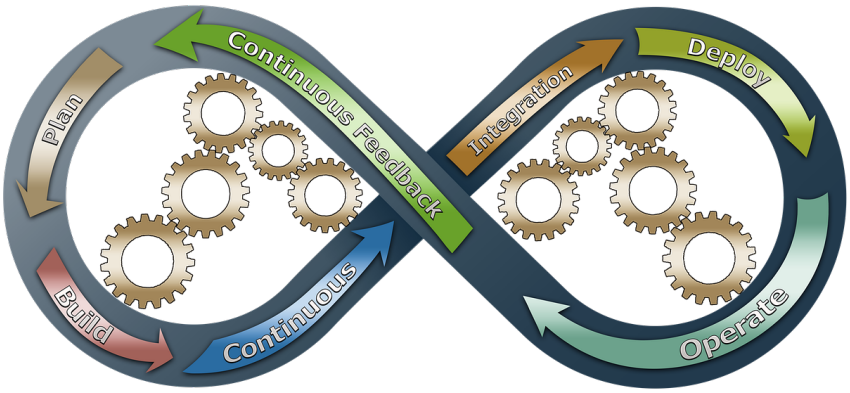


Figura 2.1. Approccio agile

# Capitolo 3

# Requisiti

I requisiti descrivono *cosa* un sistema deve fare, le sue proprietà essenziali e i vincoli a cui deve rispondere senza definire *come* verrà costruito.

La fase di redazione dei requisiti è un'attività che presenta una interazione intensa con i futuri fruitori del sistema oltre ad aspetti tecnici.

## 3.1 Attributi e requisiti essenziali

Per lo sviluppo di una cartella clinica informatizzata completa è necessario tener conto di una serie di attributi fondamentali presentati nel documento *EHR Definitional Model* da HIMSS (Healthcare Information and Management Systems Society) e che verranno di seguito descritti.

Ciascun attributo viene suddiviso in una serie di requisiti essenziali da soddisfare per la corretta realizzazione dello stesso.

1. **Fornire un accesso sicuro, affidabile e in tempo reale alle informazioni del paziente, dove e quando è necessario per supportare l'assistenza:**

   - è necessario fornire strumenti per garantire la riservatezza e la sicurezza delle informazioni sulla salute del paziente. Tali strumenti devono includere anche gli audit trail degli accessi (sequenze cronologiche non modificabili sulle transazioni);
   - il sistema sviluppato deve essere disponibile e affidabile 24 ore su 24 e 7 giorni su 7;
   - deve garantire una reattiva integrazione con il flusso di lavoro del medico;
   - deve prevedere la possibilità di accesso da remoto, in corsia di reparto e ovunque si trovi il medico o qualunque operatore sanitario.

2. **Acquisire e gestire le informazioni in maniera episodica e longitudinale:**

   - le informazini devono essere registrate o importate in maniera corretta con indicazione di timestamp e audit trail delle modifiche;

- possibilità di accettare informazioni da sistemi esterni e dispositivi di acquisizione dati automatizzati come monitor pazienti, apparecchiature per analisi di laboratorio e lettori di codici a barre tramite utilizzo di standard approvati;
- possibilità di consentire l'immissione efficiente dei dati e di documentazione da parte dei medici autorizzati; possibilmente dovrebbe supportare diverse tipologie di immissione dati (da tastiera, via voce o tramite riconoscimento della grafia);
- supporta la firma elettronica dove consentito per legge;
- si deve poter distinguere tra i dati storici del paziente rispetto ai dati episodici.

3. **Funzionare come la principale risorsa di informazioni per i medici durante la fornitura di assistenza al paziente:**

   - include l'elenco dei problemi del paziente, anamnesi, esame fisico, allergie, farmaci erogati e somministrati, risultati diagnostici con immagini e segni vitali più recenti;
   - offre un accesso facilitato alle informazioni sul paziente tramite visualizzazioni integrate e differenziate per specialità, contrassegnando anche le informazioni fuori dai valori normali;
   - fornisce strumenti di accesso e visualizzazione adattate ai ruoli e personalizzabili in base alle preferenze dell'utente finale;
   - permette l'accesso alle informazioni in qualsiasi punto del flusso di lavoro clinico;
   - implementa la possibilità di organizzare e assegnare priorità diverse alle comunicazioni relative al paziente come risultati di esami diagnostici, di laboratorio o di visite specialistiche.

4. **Assistere nel lavoro di pianificazione e fornitura di cure basate sull'evidenza a singoli e gruppi di pazienti:**

   - supporto per la valutazione della situazione clinica;
   - deve supportare la pianificazione, il monitoraggio e l'erogazione in maniera interdisciplinare basandosi su evidenze temporali e sui risultati ottenuti dai pazienti;
   - fornisce strumenti per supportare il lavoro del personale sanitario per i singoli pazienti: elenchi di pazienti, elenchi di attività e completamento dell'attività;
   - devono essere presenti strumenti per la pianificazione e l'organizzazione del lavoro del personale sanitario: oggi, questo turno, questa sessione clinica, durante l'orario di ufficio, ecc;
   - sono necessari strumenti per facilitare il lavoro di squadra e il processo di coordinamento: copertura, passaggi di consegne, escalation e delega;
   - deve fornire strumenti per il monitoraggio della conformità alle policy, notifica rapida dei cambiamenti nello stato del paziente e potenziali eventi avversi;
   - fornisce strumenti per facilitare e gestire la comunicazione con le aree diagnostiche e terapeutiche e monitorare il processo di completamento;

- include strumenti di supporto decisionale per guidare e valutare la somministrazione dei farmaci: paziente giusto, farmaco giusto, dose giusta, momento giusto, percorso giusto;
- può fornire raccomandazioni e avvisi su misura per le condizioni, la situazione e le preferenze del singolo paziente;

5. **Acquisire dati per migliorare continuamente la qualità, le modalità di utilizzo, la gestione del rischio, la pianificazione delle risorse e la gestione delle prestazioni:**
   - presenza di reportistica per la valutazione dei processi e degli esiti dell'assistenza fornita;
   - supporta le segnalazioni in merito alle conformità con gli standard di cura e di processo;
   - permette di integrare informazioni finanziarie e altri dati esterni come la soddisfazione del paziente e i dati comparativi del settore allo scopo di analizzare i processi e le prestazioni;
   - supoprta strumenti di *data modeling* per valutare potenziali cambiamenti;
   - registra i dati relativi alla salute del paziente necessari per identificare l'intensità del servizio per l'allocazione predittiva delle risorse;
   - può supportare la sorveglianza in tempo reale e la segnalazione di potenziali eventi avversi;
   - può fornire cure simultanee tramite visualizzazioni di riepilogo in tempo reale basate su informazioni per gruppi (coorti) di pazienti (ad esempio tutti i pazienti in una specifica unità di cura, tutti i pazienti assegnati a uno specirfico gruppo di studio, tutti i pazienti con sintomi specifici e dati demografici, ecc...).
6. **Registrare informazioni relative alla salute del paziente necessarie per il rimborso e la fatturazione:**
   - cattura informazioni basate sugli episodi da passare alla fatturazione (grafici di somministrazione dei farmaci, grafici e tabelle sui risultati ottenuti, ecc...);
   - recupera automaticamente le informazioni necessarie per verificare la copertura e le necessità mediche.
7. **Fornire informazioni longitudinali anonimizzati per supportare la ricerca clinica, i rapporti sulla salute pubblica e iniziative per la salute della popolazione:**
   - identifica la popolazione di pazienti che può beneficiare di iniziative di gestione della salute;
   - idendifica e tiene traccia di pazienti che sono iscritti a programmi di gestione della salute;
   - fornisce supporto integrato nella gestione della malattia per educare, sensibilizzare e assistere i pazienti ingaggiati;

- supporta la rendicontazione sulla responsabilità del prodotto e sul benessere sociale.

8. **Supportare studi clinici e ricerche basati sull'evidenza**
   - supporta l'identificazione dei pazienti per il reclutamento;
   - può includere protocolli, documentazione e report aggiuntivi necessari per gli studi clinici.

## 3.2 Requisiti funzionali

In questo paragrafo vengono descritti i requisiti funzionali presi in considerazione per la realizzazione del progetto. Tali requisiti sono presentati sotto forma di elenco numerato, in questo modo possono essere identificati durante la descrizione delle varie parti del sistama sviluppato.

1. Il trattamento dei dati sensibili e personali deve essere garantito seguendo le normative vigenti.
2. L'accesso alla piattafarma deve avvenire in maniera differenziata a seconda dei ruoli associati all'utente. Ogni utente può avere più ruoli e per ogni ruolo si deve poter indicare quali sono le azioni che può svolgere.
3. Deve essere previsto un meccanismo di recupero delle credenziali in maniera automatica senza l'intervento dell'amministratore di sistema.
4. Il numero di utenti che possono fare accesso alla piattaforma non è definito a priori e può crescere, non deve esserci un limite sulla creazione di utenze.Il gestore del sistema deve avere completa autonomia nella gestione degli utenti.
5. Ogni operazione eseguita dall'utente sul dato deve essere tracciata in modo tale da poter ricostruire l'informazione.
6. L'accesso ai log dell'applicativo deve essere garantito al gestore tramite interfacce intuitive.
7. Deve esserci la possibilità di creare notifiche con la registrazione dei visualizzatori.
8. Indispensabile avere una separazione tra le informazioni anagrafiche del paziente e tutte le altre informazioni relative al ricovero.
9. L'elenco dei pazienti attualmente in carico deve mettere in risalto alcune informazioni fondamentali: tipologia di ricovero, eventuali terapie che richiedono un'attenzione particolare, eventuali ausili in uso.
10. La navigazione tra un paziente e l'altro deve avvenire in maniera agile e intuitiva.
11. L'apertura di una cartella può avvenire anche tramite lettura di barcode presente su braccialetto del paziente in modo tale da garantire l'accesso sicuro e certo all'informazione del paziente.

12. L'apertura di un record in modifica è esclusiva: nel caso in cui si tenti un accesso concorrente deve essere notificato quale utente ha l'uso esclusivo di quel record.

13. Sul FUT (Foglio Unico di Terapia) è necesario che siano visibili e chiare le date di inizio e fine validità e lo stato attuale.

14. I FUT archiviati devono essere consultabili in maniera agevole. Tra le informazioni è necessario rendere visibile chi ha validato il FUT e chi lo ha archiviato.

15. Sul FUT non devono esserci vincoli riguardanti il numero massimo di prescrizioni e il numero massimo di somministrazioni.

16. Il FUT può generare un documento elettronico (pdf) che può essere firmato elettronicamente o utilizzato come backup in caso di mancato funzionamento del sistema.

17. Se il paziente va in permesso, a fronte dell'inserimento di data e ora inizio e fine, deve essere possibile creare la terapia che deve seguire il paziente nell'intervallo di tempo impostato.

18. L'operatore infermieristico deve poter accedere alla terapia in corso indicando se ha eseguito la somministraizione oppure no in maniera semplice e intuitiva.

19. Si deve prevedere un meccanismo di conferma doppia sulla somministrazione per particolari tipi di farmaci.

20. La scelta del farmaco da parte del medico deve poter avvenire sia per nome commerciale sia per principio attivo.

21. La lista dei farmaci deve far riferimento al prontuario farmaceutico di AIFA (Agenzia Italiana del Farmaco).

22. Il flusso di valutazione del medico deve essere guidato dall'anamnesi patologica prossima fino agli aggiornamenti funzionali.

23. È necessario predisporre una serie di inquadramenti che permettano di impostare automaticamente alcune valutazioni o scale di valutazione da eseguire.

24. Non è specificato a priori il numero di valutazioni specialistiche e di aggiornamenti funzionali che vengono fatti sul paziente.

25. Le scale di valutazione devono essere consultabili e compilabili agevolmente. Devono essere presenti le informazioni di compilazione.

26. L'accesso ai parametri vitali del paziente (pressione, temperatura corporea, ecc ... ) deve essere immediato e la loro compilazione deve richiedere poche operazioni.

27. La compilazione dei diari di tutte le figure sanitarie deve essere semplice e prevedere la possibilità di inserimento frasi precompialte. Una volta consolidato il salvataggio si deve poter eseguire una correzione purché questa sia evidenziata.

28. Ogni figura professionale deve avere a disposizione dei moduli per la stesura delle proprie valutazioni.

29. L'inserimento di una consegna deve essere agevole. L'utente deve avere la possibilità di scegliere se la consegna riguarda soltanto la propria figura professionale o se è comune a tutte o solo alcune delle altre figure professionali presenti nella struttura.

30. Ogni figura professionale avrà accesso alle consegne a cui è abilitata.

31. È necessario integrare la produzione di documentazione stampabile per le informative sulla privacy, il consenso informato, ecc ...

32. Si deve favorire l'accesso a informazioni relative a precedenti ricoveri: possibilità di import e di collegamenti diretti.

33. È necessario prevedere la possibilità di allegare qualsiasi tipo di documento alla cartella. Tali allegati devono essere supportati da metadati.

34. Non appena la cartella viene chiusa non deve essere possibile nessuna modifica a nessun record salvo giustificati e documentati motivi. Tale documentazione dovrà essere allegata alla cartella stessa.

35. Oltre alla consultazione in tempo reale delle informazioni sul paziente, deve essere sviluppato anche un ambiente di esportazione dati con anonimizzazione dell'informazione.

## 3.3 Schema generale

Il class diagram in figura 3.1 presenta le principali componenti che deve avere una cartella clinica informatizzata.

Il punto di partenza è rappresentato dal *paziente*: questo viene identificato dal suo codice fiscale e da altri campi anagrafici come cognome, nome, luogo e data di nascita e così via. Al paziente sono collegate delle entità aggiuntive in modo tale da avere una modularità elevata delle informazioni:

- *contatto*: entità generica per indicare i numeri di telefono e le e-mail associate al paziente;

- *privacy*: raccoglie tutti i moduli che il paziente ha firmato o dovrà firmare o far firmare per il consenso al trattamento dei propri dati personali;

- *allergia*: si tratta di un'entità molto importante in quanto permette di elencare tutte le eventuali allergie o intolleranze sofferte dal paziente; è collegata al paziente in modo tale da non dover essere compilate ad ogni ricovero.

All'entità *paziente* è direttamente collegata l'entità *cartella* con cardinalità 0 .. n. Quest'ultima raccoglie informazioni riguardanti il numero identificativo (solitamente formato da caratteri alfa numerici concatenati all'anno di ricovero), la data di ricovero e di dimissione, la tipologia del ricovero e così via. A questa entità sono collegati tutti gli elementi che andranno a comporre la cartella clinica:

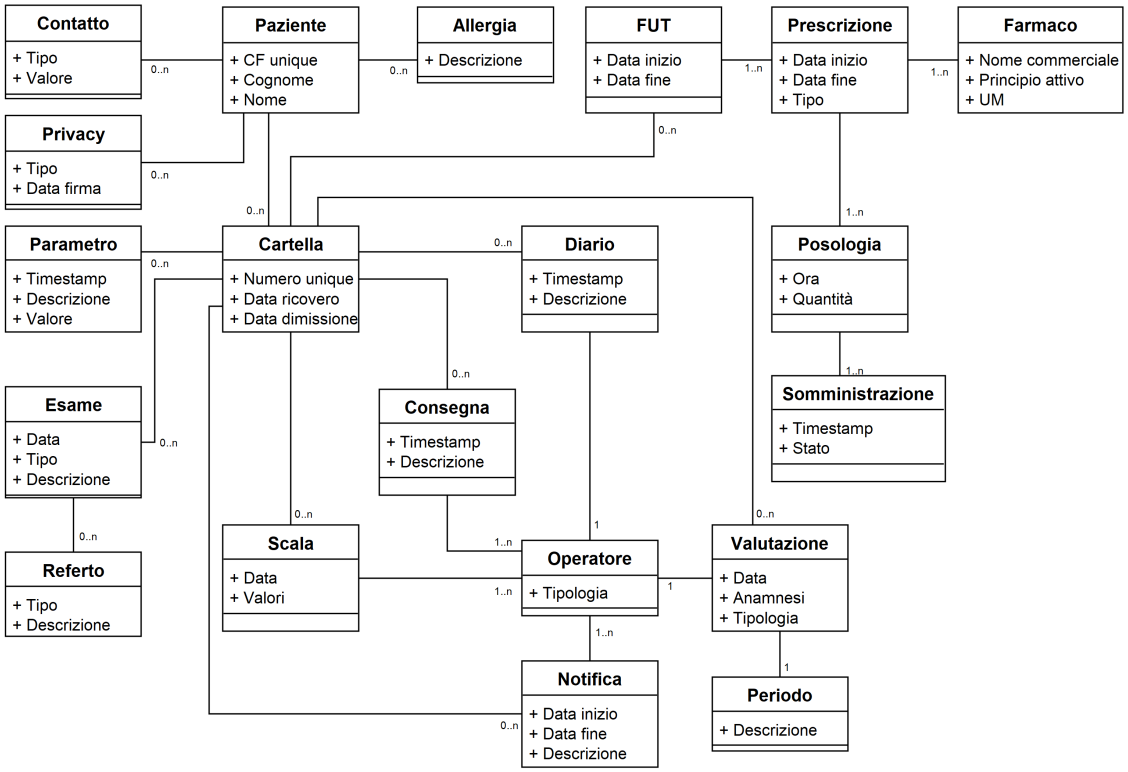


Figura 3.1. Class diagram generale

- *FUT*: è il Foglio Unico di Terapia con cardinalità 0 .. n. Gli attributi principali saranno la data inizio e fine validità e lo stato (bozza, valido, archiviato). Il FUT si compone di un numero non specificato di *prescrizioni*. Ogni prescrizione avrà una data inizio e una data fine, la tipologia, un certo numero di *farmaci* e una serie di *posologie*. Queste ultime raccolgono l'ora in cui il *farmaco* deve essere somministrato e la quantità. A questa entità sarà collegata la *somministrazione*, con cardinalità 1 .. n, che avrà come attributi minimi lo stato (somministrato, non somministrato, indefinito) e il timestamp di somministrazione.

- *Valutazione*: è l'entita che permette di gestire le valutazioni fatte dai vari *operatori* con cardinalità 0..n. Come attributi principali avrà la data di valutazione, l'anamnesi e la tipologia. Ogni valutazione sarà associata ad un solo *periodo* di ricovero e ad una sola tipologia di *operatore*.

- *Diario*: si tratta di un'entità fondamentale per l'andamento del ricovero. Ha cardinalità 0..n ed è collegato all'entità *operatore* con cardinalità 1. Questo elemento deve permettere di prendere nota di informazioni sintetiche collocate in un dato timestamp.

- *Consegna*: molto importante per permettere il passaggio delle informazioni tra colleghi della stessa figura professionale (operatore) oppure tra figure professionali diverse, motivo per cui la cardinalità con l'entità *operatore* è di 1..n.

- *Parametro*: prende in considerazione la possibilità di registrare i parametri vitali e non vitali del paziente come la temperatura, la pressione arteriosa, l'ossigenazione del sangue e così via dicendo. Ha cardinalità 0..n.
- *Esame*: per permettere la registrazione di esami specialistici eseguiti anche esternamente alla struttura. Alcuni esempi di tipologia possono essere: esami del sangue, esami diagnostici (radiografie, risonanze, ecografie), visite specialistiche (fisiatriche, ortopediche, cardiologiche). Solitamente agli esami verranno collegati uno o più *referti* che permettono di esplicitare i risultati o i dati ottenuti.
- *Scala*: è un'entità che permette la compilazione di scale di valutazione specifiche. Ha una cardinalità 0..n e può essere compilata da uno o più operatori se coinvolge più figure professionali.
- *Notifica*: questo elemento è importante per mettere in risalto alcune situazioni che riguardano il paziente e fare in modo che tutti gli operatori le leggano non appena possibile, con la registrazione del timestamp di lettura.

# Capitolo 4

# Architettura

## 4.1 Infrastruttura

L'applicazione web è stata sviluppata seguendo il pattern architetturale MVC (Model View Controller - figura 4.1). Questo pattern, che può essere visto come un modello a tre strati [8], è basato sulla separazione dei compiti fra i componenti software:

- Strato di presentazione (View): visualizza i dati applicativi in una forma leggibile alle persone attraverso un'interfaccia, definita tramite linguaggi di markup, codici di scripting e fogli di stile;
- Strato intermedio o di logica (Controller): eroga e gestisce i servizi offerti, applica le politiche aziendali e la logica di business, può essere decomposto in sotto-strati per migliorare la flessibilità, la manutenibilità, la riusabilità e la scalabilità del sistema ed è indipendente sia dallo strato di presentazione sia dallo strato dati;
- Strato dei dati (Model): fornisce al controller i meccanismi per la memorizzazione e l'accesso ai dati e garantisce la coerenza e la conservazione dei dati.

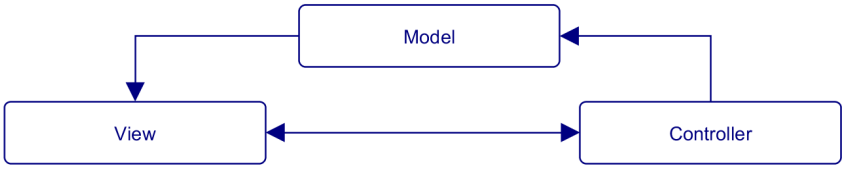


Figura 4.1. Pattern Model View Controller (MVC)

Le applicazioni web hanno una serie di caratteristiche che permettono di definirle come sistemi distribuiti e reattivi [8]. Sono sistemi distribuiti perché una richiesta può arrivare al server in qualunque momento e l'interazione è basata sullo scambio di messaggi testuali, inoltre non esiste un ordinamento intrinseco delle richieste effettuate dal client e più persone possono accedere allo stesso servizio nello stesso momento. Sono sistemi reattivi perché non è possibile stabilire a priori quale azione verrà effettuata dall'utente e ciascuna azione viene tradotta in una richiesta alla quale il server reagisce eseguendo una serie di operazioni.

Per il corretto funzionamento dell'applicazione web, l'infrastruttura client-server (figura 4.2) deve soddisfare una serie di requisiti:

- Lato client
  - accessibilità: deve essere di facile utilizzo con tempi di apprendimento rapidi e con interfacce utenti accattivanti;
  - universalità: deve essere compatibile con il maggior numero di piattaforme hardware e software;
- Lato server
  - garantire alte prestazioni tramite caching di informazioni statiche, pooling di risorse comuni, bilanciamento del carico, gestione trasparente dei guasti;
  - semplificare la gestione del sistema tramite aggiornamento e riconfigurazione dinamica, arresti controllati e log di eventi;
  - mantenere costi di sviluppo bassi tramite integrazione con sistemi esistenti, generazione automatica di codice e riuso di componenti software.
- Lato rete
  - evitare la configurazione di dispositivi;
  - ridurre il consumo di banda sia lato server sia lato client;
  - rendere il sistema compatibile con le misure di sicurezza esistenti.

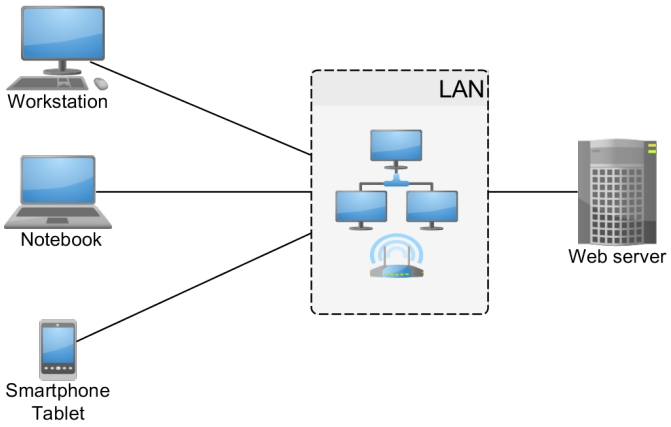


Figura 4.2. Infrastruttura client-server

Per soddisfare i requisiti lato client, vengono utilizzati browser moderni. Questi permettono l'utilizzo dell'applicazione sviluppata sui diversi dispositivi presenti sul mercato: workstation o notebook con sistemi operativi Windows, Linux o iOS, oppure smartphone o tablet con sistema operativo Android o iOS.

Per quanto riguarda la componente server, è stato scelto di utilizzare Ubuntu Server LTS 18.04, con possibilità di upgrade alle versioni successive. È stata fatta questa scelta per due motivi: possibilità di spostare in ambienti cloud che già supportano questo sistema operativo e congruità con l'ambiente di sviluppo.

## 4.2 Framework

Il framework utilizzato per lo sviluppo del progetto è Ruby on Rails o più semplicemente Rails (figura 4.3). Questo è un framework opensource per lo sviluppo di applicazioni web creato da David Heinemeir Hansson tra il 2004 e il 2005 e distribuito come una *gem* di Ruby (figura 4.4). Le *gem* sono librerie di codici o pacchetti di codici riutilizzabili.


Figura 4.3. Ruby on Rails

Rails si basa sue due principi guida: DRY (Don't Repeat Yourself) e CoC (Convention over Configuration).

- Don't Repeat Yourself: ogni informazione deve essere rappresentata in maniera unica, inequivocabile e autorevole all'interno del sistema; in questo modo il codice sarà più manutenibile, più estensibile e con meno bug.

- Convention over Configuration: il programmatore deve mettere mano ai file di configurazione soltanto per quello che non è definito dalle convenzioni del framework. Ad esempio se un model è definito dalla classe "Folder", per sfruttare appieno le convenzioni, la tabella corrispondente sul database si chiamerà "folders", il controller sarà definito dal file "folders_controller.rb" e la classe del controller sarà "FoldersController".

Ruby è un linguaggio di programmazione orientato agli oggetti creato da Yukihiro Matsumoto, il quale ha cercato di rendere il linguaggio "naturale" e non "semplice", un po' come la vita di tutti i giorni [9].

Matsumoto ha fatto una sorta di miscela dei suoi linguaggi di programmazione preferiti (Perl, Smalltalk, Eiffel, Ada e Lisp) e ha inseguito l'idea di creare un linguaggio di scripting che fosse più potente di Perl e più orientato agli oggetti di Python.

In Ruby ogni elemento è un oggetto, ogni informazione, ogni pezzo di codice ha le sue proprietà (accessibili tramite le *instance variable*) e le sue azioni (conosciute come *methods*).

Ruby ha una vasta gamma di caratteristiche, tra cui:

- gestione delle eccezioni come Java o Python, per rendere più semplice la gestione degli errori;

- garbage collector per ogni oggetto, per cui non è necessario rilasciare tutti i riferimenti e le librerie estese;
- la scrittura di estensioni in C è più semplice rispetto a Perl e a Python grazie alla API disponibile per richiamare Ruby da C;
- può caricare librerie dinamicamente se il Sistema Operativo lo permette;
- è indipendente dal sistema operativo ed è *portable* (sviluppato su GNU/Linux ma funziona bene su molti altri sistemi operativi).


Figura 4.4. Ruby

## 4.3 Rails

L'architettura di Rails (figura 4.5) si basa sul sopracitato pattern MVC oltre ad offire componenti di tipo REST (REpresentational State Transfer) per l'utilizzo di web services.

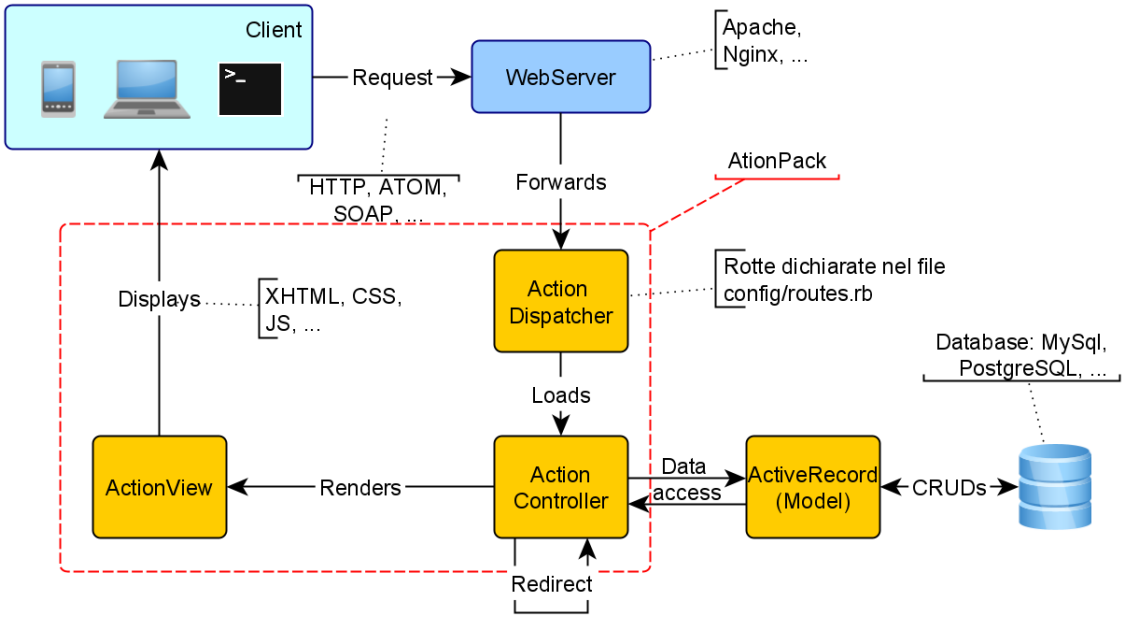


Figura 4.5. Architettura applicazioni Rails

Il modulo *ActionPack* fornisce i livelli controller e view del pattern MVC. I sotto moduli che ne fanno parte si occupano di catturare le richieste HTTP fatte dal client, mapparle in azioni definite nei controller e renderizzarle sul browser:

- **ActionDispatcher**: gestisce il routing della richiesta HTTP, analizzandone i cookie, le variabili di sessione e i metodi di richiesta (GET, POST, ecc...), indirizzandola verso l'azione corretta all'interno del controller;
- **ActionController**: fornisce un controller di base da cui possono ereditare tutti gli altri controller che conterranno le azioni utili per il controllo dei model e delle view; rende disponibili i dati secondo necessità; controlla il rendering delle views e il reindirizzamento; gestisce le sessioni utente, il flusso dell'applicazione e le funzionalità di memorizzazione nella cache;
- **ActionView**: renderizza la presentazione della pagina WEB richiesta; fornisce layout principali ed helper che supportano la generazione di HTML, feed e altri formati. Mette a disposizione i tre template html.erb, xml.erb e js.erb dove erb sta per Extended RuBy code e servono rispettivamente per le view HTML, per la creazione di documenti XML e per la creazione di codice JavaScript dinamico utile per l'implementazione di funzionalità AJAX.

Il modulo **ActiveRecord** è utilizzato per la gestione dei dati: fornisce la mappatura relazionale tra le tabelle del database (MySQL, PostgreSQL, ecc...) con le corrispondenti classi Ruby e costituisce il livello model del pattern MVC. Rails mette a disposizione gli strumenti per l'implementazione delle funzionalità CRUD (Create Read Update Delete). Un oggetto rappresenta una riga all'interno della tabella corrispondente. Questo modulo implementa anche funzionalità di ricerca avanzata e la possibilità di creare relazioni o associazioni tra modelli.

**ActiveRecord** fornisce a Rails delle importanti funzioanlità di callback chiamati *hook*. Questi permettono di eseguire metodi (o funzioni) prima o dopo la creazione, l'aggiornamento o la cancellazione di un oggetto e di iniettare *view* non ancora definite o caricate da altri moduli applicativi.

Nell'*ApplicationController* di questo progetto sono stati definiti alcuni di questi hook:

```
1  class ApplicationController < ActionController::Base
2    rescue_from ActionController::InvalidAuthenticityToken,
3        :with => :render_422
4    rescue_from ActionController::RoutingError,
5        :with => :render_404
6
7    before_action :authenticate_user! #verifica utente
8    before_action :set_paper_trail_whodunnit #salva la versione
9                  #precedente del record
10   before_action :set_gettext_locale #setta la lingua
11   before_action :_set_theme_params #setta parametri view
12   before_action :_set_folder_active_tab #imposta la tab attiva
13   after_action :page_history #aggiorna la history
14 end
```

### 4.3.1 Struttura cartelle applicativi Rails

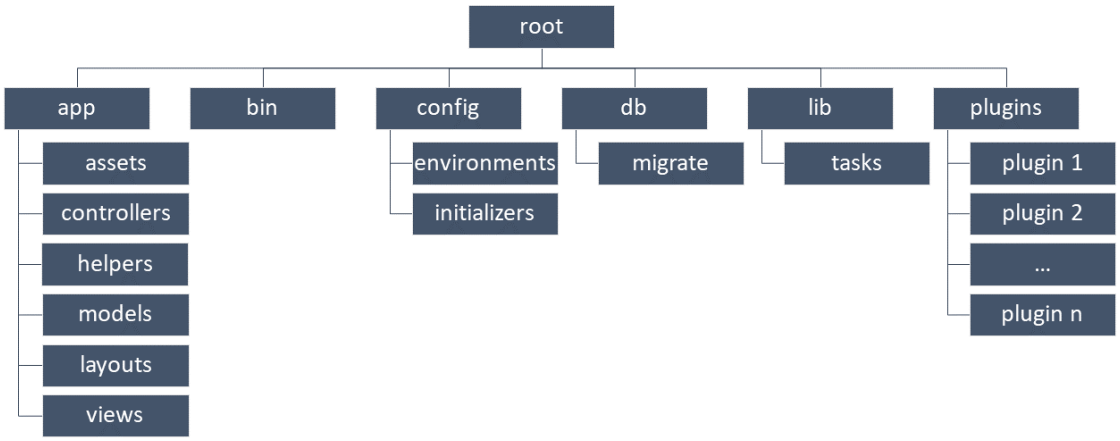


Figura 4.6. Alberatura di Rails

Le applicazioni sviluppate in Rails sono organizzate come un insieme di sottocartelle (figura 4.6). Seguendo questa struttura, è possibile comprendere con semplicità il codice di progetti realizzati da altri, oltre a poter rendere automatica la generazione di codice.

Le sotto cartelle hanno questo significato [10]:

- **app**: è il nucleo centrale dell'applicazione; essendo Rails un framework MVC, le tre sezioni Model, View e Controller vanno all'interno di questa directory.
- **app/assets**: contiene i file richiesti per il front-end dell'applicazione; questi file sono raggruppati in base al tipo (images, javascripts, css, ecc...). All'interno di questa cartella ci sono due file in particolare che assumono un ruolo particolare: *javascripts/application.js* e *scss/application.css* e sono due file di tipo "manifest". In questi file devono essere inseriti tutti i riferimenti di tutti i file javascript/css utilizzati che verranno unificati e minimizzati prima di essere inviati alla componente view.

Porzione di aplication.js

```
// Estratto di application.js
//= require bootstrap/bootstrap

// JARVIS WIDGETS
//= require smartwidgets/jarvis.widget

// DATATABLE PLUGIN
//= require plugin/datatables/jquery.dataTables
//= require plugin/datatables/dataTables.colVis
//= require plugin/datatables/dataTables.tableTools
//= require plugin/datatables/dataTables.bootstrap
//= require plugin/datatable-responsive/datatables.responsive

```

```
// JQUERY SELECT2 INPUT
//= require plugin/select2/select2

// jQuery Autocomplete
//= jquery-ui/widgets/autocomplete.js
//= require modal-responder.js

//= require sanHelper.js
```

- **app/controllers**: contiene i file richiesti per i controller che gestiscono i model e le view. Per seguire le convenzioni di Rails, i nomi dei file che descrivono un controller seguono lo "snake case" e sono composti dal nome del model al plurale concatenato a "_controller.rb". Ad esempio il controller per il model *Folder* sarà *folders_controller.rb*. Inoltre, il nome della classe seguirà il "CamelCase" e sarà, ad esempio, *FoldersController*.

Porzione di folders_controller.rb

```ruby
class FoldersController < GuiController
  # ...
  # codice
  # ...
  private
  def validate_params
    validated_params = params.require(:folder).permit!.to_h
    validated_params
  end
  def get_record
    @isReminder = params[:reminder] ? true : false
    rec = Folder.find(params[:id])
    get_related_record rec
  end

  def get_recordset
    @recordsetdim = Folder.where.not(end_date: nil) # dimessi
    Folder.where(end_date: nil)
  end
  # ...
  # altro codice
  # ...
end
```

Tra tutti i controller, quello principale è *ApplicationController* e da questo ereditano tutti gli altri. Tutti i metodi definiti al suo interno sono disponibili a tutti i controller.

- **app/helpers**: in questa cartella risiedono tutte le funzioni di supporto per le view; sono a disposizione una serie di helper di default come ad esempio *image_tag* per referenziare le immagini nelle view; possono essere create funzioni in un file helper specifico del controller seguendo la convenzione di denominazione *nome_controller_helper*. Di default esiste l'helper generale *application_helper*: i metodi scritti in questo helper sono disponibili in tutti gli altri helper e in tutte le view.

- **app/models**: in questa cartella sono presenti tutti i file relativi ai model; il model agisce come mappatore degli oggetti relazionali per le tabelle del database. La convenzione sui nomi è semplicemente *nomedelmodel.rb* e il nome del model è per convenzione la forma singolare del nome della tabella in database. Ad esempio *Folder* sarà il model che mapperà la tabella *folders* in database.

  Qui vengono gestite anche le associazioni tra le varie tabelle. In Rails, un'associazione è una connessione tra due **ActiveRecord** [11]. Le associazioni utilizzate nel progetto sono le seguenti:

  – *belongs_to*: imposta una connessione uno-a-uno con un altro model, in modo tale che ogni istanza del model dichiarante appartenga a un'istanza dell'altro model. Ad esempio il model Fut, che può appartenere solo a una cartella, avrà un *belongs_to* al model Folder (figura 4.7).

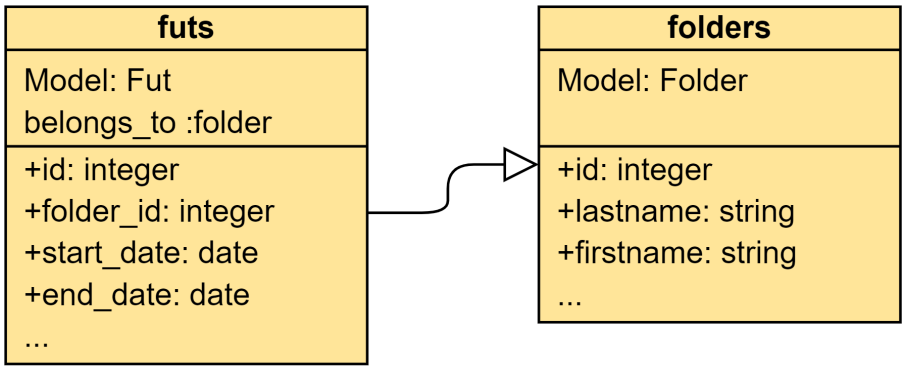


Figura 4.7. Associazione belongs to

  – *has_one*: anche questa è una connessione uno-a-uno con un altro model ma con semantica e conseguenze leggermente diverse. Indica che ogni istanza di un model contiene o possiede un altro model. Ad esempio il model Folder può avere una istanza del model NextAnamnesis.

  – *has_many*: indica una connessione uno-a-molti con un altro model. Molto spesso nell'altro model verrà indicata anche l'associazione *belongs_to*. Con questa associazione si può indicare che ogni istanza del model ha zero o più istanze di un altro model. Ad esempio il model Warfarin può avere zero o più istanze di WarfarinRow e quest'ultimo avrà una sola istanza del model Warfarin.

  – *has_many :through*: viene utilizzata per impostare connessioni molti-a-molti con un altro model. Indica che il model dichiarante può essere collegato a zero o

più istanze di un altro model passando attraverso un terzo model. Ad esempio il model Delivery può avere zero o più istanze di Role passando attraverso la relazione DeliveryRole (figura 4.8).

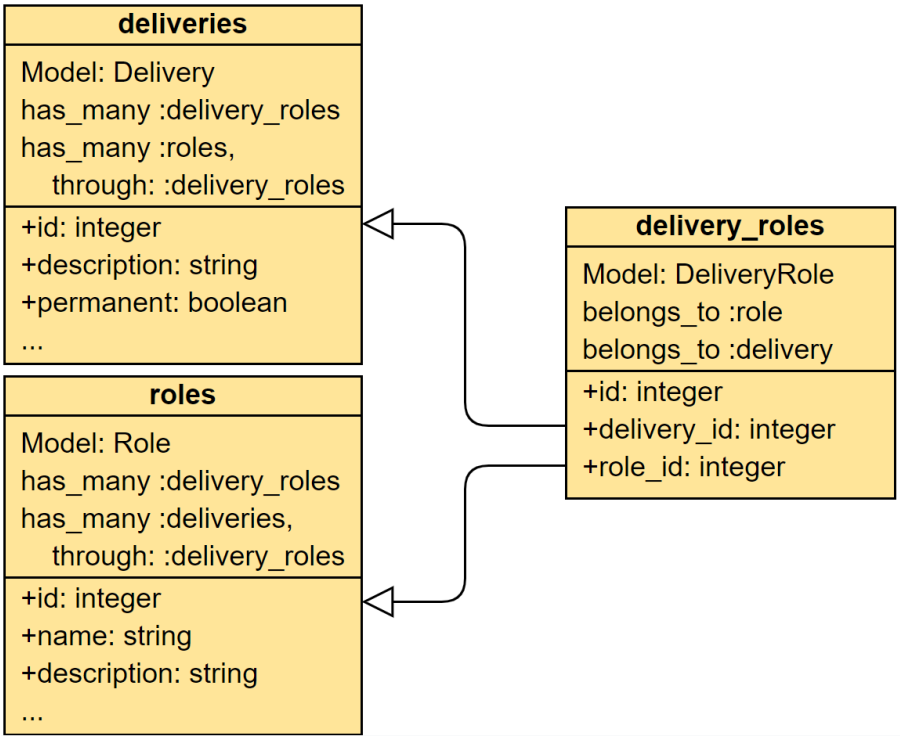


Figura 4.8. Associazione has many :through

- **app/views**: all'interno di questa cartella vanno a finire tutti i file che riguardano la terza parte del MVC, le view. I file sono una combinazione di HTML e Ruby (nominato Embedded Ruby o Erb) e sono organizzati in base al controller corrispondente. Seguendo le convenzioni di Rails, deve esistere una view per ogni azione del controller: ad esempio per l'azione *FoldersController#index* deve esistere la corrispondente view *app/views/folders/index.html.erb*.

  All'interno di questa cartella, ne esiste una che contiene i layout ereditati da tutte le view. Possono essere creati diversi layout per diversi ambiti dell'applicazione.

Contenuto del layout _header.html.erb

```
1 <header id="header">
2   <div id="logo-group">
3     <a href="/">
4     <span id="logo"> <%= image_tag "logo.png"%> </span>
5     </a>
6   </div>
7   <%= yield :top_right %>
```

```
8   <% if current_user && !@simulate_no_user %>
9     <div class="pull-right">
10      <div id="hide-menu" class="btn-header␣pull-right">
11        <span>
12    <a href="javascript:void(0);" data-action="toggleMenu">
13      <i class="fa␣fa-reorder"></i>
14    </a>
15  </span>
16      </div>
17      <div id="logout" class="btn-header␣pull-right">
18        <span>
19          <%= link_to(destroy_user_session_path) do %>
20            <i class="fa␣fa-sign-out"></i>
21          <% end %>
22        </span>
23      </div>
24      <!-- fullscreen button -->
25      <div id="fullscreen" class="btn-header␣pull-right">
26        <span>
27    <a href="javascript:void(0);" data-action="Fullscreen">
28      <i class="fa␣fa-arrows-alt"></i>
29    </a>
30  </span>
31      </div>
32    </div>
33    <!-- end pulled right: nav area -->
34  <% end %>
35 </header>
```

- **bin**: contiene i Binstubs per le applicazioni Rails. I Binstubs sono dei wrapper per eseguire le gem dell'applicazione (pacchetti di programmi e librerie scritte in Ruby). Quelli disponibili di default sono *bundle*, *rails*, *rake*, *setup* e *spring*.
- **config**: come il nome suggerisce, contiene i file di configurazione dell'applicazione. L'ambiente applicativo può essere manipolato dai file all'interno di questa cartella. È necessario scendere nel dettaglio per scoprire meglio cosa fanno alcuni di questi file e cartelle:
  - **application.rb**: contiene le configurazioni principali dell'applicazione come il timezone e la lingua utilizzata di default; le configurazioni presenti in questo file sono valide per tutti gli ambienti (sviluppo, test e produzione).
  - **application.yml**: contiene le variabili d'ambiente disponibili tramite l'array associativo *ENV*.
  - **boot.rb**: come si può immaginare, questo file è necessario per permettere l'avvio dell'applicazione. Le applicazioni Rails mantengono le dipendenze dalle gem all'interno del file *Gemfile* nella cartella root del progetto. Il *Gemfile* è obbligatorio

e **boot.rb** si occupa di verificare la sua presenza e salva il percorso verso questo file in una variabile d'ambiente chiamata *BUNDLE_GEMFILE* per usi futuri.

– **database.yml**: in questo file sono presenti le configurazioni per i database dei tre ambienti menzionati precedentemente (sviluppo, test e produzione).

Porzione di database.yml per l'ambiente di sviluppo

```
default: &default
  adapter: postgresql
  encoding: unicode
  pool: <%= ENV.fetch("RAILS_MAX_THREADS") { 5 } %>

development:
  <<: *default
  database: medrec_dev
  username: medrec
  password: <%= ENV["MEDREC_DATABASE_PASSWORD"] %>
```

Per questo progetto, come si può vedere dall'estratto di codice del file in esame, è stato scelto di utilizzare come motore di database **PostgreSQL**. Questo è un potente sistema di database relazionale, orientato agli oggetti, open source e che utilizza ed estende le funzionalità del linguaggio SQL [12]. È molto conosciuto nell'ambiente grazie alla sua architettura, affidabilità, integrità dei dati, set di funzionalità ed estensibilità.

– **environment.rb**: questo file, di concerto con *application.rb* permette di inizializzare l'applicazione Rails.

– **routes.rb**: all'interno di questo file sono presenti le rotte dell'applicazione.

Porzione di routes.rb

```
Rails.application.routes.draw do
  root "miscellaneous#blank_page"
  get "miscellaneous/login",
      to: "miscellaneous#login",
      as: :miscellaneous_login
  resources :folders do
    resources :permissions
  end
  resources :prints
  resources :users
  resources :roles
  post "api/folders/create",
    to: "api_folders#create", as: :api_folder_create
  post "api/folders/update",
    to: "api_folders#update", as: :api_folder_update
 end
```

– **environments**: questa cartella contiene i file di configurazione specifici per gli ambienti a disposizione (sviluppo, test e produzione).

  - **initializers**: è una cartella che contiene tutti i file che devono essere eseguiti durante l'inizializzazione di Rails. Tutti i file **.rb* presenti all'interno di questa cartella vengono eseguiti automaticamente in maniera sequenziale alfabetica, grazie al comando *Rails.application.initialize!* presente in *config/environment.rb*.

- **db**: tutti i file relativi al database vanno a finire in questa cartella (file di migrazione, schema, configurazione e seed).

- **lib**: raccoglie le librerie specifiche dell'applicazione. I file presenti in questa cartella sono estratti di codice riutilizzabile, include la sottocartella **tasks** che colleziona i Rake tasks richiesti dai *Rakefile*; i *Rakefile* sono invocati tramite *Rails.application.load_tasks* e richiedono *application.rb*.

```
#task per la creazione delle somministrazioni in maniera
#programmatica. Viene eseguito sul server ogni giorno alle
#00:01 per poter avere a disposizione le somministrazioni
#del giorno corrente. Esecuzione tramite il comando
#bundle exec rake create_fut_admins:create
namespace :create_fut_admins do
  desc "Create fut admins for all hospitalized patients"
  task :create, [] => [:environment] do |t, args|
    puts "Start #{DateTime.now}"
    Folder.where(end_date: nil).each do |folder|
      active_fut = folder.futs.find_by(state: 1)
      warfarin = folder.warfarins.find_by(state: 1)
      d = Date.today
      n = DateTime.now
      fut_admins = FutPluginHelper.get_today_fut_admins(
        folder, active_fut, warfarin , false, d, n, n)
      puts "Folder #{folder.number}: #{fut_admins.count}"
    end
    puts "End #{DateTime.now}"
  end
end
```

- **plugins**: questa cartella raccoglie tutti i plugin dell'applicazione. I plugin di Rails sono estensioni o modifiche al nucleo dell'applicazione [13]. Forniscono: un modo per condividere idee innovative senza danneggiare il codice base dell'applicativo; un'architettura segmentata in modo da poter aggiornare singole funzionalità in base alle proprie pianificazioni di rilascio; un modo per non includere tutte le funzionalità dell'applicativo e poter eseguire installazioni modulari in base alle richieste del cliente. Rails fornisce la possibilità di creare plugin come delle *gem*, **gemified plugins**. Gli sviluppatori di SAN hanno scelto di seguire un metodo differente messo a disposizione dal progetto *Redmine*.

### 4.3.2 Redmine

Si tratta di un'applicazione nata per la gestione di progetti, sviluppata in Ruby on Rails e multipiattaforma [14]. Redmine è opensource e distribuita con licenza GNU GPL v2.


Figura 4.9. Redmine

In questo progetto sono state integrate le funzionalità di gestione del menù e di gestione dei plugin. In particolare mi sono occupato di quest'ultima, della quale se ne parlerà nel capitolo 6.

### 4.3.3 Smartadmin

Vista la necessità di concentrarsi sullo sviluppo di funzionalità logiche, per velocizzare la progettazione e la scrittura delle interfacce grafiche e della loro interazione con il backand, SAN ha acquistato il pacchetto Bootstrap SmartAdmin.

Questo pacchetto contiene un'ampia raccolta di componenti dell'interfaccia utente riutilizzabili, ottimizzati e integrati con i plug-in jQuery.


Figura 4.10. Smartadmin

È stato scelto *Smartadmin* per le funzionalità di base a disposizione e per il continuo mantenimento dei pacchetti. Alcune di queste funzionalità base sono:

- Varie soluzioni di layout disponibili modificando soltanto la scelta del CSS da caricare;
- Integrazione con plugin molto utili come *Datatable* e *Fontawesome*;
- Configurato per la gestione delle traduzioni tramite il plugin i18n;
- Supporta tutte le ultime librerie per la gestione dei grafici come *Flot*, *Chart.js*, *Easy Pie Chart*.

## 4.4 Struttura dell'applicazione

Come si può vedere dalla figura 4.11, l'applicazione web è costituita da un nucleo centrale, dove sono definiti model e controller orizzontali utilizzabili in tutto l'ambito applicativo, e da una serie di plugins, dove sono definite funzionalità verticalizzate.

Fanno parte del nucleo dell'applicazione, che verrà discusso più approfonditamente nel capitolo successivo, i seguenti elementi:

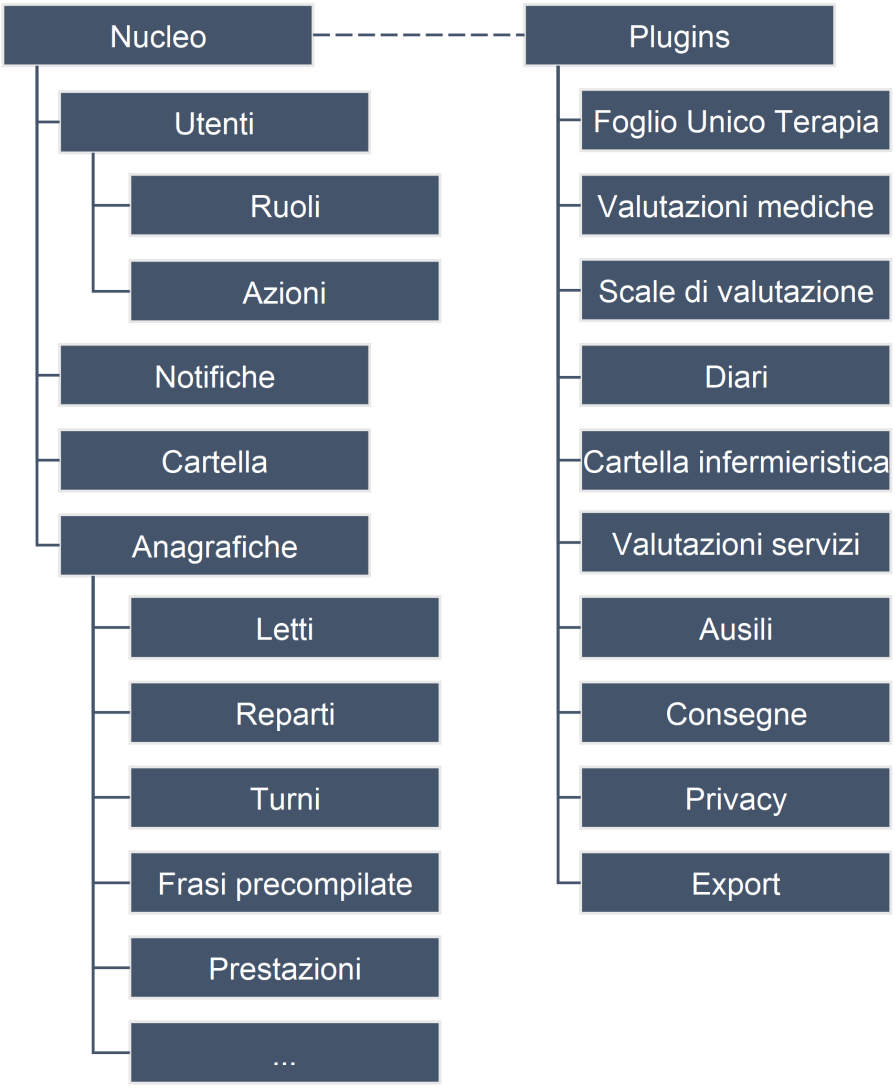


Figura 4.11. Struttura applicazione

- **Utenti**: è la componente che si occupa di creare e gestire le credenziali; serve per identificare la persona che fa accesso all'applicativo. Per il controllo delle autorizzazioni si serve di:

  - **Ruoli**: ogni utente può avere più ruoli in base alle funzioni espletate all'interno della struttura;
  - **Azioni**: sono strettamente collegate ai ruoli e ai controller; per ogni controller presente nell'applicazione (sia del nucleo centrale, sia dei vari plugin) si può definire se quel determinato ruolo può eseguire l'azione presa in considerazione (ad esempio accesso in modalità *modifica* sulla cartella);

- **Notifiche**: necessarie per la generalizzazione di messaggi da dare alle diverse tipologie di utenti in base all'ambito applicativo in cui ci si trova;
- **Cartella**: rappresenta la cartella del paziente come raccoglitore unico di tutte le informazioni anagrafiche (cognome, nome, codice fiscale, data e luogo di nascita, ecc...) e di ricovero (data di ingresso e di dimissione, reparto, letto, numero identificativo, ecc...);
- **Anagrafiche**: grazie alle quali vengono definite tutte le anagrafiche utilizzate in maniera trasversale sull'applicativo, le principali sono:
  - **Reparti**: definiscono i reparti all'interno della struttura, utilizzati all'interno della sezione *ricovero* della cartella; sono indispensabili per presentare in maniera suddivisa tutte le cartelle;
  - **Letti**: strettamente collegati ai reparti; permettono di inserire l'informazione dell'identificativo del letto (solitamente un numero progressivo) e dell'identificativo di camera (solitamente un numero progressivo);
  - **Turni**: ogni figura professionale può avere turni differenti, quindi il turno è dipendente dal ruolo; servono per dividere le informazioni principali nella home page e sono utilizzati nel plugin *FUT* per raggruppare le somministrazioni;
  - **Frasi precompilate**: sono definite per permettere alle varie figure professionali una compilazione più rapida quando si trovano in situazioni standard;
  - **Prestazioni**: definisce le prestazioni da nomenclatore nazionale; permette di distinguere per raggruppamento, branca e codice; viene introdotta la possibilità di distinguere le prestazioni erogabili con l'inserimento di alias e codici mnemonici.

I plugins, per i quali sono stato coinvolto nell'analisi implementativa e nello sviluppo durante il lavoro di tesi e che verranno approfonditi nel capitolo 6, sono i seguenti:

- **Foglio Unico di Terapia - FUT**: all'interno del quale viene gestita la terapia farmacologica del paziente, permette la gestione delle versioni della terapia, schema differenziato per terapia specifica del *warfarin* e la registrazione puntuale delle somministrazioni;
- **Valutazioni mediche**: in cui vengono gestite le valutazioni dei medici, a partire dall'anamnesi patologica prossima fino ad arrivare al decorso del ricovero; grazie a questo plugin vengono integrate anche le anagrafiche relative alle diagnosi, seguendo il nomenclatore ICD9-CM;
- **Scale di valutazione**: in cui vengono inserite tutte le possibili scale di valutazione che possono essere compilate durante il ricovero del paziente. Sono scale di valutazione trasversali alle varie figure professionali: si va dalla scala di Barthel alla scala di Ashworth per le figure mediche, dalla scala di Conley a quella di Braden per le figure infermieristiche, dal test MoCA al test DN4 per le figure di neuropsicologia, ecc... ;
- **Valutazioni servizi**: rappresenta la gestione delle valutazioni dei vari servizi professionali, che possono essere fisioterapia, terapia occupazionale, logopedia, neuropsicologia, ecc...;

- **Consegne**: con cui vengono generate le consegne sia generiche messe a disposizione di tutti gli operatori che hanno accesso all'applicativo, sia chiuse alle singole figure professionali; oltre ad avere una visione delle consegne per il singolo paziente, viene generato un indice di tutte le consegne del singolo reparto, molto utile agli operatori durante le riunioni di equipe;
- **Export**: che serve per l'esportazione di dati ai fini statistici.

Oltre ai plugins sopra citati parte integrante della tesi, questa cartella informatizzata ne mette altri a disposizione della struttura:

- **Diari**: sono divisi per figura professionale e sono indispensabili per tenere traccia dell'andamento del ricovero;
- **Cartella infermieristica**: presenta diversi elementi al suo interno come la diagnosi infermieristica, gli schemi di medicazione di eventuali ferite chirurgiche, la registrazione di parametri vitali (temperatura, pressione, SpO2, ecc...);
- **Ausili**: permette di tenere traccia degli ausili in uso al paziente oltre alla possibilità di prescriverne di nuovi in base alla normativa vigente (attualmente è integrata la modalità di prescrizione definita dalla Regione Piemonte);
- **Privacy**: si presenta come un registro dei moduli privacy e di consenso informato firmati dal paziente o da un suo delegato.

# Capitolo 5

# Nucleo dell'applicazione

Il nucleo dell'applicazione si compone di elementi orizzontali utilizzati nell'intera applicazione. Alcuni di questi elementi sono stati sviluppati per permettere la scrittura di codice in maniera agile, veloce e riutilizzabile.

Non sono stati oggetto di questa tesi gli helper generici per la creazione di campi, come il GuiHelper, i controller generici, come il GuiController, il PolymorphicController e l'HookedController che ereditano dal controller ApplicationController e che sono ereditati da tutti gli altri controller, la gestione del menù di navigazione dell'applicazione (il quale viene presentato in maniera riassuntiva). Questi sono stati progettati e sviluppati dagli sviluppatori di SAN e vengono utilizzati in maniera trasversale su più progetti.

In questo capitolo vengono presentate le componenti sulle quali sono stato coinvolto.

## 5.1 Menù di navigazione

Il menù di navigazione si trova sulla sinistra ed è diviso in tre diverse sezioni:

- *Generale* (figura 5.1): qui sono presenti tutti i reparti dell'ospedale, un link diretto a tutti i pazienti caricati sulla piattaforma, un link ai pazienti ricoverati nel giorno e uno ai pazienti in dimissione nel giorno. Esplodendo il menù per ogni reparto si trovano i link iniettati dai vari plugin (tramite il meccanismo degli hook) come le consegne, e i link diretti alle cartelle dei vari pazienti.
- *Gestione anagrafiche* (figura 5.2 sinistra): visibile soltanto agli utenti con ruolo *Gestione*. Sono disponibili tutte le anagrafiche utilizzate all'interno dell'applicativo e vengono inietatte tutte le anagrafiche utilizzate dai vari plugins.
- *Impostazioni* (figura 5.2 destra): è visibile soltanto agli utenti che hanno il flag superuser attivo. Da qui vengono gestiti gli utenti, i ruoli, le traduzioni dei campi, le preferenze.

Il menù è gestito dalla classe MenuManager ed è renderizzato grazie alla funzione left_menu _entries definita all'interno dell'ApplicationHelper:

```
1 def left_menu_entries
2   #seleziono tutti i reparti
```

```
  division = Folder.distinct.pluck(:division).sort
  #codice riservato per il caricamento delle singole entry dello
  #application_menu
  output = ""
  division.each do |d|
    output += "<div id='#{tmp}_menu'>"
    output += render_menu("#{tmp}_menu".to_sym)
    output += '</div>'
  end
  output += '<div id="application_menu">'
  output += render_menu(:application_menu)
  output += '</div>'
  #se l'utente ha il ruolo gestione
  if hasRole?("Gestione")
    management_menu = '<div id="management_menu">'
    management_menu += render_menu(:management_menu)
    management_menu += '</div>'
    output += management_menu
  end
  #se l'utente ha il flag superuser attivo
  if current_user.superuser
    output += '<div id="superuser_menu">'
    output += render_menu(:superuser_menu)
    output += '</div>'
  end
  output.html_safe
end
```


Figura 5.1. Menù generale e dettaglio sul reparto


Figura 5.2. Menù gestione anagrafiche e impostazioni

## 5.2 Utenti

Definiti dalla tabella in database *users*, dal model *User* e dal controller *UsersController*, rappresentano gli utenti che hanno accesso all'applicazione.

La gestione degli accessi è fatta tramite la *gem* **Devise**. Questa si appoggia al model *User* all'interno del quale sono definiti una serie di campi necessari per il riconoscimento degli utenti:

- email: utilizzata per le comunicazioni inerenti all'applicazione, è un valore univoco in database;
- username: è il nome utente utilizzato per l'accesso; è un valore univoco in database;
- lastname, firstname, initials: descrivono l'utente. Le iniziali devono essere univoche perché utilizzate nella creazione dei documenti PDF;
- division: è il reparto in cui l'utente presta servizio;
- superuser: è un campo booleano utilizzato per indicare che l'utente ha diritti di super utente;
- roles: i ruoli che l'utente può avere; se un utente ha più ruoli la combinazione delle azioni disponibili è rappresentata dall'unione dei due ruoli.

Gli amministratori dell'applicativo, superuser, hanno la possibilità di modificare tutti i campi in questo ambiente mentre un utente base potrà modificare soltanto le informazioni relative al reparto e alla password.

Model di User

```ruby
class User < ApplicationRecord
  ...
  has_many :roles, through: :user_roles
  has_many :user_roles
  devise :database_authenticatable, :registerable,
         :recoverable, :trackable, :validatable, :lockable,
         :timeoutable
  ...
  def display_name
    "#{lastname} #{firstname}"
  end
end
```

### 5.2.1 Devise

Devise è una soluzione flessibile per l'autenticazione su Rails basata su Warden (meccanismo di autenticazione a rack per applicativi Ruby) [15]. Si compone di dieci moduli:

- *Database Authenticable*: responsabile della codifica hash delle password, della loro memorizzazione e della convalida dell'autenticità di un utente durante l'accesso; l'autenticazione può avvenire sia tramite richieste POST sia tramite HTTP Basic Authentication;
- *Omniauthable*: fornisce il supporto a OmniAuth, una libreria che standardizza l'autenticazione multiprovider per le applicazioni web con la possibilità di utilizzo di credenziali provenienti da Google, Facebook, LDAP, ActiveDirectory, ecc;
- *Confirmable*: per l'invio di e-mail con le istruzioni di conferma e verifica se un account è stato già confermato durante l'accesso;
- *Recoverable*: resetta la password dell'utente e invia le istruzioni per concludere la procedura;
- *Registreable*: gestisce il processo di registrazione dell'utente (in questo applicativo questo modulo non è abilitato);
- *Rememberable*: gestisce, genera e crea token per ricordare l'utente in base ai cookie salvati;
- *Trackable*: permette di tenere traccia di alcune informazioni di accesso (timestamp, indirizzo IP, numero di accessi errati, numero di accessi validi, ecc...);
- *Timeoutable*: necessario per far scadere le sessioni utente che non sono attive per un periodo di tempo specificato (in questa applicazione il timeout è impostato a 30 minuti);

- *Validatable*: modulo opzionale che permette di validare l'email, lo username e la password (come il numero minimo di caratteri accettati, quali caratteri vengono accettati, ecc...);
- *Lockable*: permette di impostare alcune regole per il blocco delle utenze (dopo un certo numero di login falliti).

## 5.3 Ruoli

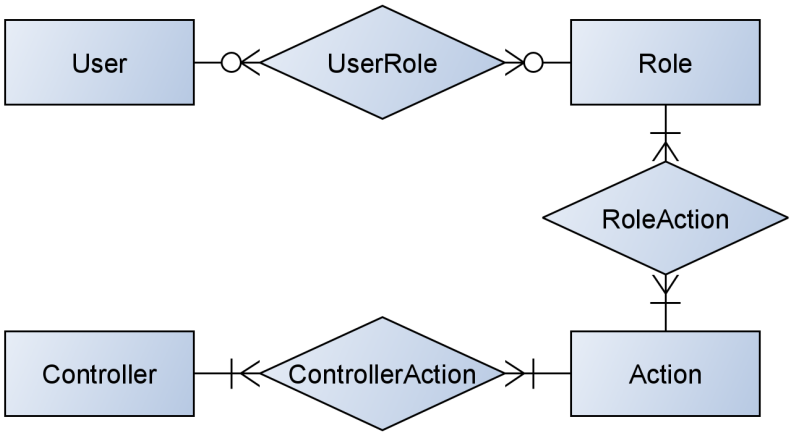


Figura 5.3. Modello ER Ruoli

Come è possibile vedere nel modello ER in figura 5.3, ad ogni utente si possono associare più ruoli.

I ruoli attualmente definiti nell'applicativo sono: medico, infermiere, fisioterapia, logopedia, neuoropsicologia, terapia occupazionale, gestione, accettazione. Ad ogni ruolo sono associate tutte le azioni definite all'interno di tutti i controller, a partire da quelle base (show, edit, update, delete, index) fino a quelle personalizzate (stampa, apertura con determinati parametri, ecc.).

Ad ogni azione è assegnato un flag booleano che determina se l'utente con quel ruolo specifico può eseguire l'operazione richiesta. Nella figura 5.4 viene presentato un estratto del form di compilazione dei ruoli. Sono presenti tre campi principali:

- *identificativo*: campo obbligatorio che identifica il ruolo;
- *descrizione*: campo aggiuntivo per inserire informazioni descrittive;
- *figura professionale*: campo booleano utilizzato per distingure i ruoli associati a figure professionali dai ruoli di gestione.

Le azioni di tutti i controller vengono caricate in maniera dinamica. Sono presenti due funzioni generali per abilitare o disabilitare tutte le azioni di tutti i controller. Inoltre, per ogni controller, sono definite in maniera automatica due funzioni per abilitare o disabilitare tutte le azioni di quel controller.

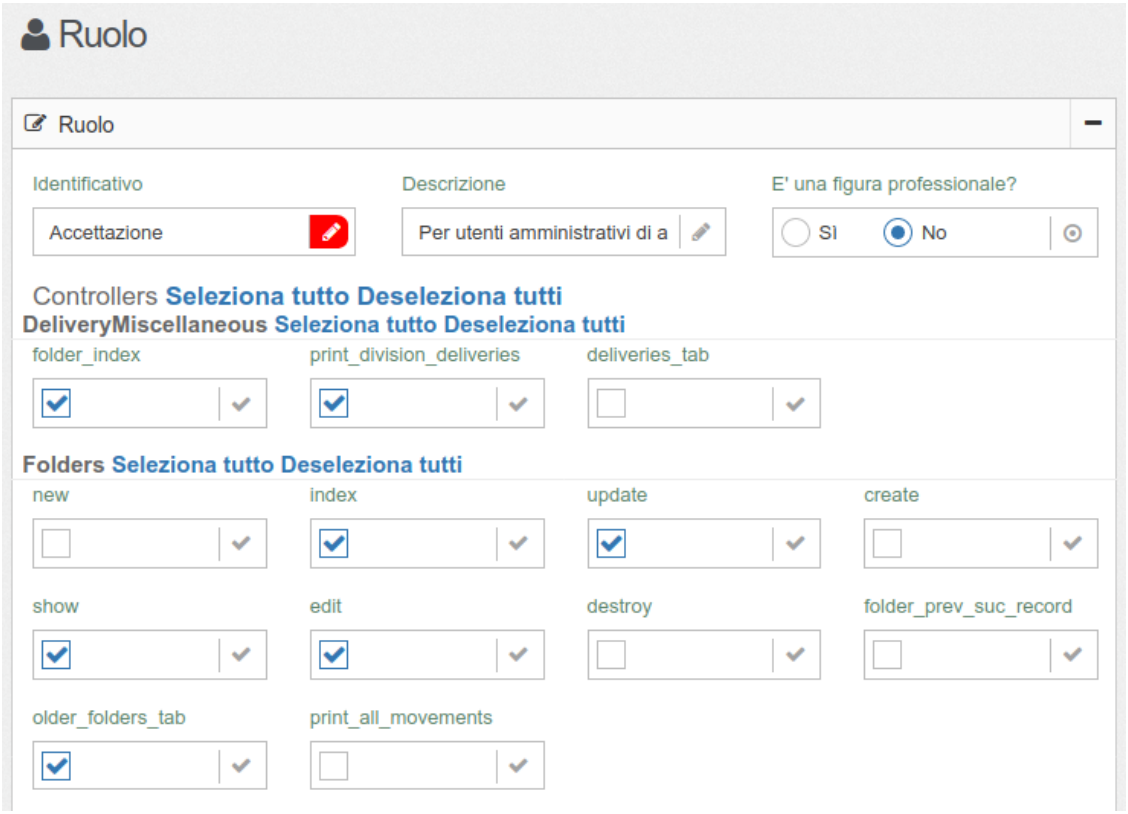


Figura 5.4. Form ruolo

La gestione dei ruoli è demandata agli utenti che hanno il flag *superuser* abilitato.

La verifica dei diritti associati all'utente tramite il ruolo viene effettuata grazie alla *before action* **checkRights** definita nel *GuiController*. Questo metodo restituisce un alert *general.action.unauthorized* se si verificano le seguenti condizioni:

- l'utente non ha il flag di superuser abilitato;
- l'utente non si trova nel root_path (home page dell'applicazione);
- l'azione richiesta non è abiliatata per l'utente corrente.

```
1 class GuiController < ApplicationController
2   include GuiHelper
3   respond_to :html, :json
4   #verifica che la sessione utente sia attiva
5   before_action :authenticate_user!
6   before_action :checkRights #verfica i diritti dell'utente
7   ...
8  end
9
```

```
module ApplicationHelper
  def checkRights
   if !(current_user.superuser ||
  root_path == request.fullpath ||
  _controllerRights(controller_name).try(:include?, action_name))
      unless params[:redirect] == "false"
        flash[:alert] = _("general.action.unauthorized")
        redirect_to page_back_path
      else
        render text: _("general.action.unauthorized")
      end
    end
  end
  ...
  private
  ...
  def _controllerRights(controller)
    controllerRights = _getRights()
    controllerRights[controller.classify.pluralize]
  end
  def _getRights()
    #se le azioni sono state precaricate allora ritorno
    return session[:enabled] if !session[:enabled].nil?
    actions =  Hash.new { |hash, key| hash[key] = [] }
    _current_user_roles.each do |role_id|
      rights = RoleRight.includes(action: :controller)
      rights = rights.where(role_id: role_id)
      #prendo tutte le azioni attivate per il ruolo
      rights.where(rights: true).each do |r|
        a = r.action
        actions[a.controller.controller_name] << a.action_name
      end
    end
    session[:enabled] = actions
  end
end
```

## 5.4 Gestione accesso condiviso

Ogni record dell'applicazione può essere consultato in visualizzazione da un numero indefinito di utenti in contemporanea. Diversa è la situazione se su quel record si richiede l'accesso in modifica.

Per gestire l'accesso esclusivo ai vari record è stato sviluppato un meccanismo di *locking*, grazie al quale l'utente che tenta di fare accesso a un record *bloccato in modifica* da un altro utente riceve l'informazione.

In fase di logout e in fase di login vengono sbloccati tutti i record che hanno un lock per quell'utente, rispettivamente grazie al metodo *destroy* di **SessionController** e al metodo *after_sign_in_path_for* di **ApplicationController**.

```
class SessionsController < Devise::SessionsController
  ...
  def destroy
    #seleziono tutti i record bloccati della sessione dalla quale
    #si sta facendo logout e li aggiorno inserendo nel timestamp
    #di rilascio l'ora e la data attuale
    lr = LockedRecord.where(released: nil, session_id: session.id)
    lr.update(released: DateTime.now)
    super
    session[:userRights] = nil
    flash.delete(:notice)
    flash.delete(:alert)
  end
end

class ApplicationController < ActionController::Base
  ...
  def after_sign_in_path_for(resource_or_scope)
    #seleziono tutti i record bloccati dell'utente che sta
    #eseguendo il sign in e li aggiorno inserendo nel timestamp
    #di rilascio l'ora e la data attuale
    lr = LockedRecord.where(released: nil, user: current_user)
    lr.update(released: DateTime.now)
    super
  end
end
```

Nel metodo che viene richiamato per eseguire il render del record in modifica viene eseguito, prima della risposta al client, il metodo privato *_checkEditLock*. Questo metodo verifica se esiste almeno un record bloccato in modifica:

- se non esiste allora crea un nuovo lock tramite il metodo *_createLock*;
- se esiste allora viene richiamato il metodo *_checkOrUpdateLockRecord*:
  - se il record è bloccato con diritto da un altro utente allora viene visualizzato il messaggio di warning "Record bloccato: Cognome e nome utente che ha bloccato il record";
  - se il record è bloccato ma la sessione dell'utente che lo blocca è scaduta allora verrà rilasciato il lock precedente e sarà possibile modificare il record stesso.

```
class GuiController < ApplicationController
  ...
  def _checkEditLock
```

```
4     ret = true
5     rr = @request_record
6     lr = LockedRecord.where(released: nil)
7     lr = lr.where(item_type: rr.class.name, item_id: rr.id)
8     if lr.blank?
9       _createLock
10      ret = false
11    else
12      ret = _checkOrUpdateLockRecord(lr)
13    end
14    ret
15  end
16
17  def _checkOrUpdateLockRecord(l_rec)
18    return false if l_rec.blank?
19    ret = true
20    if l_rec.session_id != session.id
21      u = l_rec.user
22      if u.timedout?(u.current_sign_in_at)
23        l_rec.update(released: DateTime.now)
24        _createLock
25        ret = false
26      else
27        @msglocked = "Record bloccato: #{u.display_name}"
28      end
29    else
30      ret = false
31    end
32    ret
33  end
34
35  def _createLock
36    LockedRecord.create(
37      item_type: @hooked_record.class.name,
38      item_id: @hooked_record.id.to_i,
39      user: current_user,
40      session_id: session.id,
41      reference_item_type: @ref_record.class.name,
42      reference_item_id: @ref_record.id)
43  end
```

Oltre all'automatismo sopra descritto è stata implementata una funzione che permette agli utenti superuser di sbloccare in maniera manuale i record bloccati: può accadere che l'utente non abbia eseguito il logout e un altro utente abbia urgenza di modificare il record. Questi record sono accessibili dal menù "Impostazioni -> Record bloccati" e resi disponibili tramite il metodo *_releaseRecord* (figura 5.5).

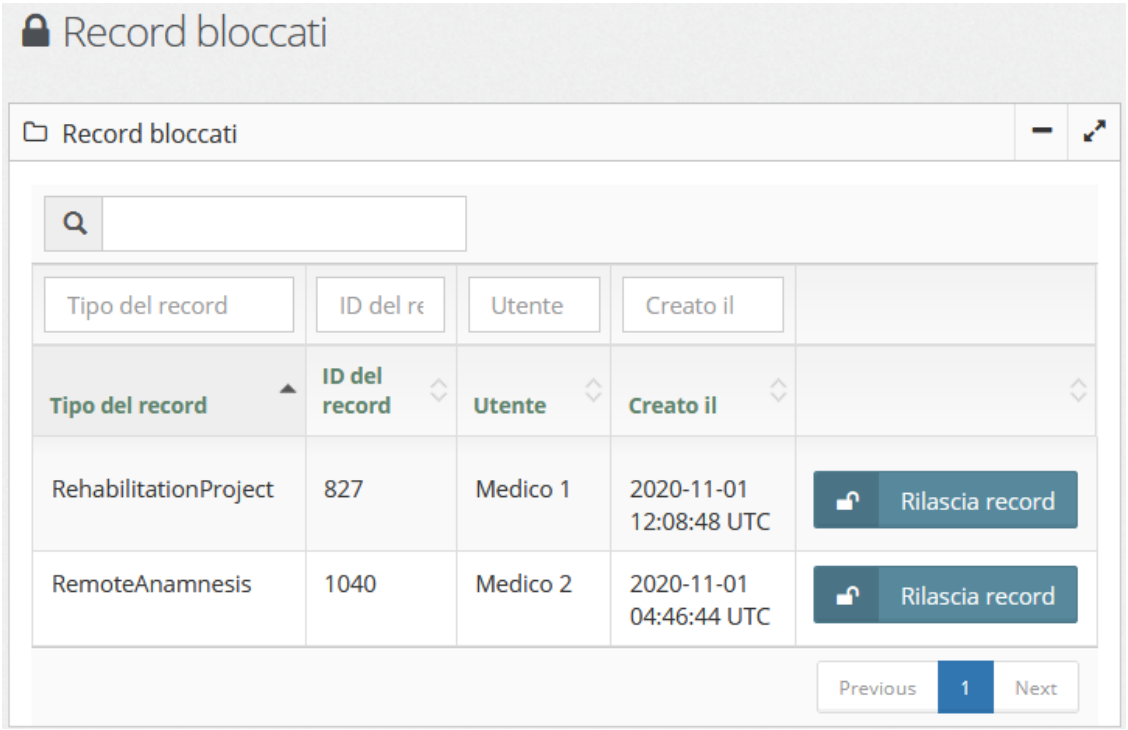


Figura 5.5. Record bloccati

## 5.5 Gestione alert

La gestione degli alert è stata sviluppata per permettere la creazione di messaggi personalizzati, ripetibili e tracciabili per un dato periodo di tempo senza l'intervento degli sviluppatori. La definizione avviene a partire dal menù "Anagrafiche -> Alert", disponibile quindi per le utenze con ruolo "Gestione" abilitato e tramite il form di inserimento in figura 5.6.

I campi sono i seguenti:

- *tipo record*: obbligatorio, indica su che tipologia di record deve essere visualizzato l'alert;
- *data inizio* e *data fine*: indicano la data di inizio (obbligatoria) e la data di fine durante le quali verrà visualizzato l'alert;
- *tutti i giorni*: definisce se l'alert deve essere visualizzato ogni giorno;
- *descrizione*: obbligatorio, rappresenta il messaggio da dare in maniera sintetica;
- *messaggio*: obbligatorio, indica il messaggio visualizzato all'utente;
- *a quali ruoli*: vengono indicati a quali ruoli deve essere visualizzato il messaggio.

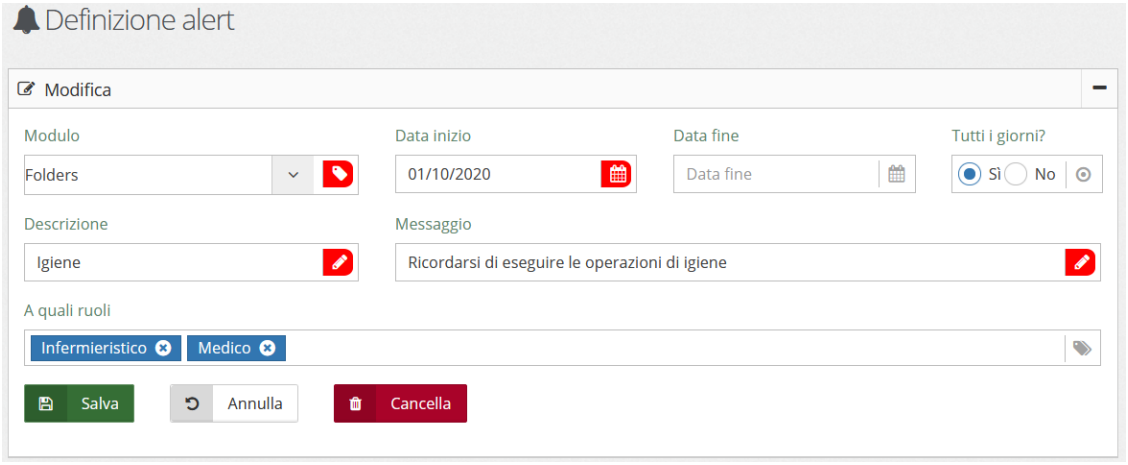


Figura 5.6. Definizione alert

L'utente che ha uno dei ruoli definiti nell'alert visualizzerà il messaggio come in figura 5.7 fino a quando non imposterà il flag "Ho capito". Nel caso in cui l'utente volesse rileggere nuovamente il messaggio il giorno successivo, basterà settare il flag "Chiedi di nuovo".

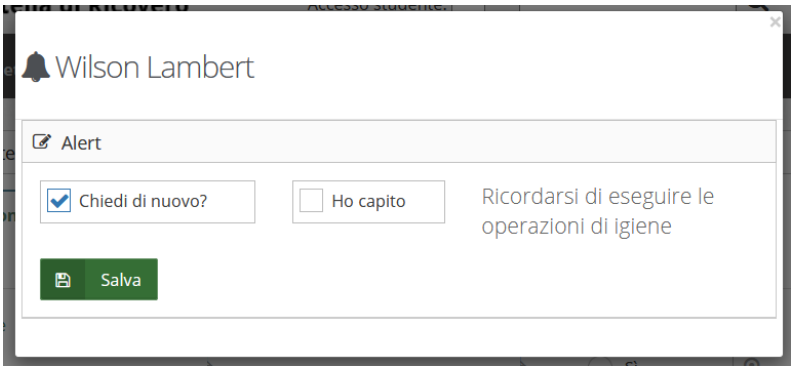


Figura 5.7. Visualizzazione alert

La verifica sulla visualizzazione degli alert viene eseguita all'interno del metodo *get _related_record* di **GuiController**.

## 5.6 Cartella

La cartella è il fulcro di questo applicativo e si presenta come il raccoglitore di tutte le informazioni raccolte nei model legati al paziente e definiti all'interno dei plugin. Questi ultimi sono caricati tramite il meccanismo degli *hook*: viene istanziato il metodo di call_back *controller_folder_get_modal_releted_record* richiamato, dove necessario, nell'inizializzazione dei plugin.

Nel *model* base della cartella vengono definiti alcuni metodi per accesso facilitato alle informazioni:

```
class Folder < ApplicationRecord
  belongs_to  :gender
  ...
  has_many    :folder_reminders
#valore di ritorno: stringa con letto, cognome e nome del paziente
  def display
    "#{bed} - #{lastname} #{firstname}"
  end
  #valore di ritorno: anni del paziente fino alla data di
   dimissione
  def age_value
    value = end_date.blank? ? end_date - birth_date : nil
    #se value viene settato con nil allora paziente in ricovero
    value ||= Date.today - birth_date
    value = (value.to_i)/365
    "#{value}"
  end
  #valore di ritorno: giorni di ricovero
  def hospitalization_days
    value = end_date.blank? ? end_date - start_date : nil
    #se value viene settato con nil allora paziente in ricovero
    value ||= Date.today - start_date
    value = (value.to_i)/365
    "#{value}"
  end
  ...
end
```

Il *model* fa riferimento alla tabella *folders* definita su PostgreSQL della quale vengono esposti i campi e i riferimenti principali nel paragrafo 5.6.2.

La creazione di una cartella può avvenire in due modi differenti:

- tramite il normale flusso web, definito quindi dai metodi *new* e *create* di **GuiController**;
- tramite servizio web utilizzando l'API **ApiFoldersController**, creata per integrare le funzionalità di questo applicativo con i gestionali già esistenti nella struttura ospedaliera. Questa API espone due metodi, *create* e *update*, che rispondono a richieste di tipo JSON.

### 5.6.1 Le *view* della cartella

**FORM: utilizzato per visualizzazione e modifica**

Il form si presenta diviso in due parti logiche: nella prima parte (figura 5.8) sono presentate le informazioni anagrafiche e di contatto, le informazioni principali di ricovero

(data ricovero/dimissione, reparto, letto, tipo del ricovero) e la raccolta delle notifiche presentate nel paragrafo precedente; nella seconda parte (figura 5.9) sono agganciati i plugin grazie alla definizione di tre *hook* diversi:

- *view_folder_li* richiamato per renderizzare l'elenco delle funzionalità;
- *view_folder_tab_content* richiamato per renderizzare il contenuto delle funzionalità;
- *view_folder_prints_tab_content* richiamato per renderizzare scorciatoie per la realizzazione di stampe elettroniche.

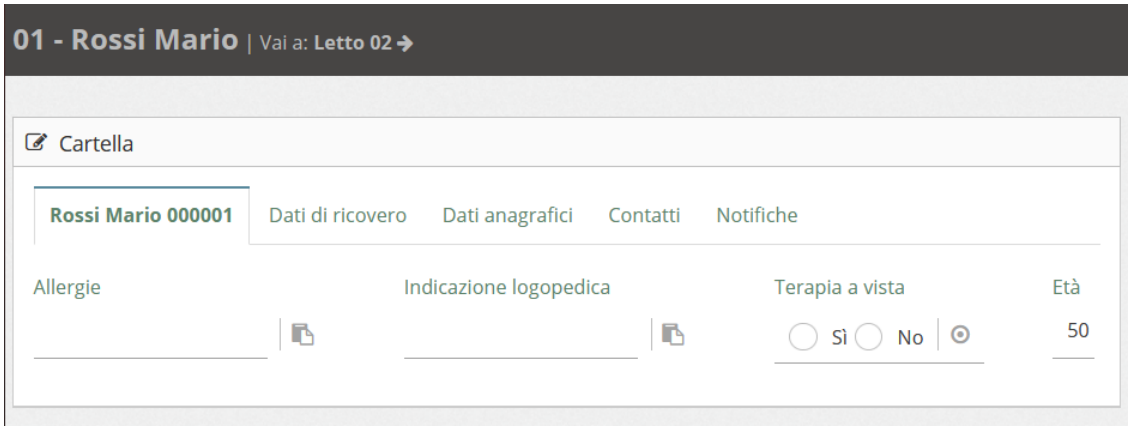


Figura 5.8. Form cartella: anagrafica e informazioni trasversali

Di seguito viene presentato un estratto di _form.html in cui è presente codice Ruby e dove sono definiti gli *hook* descritti in precedenza:

```
<%= guiWellOpen("row") %>
  <div class="tabs-left">
    <ul class="nav␣nav-tabs␣tabs-left" id="folder-tabs">
    #hook utilizzato dai plugin per renderizzare l'elenco degli
    # oggetti disponibili
      <%= call_hook(:view_folder_li) %>
      <li><a href="#folder-prints-tab" data-toggle="tab" >
        <i class="fa␣fa-print"></i> <%= _("folder_prints_tab") %>
      </a></li>
    </ul>
    <div class="tab-content" id="folder-tabs-content">
    #hook utilizzato dai plugin per renderizzare il contenuto
    # oggetti disponibili
      <%= call_hook(:view_folder_tab_content) %>
      <div class="tab-pane" id="folder-prints-tab">
        <div class="row">
    #hook utilizzato dai plugin per renderizzare le stampe
    #in pdf che si possono generare
```

```
19            <%= call_hook(:view_folder_prints_tab_content) %>
20          </div>
21        </div>
22      </div>
23    </div>
24  <%= guiWellClose %>
```

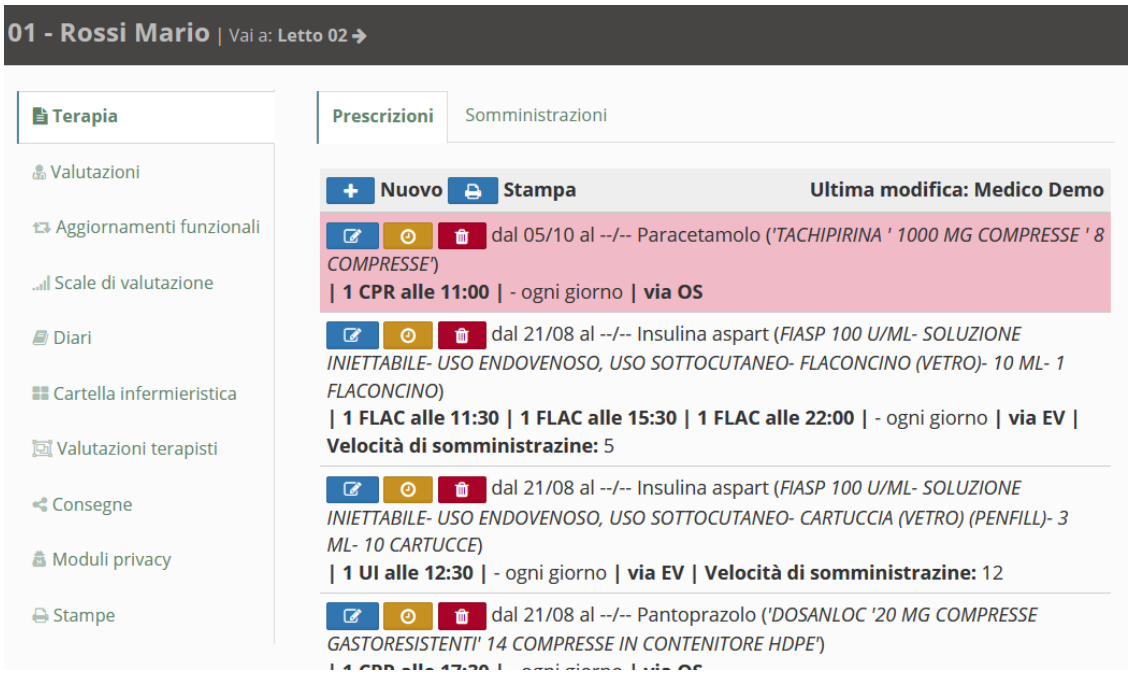


Figura 5.9. Form cartella: funzionalità caricate dinamicamente dai plugin

### INDEX: utilizzato per elencare le cartelle

Sono previste due modalità di visualizzazione per gli elenchi delle cartelle.

La prima modalità di visualizzazione (figura 5.10) presenta tutti i pazienti, in ricovero e dimessi, sotto forma di tabella. Viene sfruttato il plugin *DataTables* di *jQuery*.

In questa visualizzazione è possibile eseguire ricerche in base al numero di cartella, alla data di ricovero, alla data di dimissione, al cognome e al nome del paziente, al reparto e al letto. Vista la grande quantità di dati che verrebbero caricati si è scelto di utilizzare la modalità *server side*: in questo modo i dati vengono caricati man mano che si naviga tra di essi.

Porzione di index.html.erb

```
1 #inserisco i tag di apertura della datatable. Tramite il parametro
2 #src dichiaro che si tratta di una tabella di tipo server side
3 <%= guiDatatableOpen({id: "folders",
4     src: "#{folders_path(format:␣"json"}")%>
5   <thead>
```

```
    <tr>
   #in questa intestazione vengono inseriti i campi per la ricerca
      <%= guiDatatableThFilter(@rset, :number)%>
      <%= guiDatatableThFilter(@rset, :start_date)%>
     #aggiungo tutte le colonne necessarie
    </tr>
    <tr>
   #in questa intestazione vengono inseriti i titoli delle colonne
      <th><%=guiTrFieldLabel(@rset, :number)%></th>
      <th><%=guiTrFieldLabel(@rset, :start_date)%></th>
      #aggiungo tutte le colonne necessarie
    </tr>
  </thead>
  <tbody>
    <% @rset.each do |f| %>
      <tr>
        <td><%= guiFieldText(f, :number) %></td>
        <td><%= guiFieldDatetime(f, :start_date) %></td>
       #aggiungo tutte le colonne necessarie
        <td><%= guiButtonShow(f) %></td>
      </tr>
    <% end %>
  </tbody>
<%= guiDatatableClose %>
```

La seconda modalità di visualizzazione (figura 5.11) riguarda i pazienti ricoverati ed è stata pensata per avere un impatto visivo diretto su alcuni aspetti come il letto, la data di ricovero, la tipologia del ricovero, ecc...

In questa visualizzazione è stato inserito l'*hook* **view_folder_icons** che può essere richiamato dai vari plugin per poter aggiungere icone, all'interno del riquadro identificativo del paziente, per porre all'attenzione di chi guarda determinati tipi terapia in corso, eventuali ausili in uso, parametri vitali da misurare e così via dicendo.

Porzione di index.html.erb

```
<%= link_to folder_path(f) do %>
  <% bg = f.hospitalization_burden.try(:color).to_s %>
  #imposto il background in base all'onere degenza
  <div class="quarto" style="background:␣<%=␣bg␣%>">
    Letto <b><%= f.bed%></b> - Cartella <b><%= f.number%></b><br>
    <%= "#{f.display_name}" %><br>
    Ricovero <%= f.start_date.strftime("%d/%m/%Y") %><br>
    <div style="margin-top:␣5px">
   #hook per il rendering delle icone caricate dai plugin
      <%= call_hook(:view_folder_icons) %>
    </div>
  </div>
<% end %>
```

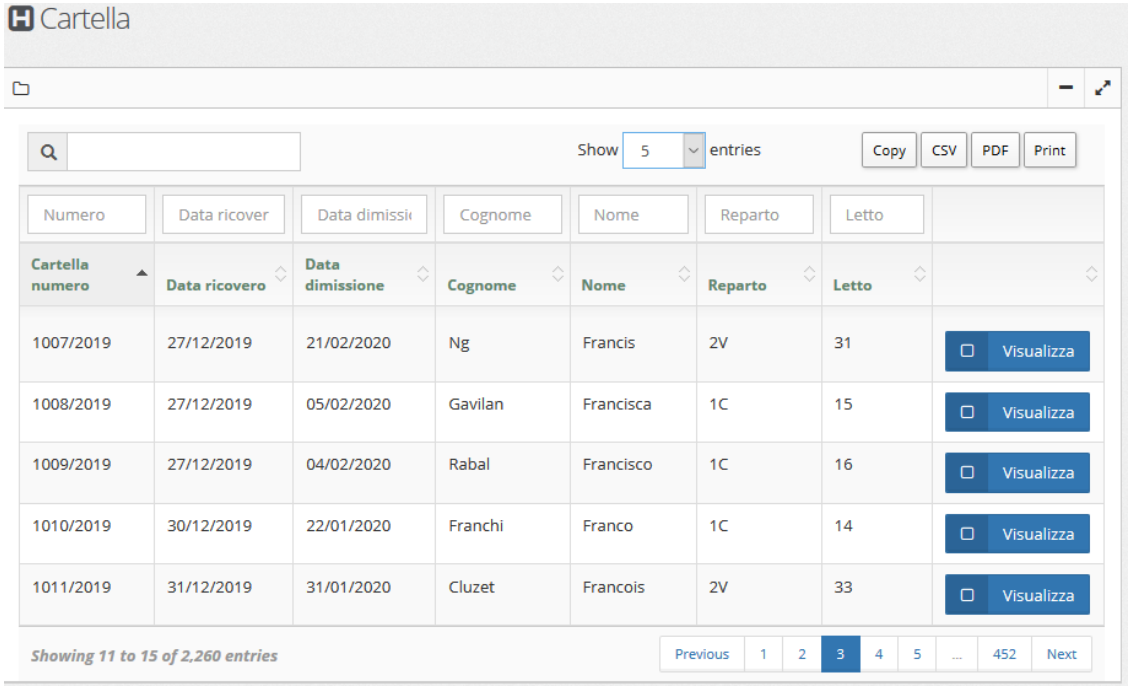


Figura 5.10. Elenco pazienti

### 5.6.2 Approfondimento sulla tabella folders

Sulla tabella folders (5.1) fanno riferimento tutti i record presenti in questo applicativo.

| Column | Type | Nullable | Descrizione |
|---|---|---|---|
| id | integer | not null | |
| number | character varying | not null | numero della cartella |
| start_date | timestamp without time zone | not null | data di ricovero |
| end_date | timestamp without time zone | | data di dimissione |
| lastname | character varying | not null | cognome del paziente |
| firstname | character varying | not null | nome del paziente |
| cf | character varying | not null | codice fiscale del paziente |
| birth_date | timestamp without time zone | not null | data di nascita del paziente |
| patient_code | character varying | not null | codice esterno del paziente |
| division | character varying | | reparto |
| bed | character varying | | letto |
| created_at | timestamp without time zone | not null | timestamp creazione |
| updated_at | timestamp without time zone | not null | timestamp aggiornamento |
| ... | ... | ... | ... |

Tabella 5.1. Tabella folders in PostgreSQL

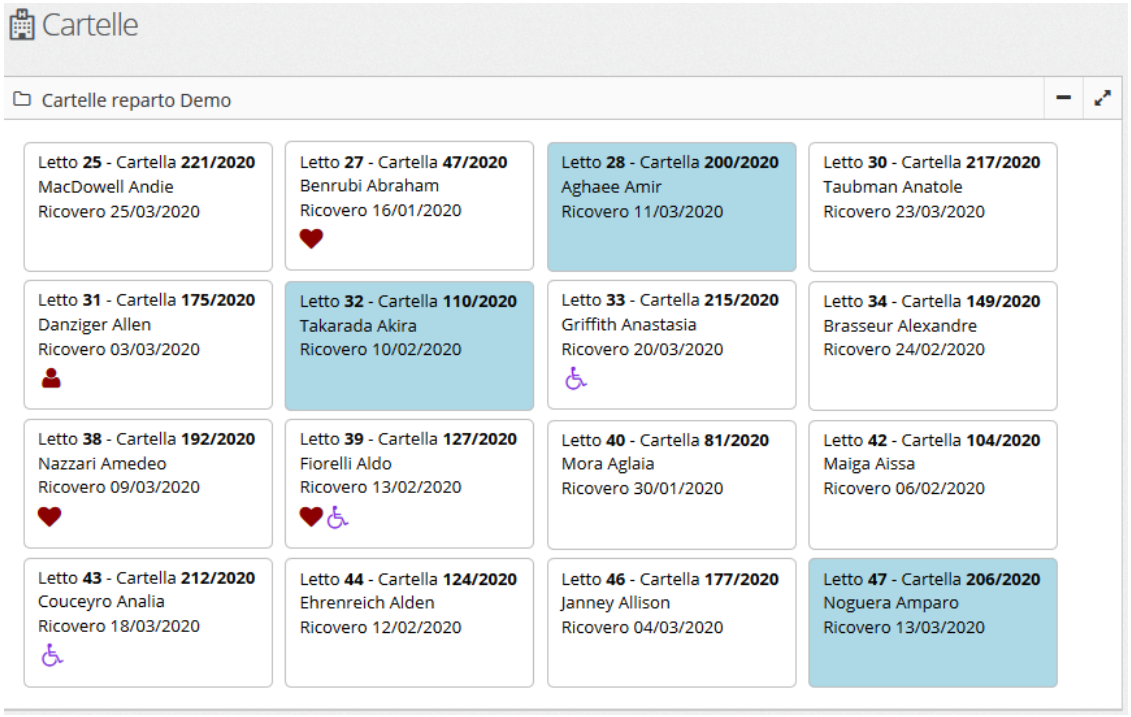


Figura 5.11. Elenco pazienti ricoverati al reparto Demo

Su questa tabella sono presenti i seguenti indici principali:

- *"folders_pkey" PRIMARY KEY, btree (id)*, rappresenta la chiave primaria del record;
- *"unique_number" UNIQUE CONSTRAINT, btree (number)*, vincolo di univocità sul numero della cartella;
- *"index_folders_on_number" btree (number), "index_folders_on_lastname" btree (lastname), "index_folders_on_start_date" btree (start_date)*, utilizzati per rendere più performanti le operazioni di ricerca.

Sulla tabella *folders* sono presenti alcuni vincoli di chiavi esterne, tra i quali:

- *"fk_rails_33de01c54a" FOREIGN KEY (provenance_id) REFERENCES provenances(id)*, su provenienza assistito;
- *"fk_rails_c833aa77b9" FOREIGN KEY (hospital_bed_id) REFERENCES hospital_beds(id)*, sui letti del reparto presenti in ospedale;
- *"fk_rails_e3ff97cdf3" FOREIGN KEY (hospitalization_burden_id) REFERENCES hospitalization_burdens(id)*, sull'onere degenza;
- *"fk_rails_e73378ab0c" FOREIGN KEY (hospitalization_type_id) REFERENCES hospitalization_types(id)*, sulla tipologia del ricovero.

La presente tabella, come detto precedentemente, è referenziata da tutti i record che si riferiscono al paziente definiti in questa applicazione. In particolare:

- *TABLE "folder_reminders" CONSTRAINT "fk_rails_be78a10432" FOREIGN KEY (folder_id) REFERENCES folders(id)*, per la gestione delle notifiche;
- *TABLE "futs" CONSTRAINT "fk_rails_a1e4514ab0" FOREIGN KEY (folder_id) REFERENCES folders(id)*, foglio unico di terapia, referenziata al caricamento del plugin **FUT** che si occuperà di caricare anche i riferimenti per **warfarin** e **permessi**;
- *TABLE "next_anamneses" CONSTRAINT "fk_rails_d68f1498e6" FOREIGN KEY (folder_id) REFERENCES folders(id)*, anamnesi patologica prossima, referenziata al caricamento del plugin **Valutazioni mediche** che si occuperà di caricare anche i riferimenti per **diagnosi**, **esame obiettivo**, **aggiornamenti funzionali**, ecc.;
- *TABLE "barthels" CONSTRAINT "fk_rails_1a5ca089ba" FOREIGN KEY (folder_id) REFERENCES folders(id)*, scala di Barthel, referenziata al caricamento del plugin **Scale di valutazione** che si occuperà di caricare anche i riferimenti per **FIM**, **Nine hole test**, **Scala di Ashworth** e molte altre scale di valutazione;
- *TABLE "fkt_evaluations" CONSTRAINT "fk_rails_258ccae7ba" FOREIGN KEY (folder_id) REFERENCES folders(id)*, valutazioni fisioterapisti, referenziata al caricamento del plugin **Valutazioni dei servizi** che si occuperà di caricare anche i riferimenti per le valutazioni di **logopedia**, **terapia occupazionale** e **neuropsicologia**;
- *TABLE "deliveries" CONSTRAINT "fk_rails_95c91eb2b9" FOREIGN KEY (folder_id) REFERENCES folders(id)*, consegne, referenziata al caricamento del plugin **Consegne**;
- *TABLE "privacies" CONSTRAINT "fk_rails_f38864bda5" FOREIGN KEY (folder_id) REFERENCES folders(id)*, moduli privacy, referenziata al caricamento del plugin **Privacy**.

# Capitolo 6

# Plugin

## 6.1 Creazione di un plugin e fusione col nucleo

La creazione di un plugin avviene tramite il comando *bundle exec ruby bin/rails generate redmine_plugin <nomeplugin>*. Questo creerà, all'interno della cartella plugins, la cartella *nomeplugin* con tutto il sottoalbero necessario per il corretto funzionamento del plugin stesso (avremo la struttura molto simile ad un'applicazione Rails a sé stante).

Uno dei file creato con il comando precedente è **init.rb**. Come dice il nome stesso, questo file è il responsabile dell'inizializzazione del plugin: include informazioni riguardanti il nome, l'autore, la descrizione, la versione, l'url. In questo file vengono dichiarate anche le varie dipendenze da *hook* e *patch* oltre all'inserimento di nuove voci nel menù.

init.rb del plugin Evaluations

```
1  require 'redmine' #necessario per la gestione del plugin
2  #registro i listener su eventuali hook
3  require_dependency 'evaluation_hook_listener'
4  #registro dipendenze sui model del nucleo dell'applicazione
5  require_dependency 'patches/evaluation_folder_patch'
6  require_dependency 'patches/evaluation_role_patch'
7
8  Redmine::Plugin.register :a010_evaluation do
9    name 'Evaluation plugin'
10   author 'Carmelo Metallo'
11   description 'Valutazioni per Cartella informatizzata'
12   version '1.0.1'
13   url 'https://gitlab.san.it/san/ita-medrec-evaluation.git'
14   author_url 'http://www.san.it'
15 end
16 #aggiungo le voci di menu: viene indicato per il controller
17 #l'azione da richiamare (in questo caso index di framings);
18 #il parent nel caso in cui fosse annidato; caption, label e title
19 #utilizzando la funzione _ necesssaria per le traduzioni.
20 Redmine::MenuManager.map(:management_menu) do |menu|
```

```
21   menu.push(:framings, {controller: 'framings', action: 'index'},
22       {caption: _('menu.framings.title'),parent: :management,
23        icon: "fa-files-o", label: _('menu.framings.label'),
24        title: _('menu.framings.title')})
25 end
26 #includo nei model del nucleo dell'applicazione le relazioni con
27 #i model del plugin
28 ActionDispatch::Callbacks.to_prepare do
29   ::Folder.send(:include, Evaluations::FolderPatch)
30   ::Role.send(:include, Evaluations::RolePatch)
31 end
```

I plugin vengono caricati in maniera automatica all'interno dell'*initializer* **14-plugin_and_menu.rb** definito nel nucleo applicativo, il quale richiama la funzione **load** della classe **Plugin**:

```
1  def self.load
2    #ciclo su tutti plugin, in ordine alfabetico, definiti nella
3    #directory plugins
4    Dir.glob(File.join(self.dir, '*')).sort.each do |directory|
5      if File.directory?(directory)
6        lib = File.join(directory, "lib")
7      #all'interno della cartella lib dei plugin vengono inserite le
8      #dipendenze al livello 'model'
9        if File.directory?(lib)
10         $:.unshift lib
11         ActiveSupport::Dependencies.autoload_paths += [lib]
12       end
13     #il file 'init.rb' permette l'inizializzazione del plugin
14       initializer = File.join(directory, "init.rb")
15       if File.file?(initializer)
16         require initializer
17       end
18     end
19   end
20 end
```

In tutti i plugin che hanno delle dipendenze o che aggiungono delle dipendenze al nucleo applicativo, vengono definite delle patch sotto forma di classi all'interno della cartella *patches* di *lib* del plugin stesso.

therapist_evaluation_folder_patch.rb - Valutazioni dei servizi

```
1  #viene richiamato il model di base, in questo caso 'folder'
2  require_dependency 'folder'
3  module TherapistEvaluations
4    module FolderPatch
5      def self.included(base)
6        base.class_eval do
```

```
    #vengono aggiunte le relazioni come all'interno dei model
        has_many    :fkt_evaluations
        has_many    :to_evaluations
        has_many    :logo_evaluations
        has_many    :neuro_psyco_evaluations
        has_many    :food_cards
      end
    end
  end
end
```

Gli *hook listener* registrati tramite il file **init.rb** ereditano dalla classe **Redmine::Hook::ViewListener** e servono per renderizzare view o parti di view in determinati punti prestabiliti e per estendere funzionalità dei controller dove sono state inserite delle callback (come nel metodo *get_related_record* del controller *folders*).

evaluation_hook_listener.rb

```
class EvaluationHookListener < Redmine::Hook::ViewListener
#vengono aggiunte le voci nell'elenco delle tab disponibili
#all'interno della cartella tramite questo render
  render_on :view_folder_li,
   partial: "evaluation_miscellaneus/liview"
#vengono aggiunti i contenuti delle tab nella cartella
  render_on :view_folder_tab_content,
   partial: "evaluation_miscellaneus/nested_index"
#create per aggiungere eventuali pulsanti per la generazione
#di documeti in PDF
  render_on :view_folder_prints_tab_content,
   partial: "evaluation_miscellaneus/prints_tab"
#context rappresenta la variabile di contesto accessibile nelle
#view tramite @hook_variable
  def controller_folder_get_releted_record(context={ })
    obj = context[:hook_variable]
#viene utilizzato il flag previous_recovery per caricare il
#pulsante di import dell'eventuale ricovero precedente
    obj[:previous_recovery] =
       Folder.where(cf: obj[:rec].cf).length > 1
       && obj[:rec].framing.nil?
  end
end
```

Una ulteriore caratteristica comune a tutti i form caricati tramite i plugin riguarda la visualizzazione della storia delle modifiche (figura 6.1). Questa è sempre disponibile in primo piano ed è renderizzata tramite il richiamo del metodo *GuiShowHistory*.

```
def guiShowHistory(obj, reverse=false, opt={})
  out = ""
 #vengono impostate delle classi css di default se non esplicitate
```

```
4    opt[:section_class]||= 'col␣col-10'
5    opt[:div_class]||= 'riga'
6   #il parametro reverse determina se visualizzare in ordine
7   #cronologico inverso
8    versions = reverse ? obj.versions.reverse : obj.versions
9    out += "<section␣class='#{opt[:section_class]}'>"
10   versions.each do |v|
11    out += "<div␣class='#{opt[:div_class]}'>"
12    out += v.event == "create" ?
13      "#{_('version.created_at')}␣" "#{_('version.updated_at')}␣"
14    out += "<strong>#{v.created_at.localtime.strftime
15␣␣␣␣␣'%d/%m/%Y␣%H:%M'}␣</strong>"
16    out += "<strong>#{User.find(v.whodunnit).display_name}"
17    out += "</strong></div>"
18   end
19   out += "</section>"
20   out.html_safe
21 end
```

Aggiornato il **04/11/2020 16:30** da **Medico Due**

Aggiornato il **04/11/2020 15:43** da **Medico Due**

Aggiornato il **04/11/2020 11:29** da **Medico Uno**

Creato il **04/11/2020 11:28** da **Medico Uno**

Figura 6.1. Render generato da GuiShowHistory

Grazie all'utilizzo della *gem* **paper_trail**, ogni operazione sui record genera una nuova versione dello stesso oltre a creare un record di versione nella tabella **versions** (figura 6.2).

Questa tabella è composta da:

- id: chiave primaria che si autoincrementa, è l'identificativo della versione;
- item_type: tipo del record, salvato nella forma camel case (nella figura d'esempio ObjectiveEamination);
- item_id: identificativo del record;
- event: evento che ha generato la versione, può essere *create*, *update*, *destroy*;
- whodunnit: identificativo dell'utente che ha generato la versione, se la versione è stata generata da console questo valore è nullo;
- object: rappresenta l'oggetto nella versione precedente a quella appena salvata;
- created_at: timestamp di creazione della versione.

| id | item_type | item_id | event | whodunnit | object | created_at |
|---|---|---|---|---|---|---|
| 196491 | ObjectiveExamination | 1 | create | 1 | | 2020-11-04 10:28:08.407247 |
| 196492 | ObjectiveExamination | 1 | update | 1 | --- | 2020-11-04 10:29:17.954513 |
| | | | | | id: 1 | |
| | | | | | folder_id: 1 | |
| | | | | | print_id: 17 | |
| | | | | | consciousness_state_id: 1 | |
| | | | | | pulmonary_id: 1 | |
| | | | | | abdomen_id: | |
| | | | | | cardiac_id: | |
| | | | | | cutis_id: | |
| | | | | | evaluation_date: | |
| | | | | | pulmonary_note: | |
| | | | | | abdomen_note: | |
| | | | | | cardiac_note: | |
| | | | | | cutis_note: | |
| | | | | | consciousness_state_note: | |
| | | | | | created_at: 2020-11-04 10:28:08.407247000 Z | |
| | | | | | updated_at: 2020-11-04 10:28:08.407247000 Z | |
| | | | | | more: | |

Figura 6.2. Estratto da tabella versions

## 6.2 Valutazioni mediche

Tramite questo plugin vengono gestite tutte le valutazioni legate al paziente per tutte le fasi del ricovero: ingresso, intermedie, dimissione.

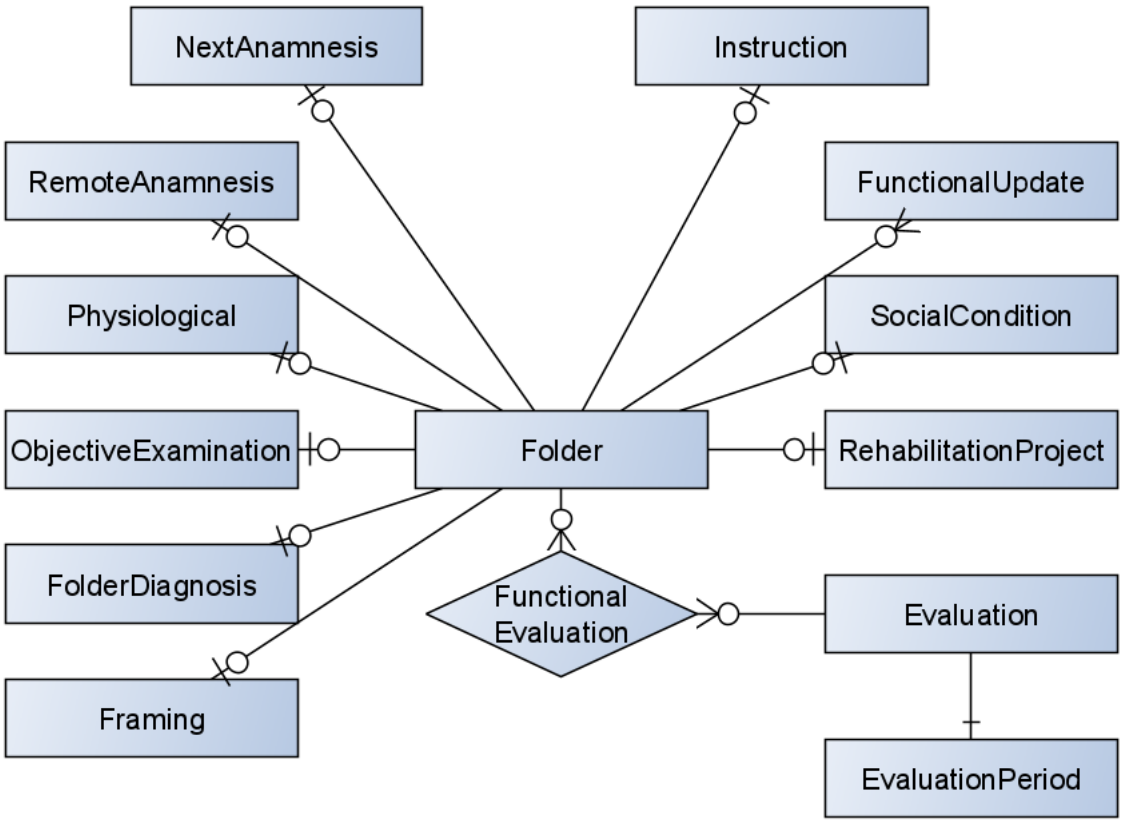


Figura 6.3. Modello ER delle valutazioni mediche

Nel modello ER in figura 6.3 sono presentati i vari model che ne fanno parte:

- *NextAnamnesis*: relazione has_one con la cartella, rappresenta l'anamnesi patologica prossima;
- *RemoteAnamnesis*: relazione has_one con la cartella, descrive l'anamnesi patologica remota;
- *Physiological*: relazione has_one con la cartella, si compone di informazioni sulla valutazione fisiologica;
- *ObjectiveExamination*: realzione has_one con la cartella, rappresenta l'esame obiettivo;
- *FolderDiagnosis*: relazione has_one con la cartella, raccoglie le diagnosi di ingresso per il paziente;
- *Framing*: relazione belongs_to con la cartella, definisce l'inquadramento generale assegnato al paziente;
- *FunctionalEvaluation*: relazione has_many :through, colleziona tutte le valutazioni funzionali, ogni valutazione ha una relazione belongs_to con il periodo di compilazione (ingresso, intermedio, dimissione - la struttura ospedaliera può decidere quanti periodi rendere disponibili);
- *RehabilitationProject*: relazione has_one con la cartella, rappresenta il progetto riabilitativo;
- *SocialCondition*: relazione has_one con la cartella in cui vengono indicate le condizioni socio ambientali del paziente;
- *FunctionalUpdate*: relazione has_many con la cartella, necessario per la registrazione degli aggiornamenti funzionali durante il ricovero del paziente;
- *Instruction*: relazione has_one con la cartella, include le indicazioni e il decorso del ricovero in maniera descrittiva.

Tutti i controller di questo plugin, ad eccezione del *framing*, ereditano dal controller generico **HookedGuiController**.

La parte grafica è organizzata in diverse tab, caricate e renderizzate soltanto se richiesto tramite chiamate XHR. La richiesta inoltrata al server genera il form annidato del record impostando il salvataggio automatico ogni x secondi (il valore di x dipende dal parametro applicativo *form.autosave.timeout*).

Estratto dalla funzione helper GuiForm per salvataggio automatico

```
...
out += "<script␣type='text/javascript'>"
#compongo l'identificativo del form in base al fatto che sia un
#nuovo record oppure una modifica
form_id = _record.persisted? ? "edit_" : "new_"
```

```
form_id += _record.class.name.underscore
form_id += _record.persisted? ? "_#{_record.id}" : ""
#se viene impostata l'opzione autosave e ci troviamo in modifica
if opt[:autosave] && @guiRenderEdit
  #se il parametro form.autosave.timeout non viene definito
  #viene assegnato il valore 120000 equivalente a 2 minuti
  timeout = ApplicationParameter
   .find_by_name("form.autosave.timeout").try(:parameter_value)
  timeout ||= 120000
 #viene impostato l'intervallo indicato al termine del quale viene
 #eseguita la funzione javascript san.submitFormByID che esegue
 #il submit del form
  out += "san.setCustomInterval(san.submitFormByID,
␣␣␣␣#{timeout},␣true,␣'#{form_id}');"
end
out += "</script>"
...
```

Tutte le informazioni compilate in questo plugin vengono unite in due documenti distinti che vengono consegnati al paziente: la valutazione di ingresso e la valutazione alla dimissione. In quest'ultima, grazie al meccanismo degli *hook*, vengono inserite anche le scale di valutazione compilate.

### 6.2.1 Anamnesi patologica prossima

L'anamnesi patologica prossima riguarda il motivo per cui il paziente è stato ricoverato, il medico interroga il paziente.

Questo specifico record, pensato per le case di cura che si occupano di riabilitazione, è composto dai seguenti campi:

- *data evento acuto*: è la data in cui il motivo del ricovero si è verificato;
- *data intervento*: è la data in cui è stato eseguito l'eventuale intervento per la risoluzione dell'evento acuto;
- *lateralità*: serve per indicare il lato del corpo del paziente colpito dall'evento acuto, è un campo di scelta tra sinistra, destra o entrambi;
- *motivo del ricovero*: breve descrizione dell'evento che ha condotto il paziente al ricovero;
- *provenienza*: da dove arriva il paziente, caricato in maniera automatica grazie alle API di creazione della cartella se è indicato in accettazione;
- *anamnesi*: è un campo descrittivo aperto in cui il medico appunta tutto quello che il paziente racconta sull'evento.

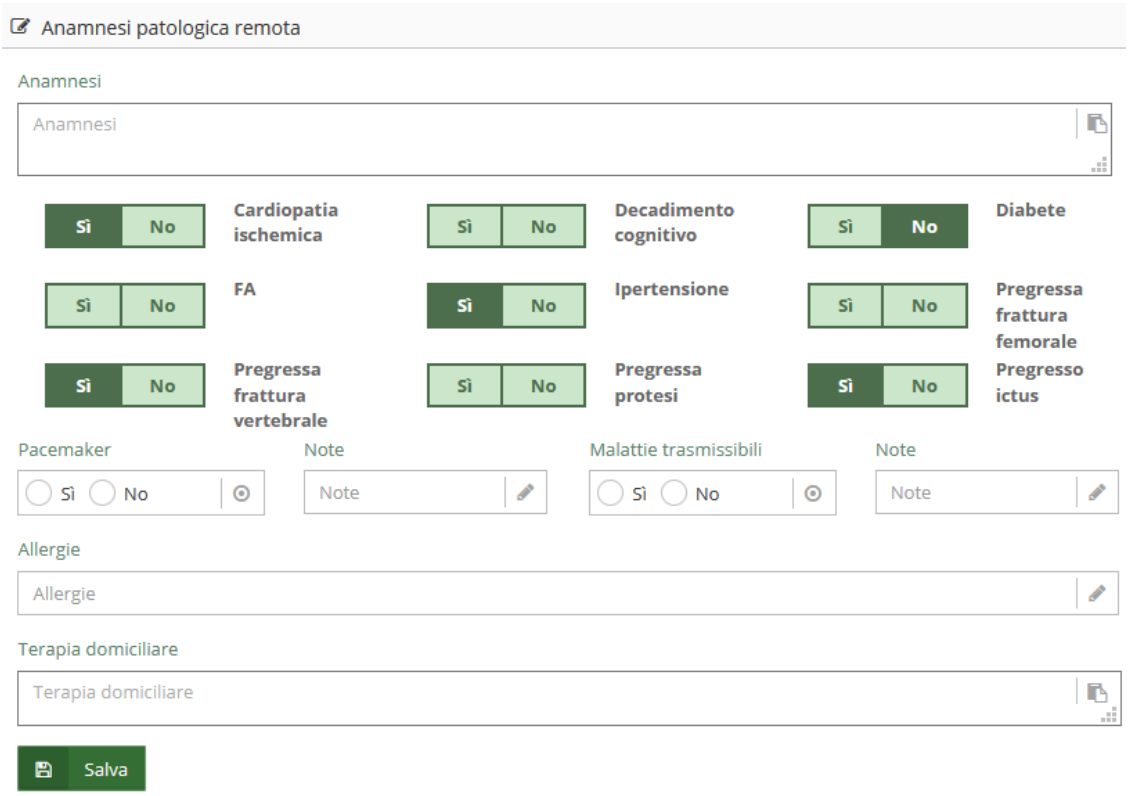


Figura 6.4. Form per anamnesi patologica remota

### 6.2.2 Anamnesi patologica remota

Consiste nell'indagine cronologica e ordinata sulle malattie, traumi ed interventi sofferti dal paziente nel passato.

Come si può vedere in figura 6.4, sono presenti i seguenti campi:

- *anamnesi*: è un campo aperto in cui il medico scrive tutto ciò che reputa opportuno in base all'intervista fatta al paziente in merito a eventi sofferti in passato;
- *pacemaker* e *note*: campo di scelta con possibilità di aggiungere delle note;
- *malattie trasmissibili* e *note*: campo di scelta con possibilità di aggiungere delle note;
- *allergie*: campo molto importante in quanto vengono riportate qui tutte le possibili allergie che presenta il paziente, la compilazione di questo campo genera una visualizzazione del suo contenuto in primo piano all'apertura della cartella;
- *terapia domiciliare*: campo aperto in cui il medico inserisce le informazioni relative a eventuali terapie seguite dal paziente non collegate all'evento che lo ha condotto in struttura.

Oltre alla compilazione di questi campi, il medico ha a disposizione una serie di patologie sulle quali deve indicare se il paziente ne ha sofferto o meno. Queste sono state inserite per poter eseguire ricerche ed esportazioni del dato puntuali. Tali patologie sono quelle che più di frequente si trovano nei pazienti che vengono ricoverati nelle strutture prese in esame e sono gestite dal model *Pathology*.

### 6.2.3 Valutazione fisiologica

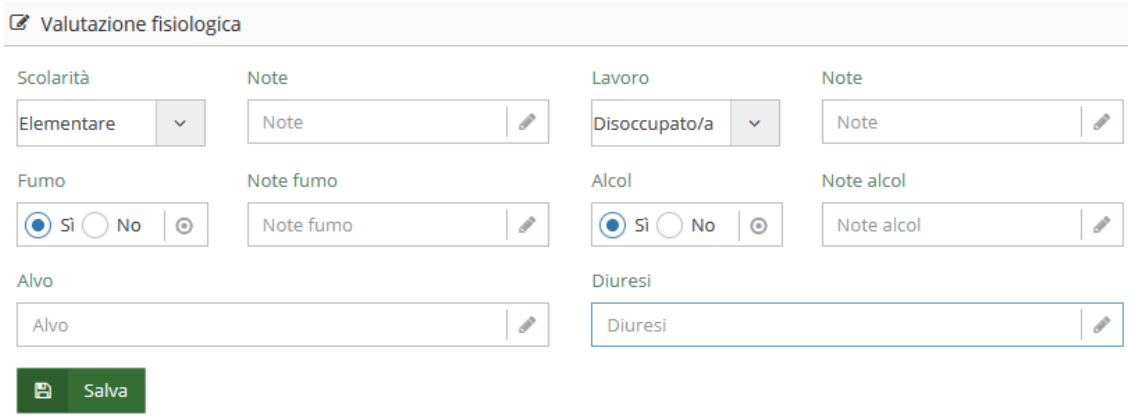


Figura 6.5. Form per valutazione fisiologica

È costituita da tutte quelle informazioni sulle funzioni fisiologiche del paziente.
Il medico raccoglie informazioni in merito a:

- *scolarità*: campo che indica il livello di istruzione del paziente, le scelte sono gestite dal model *PhysiologicalSchool* (elementi attuali sono Elemantare, Superiore di primo livello, Università triennale e Università magistrale), è possibile aggiungere delle note per approfondire la scelta;

- *lavoro*: utilzzato per indicare il tipo di impiego del paziente, le scelte sono disponibili grazie al model *PhysiologicalWork* (record presenti sono Impiegato/a, Disoccupato/a, In pensione, Operaio/a), è possibile aggiungere delle note per argomentare la scelta;

- *fumo*: campo di scelta (Sì/No) accompagnato da un campo per aggiungere note;

- *alcol*: campo di scelta (Sì/No) accompagnato da un campo per aggiungere note;

- *alvo*: indica il canale intestinale, in questo campo viene indicata la regolarità del funzionamento con la possibilità di descrivere eventuali problematiche;

- *diuresi*: descrive il volume di urine secreto dai reni in un dato periodo di tempo.

### 6.2.4 Esame obiettivo

L'esame obiettivo consente al medico di effettuare una valutazione obiettiva delle condizioni del paziente, basata sui segni oggettivi e dunque riscontrabili, al di là di ciò che il paziente riferisce in maniera soggettiva.

Grazie a questo form il medico può registrare la sua valutazione su polmoni, addome, apparato cardiologico, cute e stato di coscienza. Per tutte queste valutazioni ha a disposizione la scelta sullo stato, *Normale* oppure *Altro*, accompagnandolo con delle note.

Nel caso in cui la scelta ricada sullo stato *Normale* allora per ogni apparato è definita una formula descrittiva specifica: ad esempio per l'esame obiettivo polmonare in stato normale verrà indicato "*murmure vescicolare ben trasmesso su tutti campi polmonari. Non stasi alveolare. Non rumori aggiunti. Basipolmonari mobili*".

Oltre ai campi sopra citati, si mette a disposizione un ulteriore campo descrittivo se dovesse essere necessario aggiungere informazioni non previste.

### 6.2.5 Diagnosi in ingresso

In base alle valutazioni fatte da parte del medico, è possibile assegnare al paziente una serie di diagnosi.

Questo form permette di selezionare la diagnosi a partire dall'elenco fornito dal manuale della classificazione ICD9-CM: è lo strumento che riporta in modo sistematico e secondo precise regole d'uso la nomenclatura delle diagnosi, dei traumatismi, degli interventi chirurgici e delle procedure diagnostiche e terapeutiche [16].

Le diagnosi disponibili sono caricate tramite il model *Diagnosis* verso il quale il model *FolderDiagnosis* ha sei chiavi esterne (relazione *belongs_to*), una per la diagnosi principale e le altre cinque come diagnosi secondarie. I campi utilizzati per la selezione delle diagnosi sono di tipo **autocomplete** e permettono di ricercare sia per codice sia per descrizione. Questi campi sono accompagnati da ulteriori campi *note* per annotare informazioni utili.

### 6.2.6 Inquadramento

Questa tipologia di record è molto importante perché permette di impostare una serie di informazioni, valutazioni, scale e figure professionali di default in base a quanto definito dal gestore o dal referente della struttura.

La scelta dell'inquadramento definisce quali sono le valutazioni funzionali e le scale di valutazione da compilare.

In quasi tutti gli inquadramenti definiti in questo momento vengono impostate tutte le figure professionali disponibili (fisioterapista, infermiere, logopedista, medico, neuropsicologo, terapista occupazionale) e vengono generate in automatico due scale di valutazione (Motricity Index e Barthel index). Questi sono gli inquadramenti presenti in questo momento:

- **Neurologico**: genera una valutazione funzionale di tipo neurologico con la possibilità di scegliere se di tipo *emiplegico* (molto più completa e strutturata) o di tipo *generico*;
- **Ortopedico**: genera una valutazione funzionale che si distingue tra PTA (Protesi d'anca) e PTG (Protesi del ginocchio);

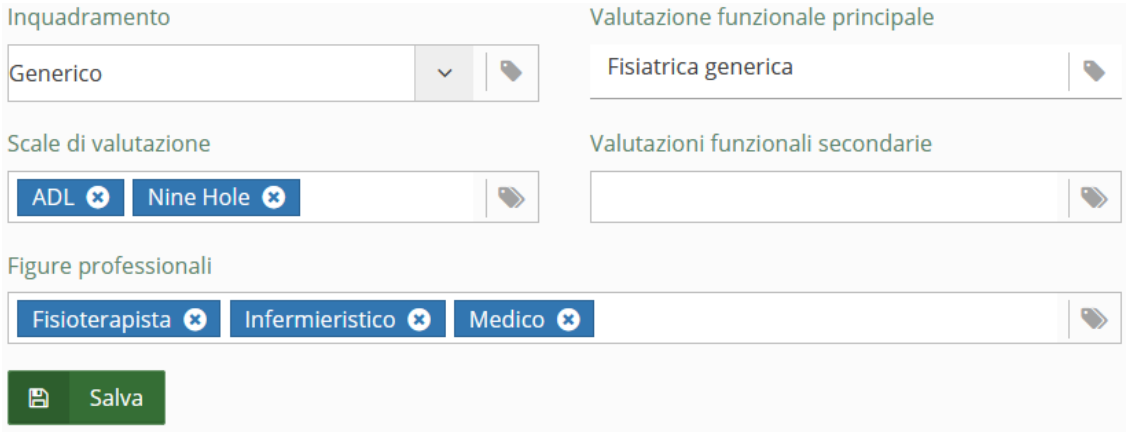


Figura 6.6. Form per la selezione dell'inquadramento

- **Generico**: imposta una valuazione funzionale di tipo fisiatrico generica, in questo caso non vengono generate ulteriori scale di valutazione e si lascia piena libertà ai medici e alle altre figure professionali.

Dopo aver scelto l'inquadramento è comunque possibile aggiungere delle ulteriori valutazioni o scale di valutazione, a discrezione del medico e specifiche per quel paziente, che dovranno essere compilate durante la fase di ricovero.

### 6.2.7 Valutazioni funzionali

La valutazione funzionale è necessaria per approfondire le disabilità del paziente e per decidere il percorso riabilitativo più adatto ad esso.

Nella figura 6.7 è presentato l'esempio della valutazione funzionale fisiatrica che si compone di un solo campo descrittivo. Le tipologie sviluppate sono:

- *Neurologica*: a sua volta si distingue in *generica* e *per emiplegici*. Tramite questo record vengono presi in considerazione aspetti generici, come la dominanza manuale, il deficit campimetrico e l'orientamento, oltre ad aspetti più specifici come la valutazione dei nervi cranici, delle alterazioni del movimento di tutti gli arti e del tronco. Si compone di molti campi a scelta multipla utilizzati per poter essere più schematici nella compilazione e per poter eseguire estrazioni dati più puntuali.

- *PTA* e *PTG*: riguardano entrambe la branca ortopedica e sono acronimi di *Protesi Totale Anca e Ginocchio*. Anche queste due tipologie si compongono di diversi campi di scelta multipla come l'atteggiamento dell'arto inferiore, il termotatto, l'eterometria, il R.O.M. (Range Of Motion - Recupero motorio), lo stato della ferita chirurgica dopo l'intervento, la scala del dolore NRS, il bilancio muscolare.

- *Fisiatrica*: questa tipologia è stata inserita per avere a disposizione una sorta di taccuino, infatti si compone di una sola textarea dove il medico può scrivere tutto in maniera discorsiva.

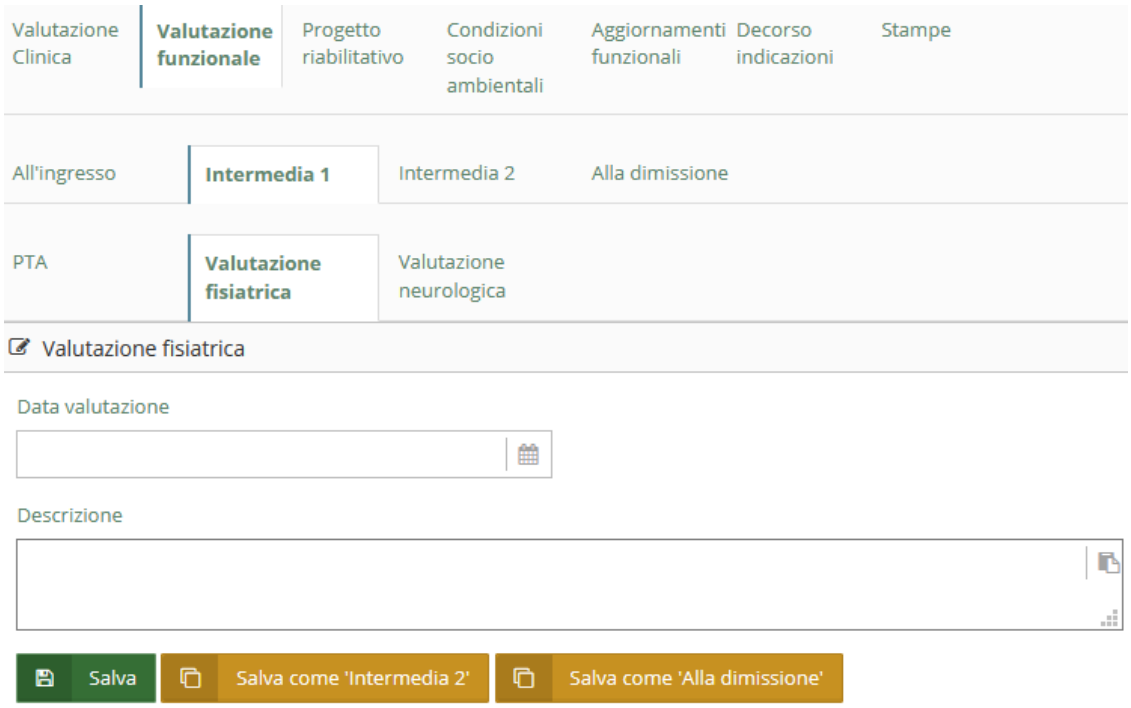


Figura 6.7. Valutazioni funzionali

Tutte le valutazioni funzionali inserite in fase di inquadramento devono essere compilate in fase di ingresso del paziente. È stato inserito un meccanismo di gestione dei periodi del soggiorno del paziente ed è la struttura ospedaliera a decidere quanti periodi rendere disponibili: ingresso, intermedio 1, intermedio n, dimissione.

Le valutazioni, all'atto della creazione, nascono nello stato *bozza*. I pulsanti di salvataggio a disposizione del medico sono di tre tipi: *Salva*, *Salva come 'periodo'* e *Consolida*. Il primo tipo continua a salvare la valutazione senza cambiargli lo stato (rimane in stato *bozza*). La tipologia *Salva come 'periodo'* serve per impostare lo stato della valutazione attuale come *consolidato* e creare una nuova valutazione in base al periodo scelto con i campi precompilati nella valutazione precedente. L'ultima tipologia permette di salvare la valutazione ed impostare lo stato *consolidato*.

Lo stato *consolidato* rende la valutazione non più modificabile ed è uno stato impostato automaticamente dal giorno dopo la data dimissione per non permettere nessuna modifica da quando il paziente non è più in carico alla struttura.

### 6.2.8 Progetto riabilitativo

Nel progetto riabilitativo si definiscono le aree di intervento specifico, gli obiettivi, i professionisti coinvolti, i setting, le metodologie e le metodiche riabilitative, i tempi di realizzazione e la verifica degli interventi che costituiscono i programmi riabilitativi [17].

Nella prima versione del progetto riabilitativo incluso in questa cartella informatizzata sono presenti solo alcune delle informazioni richieste dalle linee guida: vengono indicati

i tempi previsti di realizzazione del progetto in base alla diagnosi, i tempi effettivi che il medico prevede siano necessari per il paziente specifico, le figure professionali coinvolte (già scelte grazie alla selezione dell'inquadramento del paziente).

La descrizione del progetto riabilitativo viene inclusa in un campo di testo libero.

È già in progetto la possibilità di creare più progetti riabilitativi per lo stesso paziente in modo da poter verificare l'andamento dello stesso durante tutto il suo soggiorno in struttura. All'interno di questa nuova versione saranno indicati gli obiettivi specifici per ogni singola voce del progetto: a breve termine, a medio e a lungo termine.

### 6.2.9 Valutazione delle condizioni socio-ambientali

Questo tipo di valutazione (figura 6.8) è necessaria per inquadrare il paziente dal punto di vista sociale ed ambientale.

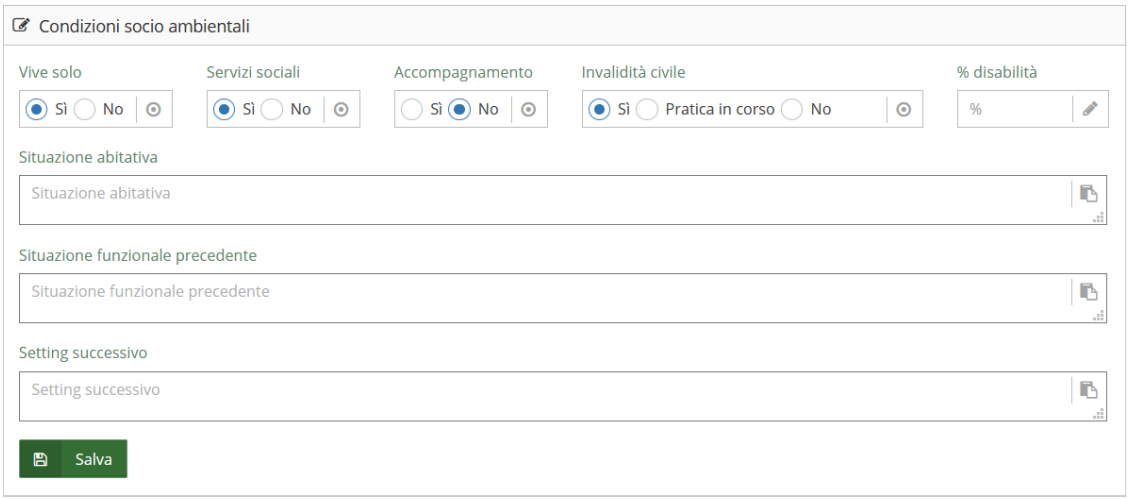


Figura 6.8. Valutazione delle condizioni socio-ambientali

Si compone di una serie di campi a risposta chiusa che indicano se il paziente vive da solo, se è seguito dai servizi sociali, se percepisce un'indennità di accompagnamento e se ha un'invalidità civile (in questo caso viene indicata anche la percentuale di tale invalidità).

Inoltre sono disponibili dei campi di testo liberi dove vengono descritti: la situazione abitativa, la situazione funzionale precedente e il setting successivo.

### 6.2.10 Aggiornamenti funzionali

Gli aggiornamenti funzionali, in aggiunta al progetto riabilitativo, servono a tenere traccia dell'andamento del soggiorno del paziente all'interno della struttura ospedaliera. Vengono utilizzati solitamente durante le riunioni di equipe e il medico appunta tutte le informazioni che reputa utili.

Come vediamo nel form in figura 6.9, sono presenti due campi di testo liberi: il primo serve per argomentare l'aggiornamento, il secondo serve per definire gli obiettivi che verranno valutati durante le varie sedute di terapia. Ulteriori campi riguardano la data di

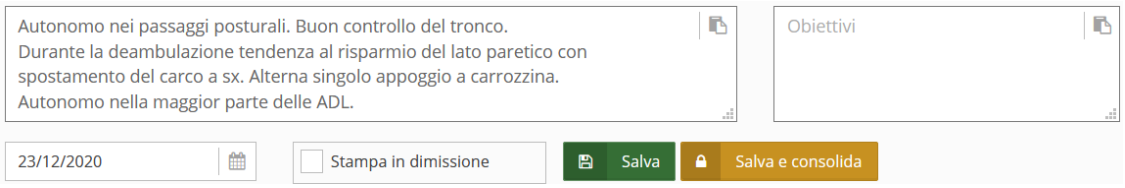


Figura 6.9. Form inserimento aggiornamento funzionale

redazione dell'aggiornamento e un flag per indicare se questo aggiornamento deve essere indicato anche nel documento di dimissione.

Questa tipologia di record può avere due stati diversi: *bozza*, il medico può scrivere tutte le sue considerazioni anche in momenti diversi, *consolidato*, il medico decide che non è più necessario apportare modifiche. Finché esiste un record nello stato *bozza* non potrà essere creato un nuovo record.

L'elenco degli aggiornamenti funzionali (figura 6.10) è presentato in ordine cronologico inverso (dalla data più vicina a quella più lontana). Quando un aggiornamento funzionale si trova nello stato *consolidato*, il medico può ancora decidere se includerlo o escluderlo dal documento di dimissione tramite i pulsanti mutualmente esclusivi **Stampa in dimissione** ed **Escludi dalla stampa**.

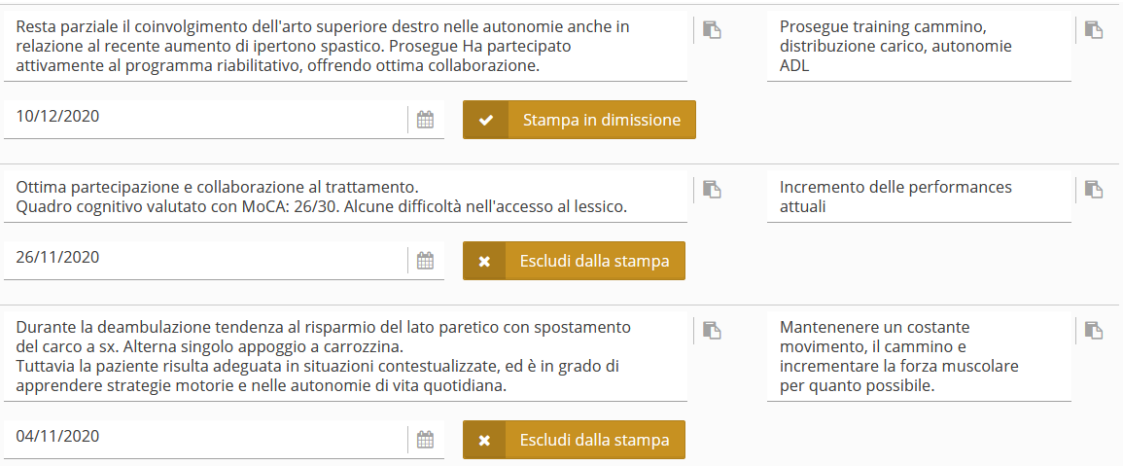


Figura 6.10. Aggiornamenti funzionali

### 6.2.11 Decorso/indicazioni

Questo è l'ultimo form che compila il medico prima che il paziente venga dimesso. Ha a disposizione due campi di testo liberi: *decorso* e *indicazioni*.

Nel primo campo viene riassunto l'andamento del ricovero dando risalto agli eventi più importanti mentre nel secondo campo vengono date delle indicazioni utili al paziente per proseguire nel miglior modo possibile con la vita di tutti i giorni.

Per la compilazione del campo *indicazioni* è stata inserita la possibilità di aggiungere frasi precompilate. In questo modo il medico ha una sorta di traccia da seguire.

## 6.3 Gli altri plugin

In questo paragrafo verranno presentati in maniera riassuntiva tutti gli altri plugin che mi hanno visto coinvolto nello sviluppo.

### 6.3.1 FUT: Foglio Unico di Terapia

Il Foglio Unico di Terapia è l'elemento più importante all'interno di una cartella clinica informatizzata. Questo plugin serve per registrare la terapia farmacologica del paziente, dalla prescrizione da parte del medico fino alla somministrazione dell'infermiere.

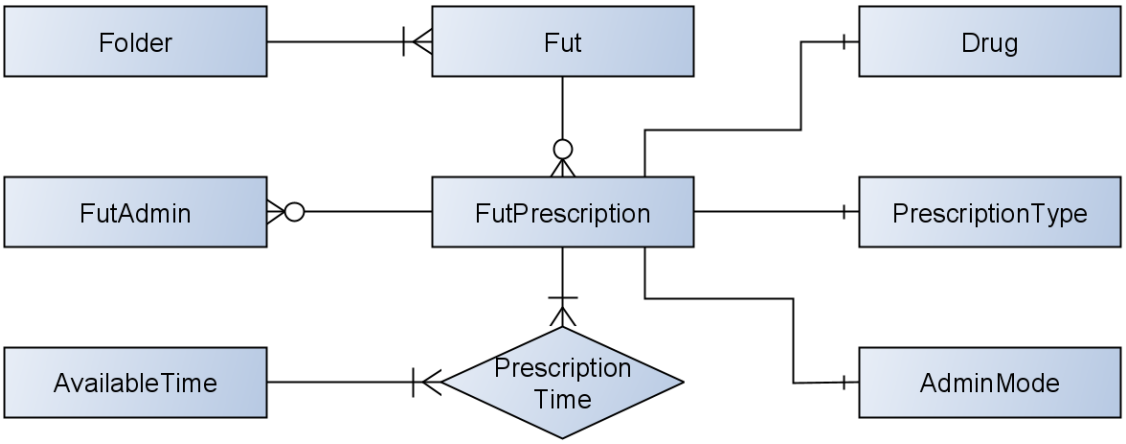


Figura 6.11. Modello ER Foglio Unico di Terapia

Ogni oggetto del modello ER in figura 6.11 assume il seguente significato:

- *Folder*: è la cartella vera e propria, all'atto della creazione viene creato anche un FUT nello stato *bozza*;
- *Fut*: può essere visto come raccoglitore di tutte le prescrizioni; questo tipo di record può essere nello stato *bozza*, *in corso*, *archiviato*; ha validità dal timestamp in cui passa dallo stato *bozza* a quello *in corso* (che avviene non appena il medico lo valida) fino alla sua archiviazione (che avviene alla creazione di un nuovo FUT oppure alla dimissione del paziente);
- *FutPrescription*: è la prescrizione vera e propria; il medico indica la data di inizio e, se lo ritiene opportuno, la data di fine; la scelta del farmaco (entità *Drug*) può avvenire per nome commerciale o per principio attivo (è a disposizione del medico il prontuario farmaceutico messo a disposizione dall'Agenzia Italiana del Farmaco - AIFA); viene indicata la tipologia della prescrizione, gestita dal model *PrescriptionType* (Ordinaria o Condizionata, nel secondo caso sarà necessario compilare il campo *condizione*); si

indica la modalità di somministrazione tramite l'entità *AdminMode* (se la scelta ricade in modalità che richiedono la compilazione della velocità di somministrazione, allora questo campo dovrà essere compilato); infine vengono definite le posologie del farmaco prescritto indicando la quantità e l'ora esatta tramite l'entità *PrescriptionTime*;

- *FutAdmin*: rappresenta il record di somministrazione collegato direttamente alla prescrizione; verranno create tante somministrazioni quante posologie inserite nella prescrizione per ogni giorno di validità del FUT; la creazione delle somministrazioni avviene alle ore 00:01 di ogni giorno oppure alla validazione di un nuovo FUT; l'infermiere avrà a disposizione l'elenco delle somministrazioni (figura 6.12) del paziente diviso per fascia oraria (il referente della struttura ospedaliera definisce le fasce orarie, solitamente i turni degli infermieri, indicando anche un colore di sfondo per ognuna di esse), su queste somministrazioni ha a disposizione il pulsante di scelta **Fatto | Da fare | Non fatto** e nel caso dovesse aggiungere delle note può compilare l'apposito campo (da compilare obbligatoriamente se si clicca su **Non fatto**).

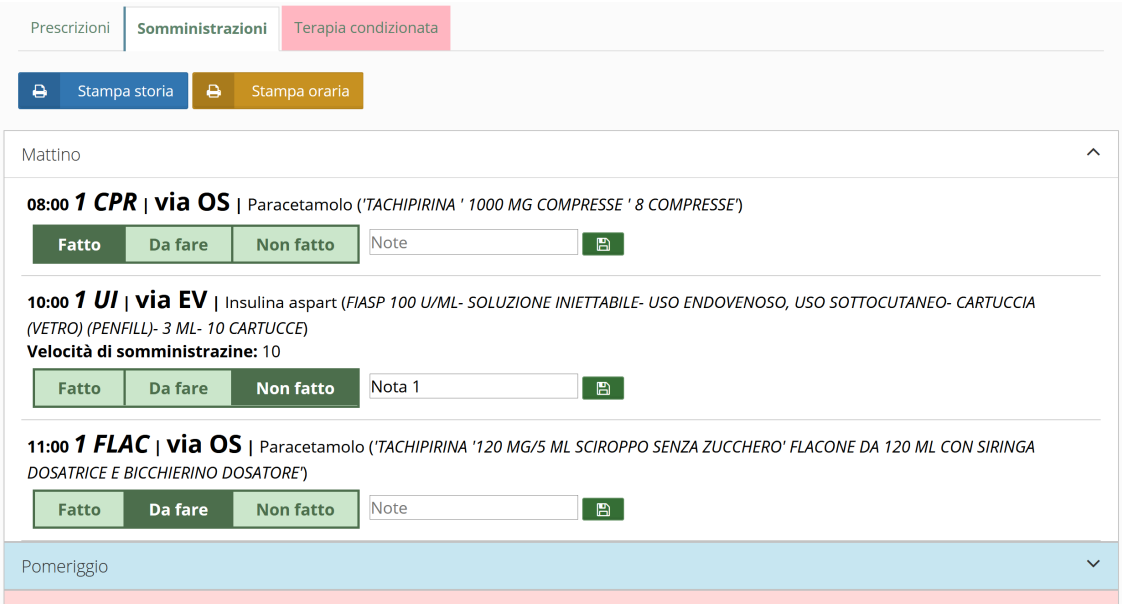


Figura 6.12. Somministrazioni viste dal ruolo infermieristico

### 6.3.2 Scale di valutazione

Le scale di valutazione redatte durante il ricovero del paziente hanno un significato importante in quanto misurano diverse tipologie di parametri seguendo regole prestabilite in base alla scala scelta.

Come si può vedere nella figura 6.13, le scale di valutazione vengono raggruppate in base a quanto scelto dai referenti della struttura ospedaliera: ci possono essere raggruppamenti in base alla branca medica (Neurologia, Ortopedia, ecc...) oppure in base a specifiche

malattie (Parkinson, sclerosi multipla, ecc...). Ogni scala di valutazione può essere inserita in più raggruppamenti.

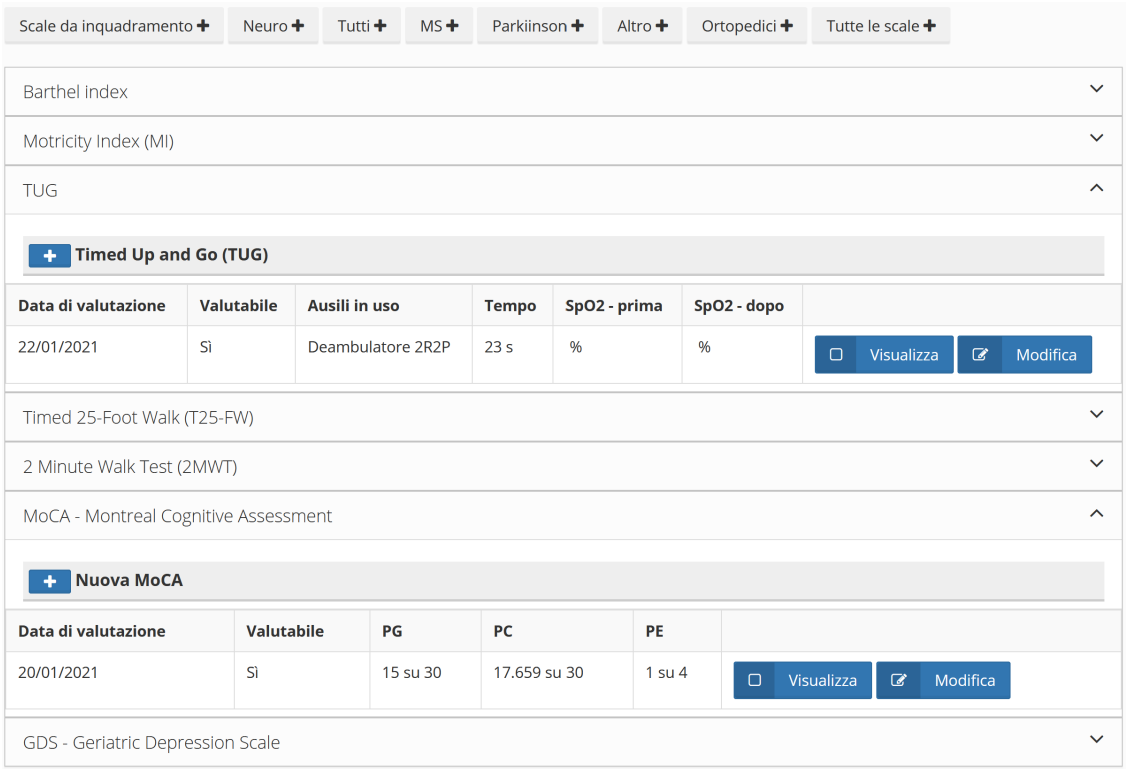


Figura 6.13. Elenco scale di valutazione compilate

Tra le scale di valutazione le più rilevanti sono:

- *Barthel index*: è stata codificata alla fine degli anni cinquanta ed ha lo scopo di stabilire il grado di indipendenza del paziente. Si compone di 10 item che prevedono le comuni attività quotidiane, con valori che variano da 0 a 15, per arrivare ad un punteggio massimo pari a 100. La somma di tutti i punteggi ottenuti sugli item determina il grado di autonomia del paziente [18].

- *FIM - Functional Independence Measure*: è uno standard internazionale di misura della disabilità, questa scala si presenta come un questionario che censisce 18 attività della vita quotidiane (13 motorio-sfinteriche - come nutrirsi, lavarsi, igiene personale, alvo, cammino - e 5 cognitive - come comprensione, rapporto con gli altri, memoria). Ogni attività può ricevere un punteggio tra 1 (completa dipendenza dagli altri) e 7 (completa autosufficienza) per un totale che può variare tra 18 e 126 [19].

- *9HPT - Nine Hole Peg Test*: è una misura quantitativa della funzionalità degli arti superiori ed è previsto dal protocollo di Minima Ictus Cerebrale. Il test consiste nell'inserire 9 piolini uno alla volta nei buchi, il più veloce possibile e in qualsiasi ordine,

usando una sola mano, fino a quando non sono stati riempiti tutti i buchi. Successivamente, senza fare pause, rimuovere tutti i 9 piolini, uno alla volta, e riportarli nel loro contenitore, usando una sola mano e nel minor tempo possibile. Il form sviluppato permette al terapista di registrare le tempistiche ed eventuali note salienti.

- *TUG - Timed Up and Go*: misura il livello di mobilità di una persona registrando il tempo che si impiega per alzarsi da una sedia, camminare per tre metri, girarsi, tornare alla sedia e sedersi di nuovo [20]. Oltre al tempo impiegato vengono registrati gli ausili utilizzati, il livello di ossigeno nel sangue (SpO2) prima e dopo il test ed eventuali note.

Tutte le scale hanno in comune due campi obbligatori: la **data di valutazione**, necessaria per congelare nel tempo la scala stessa, il **flag di valutabilità**, importante per definire se il paziente può essere valutato o meno con la scala di valutazione in esame.

### 6.3.3 Valutazioni dei servizi

Le valutazioni dei servizi seguono lo stesso concetto delle valutazioni funzionali, cioè ne possono essere redatte tante quanti i periodi definiti dai referenti di struttura.

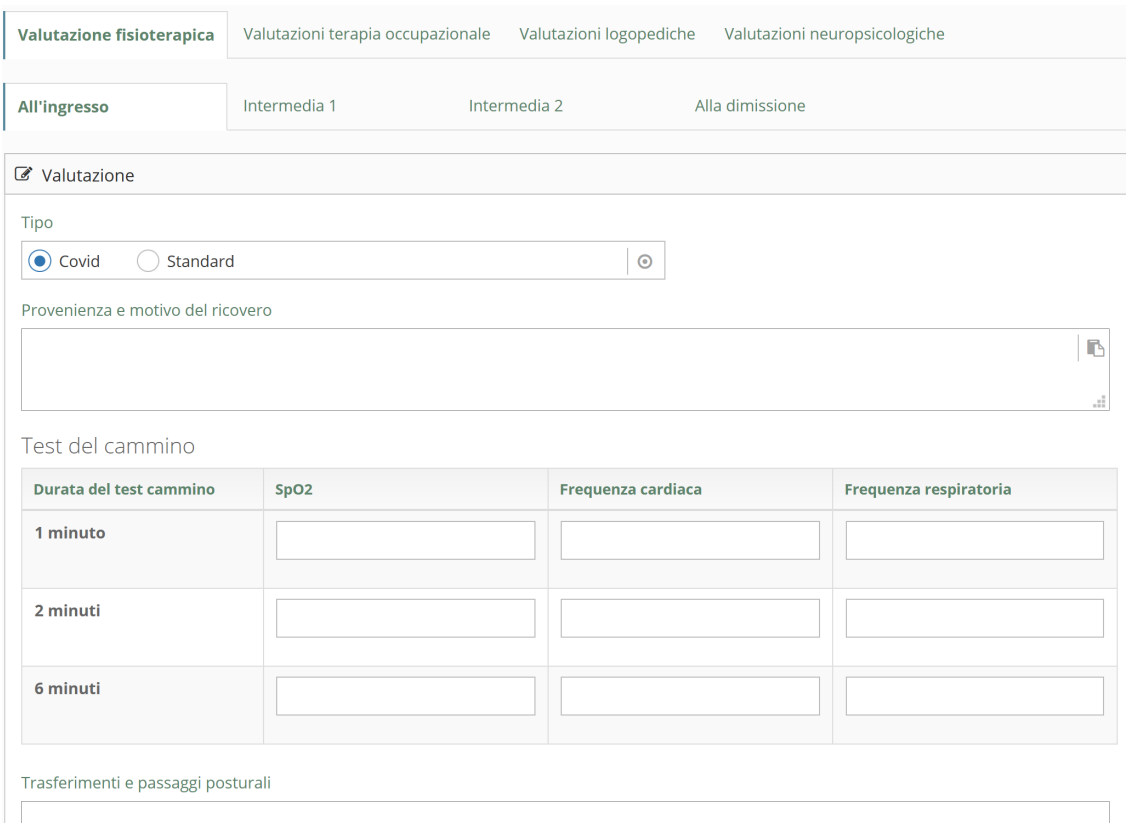


Figura 6.14. Esempio valutazione fisioterapisti del plugin TherapistEvaluation

Le figure professionali per le quali è stata definita una valutazione sono: fisioterapista, logopedista, neuropsicologa, terapista occupazionale. Queste tipologie di valutazioni hanno a disposizione un campo di testo libero all'interno del quale le varie figure professionali descrivono le proprie relazioni.

In aggiunta a quanto detto prima, per la valutazione dei *fisioterapisti* possono essere indicate ulteriori informazioni come la tipologia (aggiunta di recente per far fronte all'emergenza Covid), la provenienza e il motivo del ricovero, i trasferimenti e passaggi posturali, la deambulazione e il controllo del tronco.

Inoltre la figura professionale *logopedista* ha a disposizione la compilazione di una scheda alimentare che viene messa a disposizione degli altri operatori sanitari che si occupano dei pasti dei pazienti.

### 6.3.4 Consegne

Il passaggio di consegne, e quindi di informazioni tra figure professionali, è una delle operazioni più critiche per garantire la continuità assistenziale al paziente in ricovero. Un buon passaggio di consegne deve essere un processo strutturato che comprende non solo uno scambio di informazioni, ma anche suggerimenti per eventuali altri passaggi.

Durante lo sviluppo di questo plugin si è scelto di poter avere delle consegne tra figure professionali oltre a consegne di interesse generale e trasversale.

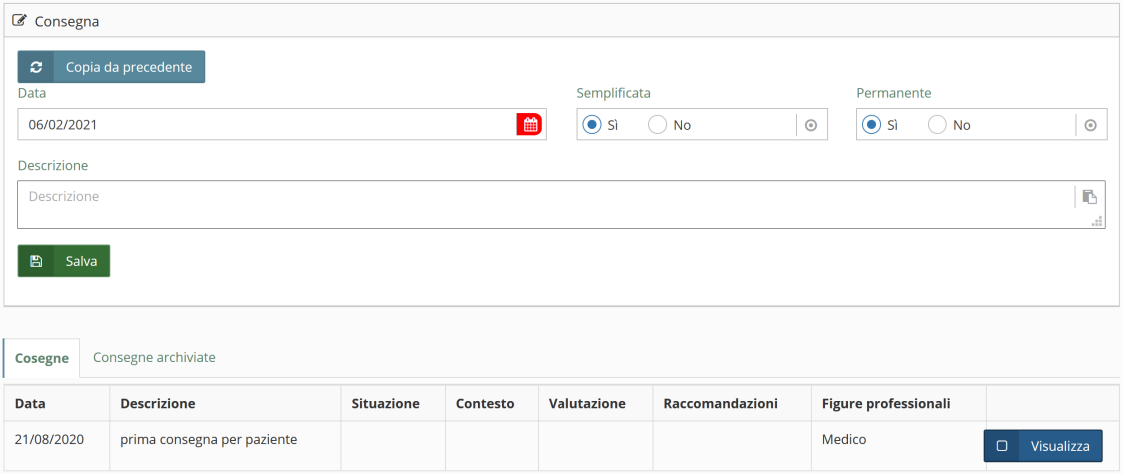


Figura 6.15. Consegne

L'operatore sanitario ha la possibilità di scegliere la modalità di inserimento della nuova consegna: semplificata oppure seguendo il metodo SBAR. La prima modalità di inserimento prevede un solo campo di testo libero. L'utilzzo del metodo SBAR prevede invece la compilazione di quattro campi ben distinti:

- **S**ituation: la situazione clinica del paziente che inquadra lo stato attuale dello stesso;
- **B**ackground: la storia clinica del paziente comprendente anche eventuali allergie;

- **A**ssessment: cosa è stato fatto fino ad ora, compresi alcuni parametri vitali e indagini diagnostiche in atto;
- **R**ecommendation: rappresenta quello che deve essere fatto e come nel breve termine.

Nel caso in cui ci fossero delle consegne che non cambiano nel tempo è stato inserito un flag *permanente* che permette di mettere quella consegna sempre in primo piano.

Per agevolare la compilazione di nuove consegne è stata creata la funzionalità *Copia da precedente* che precompila tutti i campi in base all'ultima consegna inserita aggiornando la data a quella odierna.

### 6.3.5 Export dei dati

La possibilità di esportazione dati è diventata molto importante per poter effettuare ricerche scientifiche sulla tipologia di pazienti passati dalla struttura ospedaliera.

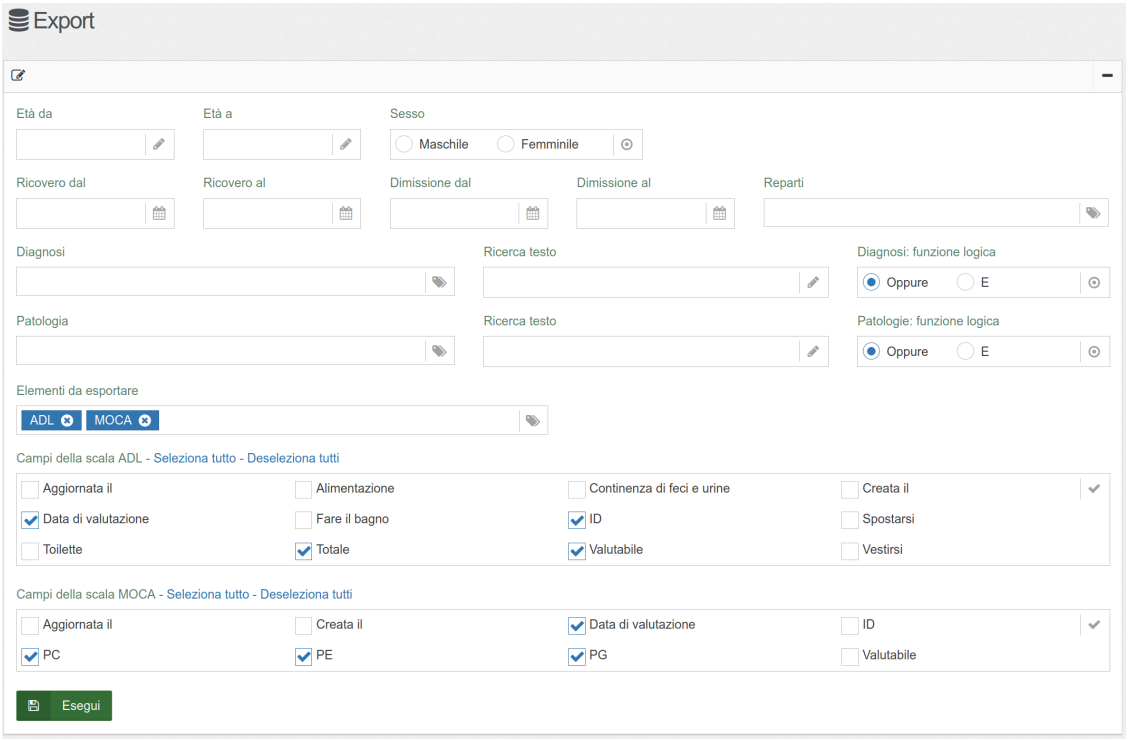


Figura 6.16. Form di esportazione dati

Questo plugin mette a disposizione un form (figura 6.16), da compilare preventivamente, che ha la funzionalità di filtrare opportunamente i dati. I filtri applicabili sono:

- *età*: composto da due campi che impostano l'età minima e l'età massima che si vuole includere nell'esportazione;
- *sesso*: maschile, femminile o altro;

- *date del ricovero*: utilizzate se si desidera esportare campioni di dati in base alla data di ricovero e/o di dimissione;
- *reparto*: per poter restringere il campione a uno o più reparti della struttura;
- *diagnosi*: si possono impostare filtri sulle diagnosi ricercate per codice/descrizione, oppure su quanto scritto all'interno dell'anamnesi patologica prossima da parte del medico; in questo caso si può scegliere la funzione logica che andrà a processare questi filtri (**AND** oppure **OR**);
- *patologia*: imposta i filtri sulle patologie e/o su quanto scritto dal medico nell'anamnesi patologica remota; anche in questo caso si può scegliere la funzione logica che andrà a processare i filtri (**AND** oppure **OR**).

Gli elementi da esportare sono tutti i record collegati alla cartella: dall'anamnesi patologica prossima all'esame obiettivo, dal fut alla somministrazione, dalla scala di Barthel alla scala di Ashworth, ecc... Si possono scegliere più tipologie di record da esportare per volta e per ogni tipologia si possono scegliere i campi desiderati.

Ogni riga del file di esportazione corrisponde ad una cartella, tutti i record collegati vanno a popolare le colonne: se alla cartella sono collegati più record dello stesso tipo saranno etichettati con T0 ... Tn.

Estratto dalla funzione che esegue il filtro dei dati

```
#accetto solo i parametri del form Export
v_p = params.require(:export).permit!.to_h
#Preparo la prima parte di query data dai filtri costanti
qry = []
unless v_p[:start_date_from].blank? #data di ricovero da
  qry << "start_date >= date('#{v_p[:start_date_from].to_date}')"
unless v_p[:start_date_to].blank? #data di ricovero a
  qry << "start_date <= date('#{v_p[:start_date_to].to_date}')"
unless v_p[:end_date_from].blank? #data di dimissione da
  qry << "end_date >= date('#{v_p[:end_date_from].to_date}')"
unless v_p[:end_date_to].blank? #data di dimissione a
  qry << "end_date <= date('#{v_p[:end_date_to].to_date}')"
unless v_p[:gender_id].blank? #sesso
  qry << "gender_id = '#{v_p[:gender_id]}'"
...
#i filtri vengono uniti in un'unica query in AND
folders = Folder.where("#{qry.join(' AND ')}")
...
#elimino i campi vuoti per le patologie
v_p[:pathology_ids].delete('')
unless v_p[:pathology_ids].blank? && v_p[:pathology_text].blank?
  if v_p[:pathology_logic] == 'true' #funzione logica OR
    unless v_p[:pathology_ids].blank?
       query = "remote_anamnesis_rows.pathology_id in (?) and
   remote_anamnesis_rows.present=true"
```

```
25      unless v_p[:pathology_ids].blank?
26        unless v_p[:pathology_text].blank?
27           query += " OR remote_anamneses.anamnesis ilike '%#{v_p[:
    pathology_text]}%'"
28        #eseguo left_join su remote_anamnesi e remote_anamnesis_row
29        folders = folders.left_joins([remote_anamnesis: [:
    remote_anamnesis_rows]]).where(query, v_p[:pathology_ids])
30      else
31        query = "remote_anamneses.anamnesis ilike '%#{v_p[:
    pathology_text]}%'"
32        #eseguo left_join solo su remote_anamnesis
33        folders = folders.left_joins(:remote_anamnesis).where(query)
34      end
35    else
36      #funzione logica AND
37      unless v_p[:pathology_ids].blank?
38        query = "remote_anamneses.id in (select remote_anamnesis_id
    from ("
39        query += "select remote_anamnesis_id, array_agg(pathology_id)
    as x from remote_anamnesis_rows where present=true group by
    remote_anamnesis_id) as t "
40        query += "where x @> ARRAY[#{v_p[:pathology_ids].join(",")}])
    "
41        unless v_p[:pathology_text].blank?
42          query += " AND remote_anamneses.anamnesis ilike '%#{v_p[:
    pathology_text]}%'"
43      else
44        query = "remote_anamneses.anamnesis ilike '%#{v_p[:
    pathology_text]}%'"
45      end
46      #eseguo left_join solo su remote_anamnesis
47      folders = folders.left_joins([:remote_anamnesis]).where(query)
48    end
49  end
50  #tutti gli id delle cartelle da esportare
51  folder_ids = folders.ids
52  #da qui in poi devo prendere tutti i record scelti che hanno un
    riferimento agli id filtrati delle cartelle
53  ...
```

# Capitolo 7

# Conclusioni

Il lavoro svolto fino a questo momento è stato lungo e importante e mi ha permesso di approfondire tutte le fasi che presenta un progetto di sviluppo: dalle interviste ai soggetti interessati all'analisi dei processi attivi, dalla progettazione di singoli form allo sviluppo delle logiche implementative fino alla presentazione dei risultati.

Gli approfondimenti sul framework Ruby on Rails e sui plugin ad esso associati mi hanno permesso di ampliare il bagaglio culturale in questo ambito, confermando che la strada giusta per crescere professionalmente e umanamente è quella di alimentare la curiosità tramite lo studio di nuove tecniche e di nuovi linguaggi.

La struttura modulare dell'applicativo oggetto di questa tesi presenta delle basi molto solide per poter procedere con ampliamenti sulla stessa, sui plugin già sviluppati e sull'integrazione di nuovi plugin.

A questo proposito, è facile pensare alla "Cartella clinica informatizzata" come progetto in continua evoluzione in quanto sono già previsti almeno due tematiche per gli sviluppi futuri: l'adesione allo standard **HL7-CDA** e la **firma elettronica qualificata**.

Lo standard **HL7-CDA** (**H**ealth **L**evel **7** - **C**linical **D**ocument **A**rchitecture) definisce un markup dei documenti che specifica la struttura semantica dei documenti clinici ai fini dello scambio di informazioni tra operatori sanitari e pazienti. Permette di definire un documento clinico avente le seguenti caratteristiche: persistenza, amministrazione, autenticazione, contesto, integrità, leggibilità umana [21].

Lo sviluppo di un interfaccia che aderisca allo standard permetterebbe a questo applicativo di potersi integrare in maniera agevole con gli altri gestionali presenti nelle strutture sanitarie. Questa interfaccia sarebbe molto utile, inoltre, per inviare e/o ricevere la documentazione verso/dal Fascicolo Sanitario Elettronico del paziente.

La **firma elettronica qualificata** - o **digitale** - è basata su un certificato qualificato e su un sistema di chiavi crittografiche, una pubblica e una privata, correlate tra loro, che consente al titolare tramite la chiave privata e al destinatario tramite la chiave pubblica, rispettivamente, di rendere manifesta e di verificare la provenienza e l'integrità di un documento informatico o di un insieme di documenti informatici ; tale procedura garantisce l'autenticità, l'integrità e il non ripudio dei documenti informatici [22].

La possibilità di apporre la firma digitale sui documenti clinici è un ulteriore sviluppo che permetterebbe la completa dematerializzazione di tutto il processo. Allo stato attuale

l'inserimento e la consultazione del dato avviene completamente in maniera digitale. Continua a sussistere l'obbligatorietà di stampa e firma autografa sui documenti clinici risultanti per poterli archiviare legalmente, con tutte le difficoltà del caso.

In questo momento tutte le operazioni eseguite sui record (creazione, modifica, eliminazione) sono univocamente attribuibili ad uno e uno solo utente. Questo meccanismo, ottenuto grazie all'autenticazione con *username* e *password*, non è sufficiente al fine di archiviare in maniera legale i record salvati all'interno del database.

È in fase di studio una modalità che permetta di firmare elettronicamente una serie di documenti prodotti dalla cartella informatizzata per adempiere a quanto richiesto dalla legge. Oltre alla firma di questi documenti sarà necessario attivare anche l'archiviazione sostitutiva senza la quale il documento comunque non avrebbe validità. Un ulteriore passo potrebbe essere rappresentato dall'apposizione della firma elettronica sui log generati dall'applicativo.

# Bibliografia


[1] Elena Sini. *Sistemi informativi per il governo delle organizzazioni sanitarie. Processi core dei sistemi sanitari*. 2020.

[2] Giuseppe Goglio. *A che punto è la Cartella Clinica Digitale*. 2019. URL: https://www.01health.it/tecnologie/a-che-punto-cartella-clinica-digitale/.

[3] HIMSS Electronic Health Record Committee. *HIMSS Electronic Health Record Definitional Model Version 1.0*. 2003. URL: http://forensik.rendygunardi.net/s1/Medical%20Report/EHRAttributes.pdf.

[4] Quest Diagnostics, Care360. *Traditional paper records VS Electronic Health Records*. 2020. URL: http://quanuminsights.questdiagnostics.com/infographic-ehr-vs-traditional-paper-records/.

[5] CRRF Mons. Luigi Novarese. *Carta dei servizi*. 2012. URL: http://www.trompone.it/wp-content/uploads/2013/11/carta_dei_servizi_crrf_mons_luigi_novarese.pdf.

[6] Presidio Sanitario San Camillo. *Carta dei servizi*. 2019.

[7] Redhat. *Cos'è la metodologia agile?* 2019. URL: https://www.redhat.com/it/devops/what-is-agile-methodology.

[8] C. Barberis, G. Malnati. *Applicazioni web - Introduzione*. 2011 - 2014.

[9] Comunità Ruby. *Documentazione Ruby*. 2020. URL: https://www.ruby-lang.org/en/.

[10] Vinoth. *A Quick Study of the Rails Directory Structure*. 2016. URL: https://www.sitepoint.com/a-quick-study-of-the-rails-directory-structure/.

[11] David Heinemeier Hansson. *Ruby guides*. 2020. URL: https://guides.rubyonrails.org/.

[12] PostgreSQL global development gruop. *About PostgreSQL*. 1996 - 2021. URL: https://www.postgresql.org/about/.

[13] Rails documentation contributors. *The basic of create Rails plugins*. 2021. URL: https://guides.rubyonrails.org/plugins.html.

[14] Jean-Philippe Lang. *Redmine project*. 2006 - 2014. URL: https://www.redmine.org.

[15] R. França, L. Tegn, C. da Silva. *RubyDoc devise gem*. 2020. URL: https://rubydoc.info/github/heartcombo/devise/master/frames.

[16] Ministero della salute. *Il manuale ICD9CM*. 2007. URL: http://www.salute.gov.it/portale/temi/p2_5.jsp?lingua=italiano&area=ricoveriOspedalieri&menu=classificazione.

[17] Ministero della salute. *I quaderni del ministero della saluta. La centralità della Persona in riabilitazione: nuovi modelli organizzativi e gestionali*. Marzo - Aprile 2011. URL: http://www.salute.gov.it/imgs/C_17_pubblicazioni_1705_allegato.pdf.

[18] Mahoney FI, Barthel DW. *Functional evaluation: the barthel index*. 1965.

[19] Fondazione Italiana Sclerosi Multipla. *La scala FIM*. 2011. URL: http://scalafim.com/pages/scala_fim.html.

[20] Wikipedia. *Test Timed Up and Go — Wikipedia, L'enciclopedia libera*. 2020. URL: http://it.wikipedia.org/w/index.php?title=Test_Timed_Up_and_Go&oldid=114431101.

[21] Health Level Seven International. *CDA® (HL7 Clinical Document Architecture)*. 2007 - 2021. URL: https://www.hl7.org/implement/standards/product_brief.cfm?product_id=496.

[22] Agenzia per l'Italia Digitale. *La firma elettronica qualificata*. 2020. URL: https://www.agid.gov.it/it/piattaforme/firma-elettronica-qualificata.