UNIVERSITÀ DEGLI STUDI DI PALERMO

SCUOLA DELLE SCIENZE DI BASE ED APPLICATE

DIPARTIMENTO DI MATEMATICA ED INFORMATICA

S.H.I.V.A.

Software for **H**ealth **I**nfrastructures and medical **V**isits **A**dministration

Documentazione di progetto per l'esame di Ingegneria del software A.A. 2015/2016

Autore 1 matricola1

Autore 2 matricola2

Indice

SPMP

Software Project Management Plan

Indice SPMP

Prefazione

Questo documento serve a descrivere e definire la pianificazione della produzione del sistema in oggetto e il suo processo tecnico, ed è pertanto indirizzato al cliente ma anche a sviluppatori ed analisti.

È il documento di controllo del progetto per lo sviluppo di SHIVA – Software For Health Infrastructures and Visits Administration.

1. Introduzione

SHIVA è un software che ha come obbiettivo quello di semplificare la gestione di operazioni comuni all'interno di strutture mediche, soprattutto in relazione a gestione di prenotazioni di visite mediche e report ad esse associate.

Il software è stato creato sulle necessità di una struttura medica che adopera al suo interno tre tipologie di impiegati: receptionist, medico e amministratore di sistema.

Ai receptionist è stata data la possibilità di avere un supporto per le operazioni di registrazione di un nuovo paziente e prenotazione di una visita medica.

Ai medici è stata data la possibilità di gestire le operazioni di inserimento e visualizzazione di reports di visite mediche, nonché di potere visualizzare le prenotazioni di visite mediche che dovranno effettuare.

L'amministratore di sistema può invece registrare nel sistema medici e receptionist, può modificare i dati degli utenti del sistema e può eliminare gli account degli utenti stessi.

1.1 Acronimi e Definizioni

Di seguito vengono riportati gli acronimi presenti nel resto dei documenti.

- SPMP – Software Project Management Plan
- RAD – Requirement Analysis Document
- SDD – System Design Document
- ODD – Object Design Document
- UML – Unified Modeling Language
- DBMS – Database Management System
- GUI – Graphical User Interface
- DAO – Data Access Object

1.2 Elenco dei deliverables

I deliverables prodotti sono i seguenti:

- Software Project Management Plan (questo documento)
- Requirements Analysis Document: il documento contenente i dettagli dell'analisi dei requisiti.
- System Design Document: documento che definisce le scelte progettuali a livello di sistema, come la decomposizione in sottosistemi e la scelta dell'architettura di sistema.
- Object Design Document: il documento che descrive nel dettaglio l'implementazione del sistema.
- Un CD contenente:
    - Il codice sorgente di SHIVA
    - Tutta la documentazione
    - Tutti i file progettuali

1.2 Overview del progetto

Le attività principali svolte per la realizzazione del progetto sono riassunte nella tabella qui di seguito:

Data	**Fase del progetto**	**Attività cardine**
Dal 15/06/2016 al 20/06/2016	Studi preliminari	
Dal 21/06/2016 al 04/07/2016	Raccolta e Analisi dei requisiti	
05/07/2016		Produzione RAD
Dal 06/07/2016 al 09/07/2016	System Design	
11/07/2016		Produzione SDD
Dal 12/07/2016 al 15/07/2016	Object Design	
16/07/2016		Produzione ODD
Dal 18/07/2016 al 21/07/2016	Revisione Generale	
22/07/2016		Consegna del progetto

1.3 Evoluzione del Software Management Plan

La pianificazione delle attività di progettazione e sviluppo del software e la relativa allocazione delle risorse è stata gestita tramite il software open source ProjectLibre. Questo documento sarà aggiornato nel caso si verifichino variazioni relative alle attività appena descritte.

2. Tools utilizzati

Per la progettazione e lo sviluppo del progetto software SHIVA sono stati usati i seguenti strumenti:

- **ProjectLibre** – Software open source per la pianificazione di progetti.
- **Astah Professional** – CASE tool utilizzato per la modellazione UML e per la generazione di codice e JavaDoc
- **SceneBuilder** – Tool utilizzato per la prototipizzazione delle interfacce utente.
- **MySQL Workbench** – Tool utilizzato per la progettazione logica del database.
- **Diagram Designer** – Tool utilizzato per la creazione dei diagrammi ER del progetto del database e per i diagrammi di navigazione.

3. Organizzazione progetto

3.1 Risorse umane impiegate

Il team di sviluppo è composto da risorse umane specializzate in diversi settori, in modo da poter garantire completezza a livello globale di team. Di seguito vengono elencati i ruoli e le capacità:

- **Responsabile del progetto**: esperienza nella gestione delle risorse umane e capacità di coordinare e pianificare le attività e i meeting del team di sviluppo. Si occupa anche di concordare i meeting con il committente.
- **Ingegnere del software**: esperto nella progettazione e nella realizzazione di sistemi software.
- **Analista**: esperto dell'analisi dei domini dei problemi che il software deve risolvere. Spesso si interfaccia con esperti del dominio per acquisire le conoscenze necessarie ad esso legate.
- **Architetto di sistema**: esperto nella scelta di architetture di sistema e nei processi di decomposizione in sottosistemi e mappatura tra hardware e software.
- **Progettista GUI**: esperto nel prototipizzare, progettare e realizzare interfacce utente.

3.2 Attività

Il progetto è stato suddiviso nelle seguenti attività:

3.2.1 Studi preliminari

Durante questa fase vengono approfondite le conoscenze relative agli strumenti che verranno utilizzati e alle metodologie adottate.

3.2.2 Pianificazione del progetto

Durante questa fase viene descritto e pianificato il progetto: se ne descrivono le attività, le relazioni tra esse e la dipendenza dalle risorse utilizzate. Questa fase porta alla stesura del SPMP (questo documento).

3.2.3 Raccolta e Analisi dei requisiti

Durante questa fase vengono raccolte le informazioni necessarie a identificare il dominio del problema e viene esaminato il problema stesso al fine di definire il dominio della soluzione. L'analisi porta all'identificazione di quattro modelli:

- Modello dei casi d'uso
- Modello degli oggetti
- Modello funzionale
- Modello dinamico

L'analisi dei requisiti porta alla stesura del RAD.

3.2.4 System Design

Durante questa fase viene effettuata una progettazione a livello di sistema. Si definisce la suddivisione in sottosistemi e si descrive la mappatura tra hardware e software. Questa fase porta alla stesura del SDD.

3.2.5 Object Design

Questa fase mira alla realizzazione dei vari sottosistemi e porta alla stesura dell'ODD.

3.2.6 Revisione generale

Durante questa fase viene effettuata una revisione finale e generale del software e della documentazione prodotti.

3.2.7 Consegna del prodotto

Questa fase prevede la consegna del prodotto al cliente. La consegna viene accompagnata da una dimostrazione.

4. Vista complessiva sulla pianificazione del progetto

Di seguito vengono riportate le informazioni risultanti dalla pianificazione del progetto. La pianificazione è stata effettuata con l'ausilio del software ProjectLibre.

4.1 Risorse umane

Di seguito vengono riportate le risorse umane utilizzate nel progetto e i relativi compensi.

		Nome	Tipo	Initiali	Tasso standard	Calendario di base
1		Analista	Lavoro	A	€ 50,00/ora	Standard
2		Responsabile del progetto	Lavoro	R	€ 100,00/ora	Standard
3		Architetto di sistema	Lavoro	A	€ 70,00/ora	Standard
4		Progettista GUI	Lavoro	P	€ 25,00/ora	Standard
5		Ingegnere del software	Lavoro	I	€ 40,00/ora	Standard

FIGURA 1 - DESCRIZIONE DELLE RISORSE UMANE

4.1.1 Utilizzo delle risorse umane

Di seguito vengono riportati gli schemi che mostrano l'utilizzo effettivo delle risorse sopra presentate.

	Nome	Lavoro		lug 2016													
				12	15	18	21	24	27	30	03	06	09	12	15	18	21
1	Analista	114 ore	Lavoro	0h	0h	0h	24h	16h	24h	24h	10h	0h	0h	0h	4h	12h	0h
	Revisione Generale	*16 ore*	Lavoro												*4h*	*12h*	
	Definizione del modello din	*16 ore*	Lavoro							*16h*							
	Progettazione concettuale	*8 ore*	Lavoro							*8h*							
	Identificazione degli scenai	*16 ore*	Lavoro				*8h*	*8h*									
	Raccolta dei requisiti	*16 ore*	Lavoro				*16h*										
	Definizione del modello dei	*16 ore*	Lavoro					*8h*	*8h*								
	Revisione raccolta e analisi	*8 ore*	Lavoro								*8h*						
	Definizione del modello deg	*16 ore*	Lavoro						*16h*								
	Produzione RAD	*2 ore*	Lavoro								*2h*						
2	Responsabile del progetto	84 ore	Lavoro	0h	0h	0h	0h	0h	0h	0h	16h	0h	16h	8h	16h	24h	4h
	Produzione SDD	*8 ore*	Lavoro										*8h*				
	Revisione Generale	*32 ore*	Lavoro												*8h*	*24h*	
	Revisione Object Design	*8 ore*	Lavoro											*8h*			
	Revisione raccolta e analisi	*8 ore*	Lavoro								*8h*						
	Consegna del progetto	*4 ore*	Lavoro														*4h*
	Produzione RAD	*8 ore*	Lavoro								*8h*						
	Revisione System Design	*8 ore*	Lavoro										*8h*				
	Produzione ODD	*8 ore*	Lavoro												*8h*		
3	Architetto di sistema	42 ore	Lavoro	0h	0h	0h	0h	0h	0h	0h	0h	24h	10h	0h	2h	6h	0h
	Mappatura Hardware-softv	*8 ore*	Lavoro									*8h*					
	Revisione System Design	*8 ore*	Lavoro										*8h*				
	Produzione SDD	*2 ore*	Lavoro										*2h*				
	Revisione Generale	*8 ore*	Lavoro												*2h*	*6h*	
	Decomposizione del sistem	*16 ore*	Lavoro									*16h*					
4	Progettista GUI	10 ore	Lavoro	0h	0h	0h	0h	0h	0h	8h	2h	0h	0h	0h	0h	0h	0h
	Mockup Interfacce grafiche	*8 ore*	Lavoro							*8h*							
	Revisione raccolta e analisi	*2 ore*	Lavoro								*2h*						
5	Ingegnere del software	44 ore	Lavoro	0h	0h	0h	0h	0h	0h	0h	0h	8h	2h	24h	4h	6h	0h
	Revisione Object Design	*8 ore*	Lavoro											*8h*			
	Revisione Generale	*8 ore*	Lavoro												*2h*	*6h*	
	Revisione System Design	*2 ore*	Lavoro										*2h*				
	Produzione ODD	*2 ore*	Lavoro												*2h*		
	Definizione contratti tra cla	*8 ore*	Lavoro											*8h*			
	Produzione JAVADoc	*8 ore*	Lavoro											*8h*			
	Progettazione logica Datab	*8 ore*	Lavoro									*8h*					

4.2 Attività

Di seguito vengono mostrate le attività nel dettaglio, il loro diagramma di Gantt e il loro diagramma di Pert.

FIGURA 1.1 – UTILIZZO DELLE RISORSE UMANE

	Nome	Durata	Avvio	Termine	Predecessori	Nome risorsa
2	**Raccolta e analisi dei requisiti**	12 giorni	**21/06/16 8.00**	**04/07/16 17.00**	**1**	
3	Raccolta dei requisiti	2 giorni	21/06/16 8.00	22/06/16 17.00		Analista
4	Identificazione degli scenari	2 giorni	23/06/16 8.00	24/06/16 17.00	3	Analista
5	Definizione del modello dei casi d'uso	2 giorni	25/06/16 8.00	27/06/16 17.00	4	Analista
6	Definizione del modello degli oggetti	2 giorni	28/06/16 8.00	29/06/16 17.00	5	Analista
7	Definizione del modello dinamico	2 giorni	30/06/16 8.00	01/07/16 17.00	6	Analista
8	Progettazione concettuale Database	1 giorno	02/07/16 8.00	02/07/16 17.00	7	Analista
9	Mockup Interfacce grafiche	1 giorno	02/07/16 8.00	02/07/16 17.00	7	Progettista GUI
10	Revisione raccolta e analisi dei requisiti	1 giorno	04/07/16 8.00	04/07/16 17.00	8;9	Analista;Progettista GUI[25%];Responsabile del progetto
11	Produzione RAD	1 giorno	05/07/16 8.00	05/07/16 17.00	2	Responsabile del progetto;Analista[25%]
12	**System Design**	4 giorni	**06/07/16 8.00**	**09/07/16 17.00**	**11**	
13	Decomposizione del sistema	2 giorni	06/07/16 8.00	07/07/16 17.00		Architetto di sistema
14	Progettazione logica Database	1 giorno	06/07/16 8.00	06/07/16 17.00		Ingegnere del software
15	Mappatura Hardware-software	1 giorno	08/07/16 8.00	08/07/16 17.00	13;14	Architetto di sistema
16	Revisione System Design	1 giorno	09/07/16 8.00	09/07/16 17.00	15	Ingegnere del software[25%];Architetto di sistema;Responsabile del progetto
17	Produzione SDD	1 giorno	11/07/16 8.00	11/07/16 17.00	12	Responsabile del progetto;Architetto di sistema[25%]
18	**Object Design**	3 giorni	**12/07/16 8.00**	**14/07/16 17.00**	**17**	
19	Definizione contratti tra classi	1 giorno	12/07/16 8.00	12/07/16 17.00		Ingegnere del software
20	Produzione JAVADoc	1 giorno	13/07/16 8.00	13/07/16 17.00	19	Ingegnere del software
21	Revisione Object Design	1 giorno	14/07/16 8.00	14/07/16 17.00	20	Responsabile del progetto;Ingegnere del software
22	Produzione ODD	1 giorno	15/07/16 8.00	15/07/16 17.00	18	Responsabile del progetto;Ingegnere del software[25%]
23	Revisione Generale	4 giorni	16/07/16 8.00	20/07/16 17.00	22	Responsabile del progetto;Analista[50%];Architetto di sistema[25%];Ingegnere del software[25%]
24	Consegna del progetto	0,5 giorni	21/07/16 8.00	21/07/16 13.00	23	Responsabile del progetto

FIGURA 2.1 - ATTIVITÀ

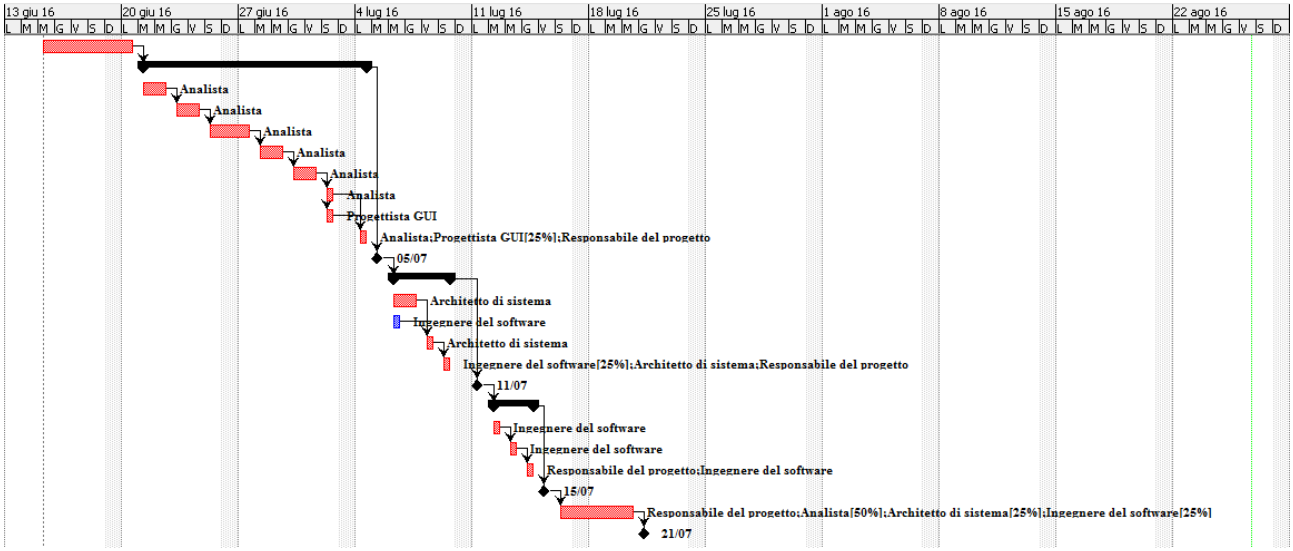


FIGURA 2.2 - DIAGRAMMA DI GANTT

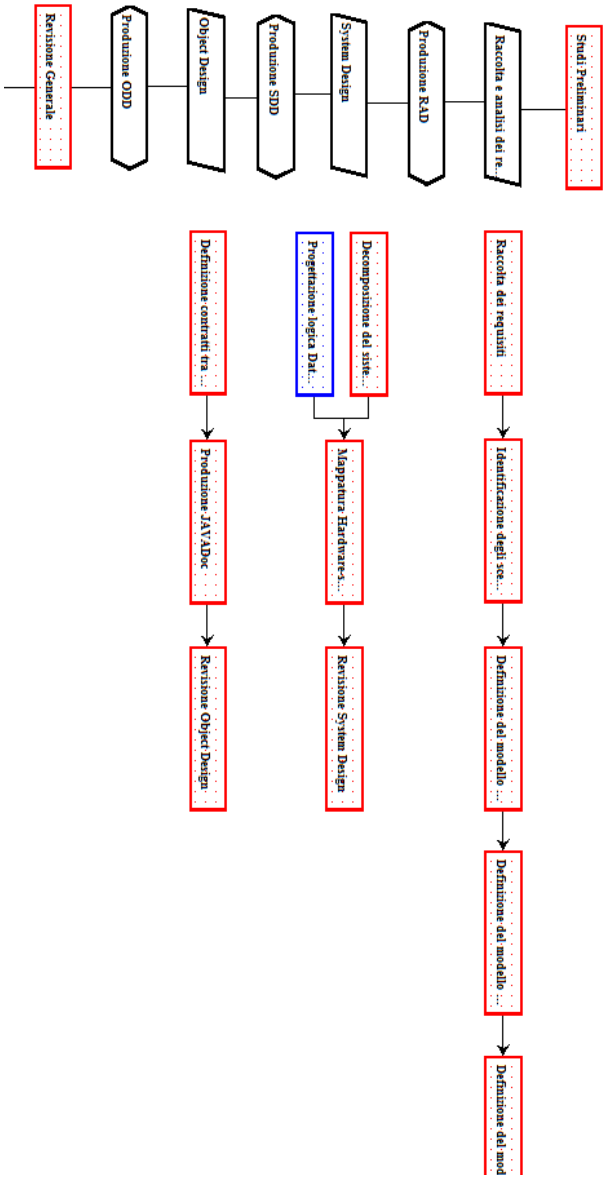


FIGURA 2.3 - DIAGRAMMA DI PERT

4.3 Stima costi progetto

Di seguito la stima dei costi del progetto ottenuta tramite l'utilizzo di ProjectLibre

	Nome	Lavoro	Costo
1	**SHIVA**	**334 ore**	**€ 19050,00**
2	Studi Preliminari	40 ore	€ 0,00
3	**Raccolta e analisi dei requisiti**	**114 ore**	**€ 5850,00**
4	Raccolta dei requisiti	16 ore	€ 800,00
5	Identificazione degli scenari	16 ore	€ 800,00
6	Definizione del modello dei casi d'uso	16 ore	€ 800,00
7	Definizione del modello degli oggetti	16 ore	€ 800,00
8	Definizione del modello dinamico	16 ore	€ 800,00
9	Progettazione concettuale Database	8 ore	€ 400,00
10	Mockup Interfacce grafiche	8 ore	€ 200,00
11	Revisione raccolta e analisi dei requisiti	18 ore	€ 1250,00
12	Produzione RAD	10 ore	€ 900,00
13	**System Design**	**50 ore**	**€ 3440,00**
14	Decomposizione del sistema	16 ore	€ 1120,00
15	Progettazione logica Database	8 ore	€ 320,00
16	Mappatura Hardware-software	8 ore	€ 560,00
17	Revisione System Design	18 ore	€ 1440,00
18	Produzione SDD	10 ore	€ 940,00
19	**Object Design**	**32 ore**	**€ 1760,00**
20	Definizione contratti tra classi	8 ore	€ 320,00
21	Produzione JAVADoc	8 ore	€ 320,00
22	Revisione Object Design	16 ore	€ 1120,00
23	Produzione ODD	10 ore	€ 880,00
24	Revisione Generale	64 ore	€ 4880,00
25	Consegna del progetto	4 ore	€ 400,00

FIGURA 2.4 - STIMA DEI COSTI DEL PROGETTO

4.4 Confronto con soluzioni esistenti

Il mercato propone numerose soluzioni software per la gestione di strutture da un punto di vista amministrativo. Meno frequenti sono le soluzioni orientate a strutture mediche, che spesso presentano costi decisamente più alti rispetto a quelli qui proposti. In più il sistema SHIVA è stato creato sulla base delle problematiche espresse dal committente, e quindi risulta più specifico e adatto rispetto alle soluzioni generali presenti in commercio.

RAD

Requirements Analysis Document

Indice RAD

1. Obbiettivi generali

Il sistema si pone l'obbiettivo di semplificare la gestione dei clienti di una struttura medica, attraverso strumenti che consentono di rendere più agevole la gestione delle prenotazioni e dei report dei clienti. Inoltre, il software fornisce ulteriori funzionalità al medico, che potrà avere visione in maniera agevole delle proprie visite da effettuare.

2. Sistema attuale

Attualmente la clinica non utilizza nessun software di gestione. Gli impiegati della clinica utilizzano suite di software orientate al generico lavoro di ufficio (LibreOffice / Microsoft Office) sia per la stesura dei documenti, come ad esempio i report, sia per le prenotazioni (tramite fogli di calcolo). Manca completamente un sistema di gestione centralizzato che semplifichi il lavoro degli impiegati della clinica, verificando la congruenza dei dati (ad esempio verificando in maniera automatica che non si verifichino collisioni di prenotazione), fornendo un supporto robusto e affidabile per la memorizzazione dei dati e migliorando in maniera trasparente le comunicazioni tra impiegati con differenti ruoli.

3. Sistema proposto

3.1 Overview

Il sistema proposto è stato realizzato col fine di agevolare la gestione di una struttura medica. Il prodotto si basa sulla divisione dei ruoli dei dipendenti che operano all'interno della struttura, dedicando ad ognuno di essi una vista specifica che fornisce gli strumenti dedicati al proprio ruolo.

3.2 Requisiti funzionali

Il sistema prevede l'utilizzo da parte di 3 categorie di utenti (receptionist, medico e amministratore di sistema) di un numero variabile di postazioni non deciso a priori.

L'utente accede al sistema tramite una qualsiasi delle postazioni completando una procedura di autenticazione, tramite cui il sistema determina il ruolo dell'utente e le operazioni a lui consentite.

Il sistema deve fornire ad un utente di tipo receptionist la possibilità di:

- Registrare i pazienti: il receptionist inserisce nel sistema i dati anagrafici, almeno un contatto telefonico e/o un contatto email. Il sistema memorizza il paziente e restituisce una conferma.
- Effettuare per loro prenotazioni per le prestazioni offerte della clinica

Il sistema deve fornire ad un utente di tipo medico la possibilità di:

- Visualizzare la lista delle visite che deve ancora effettuare.
- Creare un report relativo ad una visita e di associarlo al paziente oggetto della visita.
- Visualizzare tutti i report associati ad un paziente.

Il sistema deve fornire ad un utente di tipo amministratore di sistema la possibilità di:

- Registrare nuovi account di tipo medico o receptionist.
- Effettuare modifiche ad account di tipo medico o receptionist.
- Effettuare la cancellazione di account di tipo medico o receptionist.

3.3 Requisiti non funzionali

3.3.1 Interfaccia grafica e Fattore umano

L'interazione fra utente e sistema avviene tramite interfaccia grafica, progettata per consentirne l'utilizzo anche da parte di utenti non esperti.

3.3.2 Considerazioni hardware

Il sistema richiede una piattaforma hardware in grado di supportare il DBMS e una postazione per utente. Per quanto concerne le piattaforme hardware dedicate agli utenti finali sono sufficienti dei sistemi desktop d'ufficio. Si necessita inoltre di una struttura di rete che consenta la comunicazione fra le postazioni utente (dislocate su diversi piani) e DBMS.

3.3.3 Caratteristiche di performance

La mole di dati che viene scambiata fra le postazioni utenti e il DBMS non è tale da richiedere particolari requisiti di performance di rete. Il sistema deve comunque fornire una comunicazione preferibilmente con tempi di risposta ridotti e con bassa latenza.

3.3.4 Gestione degli errori e condizioni estreme

Ogni volta che un'operazione effettuata dal sistema non va a buon fine viene presentato all'utente un messaggio di errore, che ne comunica la natura e le possibili cause. Nel caso di operazioni sensibili che comportano la modifica e/o la creazione di record all'interno del database, un successo viene accompagnato da un messaggio di buona riuscita, confermando all'utente l'esito positivo. Per le altre operazioni, come ad esempio la visualizzazione di un report associato ad un paziente, il successo è già implicitamente denotato dalla presentazione della vista corretta.

Nel caso non sia possibile raggiungere il DBMS per problemi di connessione, il software notifica il problema all'utente e arresta la sua esecuzione.

3.3.5 Problemi di sicurezza

La sicurezza relativa all'accesso al sistema è garantita da un sistema di login a credenziali (username e password). Queste vengono fornite ai vari utenti dall'amministratore di sistema al momento della registrazione. Ogni utente accede al sistema dopo essersi autenticato con le proprie credenziali. Ad ogni utente vengono mostrate le sole funzionalità supportate dal ruolo dell'utente stesso: ciò consente di evitare l'utilizzo di particolari funzionalità del sistema da parte di ruoli utente non autorizzati, ad esempio il sistema consente la visione dei dati e dei reports dei clienti esclusivamente da parte dei medici. Considerato inoltre che la rete di comunicazione fra i client e i server è locale, non vi sono problematiche legate agli accessi esterni alla rete.

3.4 Pseudo-Requisiti

Il linguaggio d'implementazione deve essere JAVA.

Il DBMS utilizzato deve essere di tipo Relazionale. Questa scelta deriva dal fatto che i dati persistiti sono dati ben strutturati.

La connessione al DBMS deve avvenire tramite driver JDBC. Questa scelta deriva dal fatto che è l'API ufficiale delle specifiche del linguaggio.

3.5 Modello del sistema

3.5.1 Scenari

Di seguito verranno rappresentati gli scenari più comuni presenti all'interno della struttura medica.

3.5.1.1 Scenario Login

Nome scenario	Login
Attori partecipanti	Paola :Utente :DBMS
Flusso di eventi	1. Paola deve accedere al sistema per iniziare a lavorare. Avvia la sua postazione e manda in esecuzione il software. 2. SHIVA presenta a Paola la vista di login. 3. Paola inserisce le sue credenziali di accesso: username e password. 4. Il server di SHIVA verifica la correttezza delle credenziali interrogando il DBMS. 5. SHIVA presenta a Paola la schermata principale del software.

3.5.1.2 Scenario Login_errato

Nome scenario	Login_errato
Attori partecipanti	Paola :Utente :DBMS
Flusso di eventi	1. Paola deve accedere al sistema per iniziare a lavorare. Avvia la sua postazione e manda in esecuzione il software. 2. SHIVA presenta a Paola la vista di login. 3. Paola inserisce in maniera incorretta le sue credenziali di accesso: username e password. 4. SHIVA interroga il DBMS e riscontra negativamente le credenziali di accesso. 5. SHIVA presenta a Paola un messaggio di login errato. 6. Paola conferma la lettura del messaggio di errore. 7. SHIVA ripresenta a Paola la vista di login.

3.5.1.3 Scenario Logout

Nome scenario	Logout
Attori partecipanti	Francesca :Utente
Flusso di eventi	1. Francesca deve allontanarsi dalla postazione e decide di effettuare il logout. Francesca seleziona quindi la voce logout. 2. SHIVA presenta a Francesca un messaggio di logout effettuato con successo. 3. SHIVA presenta la schermata di login in attesa del successivo accesso.

3.5.1.4 Scenario Registrazione_Paziente

Nome scenario	Registrazione_Paziente
Attori partecipanti	Simona :Receptionist : DBMS
Flusso di eventi	1. Simona accoglie alla reception un paziente che deve essere registrato nel sistema. Simona seleziona quindi la voce *registra nuovo paziente*. 2. SHIVA presenta a Simona la vista di registrazione di un nuovo paziente. 3. Simona inserisce nel sistema i dati anagrafici e i recapiti relativi al paziente. 4. SHIVA presenta a Simona un messaggio di richiesta di conferma della registrazione. 5. Simona conferma la registrazione. 6. SHIVA richiede la verifica dell'eventuale presenza nel database di un altro paziente con lo stesso codice fiscale inserito al DMBS. 7. SHIVA registra nel database il paziente tramite il DBMS. 8. SHIVA restituisce un messaggio di registrazione avvenuta con successo.

3.5.1.5 Scenario Registrazione_Medico

Nome scenario	Registrazione_medico
Attori partecipanti	Roberto :Amministratore_di_sistema : DBMS
Flusso di eventi	1. Roberto deve registrare nel sistema un nuovo medico. Roberto seleziona quindi la voce *registra medico*. 2. SHIVA presenta a Roberto la vista di registrazione di un nuovo medico. 3. Roberto inserisce nel sistema i dati anagrafici e i recapiti relativi al medico. Roberto inserisce altresì le specializzazioni del medico. 4. SHIVA presenta a Roberto un messaggio di richiesta di conferma della registrazione. 5. Roberto conferma la registrazione. 6. SHIVA richiede la verifica dell'eventuale presenza nel database di un altro medico con lo stesso codice fiscale inserito al DBMS. 7. SHIVA registra nel database il medico tramite il DBMS. 8. SHIVA restituisce un messaggio di registrazione avvenuta con successo. Nel messaggio sono specificati l'username e la password generati dal sistema per l'account utente appena creato.

3.5.1.6 Scenario Registrazione_Receptionist

Nome scenario	Registrazione_receptionist
Attori partecipanti	Roberto :Amministratore_di_sistema : DBMS
Flusso di eventi	1. Roberto deve registrare nel sistema un nuovo receptionist. Roberto seleziona quindi la voce *registra receptionist*. 2. SHIVA presenta a Roberto la vista di registrazione di un nuovo receptionist. 3. Roberto inserisce nel sistema i dati anagrafici e i recapiti relativi al receptionist. 4. SHIVA presenta a Roberto un messaggio di richiesta di conferma della registrazione. 5. Roberto conferma la registrazione. 6. SHIVA richiede la verifica dell'eventuale presenza nel database di un altro receptionist cono lo stesso codice fiscale inserito al DBMS. 7. SHIVA registra nel database il receptionist tramite il DBMS. 8. SHIVA restituisce un messaggio di registrazione avvenuta con successo. Nel messaggio sono specificati l'username e la password generati dal sistema per l'account utente appena creato.

3.5.1.7 Scenario Visualizzazione_Paziente

Nome scenario	Visualizzazione_Paziente
Attori partecipanti	Paolo :Medico : DBMS
Flusso di eventi	1. Paolo vuole visualizzare le informazioni associate ad un paziente a cui deve effettuare una visita. Paolo seleziona quindi la voce *cerca paziente*. 2. SHIVA presenta a Paolo una vista tramite cui effettuare la ricerca del paziente d'interesse. 3. Paolo compila i campi di ricerca. 4. SHIVA cerca il paziente interrogando il DBMS. 5. SHIVA presenta a Paolo la lista dei risultati della ricerca. 6. Paolo seleziona il paziente dalla lista dei risultati. 7. SHIVA presenta a Paolo una vista in cui sono presenti i dati relativi al paziente e da cui può accedere ai reports associati al paziente.

3.5.1.8 Scenario Visualizzazione_Report

Nome scenario	Visualizzazione_Report
Attori partecipanti	Paolo :Medico : DBMS
Flusso di eventi	1. Paolo deve visualizzare un report relativo ad una visita medica effettuata ad un paziente. Paolo seleziona quindi la voce *visualizza reports*. 2. SHIVA presenta a Paolo una vista tramite cui effettuare la ricerca del paziente a cui è associato il report. 3. Paolo compila i campi di ricerca. 4. SHIVA cerca il paziente interrogando il DBMS. 5. SHIVA presenta a Paolo la lista dei risultati della ricerca. 6. Paolo seleziona il paziente dalla lista dei risultati. 7. SHIVA cerca tutti i reports associati al paziente selezionato interrogando il DBMS. 8. SHIVA presenta a Paolo una vista in è presente la lista dei reports associati al paziente d'interesse, ordinati cronologicamente. 9. Paolo seleziona dalla lista il report a cui è interessato. 10. SHIVA presenta a Paolo una vista in cui sono presenti i dettagli del report selezionato.

3.5.1.9 Scenario Inserimento_Report

Nome scenario	Inserimento_Report
Attori partecipanti	Paolo :Medico : DBMS
Flusso di eventi	1. Paolo deve stilare un report relativo ad una visita medica effettuata ad un paziente. Paolo seleziona quindi la voce *inserisci nuovo report*. 2. SHIVA presenta a Paolo una vista tramite cui effettuare la ricerca del paziente a cui associare il report. 3. Paolo compila i campi di ricerca. 4. SHIVA cerca il paziente interrogando il DBMS. 5. SHIVA presenta a Paolo la lista dei risultati della ricerca. 6. Paolo seleziona il paziente dalla lista dei risultati. 7. SHIVA presenta a Paolo una vista in cui può compilare il nuovo report. 8. Paolo compila il report inserendo la data in cui è stata effettuata la visita, il titolo del report, la descrizione della visita, la diagnosi e le prescrizioni mediche derivanti dalla visita. Paolo allega altresì i file relativi alle scansioni di due documenti cartacei e conferma la compilazione del report. 9. SHIVA registra il nuovo report e lo associa al paziente tramite il DBMS. 10. SHIVA mostra a Paolo un messaggio di inserimento nuovo report effettuato con successo.

3.5.1.10 Scenario Visualizzazione_visite

Nome scenario	Visualizzazione_visite
Attori partecipanti	Paolo :Medico : DBMS
Flusso di eventi	1. Paolo desidera vedere quali visite dovrà effettuare in giornata tra quelle prenotate. Paolo seleziona quindi la voce *visualizza visite*. 2. SHIVA cerca tutte le prenotazioni di visite associate a Paolo interrogando il DBMS. 3. SHIVA presenta a Paolo una vista in cui viene mostrata la lista delle visite che Paolo deve effettuare, ordinate cronologicamente.

3.5.1.11 Scenario Prenotazione_visita

Nome scenario	Prenotazione_visita
Attori partecipanti	Simona :Receptionist : DBMS
Flusso di eventi	1. Simona su richiesta di un paziente deve effettuare una prenotazione per una visita medica oculistica. Simona seleziona quindi la voce *effettua prenotazione*. 2. SHIVA presenta a Simona una vista in cui è presente una lista di tutte le tipologie di visite effettuabili presso la struttura. 3. Dalla lista Simona seleziona la voce *visita oculistica*. 4. SHIVA cerca le date in cui almeno un medico con specializzazione oculistica registrato nella struttura non ha già prenotazioni visite assegnate tramite il DBMS. 5. SHIVA presenta a Simona una vista in cui sono mostrate le date disponibili. 6. Concordata una data utile per il paziente tra quelle disponibili, Simona la seleziona dalla vista. 7. SHIVA presenta a Simona una vista tramite cui effettuare la ricerca del paziente a cui associare la prenotazione. 8. Simona compila i campi di ricerca. 9. SHIVA cerca il paziente interrogando il DBMS. 10. SHIVA presenta a Simona la lista dei risultati della ricerca. 11. Simona seleziona il paziente dalla lista dei risultati. 12. SHIVA presenta a Simona una vista di riepilogo in cui sono riassunti i dati relativi alla prenotazione. 13. Simona conferma la prenotazione. 14. SHIVA registra la prenotazione e la associa al paziente e ad un medico con specializzazione oculistica che non abbia già una prenotazione assegnata in quella data tramite il DBMS. 15. SHIVA mostra a Simona un messaggio di prenotazione effettuata con successo.

3.5.1.12 Scenario Modifica_receptionist

Nome scenario	Modifica_receptionist
Attori partecipanti	Roberto :Amministratore_di_sistema : DBMS
Flusso di eventi	1. Roberto deve modificare i dati relativi ad un receptionist registrato nel sistema. Roberto seleziona quindi la voce *modifica utente*. 2. SHIVA presenta a Roberto una vista tramite cui può effettuare la ricerca del receptionist a cui apportare le modifiche. 3. Roberto compila i campi di ricerca, selezionando come tipo di utente *receptionist*. 4. SHIVA cerca il receptionist interrogando il DBMS. 5. SHIVA presenta a Roberto la lista dei risultati della ricerca. 6. Roberto seleziona il receptionist dalla lista dei risultati. 7. SHIVA presenta a Roberto una vista in cui può modificare l'indirizzo di residenza e i recapiti del receptionist. 8. Roberto apporta le modifiche. 9. SHIVA registra le modifiche tramite il DBMS. 10. SHIVA restituisce un messaggio di modifica effettuata con successo.

3.5.1.13 Scenario Modifica_medico

Nome scenario	Modifica_medico
Attori partecipanti	Roberto :Amministratore_di_sistema : DBMS
Flusso di eventi	1. Roberto deve modificare i dati relativi ad un medico registrato nel sistema. Roberto seleziona quindi la voce *modifica utente*. 2. SHIVA presenta a Roberto una vista tramite cui può effettuare la ricerca del medico a cui apportare le modifiche. 3. Roberto compila i campi di ricerca, selezionando come tipo di utente *medico*. 4. SHIVA cerca il medico interrogando il DBMS. 5. SHIVA presenta a Roberto la lista dei risultati della ricerca. 6. Roberto seleziona il medico dalla lista dei risultati. 7. SHIVA presenta a Roberto una vista in cui può modificare l'indirizzo di residenza e i recapiti del medico. 8. Roberto apporta le modifiche tramite il DBMS. 9. Il server di SHIVA registra le modifiche nel database. 10. SHIVA restituisce un messaggio di modifica effettuata con successo.

3.5.1.14 Scenario Cancellazione_utente

Nome scenario	Cancellazione_utente
Attori partecipanti	Roberto :Amministratore_di_sistema : DBMS
Flusso di eventi	1. Roberto deve eliminare l'account di un utente del sistema. Roberto seleziona quindi la voce *elimina utente*. 2. SHIVA presenta a Roberto una vista tramite cui può effettuare la ricerca dell'utente da eliminare. 3. Roberto compila i campi di ricerca. 4. SHIVA cerca l'utente interrogando il DBMS. 5. SHIVA presenta a Roberto la lista dei risultati della ricerca. 6. Roberto seleziona l'utente dalla lista dei risultati. 7. SHIVA presenta a Roberto una vista di riepilogo dell'utente selezionato. 8. Roberto seleziona la voce *elimina*. 9. SHIVA presenta a Roberto un messaggio in cui si richiede conferma dell'operazione. 10. Roberto conferma l'eliminazione. 11. SHIVA elimina l'utente dal database tramite il DBMS. 12. SHIVA restituisce un messaggio di eliminazione avvenuta con successo.

3.5.1.15 Scenario Modifica_password

Nome scenario	Modifica_password
Attori partecipanti	Roberto :Utente : DBMS
Flusso di eventi	1. Roberto decide di cambiare la sua password. Roberto seleziona quindi la voce *modifica password*. 2. SHIVA presenta a Roberto una vista tramite cui può modificare la sua password e in cui sono presenti le informazioni riguardo il formato che la nuova password deve avere. 3. Roberto inserisce la sua vecchia password e la nuova password scelta. 4. SHIVA verifica che la vecchia password sia corretta. 5. SHIVA presenta a Roberto un messaggio in cui si richiede conferma della modifica della password. 6. Roberto conferma la modifica della password. 7. SHIVA modifica nel database la password di Roberto tramite il DBMS. 8. SHIVA restituisce un messaggio di modifica password avvenuta con successo.

3.5.2 Modelli dei casi d'uso

3.5.2.1 Attori

In seguito vengono riportati gli attori che interagiscono con SHIVA:

Nome	**Descrizione**
Receptionist	Dipendente della struttura medica. Dopo essersi autenticato correttamente ha accesso alle funzionalità di gestione delle prenotazioni e registrazione dei pazienti presso la struttura.
Medico	Dipendente della struttura medica. Dopo essersi autenticato correttamente può visualizzare i reports dei pazienti e la lista delle visite da effettuare.
Amministratore di Sistema	Dipendente della struttura medica. Dopo essersi autenticato correttamente ha accesso ai strumenti utili per la gestione degli accounts dei receptionist e dei medici.
Utente	Una generalizzazione di un dipendente della struttura medica.
DBMS	Il DBMS in cui verranno persistiti i dati.

3.5.2.2 Casi d'uso

Questo diagramma rappresenta una visione generale dei casi d'uso del sistema.

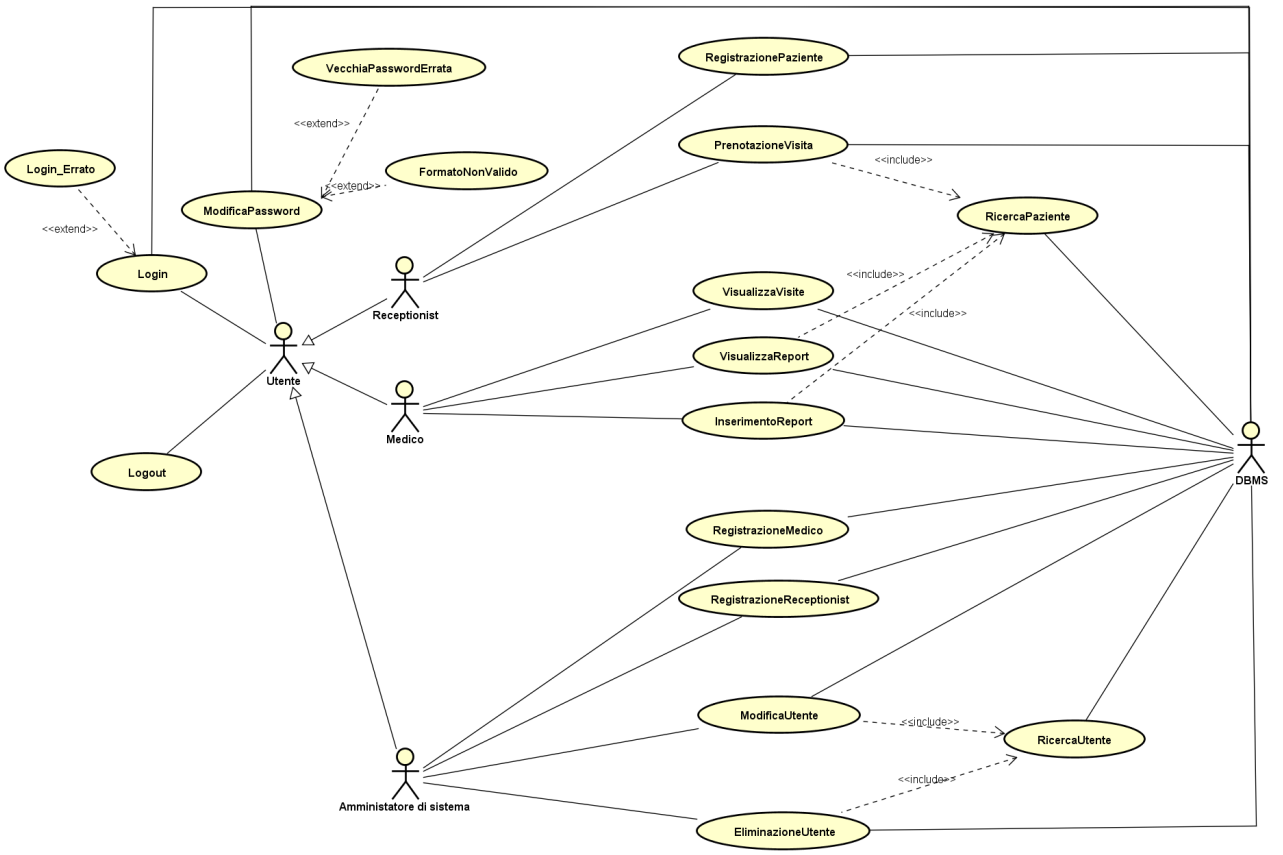


FIGURA 3 - DIAGRAMMA GENERALE DEI CASI D'USO

3.5.2.2.1 Caso d'uso Login

Nome caso d'uso	Login
Attori partecipanti	(Inizializzato da) Utente DBMS
Flusso di eventi	1. L'utente inserisce le sue credenziali di accesso (username e password) e li conferma. 2. Il server di SHIVA verifica la correttezza delle credenziali interrogando il DBMS. 3. SHIVA mostra le operazioni disponibili all'utente.
Condizione di entrata	• L'utente vuole autenticarsi al sistema
Condizione di uscita	• L'utente visualizza le operazioni disponibili
Requisiti di qualità	• Deve essere possibile connettersi al server di SHIVA. • Il server di SHIVA deve poter comunicare con il DBMS.

3.5.2.2.1 Caso d'uso Login_Errato

Nome caso d'uso	Login_Errato
Attori partecipanti	Utente
Flusso di eventi	*Questo caso d'uso estende il caso d'uso di Login quando l'autenticazione non avviene con successo a causa di username o password errati inseriti dall'utente.* 1. L'utente riceve un messaggio di errore da SHIVA. 2. SHIVA ripresenta la vista di login all'utente.
Condizione di entrata	• L'utente inserisce username o password errati
Condizione di uscita	• SHIVA consente nuovamente all'utente di autenticarsi
Requisiti di qualità	

3.5.2.2.2 Caso d'uso Logout

Nome caso d'uso	Logout
Attori partecipanti	(Inizializzato da) Utente
Flusso di eventi	1. L'utente seleziona la voce di *logout.* 2. SHIVA termina la sessione dell'utente.
Condizione di entrata	• L'utente è autenticato • L'utente vuole disconnettersi dal sistema
Condizione di uscita	• L'utente è disconnesso dal sistema
Requisiti di qualità	

3.5.2.2.3 Caso d'uso ModificaPassword

Nome caso d'uso	ModificaPassword
Attori partecipanti	(Inizializzato da) Utente DBMS
Flusso di eventi	1. SHIVA presenta all'utente la vista di modifica password. Nella vista è specificato il formato che deve avere la nuova password. 2. L'utente inserisce la vecchia e la nuova password. 3. SHIVA presenta all'utente un messaggio in cui si richiede conferma della modifica della password. 4. L'utente conferma la modifica. 5. SHIVA comunica la modifica della password dell'utente al DBMS. 6. SHIVA presenta all'utente un messaggio con cui notifica il successo dell'operazione. 7. L'utente conferma il messaggio contenente l'esito dell'operazione.
Condizione di entrata	• L'utente è autenticato. • L'utente seleziona la voce *modifica password*.
Condizione di uscita	• La password dell'utente è stata modificata
Requisiti di qualità	• Deve essere possibile connettersi al server di SHIVA. • Il server di SHIVA deve poter comunicare con il DBMS. • La vecchia password deve essere corretta. • La nuova password deve avere il formato specificato da SHIVA.

3.5.2.2.4 Caso d’uso VecchiaPasswordErrata

Nome caso d’uso	VecchiaPasswordErrata
Attori partecipanti	Utente
Flusso di eventi	*Questo caso d’uso estende il caso d’uso di ModificaPassword quando l’utente inserisce la sua attuale password non correttamente* 1. L’utente riceve un messaggio di errore da SHIVA. 2. SHIVA ripresenta la vista di modifica della password all’utente.
Condizione di entrata	• L’utente è autenticato. • L’utente inserisce la precedente password errata
Condizione di uscita	• Il sistema consente nuovamente all’utente di modificare la password
Requisiti di qualità	

3.5.2.2.4 Caso d'uso FormatoNonValido

Nome caso d'uso	FormatoNonValido
Attori partecipanti	Utente
Flusso di eventi	*Questo caso d'uso estende il caso d'uso di ModificaPassword quando l'utente inserisce la sua nuova password in un formato non valido* 1. L'utente riceve un messaggio di errore da SHIVA. 2. SHIVA ripresenta la vista di modifica della password all'utente.
Condizione di entrata	• L'utente è autenticato. • L'utente inserisce la nuova password in un formato non valido
Condizione di uscita	• Il sistema consente nuovamente all'utente di modificare la password
Requisiti di qualità	

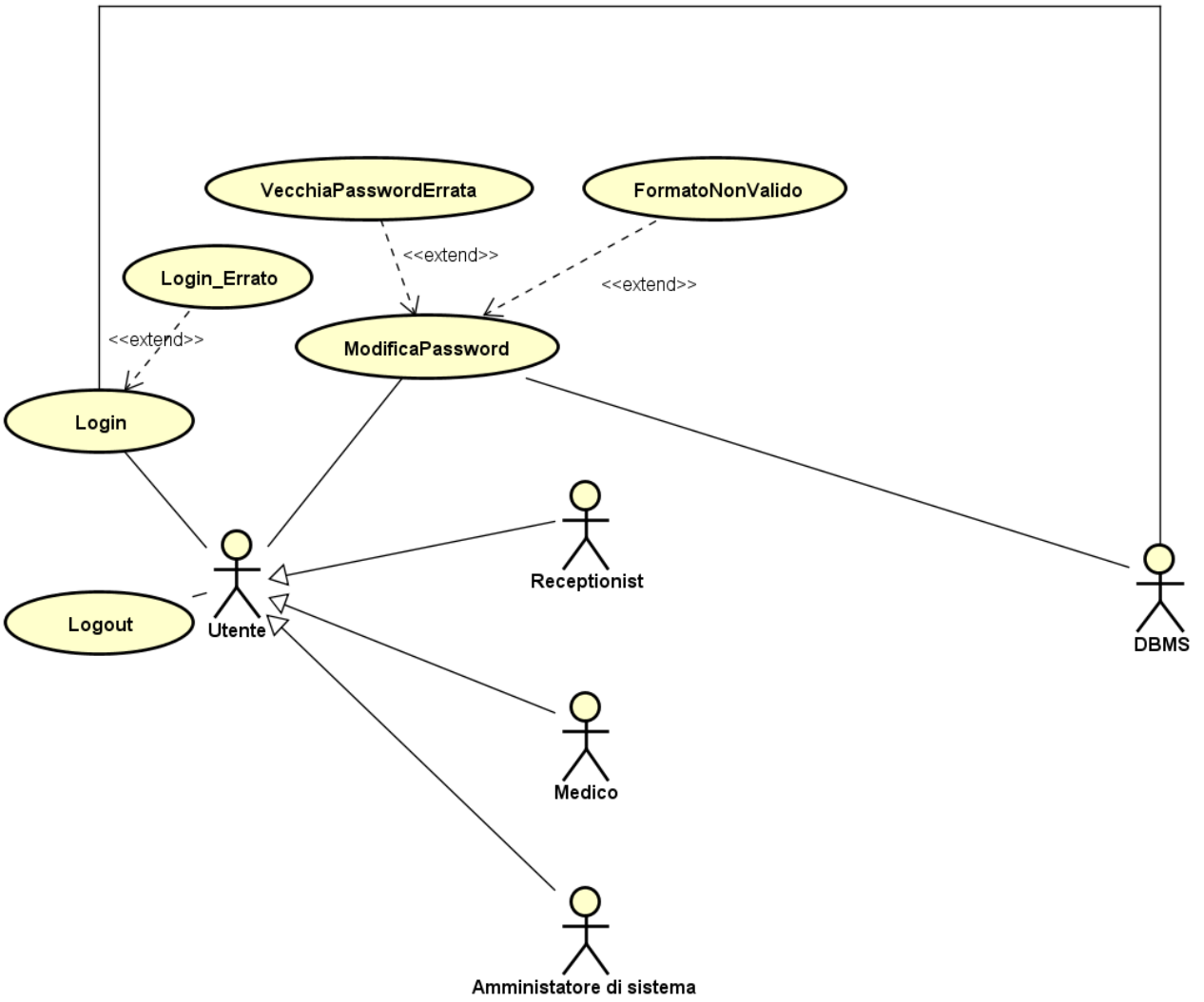


FIGURA 4 - DIAGRAMMA RIASSUNTIVO DEI CASI D'USO DELL'UTENTE

3.5.2.2.5 Caso d'uso RegistrazionePaziente

Nome caso d'uso	RegistrazionePaziente
Attori partecipanti	(Inizializzato da) Receptionist DBMS
Flusso di eventi	1. SHIVA fornisce al receptionist una vista di registrazione nuovo paziente 2. Il receptionist compila i campi della vista inserendo i dati anagrafici del paziente. 3. SHIVA presenta al receptionist un messaggio di richiesta di conferma della registrazione. 4. Il receptionist conferma la registrazione. 5. SHIVA richiede al DBMS il controllo dell'eventuale presenza di un altro paziente con lo stesso codice fiscale . 6. SHIVA registra nel database il paziente tramite il DBMS. 7. SHIVA restituisce un messaggio di registrazione avvenuta con successo.
Condizione di entrata	• Il receptionist è autenticato. • Il receptionist seleziona la voce *registra paziente*.
Condizione di uscita	• Il paziente è stato registrato con successo
Requisiti di qualità	• Il codice fiscale inserito deve essere coerente coi dati anagrafici. • Non deve essere già presente nel database un paziente con il codice fiscale inserito. • Deve essere possibile connettersi al server di SHIVA. • Il server di SHIVA deve poter comunicare con il database.

3.5.2.2.6 Caso d'uso RicercaPaziente

Nome caso d'uso	RicercaPaziente
Attori partecipanti	Receptionist/Medico DBMS
Flusso di eventi	1. SHIVA fornisce al Receptionist/Medico una vista per la ricerca di un paziente. 2. Il Receptionist/Medico compila i campi della vista inserendo i dati del paziente. 3. SHIVA cerca il paziente tramite il DBMS. 4. SHIVA presenta al Receptionist/Medico una lista contenente i risultati della ricerca. 5. Il Receptionist/Medico seleziona dalla lista il risultato cercato.
Condizione di entrata	• Il Receptionist/Medico deve essere autenticato. • SHIVA necessita del riferimento di un paziente.
Condizione di uscita	• Il Receptionist/Medico seleziona uno dei risultati ottenuti dalla ricerca in SHIVA
Requisiti di qualità	• La ricerca deve produrre almeno un risultato. • Deve essere possibile connettersi al server di SHIVA. • Il server di SHIVA deve poter comunicare con il database.

3.5.2.2.7 Caso d'uso PrenotazioneVisita

Nome caso d'uso	PrenotazioneVisita
Attori partecipanti	(Inizializzato da) Receptionist DBMS
Flusso di eventi	1. SHIVA fornisce al Receptionist una vista contenente le tipologie di visite mediche offerte dalla struttura. 2. Il Receptionist seleziona una delle tipologie di visite. 3. SHIVA cerca all'interno le date in cui almeno un medico registrato nella struttura, con specializzazione uguale al tipo di visita scelto, non ha già prenotazioni visite assegnate tramite il DBMS. 4. SHIVA presenta al Receptionist una vista in cui sono mostrate le date disponibili. 5. Il Receptionist seleziona una delle date disponibili. 6. Il Receptionist effettua la ricerca del paziente, **include** (RicercaPaziente). 7. SHIVA presenta al Receptionist una vista di riepilogo in cui chiede conferma dell'operazione. 8. Il Receptionist conferma l'operazione. 9. SHIVA registra nel database, tramite il DBMS, la prenotazione e la associa al paziente e ad un medico con specializzazione uguale al tipo di visita scelto che non abbia già una prenotazione assegnata in quella data. 10. SHIVA invia al Receptionist un messaggio di prenotazione effettuata.
Condizione di entrata	• Il Receptionist deve essere autenticato. • Il Receptionist seleziona la voce *effettua prenotazione*.
Condizione di uscita	• La prenotazione è stata registrata con successo
Requisiti di qualità	• Deve essere possibile connettersi al server di SHIVA. • Il server di SHIVA deve poter comunicare con il database.

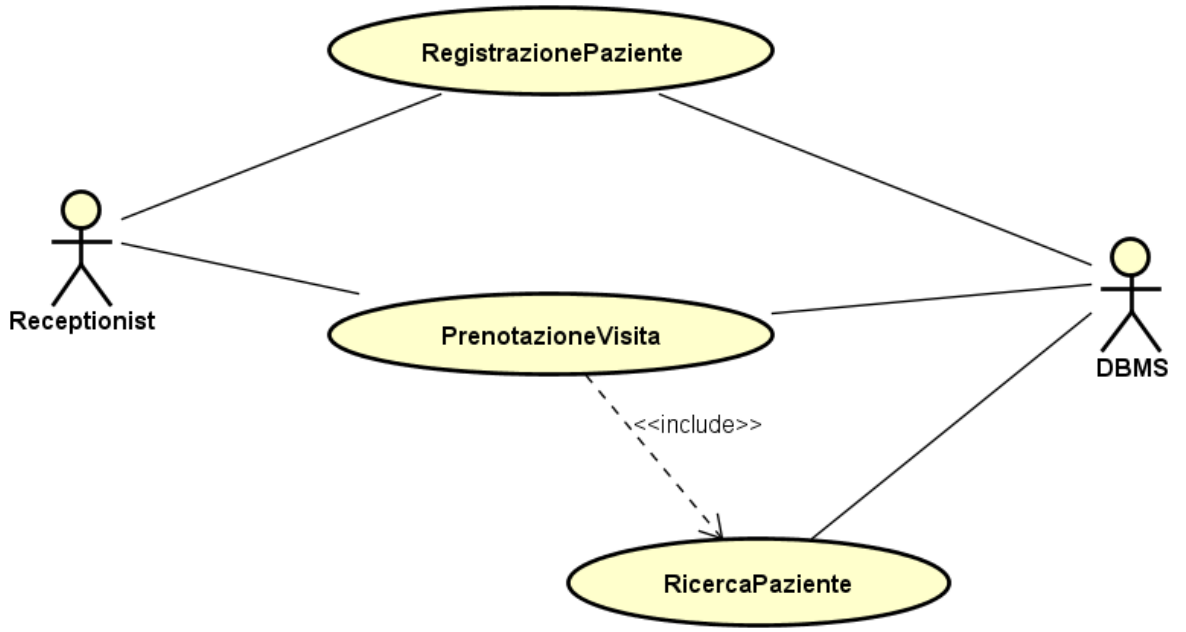


FIGURA 5 - DIAGRAMMA DEI CASI D'USO DEL RECEPTIONIST

3.5.2.2.8 Caso d'uso VisualizzaVisite

Nome caso d'uso	VisualizzaVisite
Attori partecipanti	(Inizializzato da) Medico DBMS
Flusso di eventi	1. SHIVA cerca nel database tutte le prenotazioni visite assegnate al Medico. 2. SHIVA interroga il DBMS per ottenere l'elenco delle visite da effettuare. 3. SHIVA mostra al medico una lista contenente le visite da effettuare.
Condizione di entrata	• Il Medico deve essere autenticato. • Il Medico seleziona la voce *visualizza visite*.
Condizione di uscita	• Il Medico ha visualizzato la lista delle visite da effettuare.
Requisiti di qualità	• Deve essere possibile connettersi al server di SHIVA. • Il server di SHIVA deve poter comunicare con il database.

3.5.2.2.9 Caso d'uso VisualizzaReport

Nome caso d'uso	VisualizzaReport
Attori partecipanti	(Inizializzato da) Medico DBMS
Flusso di eventi	1. Il Medico seleziona la voce *visualizza reports*. 2. Il Medico effettua la ricerca del paziente a cui è associato il report, **include** (RicercaPaziente). 3. SHIVA cerca tutti i report associati al paziente cercato tramite il DBMS. 4. SHIVA presenta al Medico una lista che contiene i reports associati al paziente. 5. Il Medico seleziona il report d'interesse dalla lista. 6. SHIVA presenta al Medico una vista in cui sono presenti i dettagli del report selezionato.
Condizione di entrata	• Il Medico deve essere autenticato. • Il Medico seleziona la voce *visualizza reports*.
Condizione di uscita	• Il Medico ha visualizzato la lista delle visite da effettuare.
Requisiti di qualità	• Deve essere possibile connettersi al server di SHIVA. • Il server di SHIVA deve poter comunicare con il database.

3.5.2.2.10 Caso d'uso InserimentoReport

Nome caso d'uso	InserimentoReport
Attori partecipanti	(Inizializzato da) Medico DBMS
Flusso di eventi	1. Il Medico effettua la ricerca del paziente a cui associare il report, **include** (RicercaPaziente). 2. SHIVA presenta al Medico una vista in cui può compilare il report. 3. Il Medico compila il report e allega gli eventuali file. 4. SHIVA registra il nuovo report e lo associa al paziente tramite il DBMS. 5. SHIVA presenta al Medico un messaggio di inserimento report effettuato con successo.
Condizione di entrata	• Il Medico è autenticato nel sistema. • Il Medico seleziona la voce *inserisci nuovo report*.
Condizione di uscita	• SHIVA registra il nuovo report e lo associa al paziente cercato.
Requisiti di qualità	• Deve essere possibile connettersi al server di SHIVA. • Il server di SHIVA deve poter comunicare con il database.

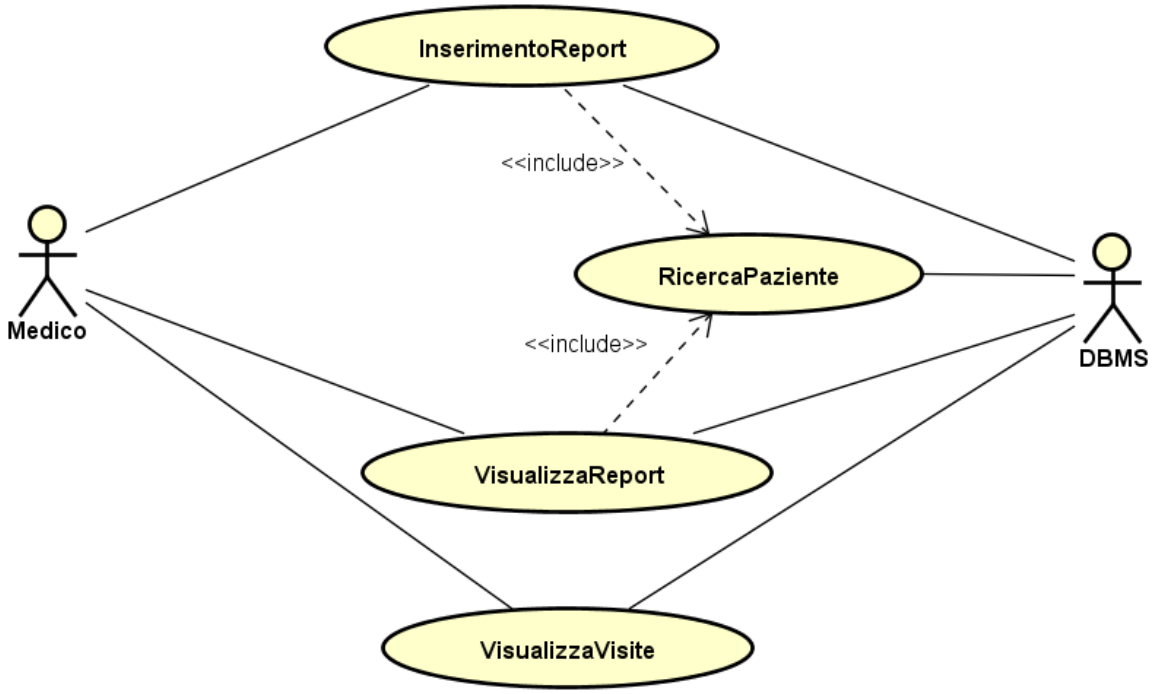


FIGURA 6 - DIAGRAMMA DEI CASI D'USO DEL MEDICO

3.5.2.2.11 Caso d'uso RegistrazioneMedico

Nome caso d'uso	RegistrazioneMedico
Attori partecipanti	(Inizializzato da) AmministratoreDiSisitema DBMS
Flusso di eventi	1. SHIVA presenta all'AmministratoreDiSistema la vista di registrazione di un nuovo medico. 2. L'AmministratoreDiSistema compila i campi nella vista inserendo i dati anagrafici, i recapiti e le specializzazioni del medico da registrare. 3. SHIVA presenta all'AmministratoreDiSistema un messaggio di richiesta di conferma della registrazione. 4. L'AmministratoreDiSistema conferma la registrazione. 5. SHIVA genera username e password per il Medico. 6. SHIVA registra il nuovo Medico tramite il DBMS. 7. SHIVA restituisce un messaggio di registrazione avvenuta con successo. Nel messaggio sono specificati l'username e la password generati dal sistema per l'account medico appena creato.
Condizione di entrata	• L'AmministratoreDiSistema è autenticato nel sistema. • L'AmministratoreDiSistema seleziona la voce *registra medico*.
Condizione di uscita	• SHIVA registra correttamente il nuovo account medico.
Requisiti di qualità	• Il codice fiscale inserito deve essere coerente con i dati anagrafici inseriti. • Non deve già essere presente in SHIVA un account medico con il codice fiscale inserito. • Deve essere possibile connettersi al server di SHIVA. • Il server di SHIVA deve poter comunicare con il database.

3.5.2.2.12 Caso d'uso RegistrazioneReceptionist

Nome caso d'uso	RegistrazioneReceptionist
Attori partecipanti	(Inizializzato da) AmministratoreDiSisitema DBMS
Flusso di eventi	1. SHIVA presenta all'AmministratoreDiSistema la vista di registrazione di un nuovo receptionist. 2. L'AmministratoreDiSistema compila i campi nella vista inserendo i dati anagrafici e i recapiti del receptionist da registrare. 3. SHIVA presenta all'AmministratoreDiSistema un messaggio di richiesta di conferma della registrazione. 4. L'AmministratoreDiSistema conferma la registrazione. 5. SHIVA genera username e password per il Receptionist. 6. SHIVA il nuovo Receptionist tramite il DBMS. 7. SHIVA restituisce un messaggio di registrazione avvenuta con successo. Nel messaggio sono specificati l'username e la password generati dal sistema per l'account receptionist appena creato.
Condizione di entrata	• L'AmministratoreDiSistema è autenticato nel sistema. • L'AmministratoreDiSistema seleziona la voce *registra receptionist*.
Condizione di uscita	• SHIVA registra correttamente il nuovo account receptionist.
Requisiti di qualità	• Il codice fiscale inserito deve essere coerente con i dati anagrafici inseriti. • Non deve già essere presente in SHIVA un account receptionist con il codice fiscale inserito. • SHIVA deve poter comunicare con il database.

3.5.2.2.13 Caso d'uso RicercaUtente

Nome caso d'uso	RicercaUtente
Attori partecipanti	AmministratoreDiSisitema DBMS
Flusso di eventi	1. SHIVA presenta all'AmministratoreDiSistema la vista di ricerca utente. 2. L'AmministratoreDiSistema compila i campi della vista e seleziona il tipo di utente che vuole cercare (Receptionist o Medico). 3. SHIVA cerca l'utente interrogando il DBMS. 4. SHIVA presenta all'AmministratoreDiSistema la lista dei risultati della ricerca. 5. L'AmministratoreDiSistema seleziona dalla lista il risultato cercato.
Condizione di entrata	• L'AmministratoreDiSistema è autenticato nel sistema. • L'AmministratoreDiSistema deve cercare un account utente.
Condizione di uscita	• L'AmministratoreDiSistema seleziona uno dei risultati ottenuti dalla ricerca in SHIVA.
Requisiti di qualità	• La ricerca deve produrre almeno un risultato. • SHIVA deve poter comunicare con il database.

3.5.2.2.14 Caso d'uso ModificaUtente

Nome caso d'uso	ModificaUtente
Attori partecipanti	(Inizializzato da) AmministratoreDiSisitema DBMS
Flusso di eventi	1. L'AmministratoreDiSistema effettua la ricerca dell'utente, **include** (RicercaUtente). 2. SHIVA presenta all'AmministratoreDiSistema la vista di modifica utente. 3. L'AmministratoreDiSistema effettua le modifiche desiderate. 4. SHIVA modifica l'utente tramite il DBMS. 5. SHIVA restituisce un messaggio di modifica effettuata con successo.
Condizione di entrata	• L'AmministratoreDiSistema è autenticato nel sistema. • L'AmministratoreDiSistema seleziona la voce *modifica utente*.
Condizione di uscita	• SHIVA registra le modifiche apportate all'account utente.
Requisiti di qualità	• SHIVA deve poter comunicare con il database.

3.5.2.2.15 Caso d'uso EliminaUtente

Nome caso d'uso	EliminaUtente
Attori partecipanti	(Inizializzato da) AmministratoreDiSisitema DBMS
Flusso di eventi	1. L'AmministratoreDiSistema effettua la ricerca dell'utente, **include** (RicercaUtente). 2. SHIVA presenta all'AmministratoreDiSistema una vista di riepilogo dell'utente selezionato. 3. L'AmministratoreDiSistema seleziona la voce *elimina*. 4. SHIVA presenta all'AmministratoreDiSistema un messaggio in cui si richiede conferma dell'eliminazione. 5. L'AmministratoreDiSistema conferma l'eliminazione. 6. SHIVA elimina l'utente tramite il DBMS. 7. SHIVA restituisce un messaggio di eliminazione avvenuta con successo.
Condizione di entrata	• L'AmministratoreDiSistema è autenticato nel sistema. • L'AmministratoreDiSistema seleziona la voce *elimina utente*.
Condizione di uscita	• SHIVA elimina l'account utente.
Requisiti di qualità	• SHIVA deve poter comunicare con il database.

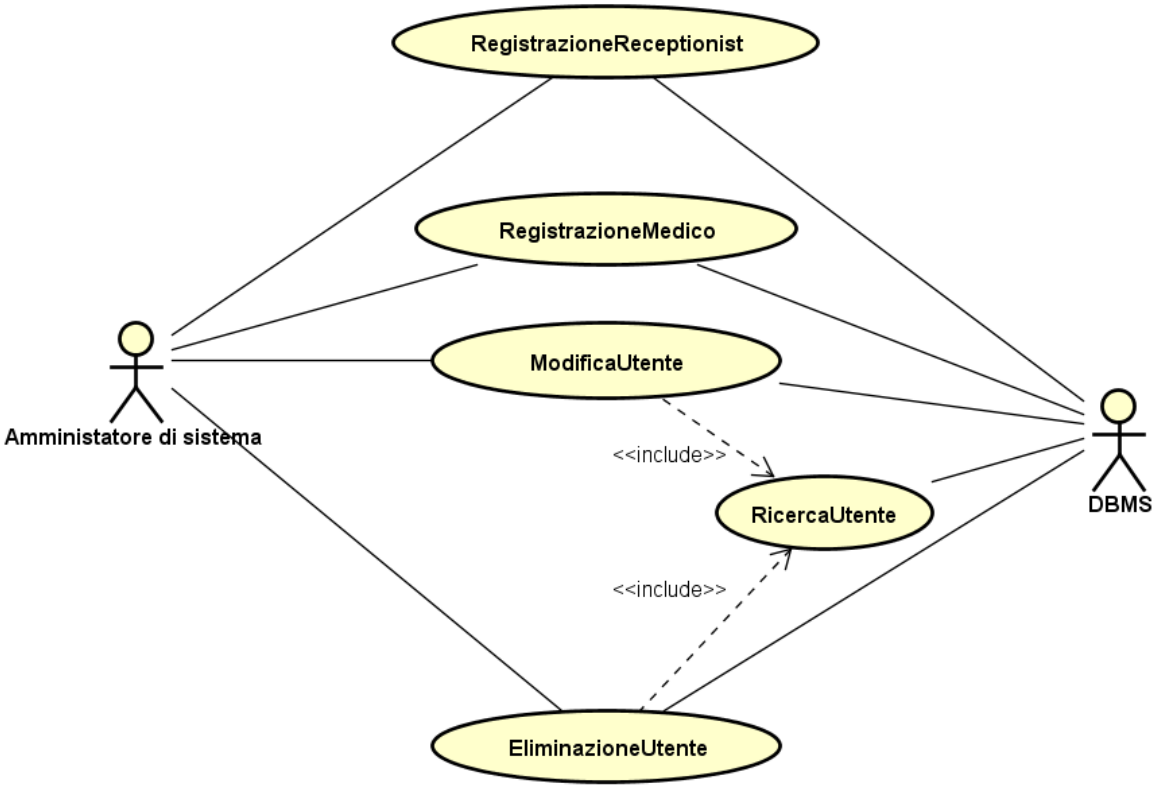


FIGURA 7 - DIAGRAMMA DEI CASI D'USO DELL'AMMINISTRATORE DI SISTEMA

3.5.3 Modelli degli oggetti

Tramite l'utilizzo della tecnica di Abbott sono state individuate le seguenti classi.

3.5.3.1 Dizionario dei dati

3.5.3.1.1 Entity Objects

Nome	Descrizione	Attributi	Metodi
Utente	Rappresenta un utente con un ruolo qualsiasi tra amministratore di sistema e receptionist.	• **username** • **password** • **anagrafica** • **ruolo:** campo che identifica il ruolo tra amministratore, receptionist e medico. • **recapiti:** lista dei recapiti (sia telefonici che email) • **password**	
Medico	Rappresenta un utente di tipo Medico. Eredita da Utente.	• **ruolo:** in Medico ruolo viene già settata a medico. • **specializzazione:** la specializzazione del Medico. • **listaPrenotazioni:** la lista delle prenotazioni a lui assegnate.	
Paziente	Rappresenta un paziente della struttura medica.	• **anagrafica** • **recapiti:** lista dei recapiti (sia telefonici che email)	
Report	Rappresenta il report associato a una visita e compilato dal medico.	• **dataVisita** • **titolo** • **descrizioneVisita** • **diagnosi** • **prescrizioniMediche** • **files :** lista di file serializzati in formato array di byte	

Prenotazione	Rappresenta le prenotazioni per le visite effettuate nella struttura.	• **dataVisita** • **tipoVisita :** corrisponde a una delle specializzazioni dei medici presenti nella struttura. • **paziente :** il paziente associato alla prenotazione. • **medico :** il paziente associato alla prenotazione.	
Anagrafica	Rappresenta i dati anagrafici di un utente o paziente.	• **nome** • **cognome** • **dataNascita** • **luogoNascita** • **sesso** • **cf:** il codice fiscale • **indirizzo:** l'indirizzo di residenza	

3.5.3.1.2 Boundary Objects

Nome	Descrizione	Attributi	Metodi
FormLoginBnd	Form utilizzato dall'Utente per inserire le credenziali di login.	• **username** • **password**	• **login:** inizializza il caso d'uso login
MsgErroreOperazioneBnd	Messaggio di errore con cui SHIVA comunica all'utente che l'operazione non è avvenuta con successo e specifica cause del fallimento.	• **descrizione Errore**	• **conferma :** conferma la lettura del messaggio.

ReceptionistMainWindowBnd	Vista principale presentata al Receptionist dopo il login. Gli permette di avviare le principali funzionalità.		• **logout :** permette di avviare l'operazione di logout • **modificaPassword :** permette di avviare l'operazione di modifica password • **registraPaziente :** permette di avviare l'operazione di registrazione paziente • **registraPrenotazione :** permette di avviare l'operazione di registra prenotazione
AmmMainWindowBnd	Vista principale presentata all'Amministratore dopo il login. Gli permette di avviare le principali funzionalità.		• **logout :** permette di avviare l'operazione di logout • **modificaPassword :** permette di avviare l'operazione di modifica password • **registraMedico :** permette di avviare l'operazione di registrazione medico. • **registraReceptionist :** permette di avviare l'operazione di registrazione receptionist • **modificaUtente :** permette di avviare

			l'operazione di modifica utente • **eliminaUtente :** permette di avviare l'operazione di eliminazione utente
MedicoMainWindowBnd	Vista principale presentata al Medico dopo il login. Gli permette di avviare le principali funzionalità.		• **logout :** permette di avviare • **modificaPassword :** permette di avviare l'operazione di modifica password • **visualizzaVisite :** permette di avviare l'operazione di visualizzazione visite • **visualizzaReport :** permette di avviare l'operazione di visualizzazione report • **inserisciReport :** permette di avviare l'operazione di inserimento report
FormModificaPassBnd	Form utilizzato dall'Utente per modificare la propria password.	• **vecchiaPassword** • **nuovaPassword**	• **modifica :** permette di inserire la vecchia password e nuova password
MsgSuccessoOperazioneBnd	Messaggio con cui SHIVA comunica all'utente il successo dell'operazione.	• **descrizione**	• **conferma :** conferma la lettura del messaggio

FormRegistraPazienteBnd	Form utilizzato dal Receptionist per registrare un paziente.	• **nome** • **cognome** • **dataNascita** • **luogoNascita** • **sesso** • **codiceFiscale** • **indirizzo** • **recapitoTel1** • **recapitoTel2** • **recapitoMail1** • **recapitoMail2**	• **registra :** permette di inserire i dati del nuovo paziente
MsgConfermaOperazioneBnd	Messaggio con cui SHIVA richiede la conferma dell'operazione da effettuare.	• **descrizione**	• **conferma** • **annulla**
FormRicercaPazienteBnd	Form utilizzato dal Receptionist/ Medico per effettuare la ricerca di un Paziente.	• **campoRicerca**	• **cercaNome :** richiede la ricerca per nome • **cercaCF :** richiede la ricerca per codice fiscale • **cercaCognome :** richiede la ricerca per cognome
RisultatiRicercaPazienteBnd	Vista con cui SHIVA comunica i risultati della ricerca e permette la selezione del paziente cercato.	• **listaPazienti** • **pazienteSelezionato**	• **selezionaPaziente** • **conferma**

FormRicercaUtenteBnd	Form utilizzato dall'Amministratore per effettuare la ricerca di un Receptionist o di un Medico.	• **campoRicerca** • **tipoUtente**	• **cercaNome :** richiede la ricerca per nome • **cercaCF :** richiede la ricerca per codice fiscale • **cercaCognome :** richiede la ricerca per cognome
RisultatiRicercaUtenteBnd	Vista con cui SHIVA comunica i risultati della ricerca e permette la selezione di un utente.	• **listaUtenti** • **utenteSelezionato**	• **selezionaUtente** • **conferma**
FormTipologiaVisiteBnd	Form utilizzato dall'Receptionist per selezionare il tipo di visita da prenotare.	• **listaTipologie** • **tipoSelezionato**	• **selezionaTipo** • **conferma**
FormSelezionaDataBnd	Form utilizzato dal Receptionist per selezionare una data per la prenotazione.	• **listaDate** • **dataSelezionata**	• **selezionaData** • **conferma**
RiepilogoPrenotazioneBnd	Vista con cui SHIVA mostra al Receptionist il riepilogo della prenotazione e ne chiede conferma.	• **tipoVisita** • **pazienteNome** • **pazienteCognome** • **mese** • **anno** • **giorno** • **medicoNome** • **medicoCognome**	• **conferma** • **annulla**

VisualizzaVisiteBnd	Una vista con cui SHIVA comunica al Medico le visite da effettuare.	• **listaVisite**	
VisualizzaListaReportBnd	Vista con cui SHIVA mostra al Medico la lista dei Report associati al Paziente precedentemente cercato e gli permette di selezionare il report a cui è interessato.	• **listaReport** • **reportSelezionato**	• **selezionaReport** • **conferma**
VisualizzaDettagliReportBnd	Vista con cui SHIVA mostra al Medico i dettagli del Report selezionato.	• **dataVisita** • **titolo** • **descrizioneVisita** • **diagnosi** • **prescrizioniMediche** • **files**	• **visualizzaReport**
FormReportBnd	Form utilizzato dal Medico per la compilazione del Report.	• **dataVisita** • **titolo** • **descrizioneVisita** • **diagnosi** • **prescrizioniMediche** • **files**	• **addFile** • **conferma**
FormRegistraMedicoBnd	Form utilizzato dall'AmministratoreDiSistema per la registrazione di un Medico.	• **nome** • **cognome** • **dataNascita** • **luogoNascita** • **sesso** • **codiceFiscale** • **indirizzoResidenza**	• **registra:** avvia la registrazione

		• **recapitoTel 1** • **recapitoTel 2** • **recapitoMa il1** • **recapitoMa il2** • **specializzaz ione**	
FormRegistraReceptio nistBnd	Form utilizzato dall'Amministr atoreDiSistem a per la registrazione di un Receptionist.	• **nome** • **cognome** • **dataNascita** • **luogoNascit a** • **sesso** • **codiceFiscal e** • **indirizzoRe sidenza** • **recapitoTel 1** • **recapitoTel 2** • **recapitoMa il1** • **recapitoMa il2**	• **registra:** avvia la registrazione
FormModificaUtenteB nd	Form utilizzato dall'Amministr atoreDiSistem a per la modifica di un Utente.	• **nome** • **cognome** • **dataNascita** • **luogoNascit a** • **sesso** • **codiceFiscal e** • **indirizzoRe sidenza** • **recapitoTel 1** • **recapitoTel 2** • **recapitoMa il1**	• **modifica**

		• **recapitoMail2**	
RiepilogoUtenteBnd	Vista con cui SHIVA mostra il riepilogo dell' Utente.	• **nome** • **cognome** • **dataNascita** • **luogoNascita** • **sesso** • **codiceFiscale** • **indirizzoResidenza** • **recapitoTel1** • **recapitoTel2** • **recapitoMail1** • **recapitoMail2**	• **conferma** • **annulla**
UtenteDAO	Si occupa di comunicare con il sistema che si occupa della persistenza effettiva degli oggetti Utente.		• **create** • **update** • **delete** • **findByNome** • **findByCognome** • **findByCF** • **findByUserPas**
PazienteDAO	Si occupa di comunicare con il sistema che si occupa della persistenza effettiva degli oggetti Paziente.		• **create** • **update** • **delete** • **findByNome** • **findByCognome** • **findByCF**
ReportDAO	Si occupa di comunicare		• **create** • **update**

	con il sistema che si occupa della persistenza effettiva degli oggetti Report.		• **delete** • **findById** • **findByTitolo** • **findByCFPaziente**
MedicoDAO	Si occupa di comunicare con il sistema che si occupa della persistenza effettiva degli oggetti Medico.		• **create** • **update** • **delete** • **findByNome** • **findByCognome** • **findByCF** • **findBySpec**
PrenotazioneDAO	Si occupa di comunicare con il sistema che si occupa della persistenza effettiva degli oggetti Prenotazione.		• **create** • **update** • **delete** • **findById** • **findByCFMedico** • **findDateByType** : trova le date libere dato un tipo di visita.

3.5.3.1.3 Control Objects

Nome	Descrizione	Attributi	Metodi
SessioneCtrl	Gestisce la sessione dell'utente, inclusi il processo di login e il processo di logout.	• **utenteLoggato** • **utenteDao** • **formLogin** • **mainWindow :** la finestra principale dell'utente autenticato	• **login** • **logout** • **init :** inizializza la sessione
ModificaPasswordCtrl	Gestisce la modifica della	• **vecchiaPassword**	• **modificaPass**

	password Utente.	• **nuovaPassword** • **formatoPassword** • **utenteDAO** • **messErrore** • **messOk** • **utente**	• **controllaFormatoPass :** controlla che il formato della nuova password sia corretto • **controllaPass :** controlla che la vecchia password sia corretta • **confermaModifica**
RegistraPazienteCtrl	Gestisce la registrazione di un nuovo Paziente.	• **PazienteDao** • **messErrore** • **messOk**	• **registraPaziente** • **confermaRegistrazione**
RicercaPazienteCtrl	Gestisce la ricerca di un Paziente.	• **formRicercaPaz** • **pazienteDao** • **paziente** • **listaPazientiTrovati** • **viewRisultati :** la vista con cui mostrerà i risultati • **stringaDiRicerca :** la stringa che verrà utilizzata per la ricerca	• **ricercaNome** • **ricercaCognome** • **ricercaCF** • **ricercaPaziente :** funzione che permette di offrire il servizio di ricerca ad altre control.
PrenotazioneVisiteCtrl	Gestisce la prenotazione della visita.	• **formTipoVisita:** form tramite cui viene selezionato il tipo della visita da prenotare • **prenotazioneDao** • **formSeleData :** form tramite cui viene	• **effettuaPrenotazione** • **confermaRiepilogo** • **registraPrenotazione** • **selezionaTipologia** • **selezionaData**

		selezionato la data per la prenotazione • **listaDateUtili** • **dataSelezionata** • **viewRiepilogo :** vista tramite cui viene mostrato il riepilogo della prenotazione • **messErrore** • **messOk** • **paziente:** paziente associato alla visita • **tipoVisita**	
VisualizzaVisiteCtrl	Gestisce la visualizzazione delle visite.	• **prenotazioneDao** • **utente :** l'utente autenticato	• **visualizzaVisite**
VisualizzaReportCtrl	Gestisce la visualizzazione dei report di un Paziente.	• **paziente** • **reportDao** • **viewListaReport** • **viewReport** • **listaReport** • **report**	• **visualizzaReport**
InserimentoReportCtrl	Gestisce l'inserimento di un nuovo report e lo associa al paziente.	• **paziente** • **formReport** • **reportDao** • **messErrore** • **messOk**	• **inserisciReport** • **getDatiReport :** metodo con cui vengono ricevuti i campi del record da comporre.
RegistraMedicoCtrl	Gestisce la registrazione di un nuovo Medico.	• **formRegMedico** • **medicoDao** • **messErrore** • **messOk** • **medico**	• **generatePassword:** genera la password del nuovo medico in

		• **password**	maniera casuale. • **registraMedico** • **confermaRegistrazione**
RegistraReceptionistCtrl	Gestisce la registrazione di un nuovo Receptionist.	• **formRegRec** • **receptionistDao** • **receptionist** • **password** • **messErrore** • **messOk**	• **generatePassword:** genera la password del nuovo receptionist in maniera casuale. • **registraReceptionist** • **confermaRegistrazione**
RicercaUtenteCtrl	Gestisce la ricerca di un Utente nel database.	• **formRicUtente** • **utenteDao** • **viewListaUtenti** • **listaUtenti** • **utente** • **tipoUtente**	• **ricercaUtente** • **ricercaCF:** ricerca tutti gli utenti per codice fiscale • **ricercaNome:** ricerca tutti gli utenti per nome • **ricercaCognome:** ricerca tutti gli utenti per cognome • **riceviSelezioneMedico:** per ricevere l'utente selezionato dalla lista dei risultati • **ricercaCfTipo:** ricerca un tipo particolare di utente per codice fiscale

			• **ricercaCognomeTipo:** ricerca un tipo particolare di utente per cognome • **ricercaNomeTio:** ricerca un tipo particolare di utente per nome • **riceviSelezioneUtente**
ModificaUtenteCtrl	Gestisce la modifica di un Utente nel database.	• **utente** • **formModUtente** • **utenteDao** • **nuovoIndirizzo** • **nuovaCitta** • **nuoviRecapiti** • **messErrore** • **messOk**	• **modificaDati** • **modificaUtente**
EliminaUtenteCtrl	Gestisce l'eliminazione di un Utente dal sistema.	• **utente** • **viewRiepilogo** • **utenteDao** • **messErrore** • **messOk**	• **confermaRiepilogo** • **confermaElimina** • **eliminaUtente**

3.5.3.2 Diagrammi delle classi

Qualora non specificate, le molteplicità delle associazioni sono da intendersi *uno a uno*.

3.5.3.2.1 Diagramma delle classi Entità

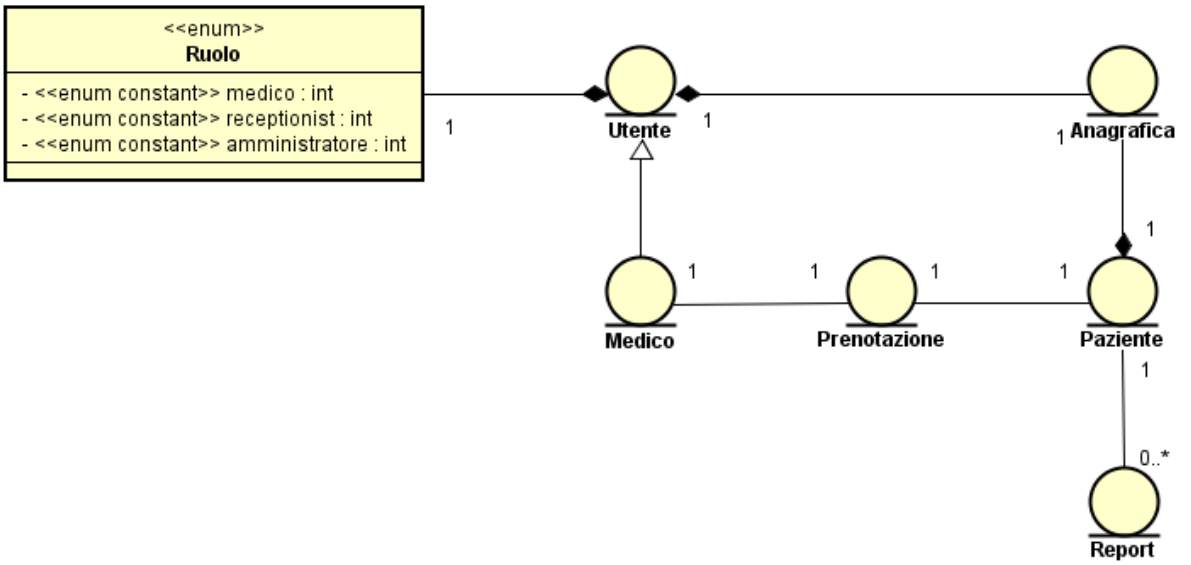


FIGURA 8 - DIAGRAMMA DELLE CLASSI ENTITÀ

3.5.3.2.2 Diagramma delle classi DAO

FIGURA 9 - DIAGRAMMA DELLE CLASSI DAO

3.5.3.2.3 Diagramma delle classi relative alle funzionalità Amministratore

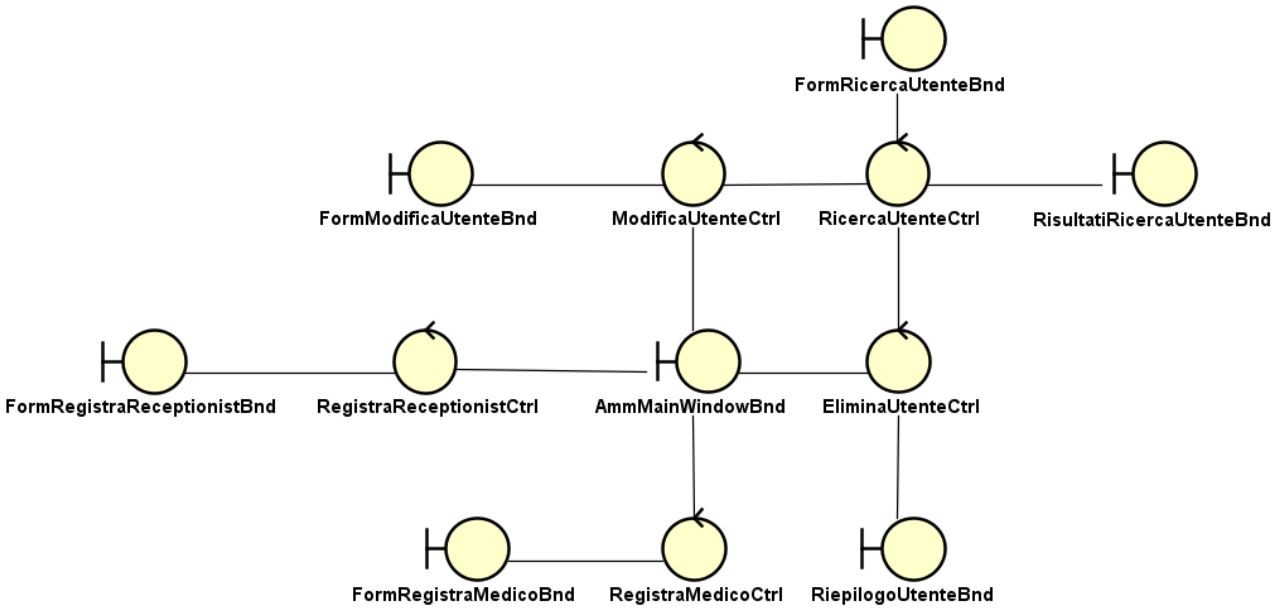


FIGURA 10 - DIAGRAMMA DELLE CLASSI RELATIVE ALLE FUNZIONALITÀ AMMINISTRATORE

3.5.3.2.4 Diagramma delle classi relative alle funzionalità Medico

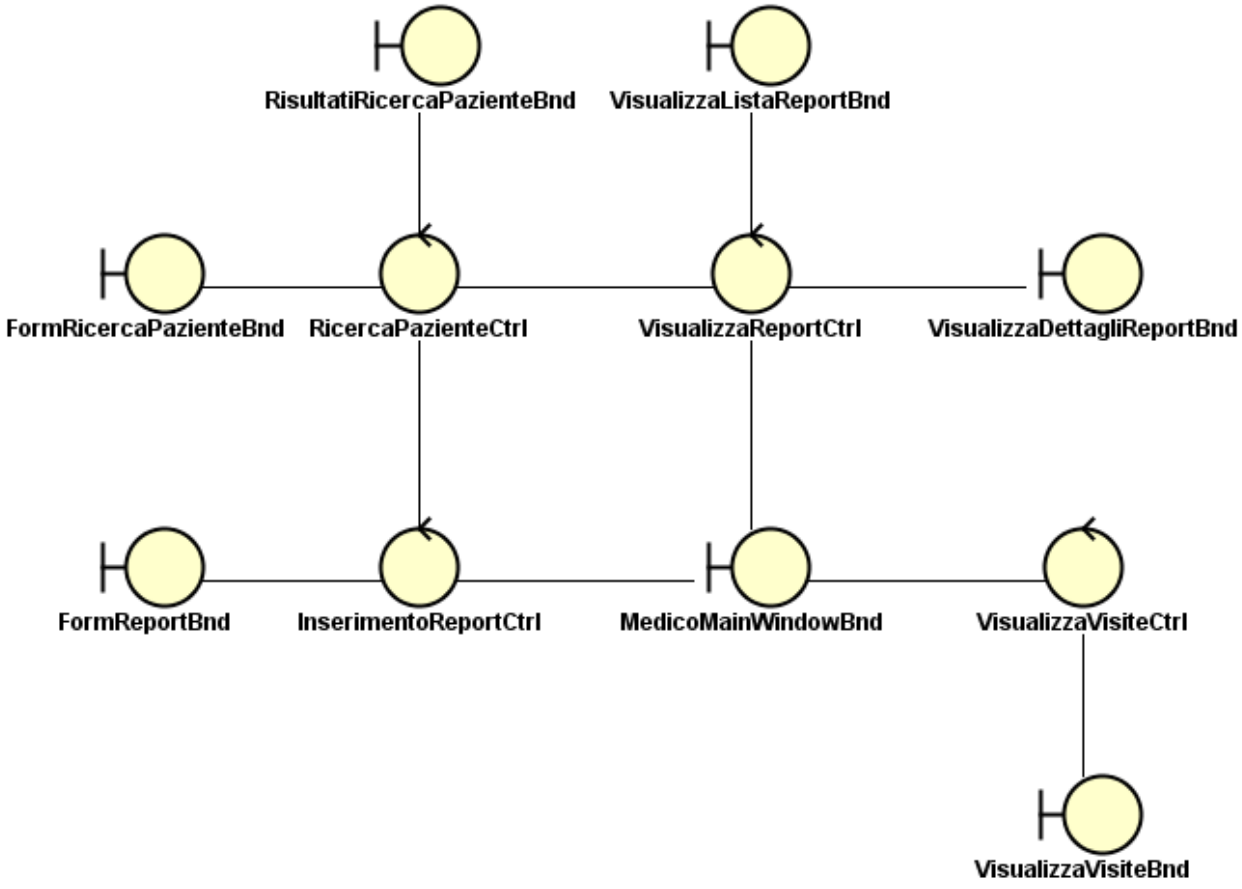


FIGURA 11 - DIAGRAMMA DELLE CLASSI RELATIVE ALLE FUNZIONALITÀ MEDICO

3.5.3.2.5 Diagramma delle classi relative alle funzionalità Receptionist

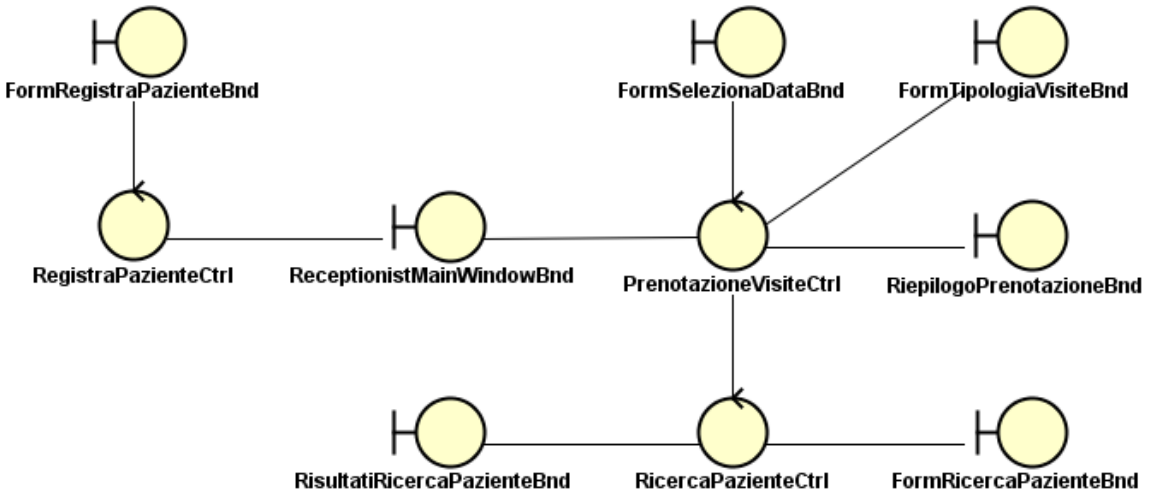


FIGURA 12 - DIAGRAMMA DELLE CLASSI RELATIVE ALLE FUNZIONALITÀ RECEPTIONIST

3.5.3.2.5 Diagramma delle classi relative alla sessione Utente

Queste sono le classi che vengono utilizzate per gestire la sessione di un Utente all'interno di SHIVA.

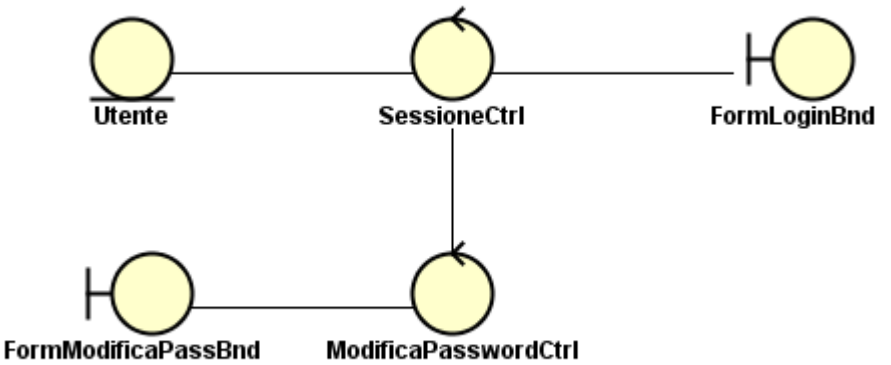


FIGURA 13 - DIAGRAMMA DELLE CLASSI RELATIVE ALLA SESSIONE UTENTE

3.5.3.2.5 Diagramma delle classi Messaggi

Queste sono le classi relative ai messaggi con cui SHIVA comunica varie informazioni all'utente.

FIGURA 14 - DIAGRAMMA DELLE CLASSI MESSAGGI

3.5.4 Progetto del Database

SHIVA necessita di una base di dati per salvare le informazioni inerenti gli utenti, i pazienti, le prenotazioni e i report. A tal fine è stato utilizzato un DBMS relazionale.

3.5.4.1 Progetto Logico

In seguito il diagramma ER del database:

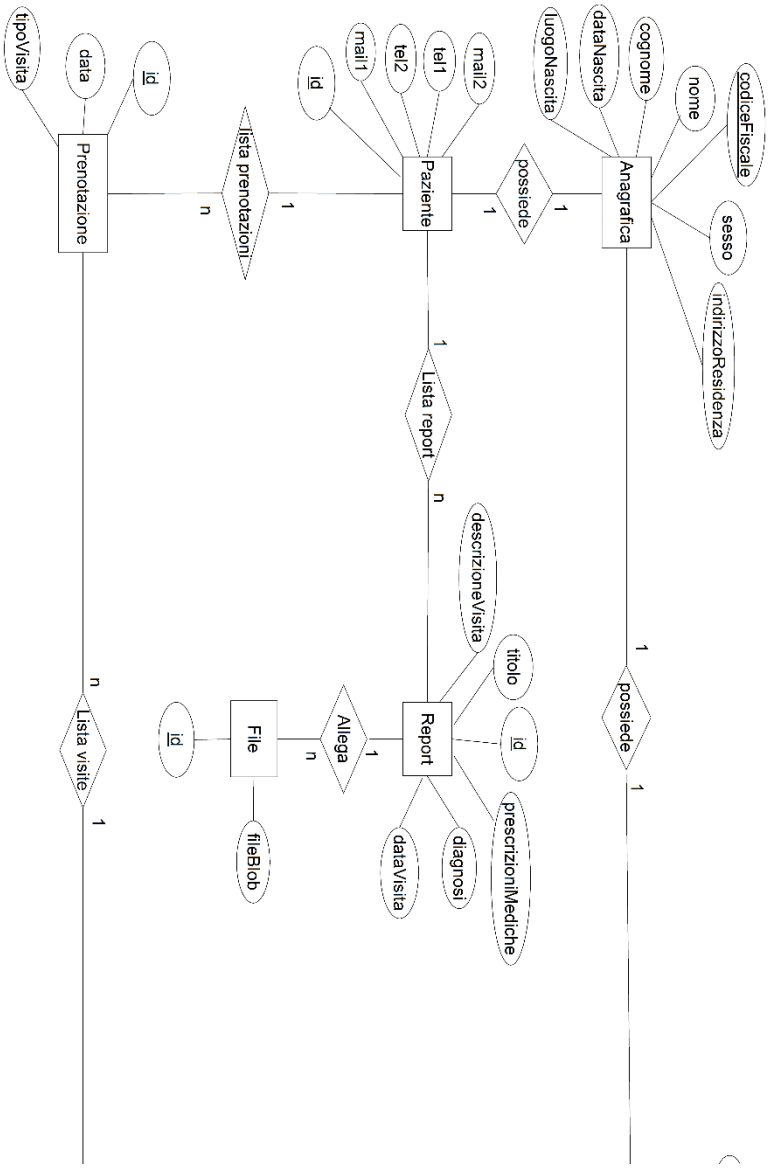


FIGURA 15 - DIAGRAMMA ER GENERALE DEL DATABASE

3.5.4.2 Descrizione Tabelle

3.5.4.2.1 Anagrafica

La seguente tabella descrive la struttura dei dati Anagrafica :

Nome	*Tipo*	*Chiave Primaria*	*Nullo*
codiceFiscale	Varchar(16)	Si	No
nome	Varchar (40)	No	No
cognome	Varchar (40)	No	No
sesso	Char(1)	No	No
dataNascita	Data	No	No
luogoNascita	Varchar (50)	No	No
indirizzoResidenza	Varchar (50)	No	No

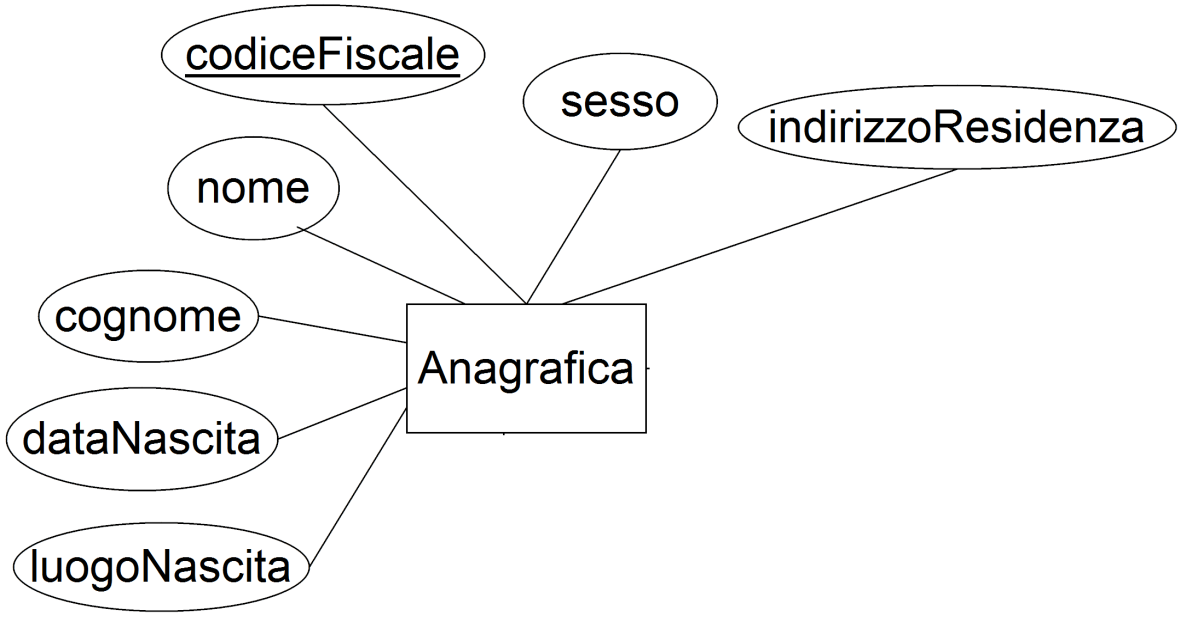


FIGURA 16 - DIAGRAMMA ER ANAGRAFICA

3.5.4.2.2 Paziente

La seguente tabella descrive la struttura dei dati Paziente:

Nome	*Tipo*	*Chiave Primaria*	*Nullo*
id	int	Si	No
tel1	Varchar (30)	No	No
tel2	Varchar (30)	No	Si
mail1	Varchar (30)	No	No
mail2	Varchar (30)	No	Si

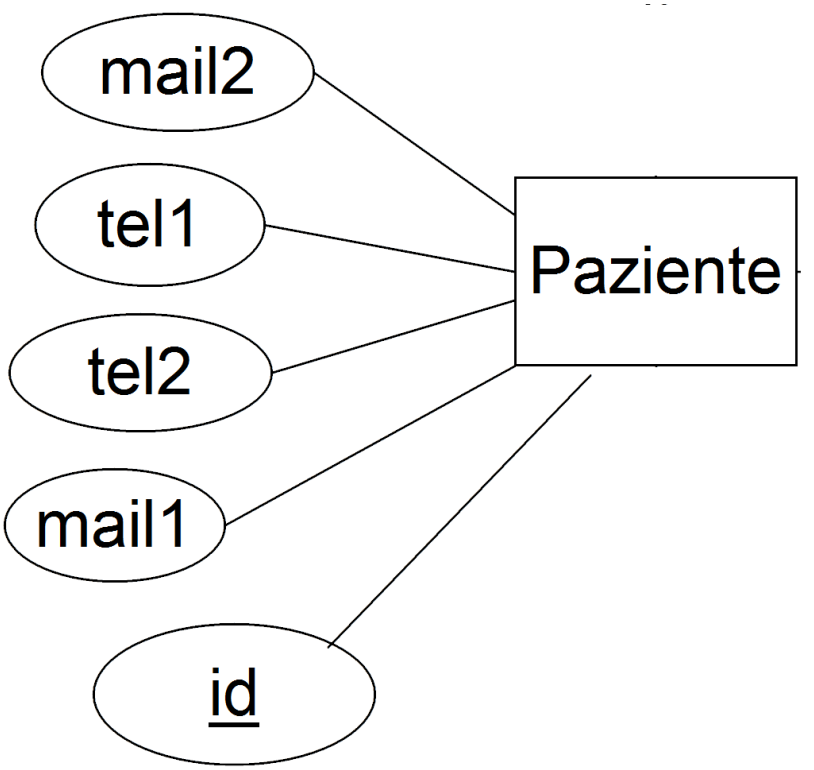


FIGURA 17 - DIAGRAMMA ER PAZIENTE

3.5.4.2.3 Prenotazione

La seguente tabella descrive la struttura dei dati Prenotazione:

Nome	*Tipo*	*Chiave Primaria*	*Nullo*
id	int	Si	No
data	Data	No	No
tipoVisita	Varchar (30)	No	No

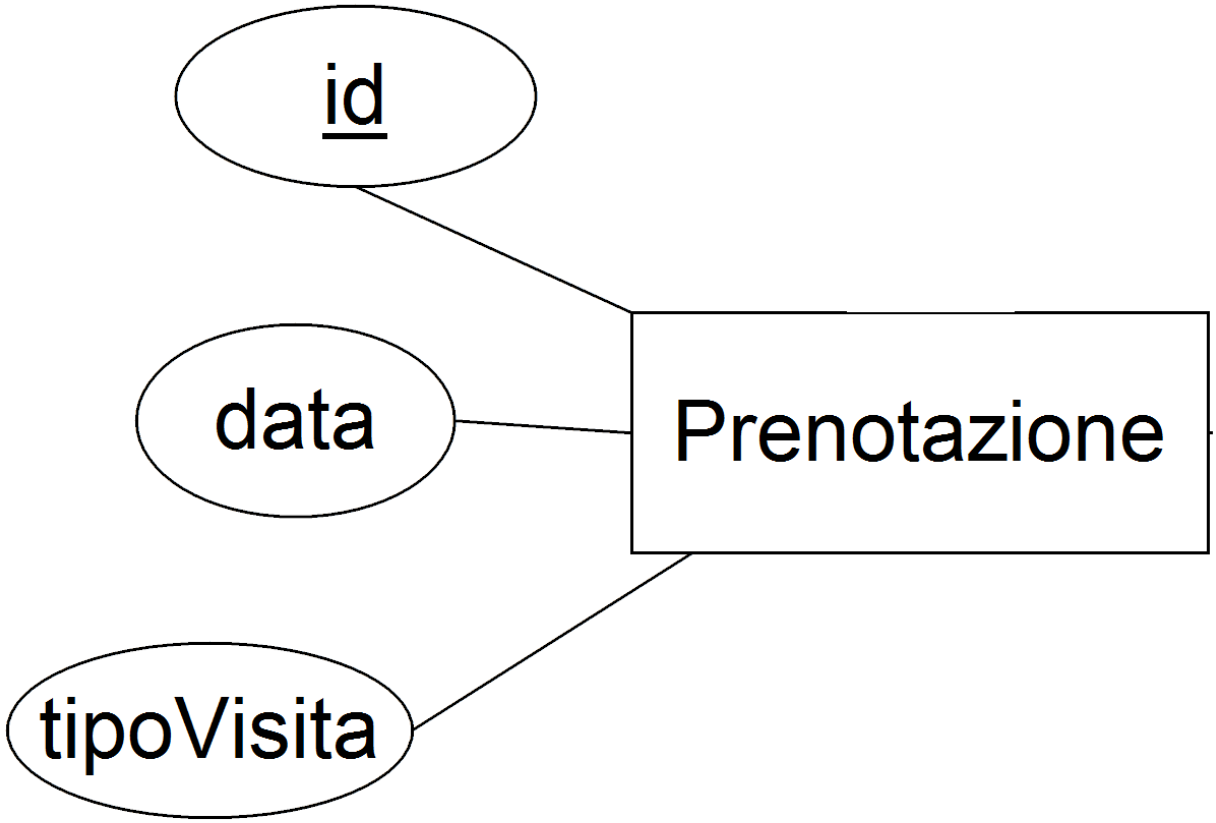


FIGURA 18 - DIAGRAMMA ER PRENOTAZIONE

3.5.4.2.4 Report

La seguente tabella descrive la struttura dei dati Report:

Nome	*Tipo*	*Chiave Primaria*	*Nullo*
id	int	Si	No
titolo	Varchar (30)	No	No
descrizioneVisita	Varchar (200)	No	No
prescrizioniMediche	Varchar (200)	No	No
diagnosi	Varchar (200)	No	No
dataVisita	Data	No	No

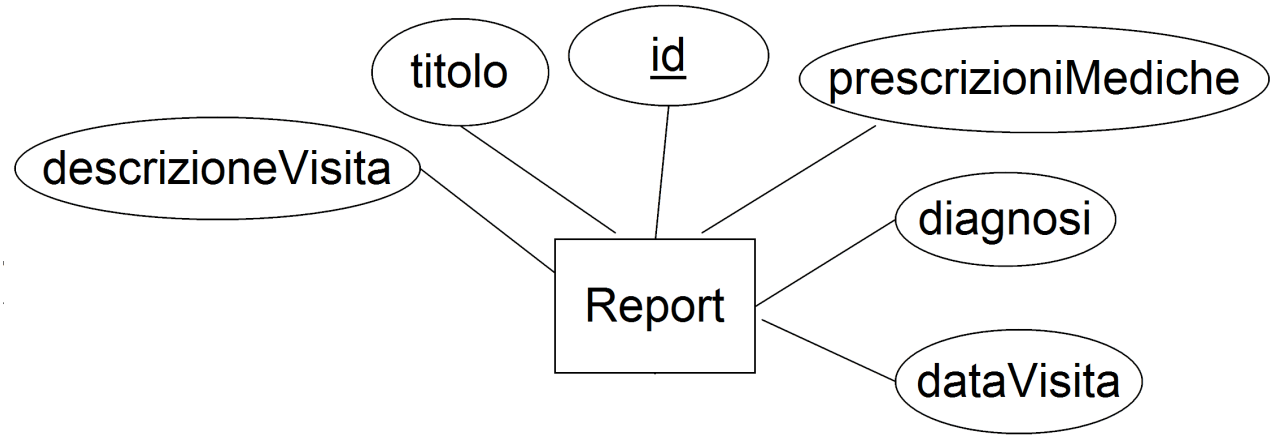


FIGURA 19 - DIAGRAMMA ER REPORT

3.5.4.2.5 File

La seguente tabella descrive la struttura dei dati File:

Nome	*Tipo*	*Chiave Primaria*	*Nullo*
id	int	Si	No
fileBlob	Blob	No	No

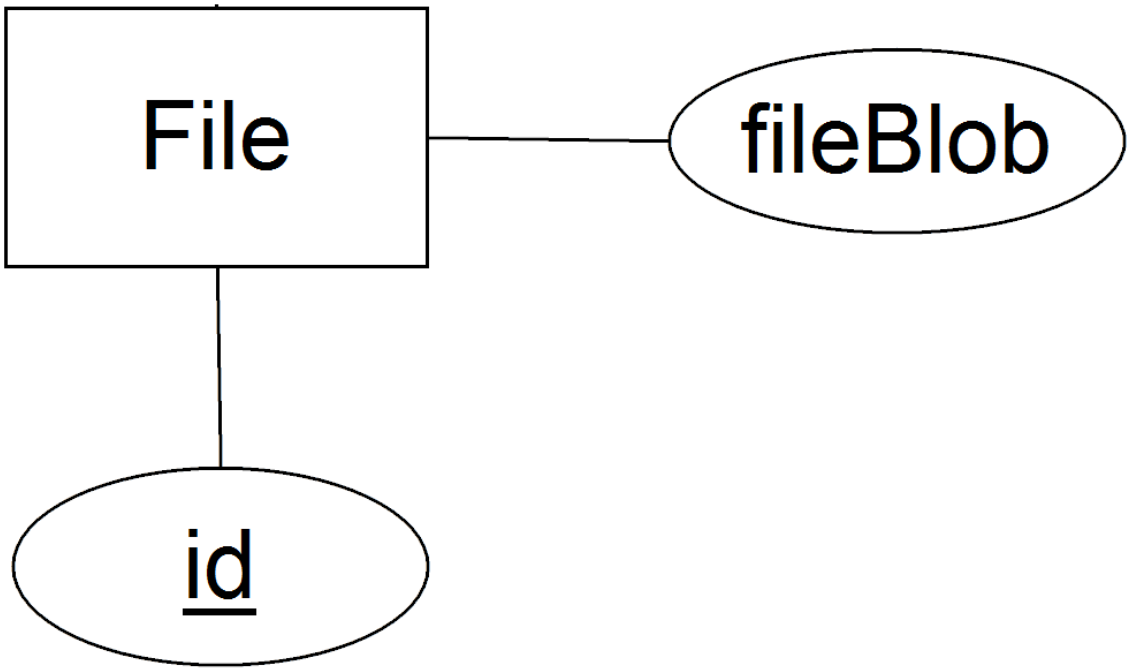


FIGURA 20 - DIAGRAMMA ER FILE

3.5.4.2.6 Utente

La seguente tabella descrive la struttura dei dati Utente:

Nome	*Tipo*	*Chiave Primaria*	*Nullo*
username	Varchar (30)	Si	No
tel1	Varchar (30)	No	No
tel2	Varchar (30)	No	Si
mail1	Varchar (30)	No	No
mail2	Varchar (30)	No	Si
password	Varchar (30)	No	No
ruolo	Varchar (30)	No	No

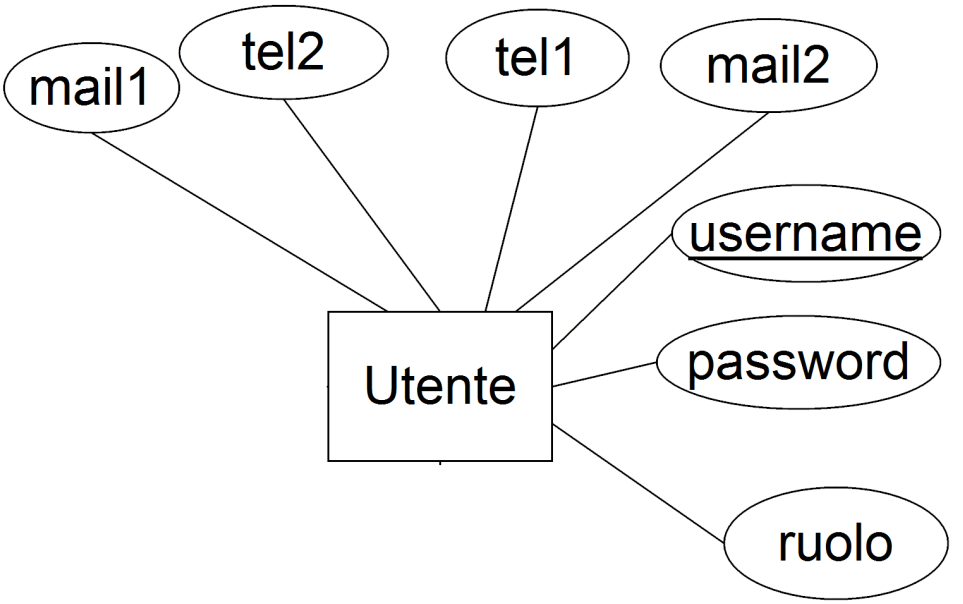


FIGURA 21 - DIAGRAMMA ER UTENTE

3.5.4.2.7 Medico

La seguente tabella descrive la struttura dei dati Medico:

Nome	*Tipo*	*Chiave Primaria*	*Nullo*
id	int	Si	No
specializzazione	Varchar (50)	No	No

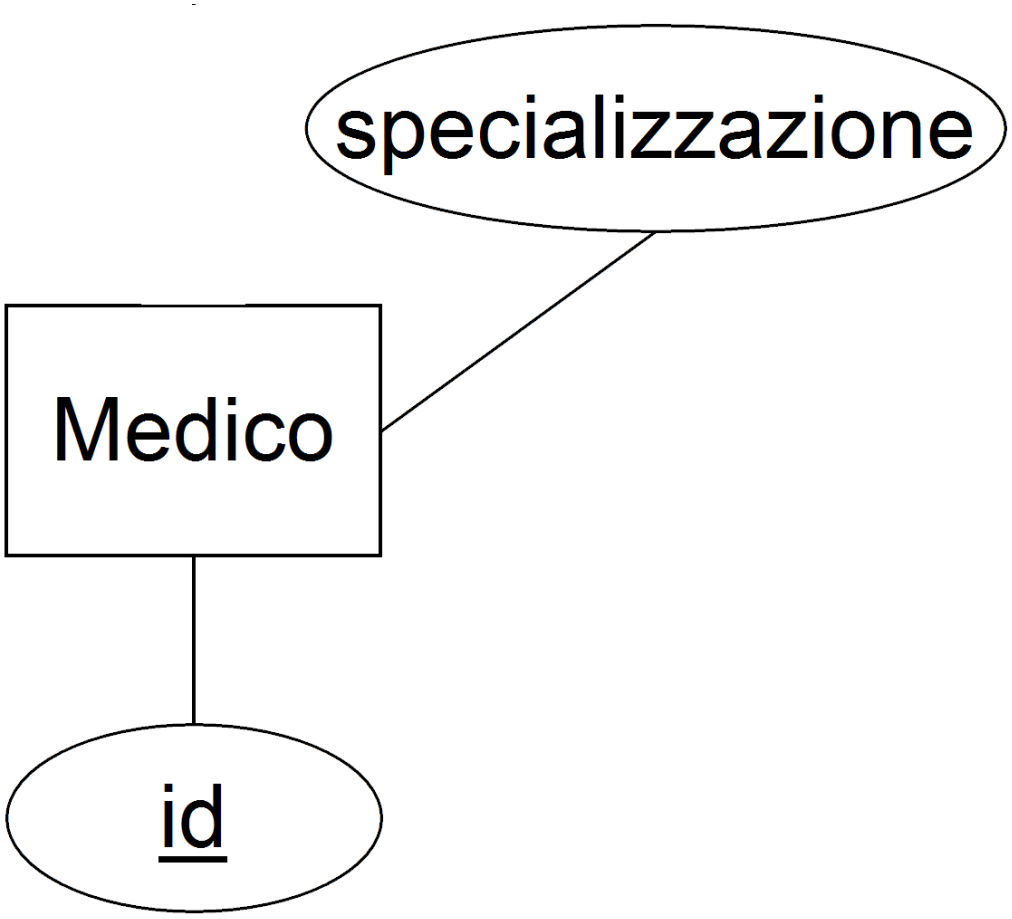


FIGURA 22 - DIAGRAMMA ER MEDICO

3.5.5 Modello Dinamico

3.5.5.1 Diagrammi di sequenza

3.5.5.1.1 Diagramma di sequenza EliminaUtente

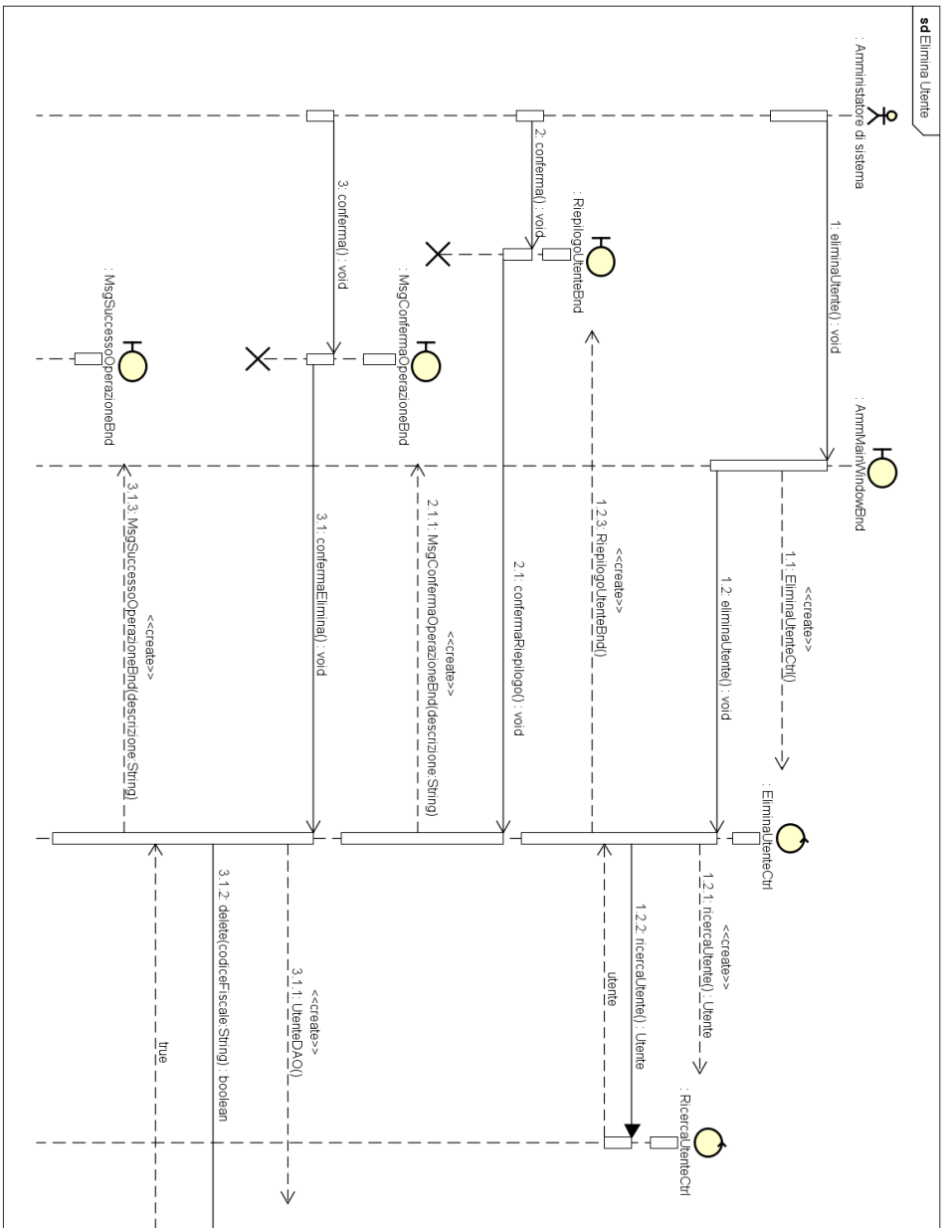


FIGURA 23 - DIAGRAMMA DI SEQUENZA ELIMINAUTENTE

3.5.5.1.2 Diagramma di sequenza FormatoPasswordErrato

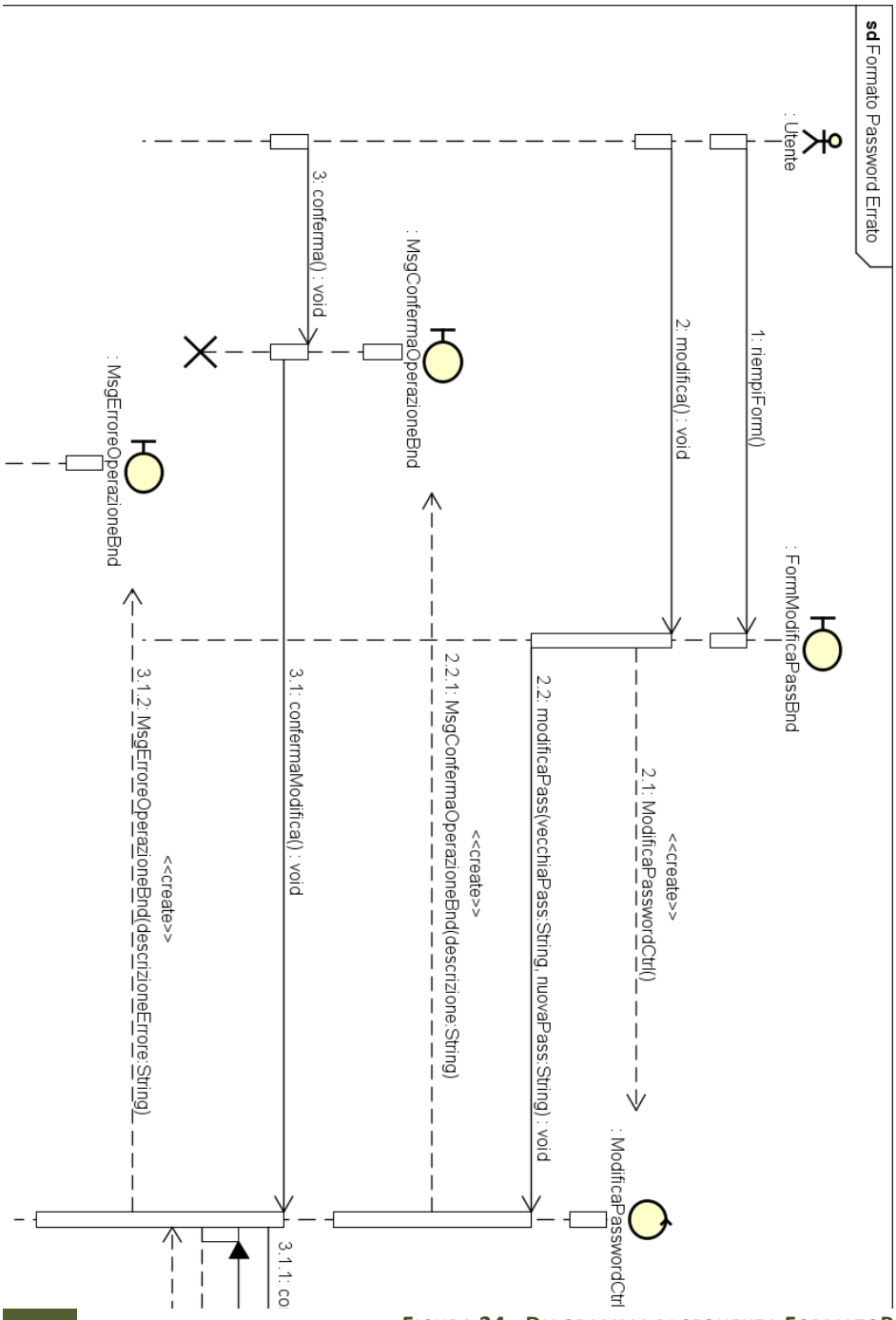


FIGURA 24 - DIAGRAMMA DI SEQUENZA FORMATOPASSWORDERRATO

3.5.5.1.3 Diagramma di sequenza InserimentoReport

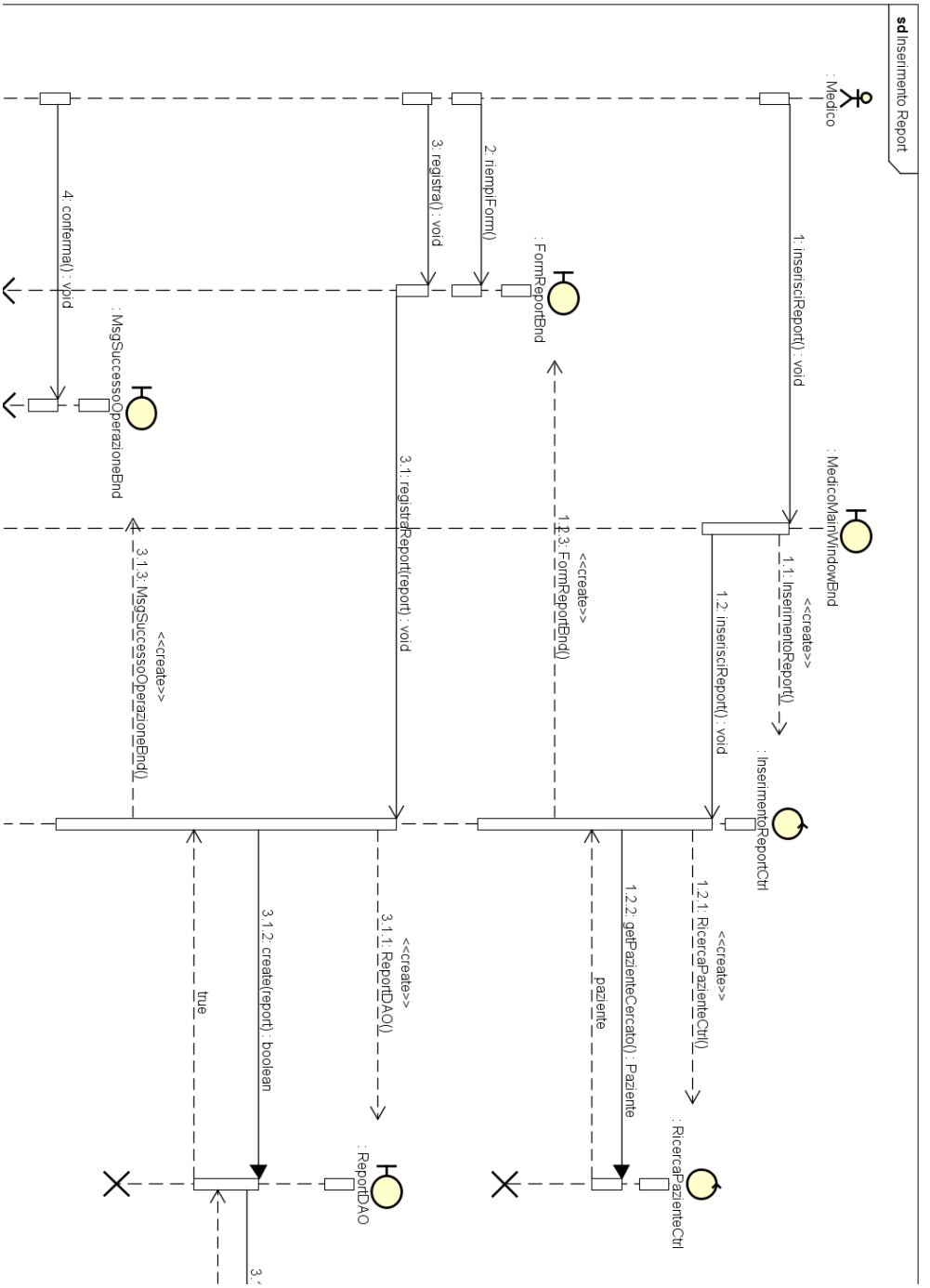


FIGURA 25 - DIAGRAMMA DI SEQUENZA INSERIMENTOREPORT

3.5.5.1.4 Diagramma di sequenza Login

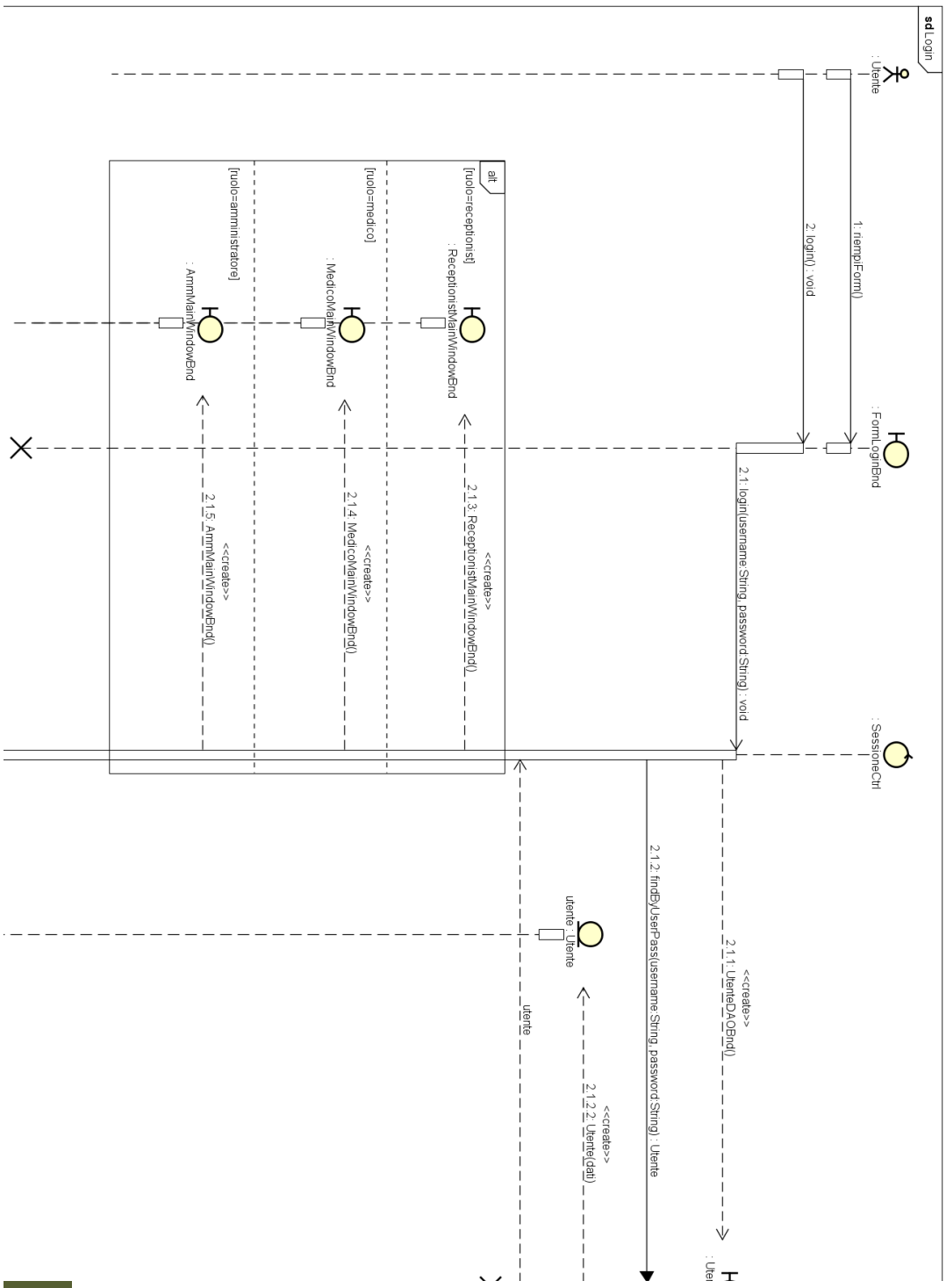


FIGURA 26 - DIAGRAMMA DI SEQUENZA LOGIN

3.5.5.1.5 Diagramma di sequenza LoginErrato

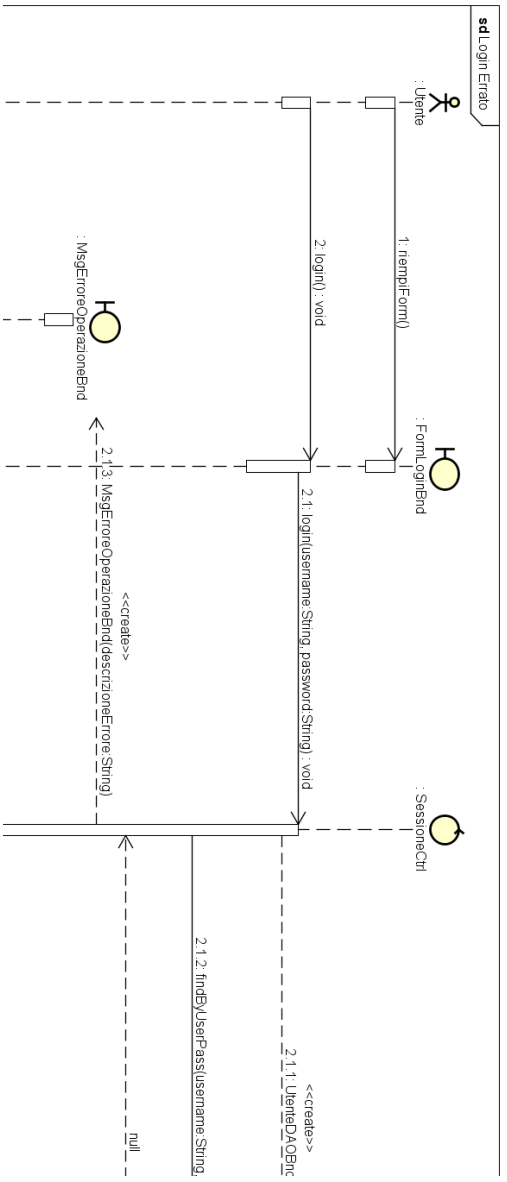


FIGURA 27 - DIAGRAMMA DI SEQUENZA LOGINERRATO

3.5.5.1.6 Diagramma di sequenza ModificaPassword

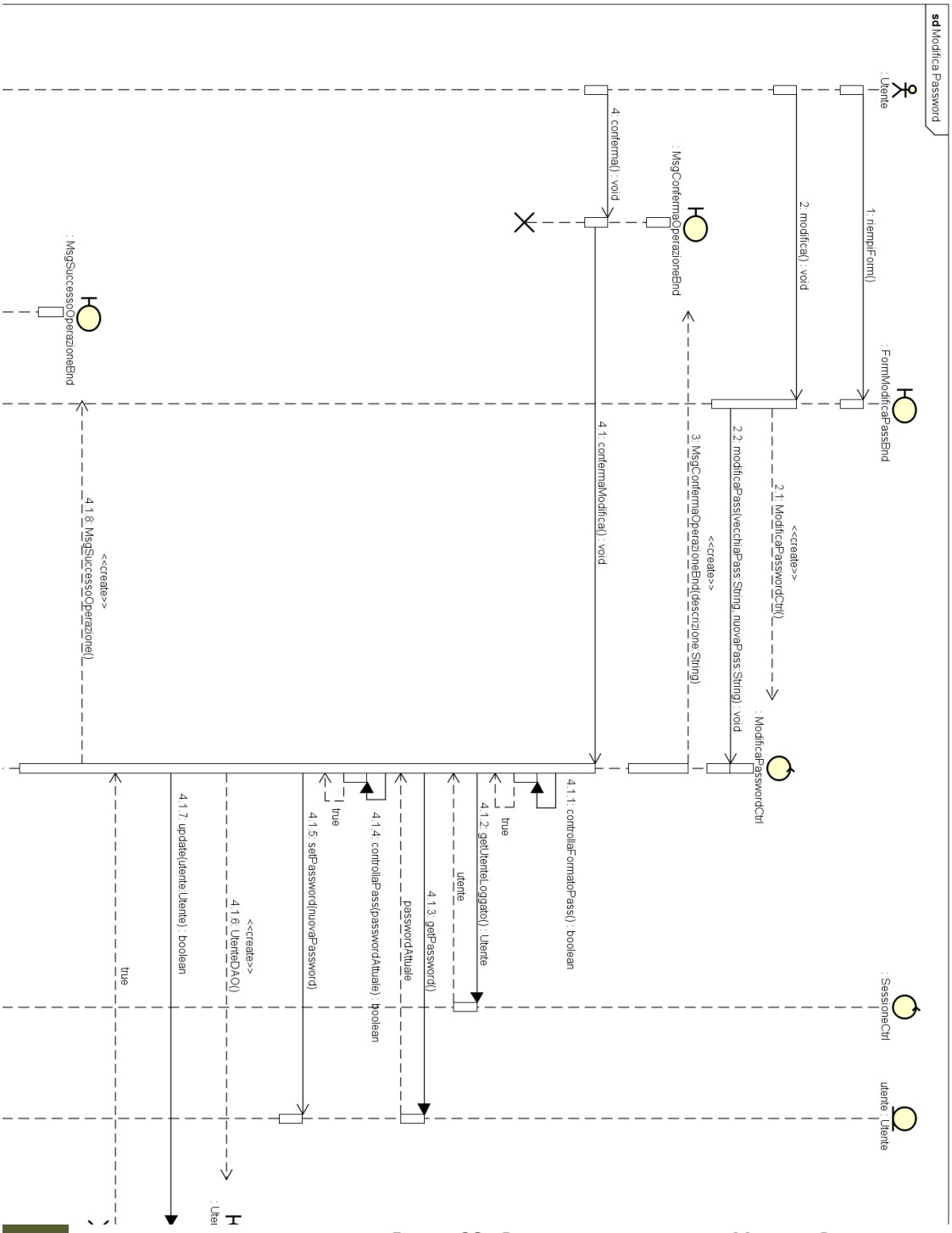


FIGURA 28 - DIAGRAMMA DI SEQUENZA MODIFICAPASSWORD

3.5.5.1.7 Diagramma di sequenza ModificaUtente

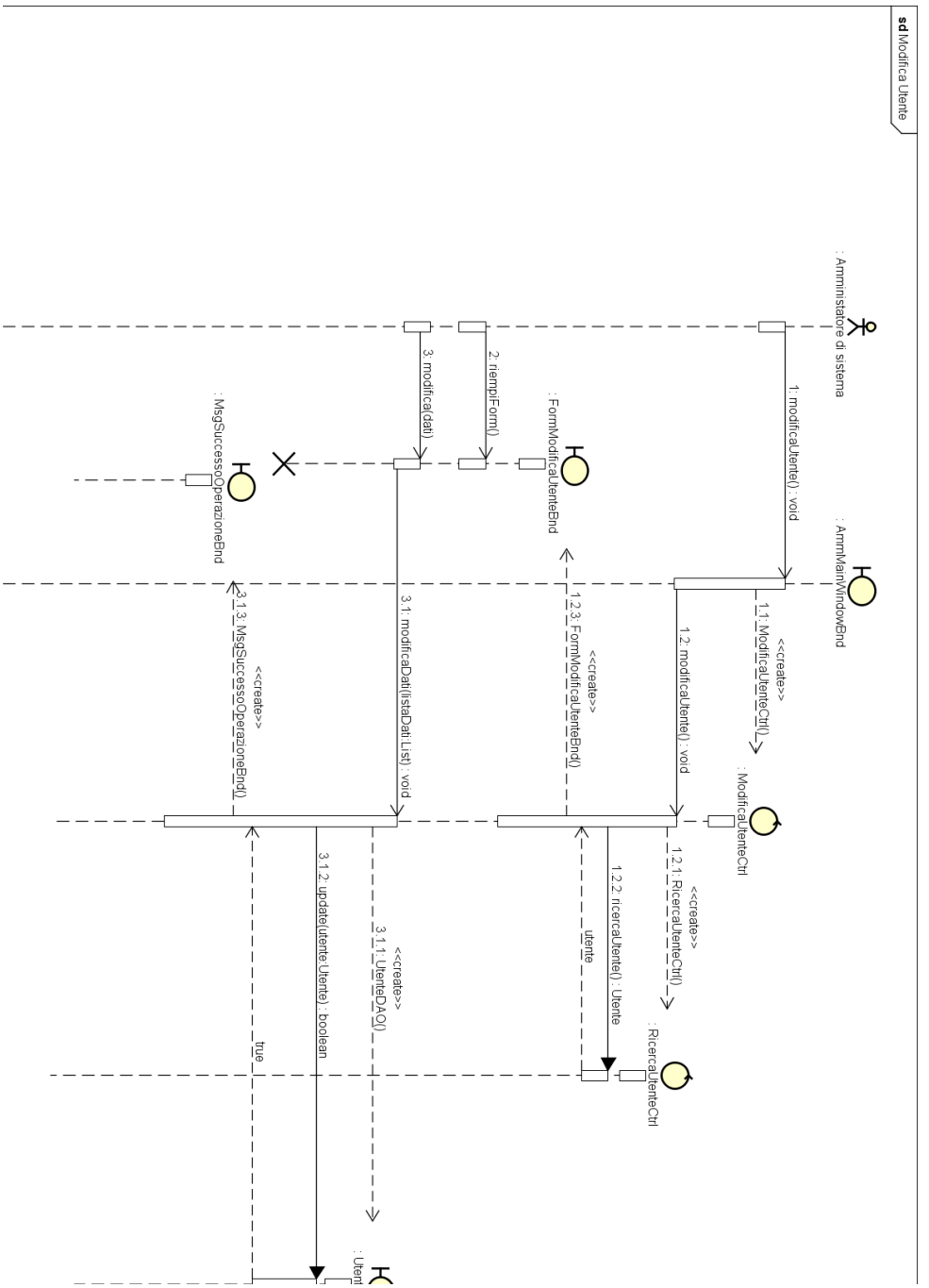


FIGURA 29 - DIAGRAMMA DI SEQUENZA MODIFICAUTENTE

3.5.5.1.8 Diagramma di sequenza PrenotazioneVisite

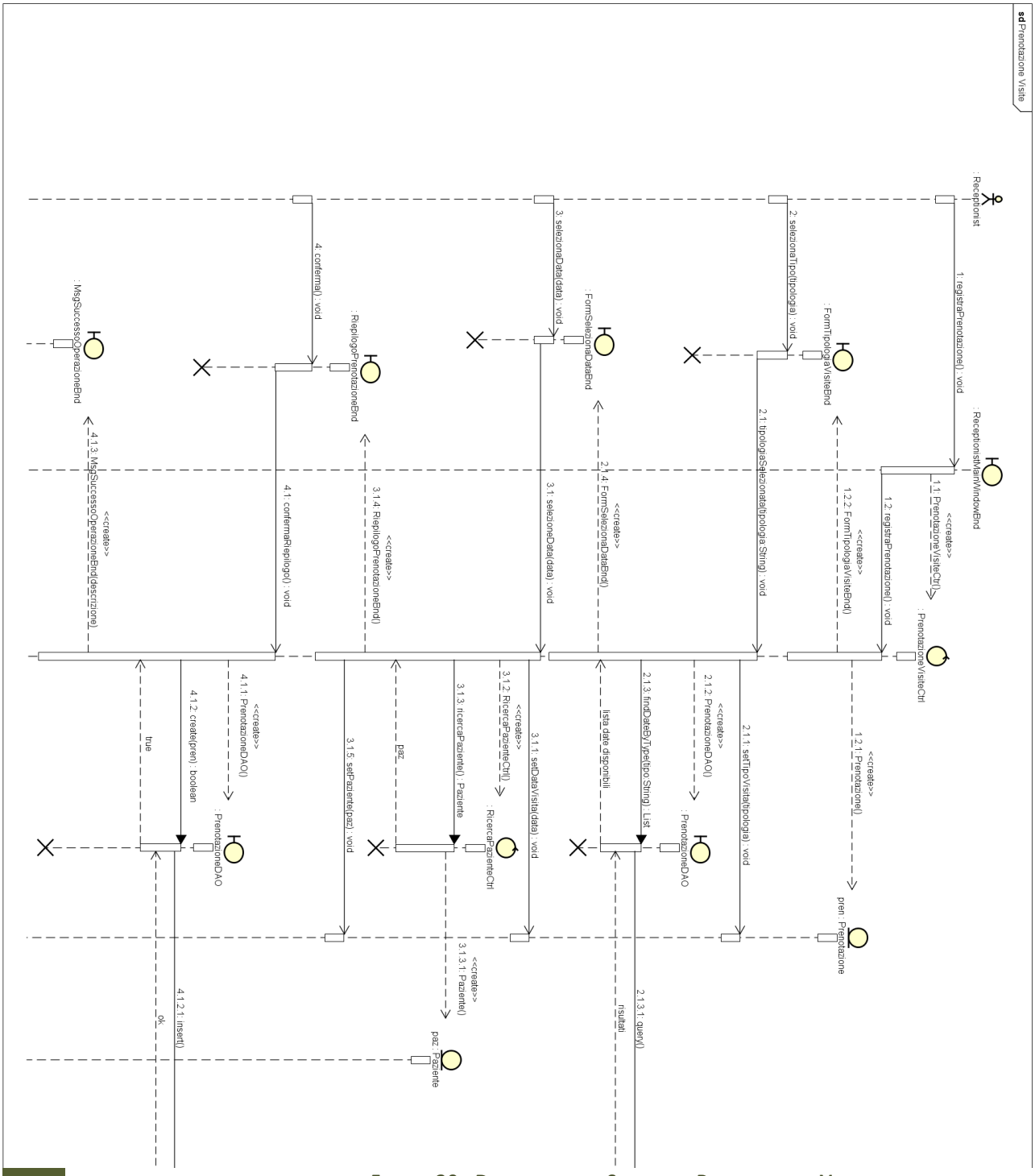


FIGURA 30 - DIAGRAMMA DI SEQUENZA PRENOTAZIONEVISITE

3.5.5.1.9 Diagramma di sequenza RegistrazioneMedico

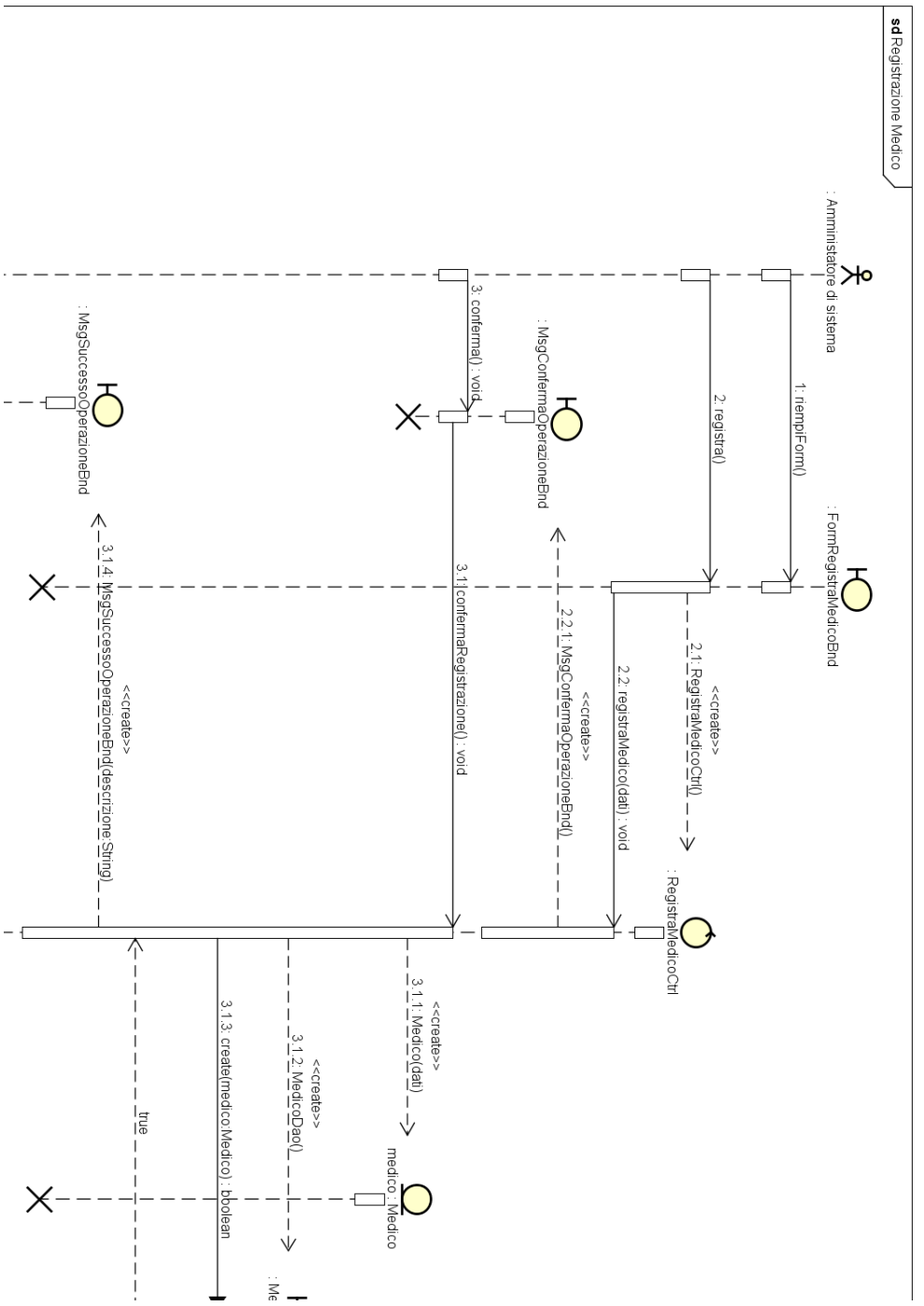


FIGURA 31 - DIAGRAMMA DI SEQUENZA REGISTRAZIONEMEDICO

3.5.5.1.10 Diagramma di sequenza RegistrazionePaziente

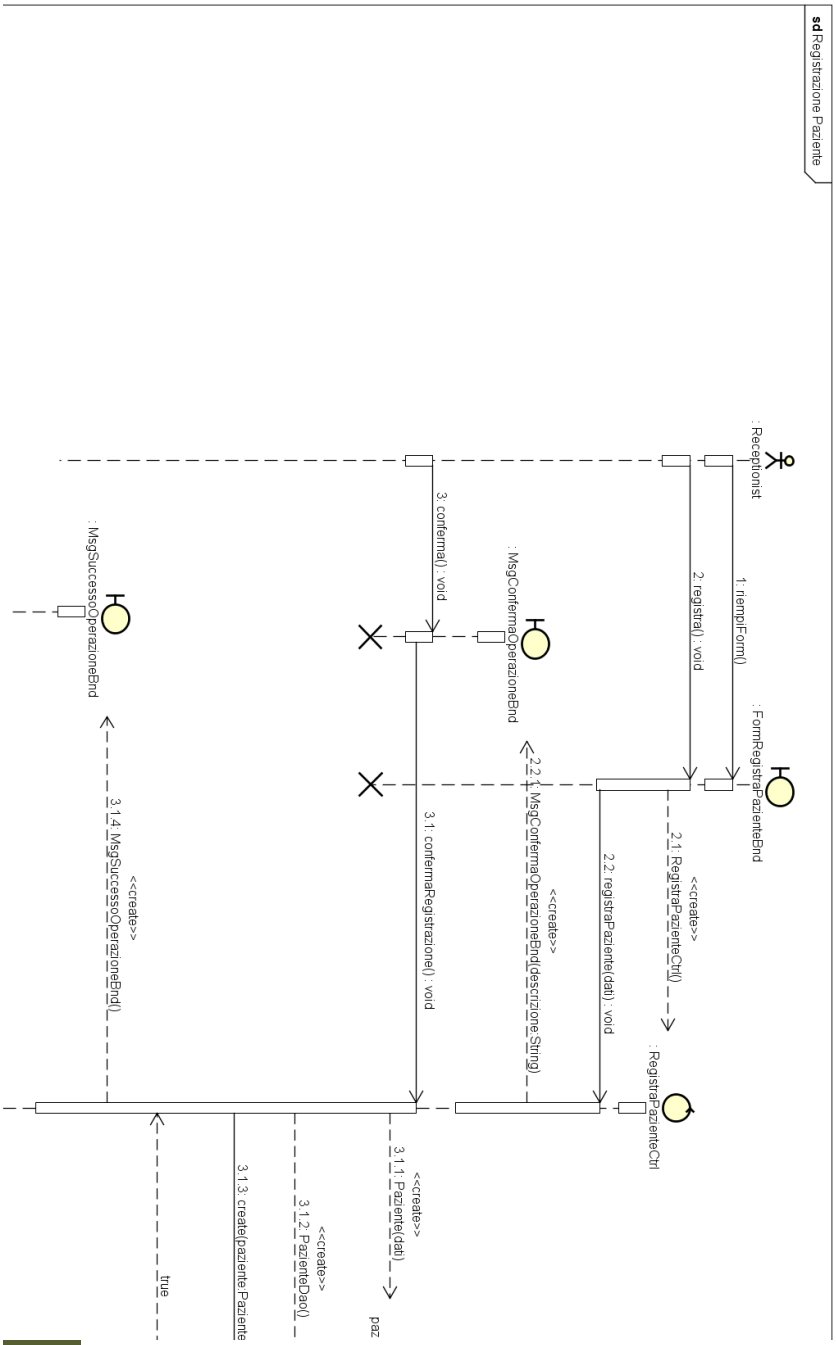


FIGURA 32 - DIAGRAMMA DI SEQUENZA REGISTRAZIONEPAZIENTE

3.5.5.1.11 Diagramma di sequenza RegistrazioneReceptionist

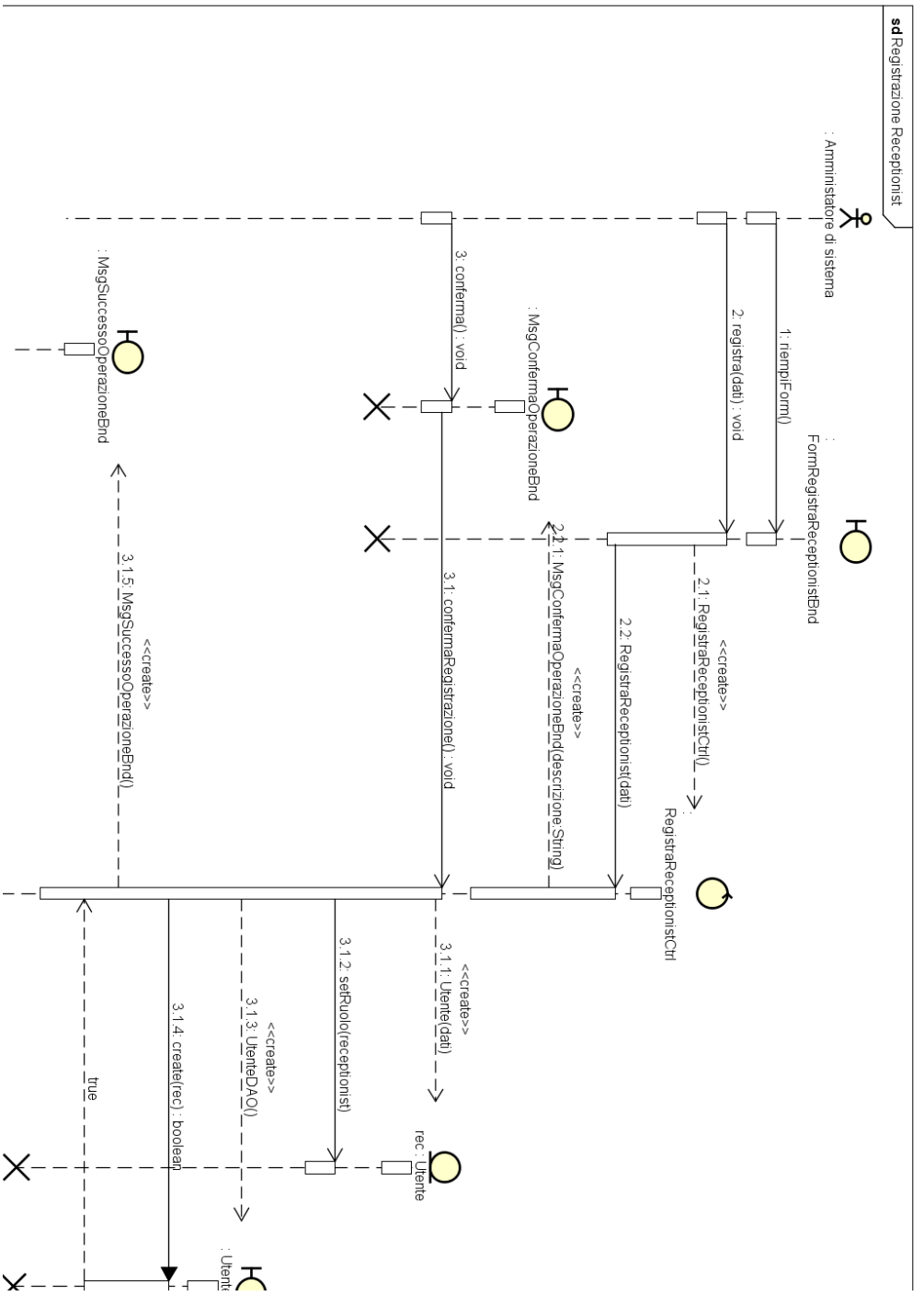


FIGURA 33 - DIAGRAMMA DI SEQUENZA REGISTRAZIONERECEPTIONIST

3.5.5.1.12 Diagramma di sequenza RicercaPazienteCF

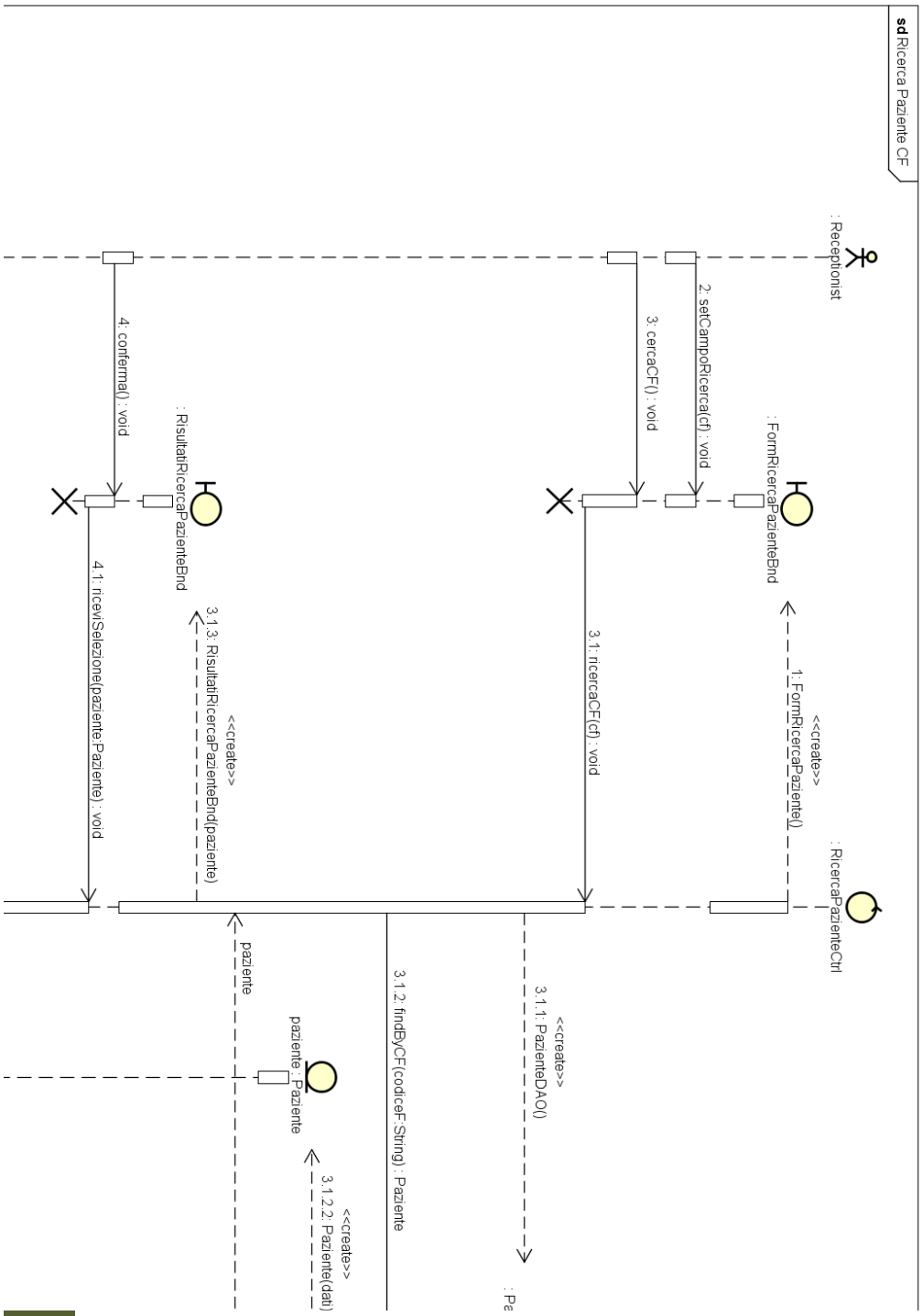


FIGURA 34 - DIAGRAMMA DI SEQUENZA RICERCAPAZIENTECF

3.5.5.1.13 Diagramma di sequenza RicercaPazienteCognome

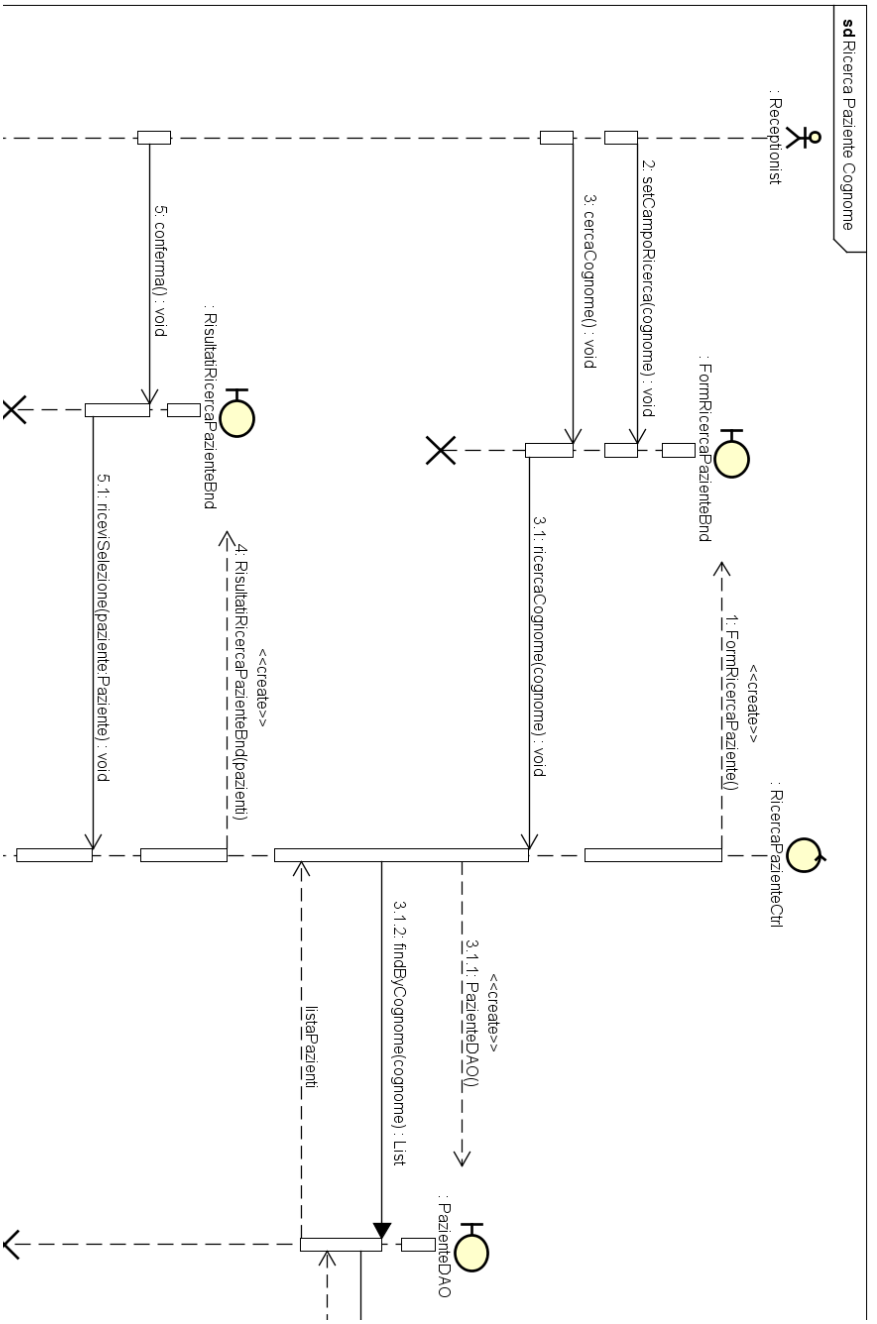


FIGURA 35 - DIAGRAMMA DI SEQUENZA RICERCAPAZIENTECOGNOME

3.5.5.1.14 Diagramma di sequenza RicercaPazienteNome

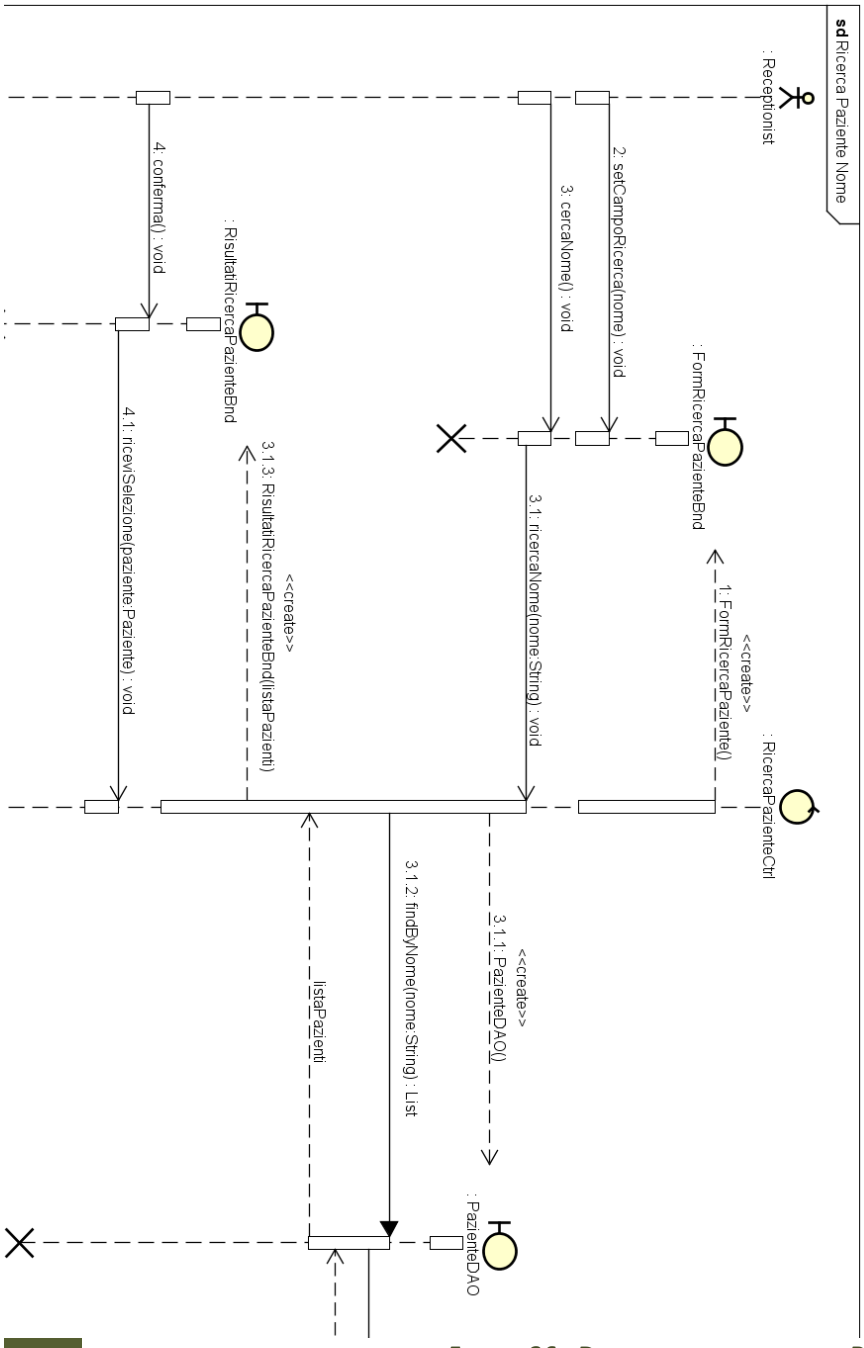


FIGURA 36 - DIAGRAMMA DI SEQUENZA RICERCAPAZIENTENOME

3.5.5.1.15 Diagramma di sequenza RicercaUtenteMedicoCF

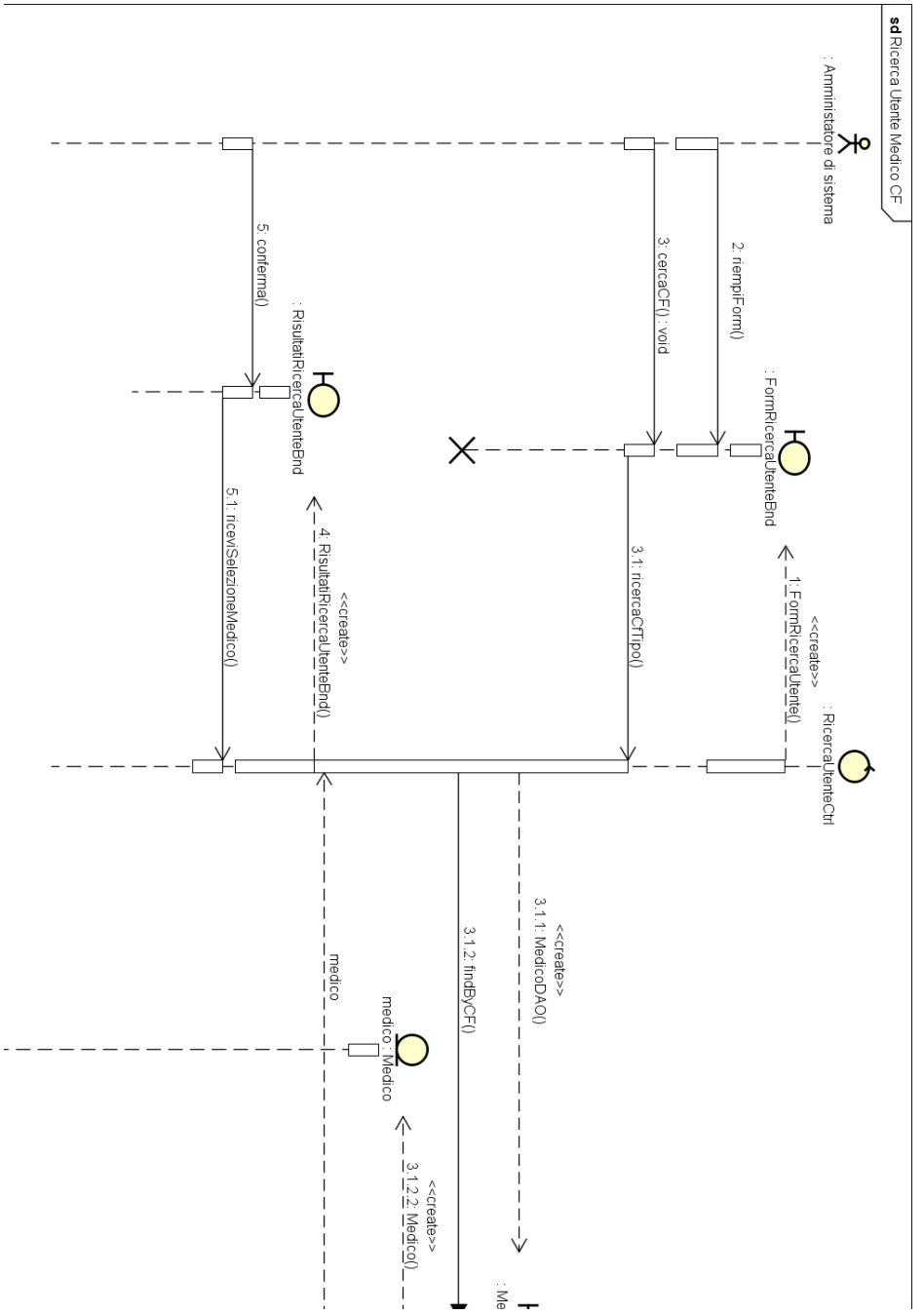


FIGURA 37 - DIAGRAMMA DI SEQUENZA RICERCAMEDICOCF

3.5.5.1.16 Diagramma di sequenza RicercaUtenteMedicoCognome

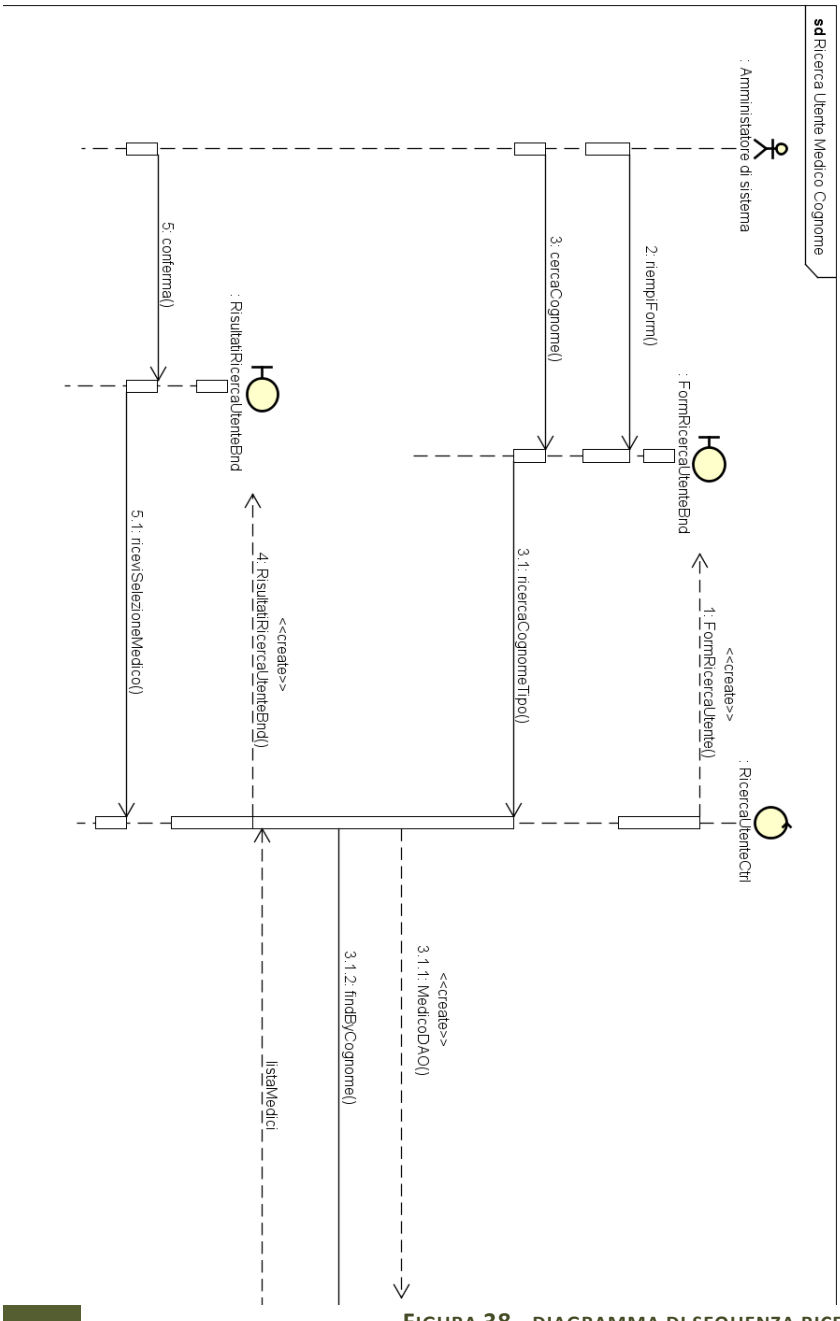


FIGURA 38 - DIAGRAMMA DI SEQUENZA RICERCAUTENTEMEDICOCOGNOME

3.5.5.1.17 Diagramma di sequenza RicercaUtenteMedicoNome

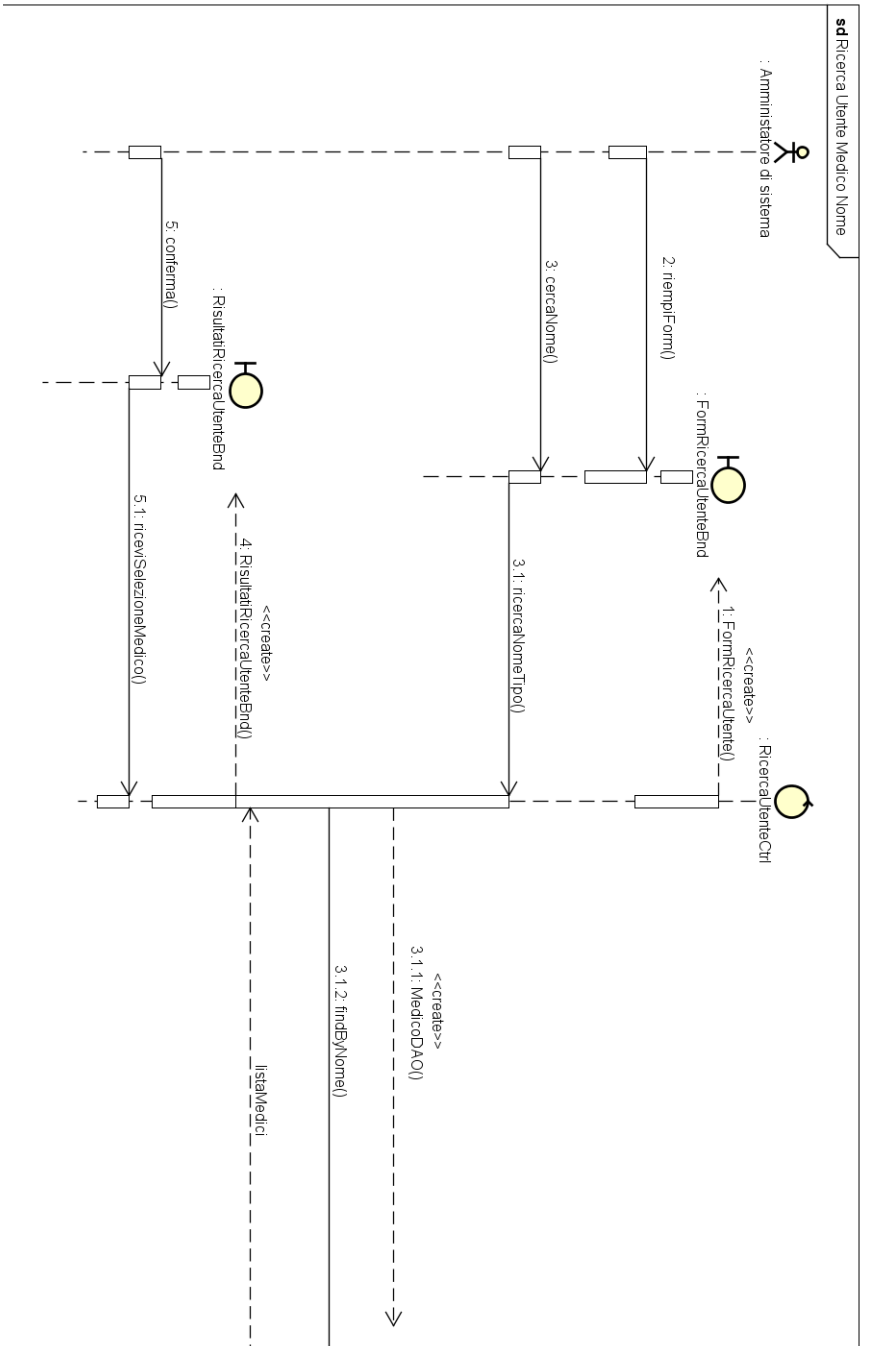


FIGURA 39 - DIAGRAMMA DI SEQUENZA RICERCAUTENTEMEDICONOME

3.5.5.1.18 Diagramma di sequenza RicercaUtenteReceptionistCF

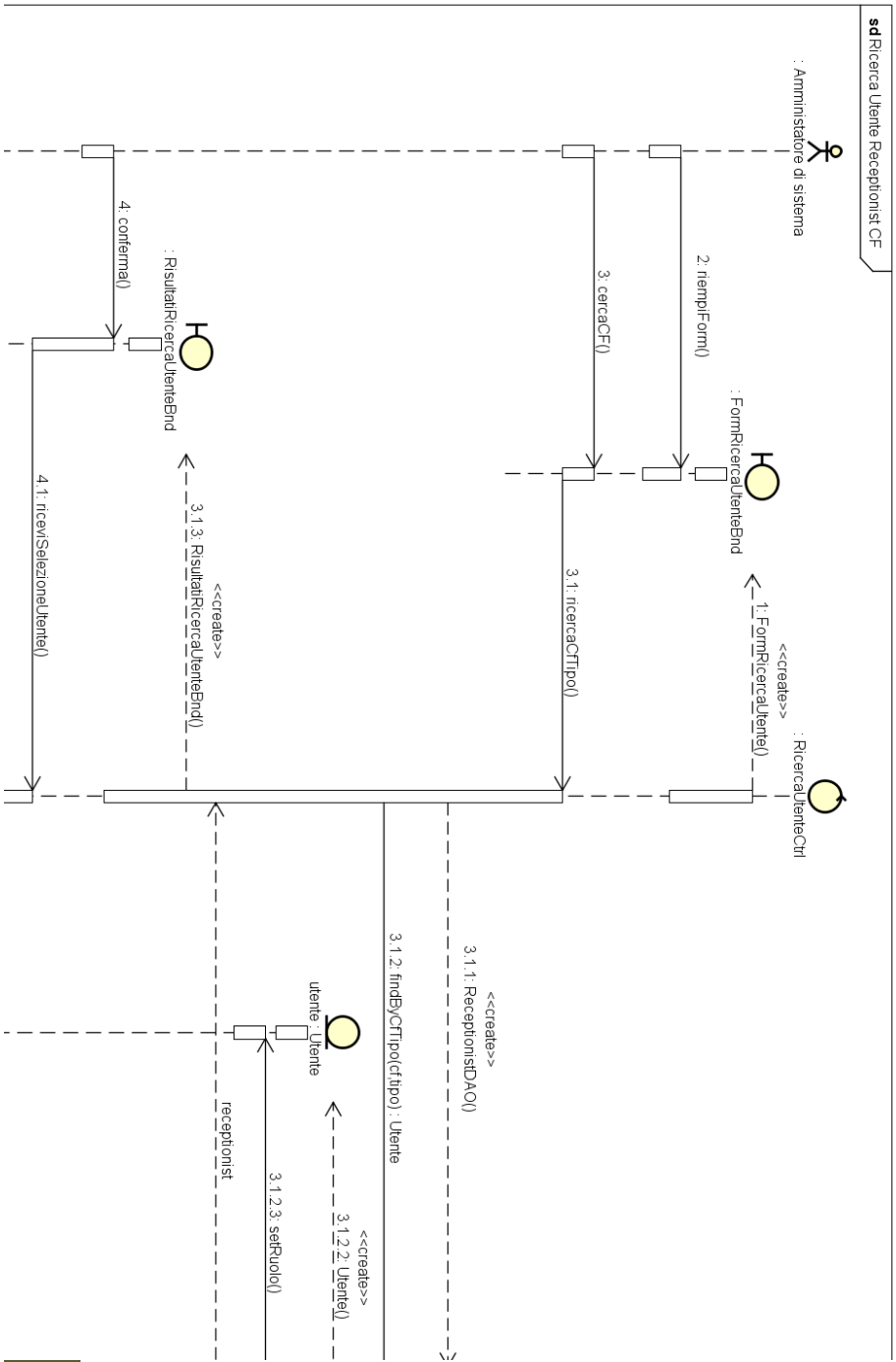


FIGURA 40 - DIAGRAMMA DI SEQUENZA DI RICERCAUTENTERECEPTIONISTCF

3.5.5.1.19 Diagramma di sequenza RicercaUtenteReceptionistCognome

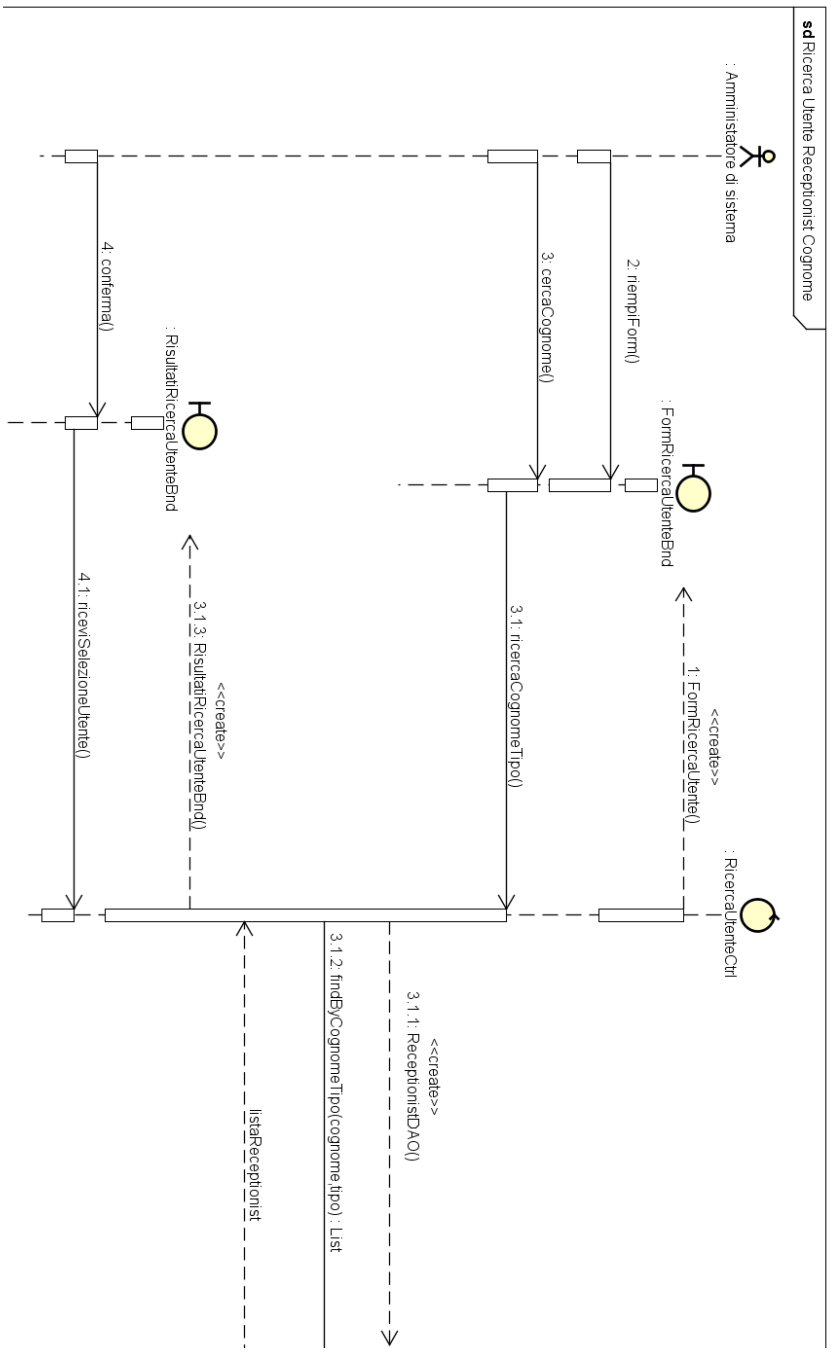


FIGURA 41 - DIAGRAMMA DI SEQUENZA DI RICERCAUTENTERECEPTIONISTCOGNOME

3.5.5.1.20 Diagramma di sequenza RicercaUtenteReceptionistNome

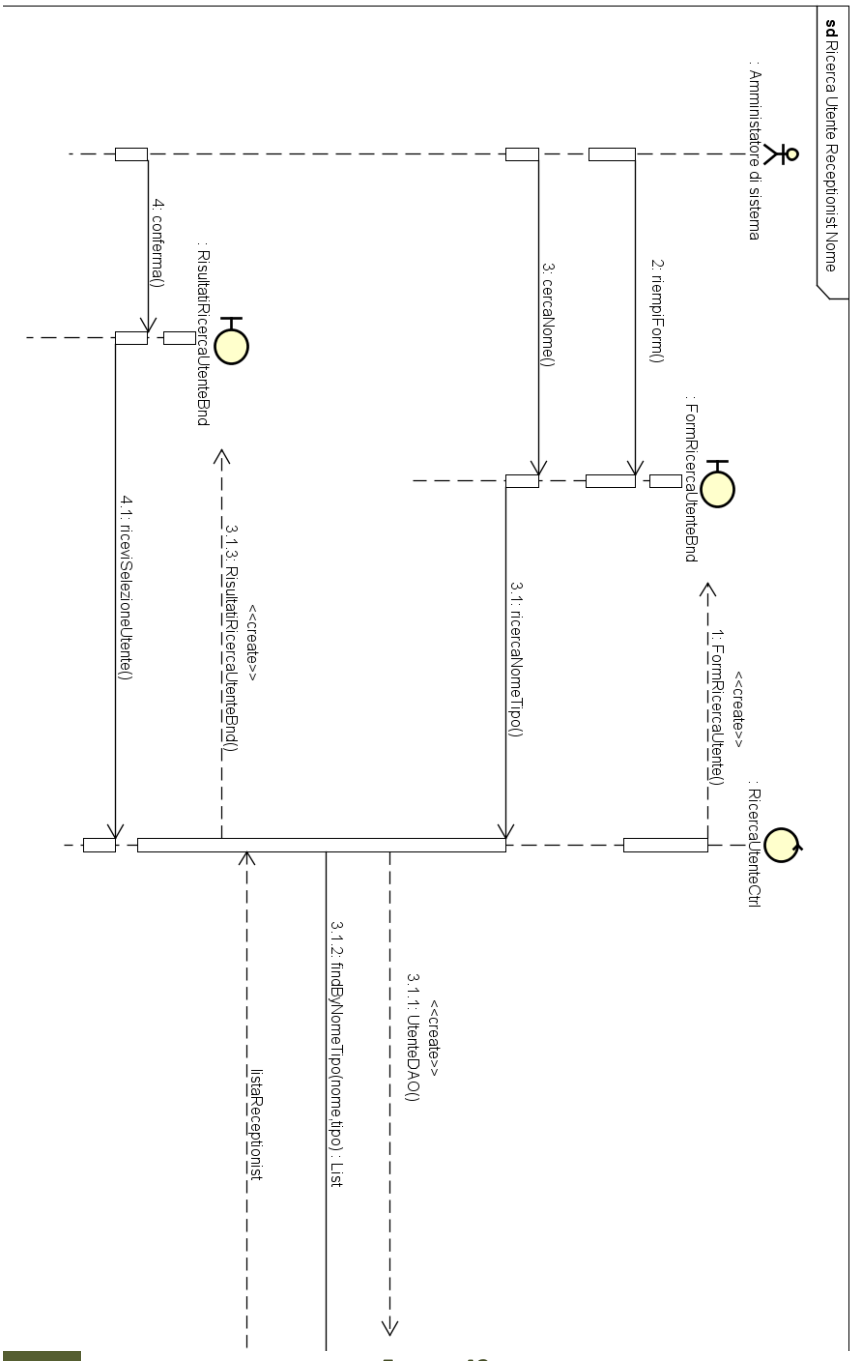


FIGURA 42 - DIAGRAMMA DI SEQUENZA DI RICERCAUTENTERECEPTIONISTNOME

3.5.5.1.21 Diagramma di sequenza VecchiaPasswordErrata

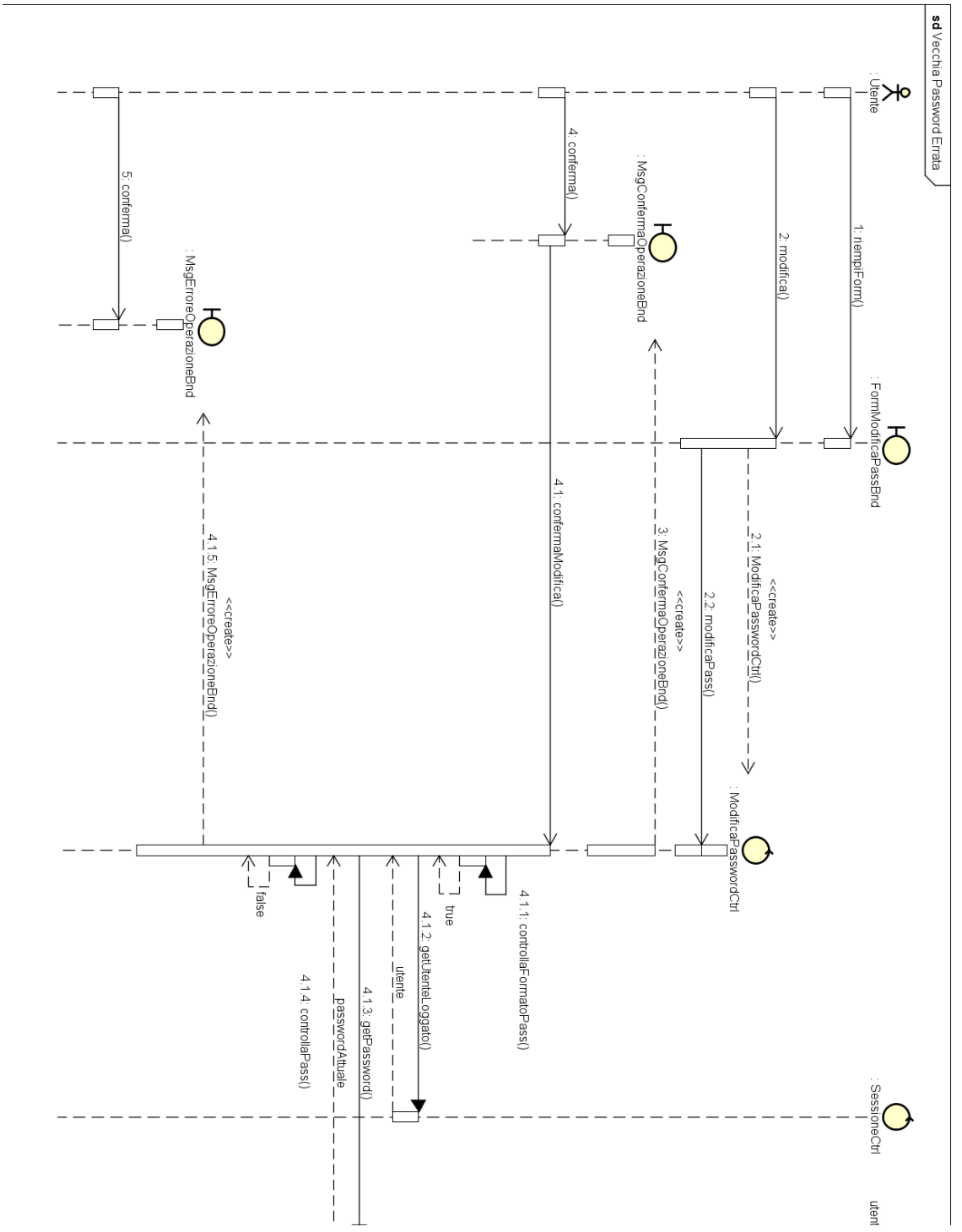


FIGURA 43 - DIAGRAMMA DI SEQUENZA VECCHIAPASSWORDERRATA

3.5.5.1.22 Diagramma di sequenza VisualizzaReport

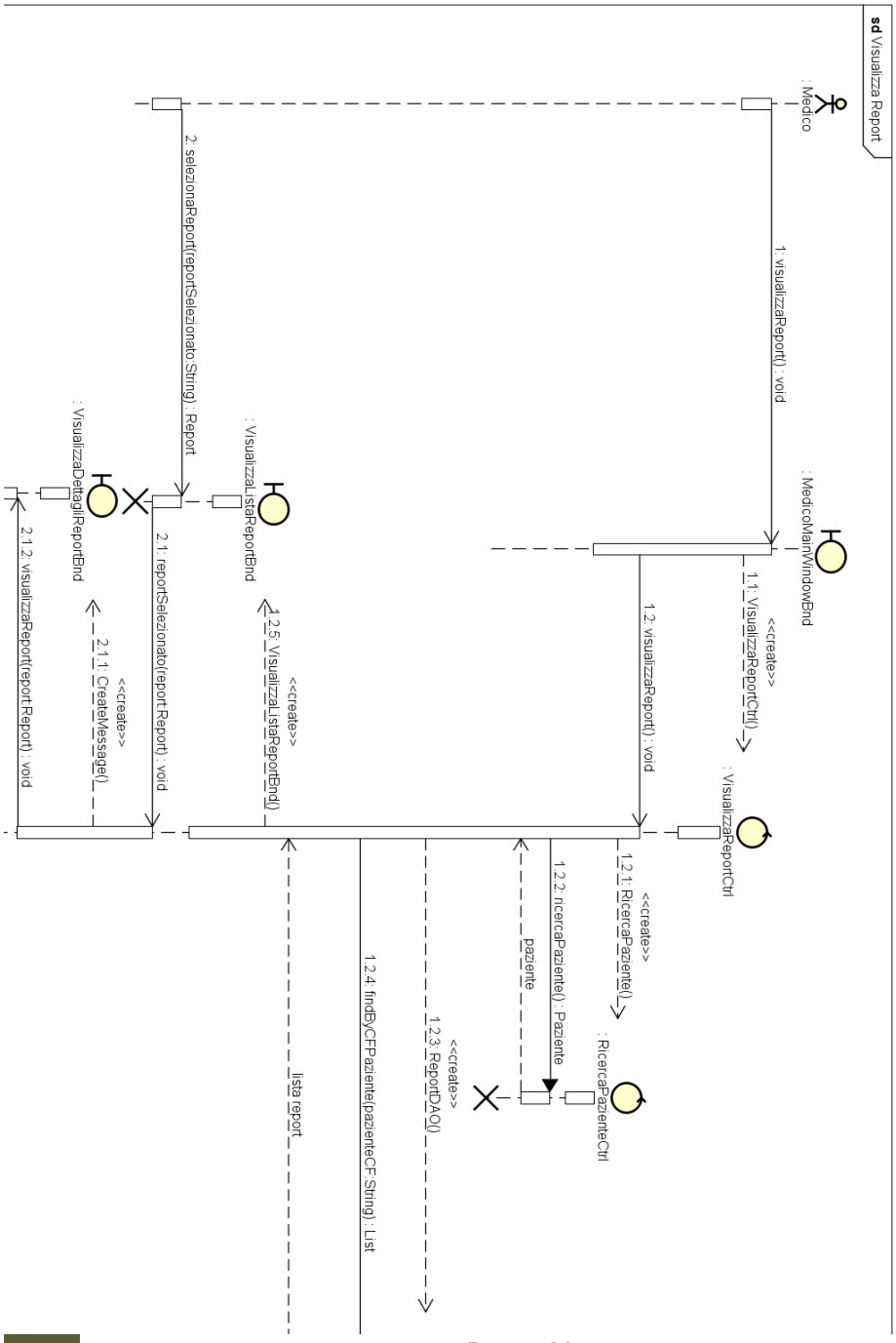


FIGURA 44 - DIAGRAMMA DI SEQUENZA VISUALIZZAREPORT

3.5.5.1.23 Diagramma di sequenza VisualizzaVisite

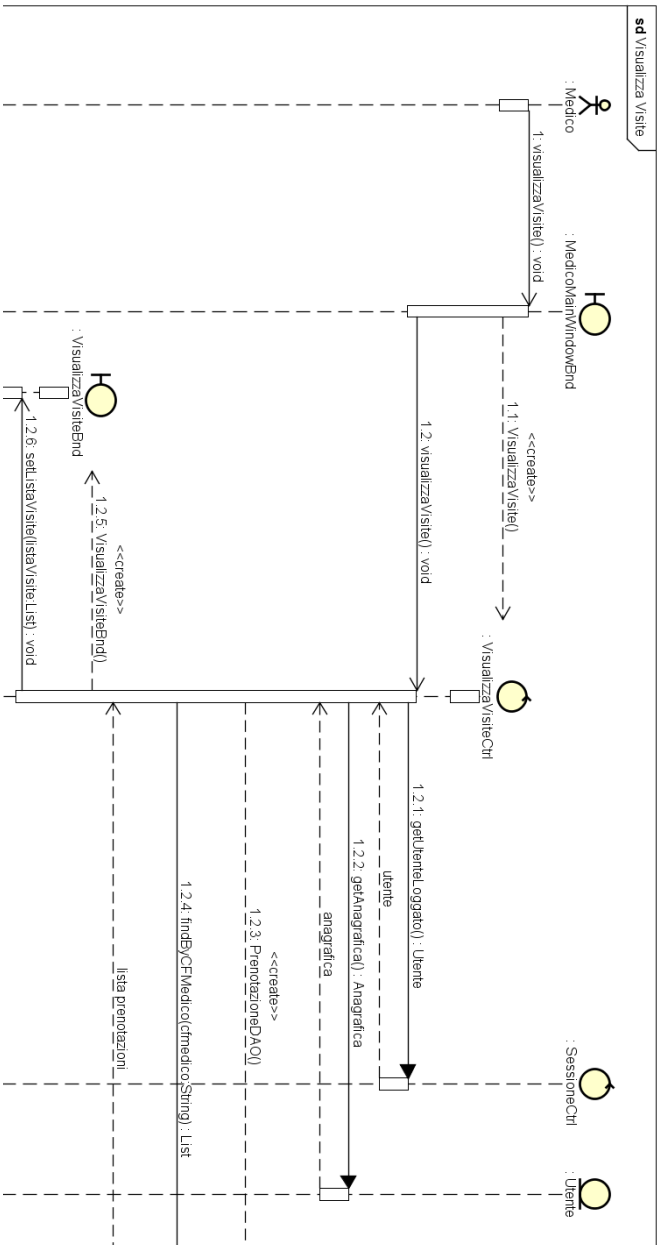


FIGURA 45 - DIAGRAMMA DI SEQUENZA VISUALIZZAVISITE

3.5.5.2 Diagrammi di attività

3.5.5.2.1 Diagramma di attività Elimina Utente

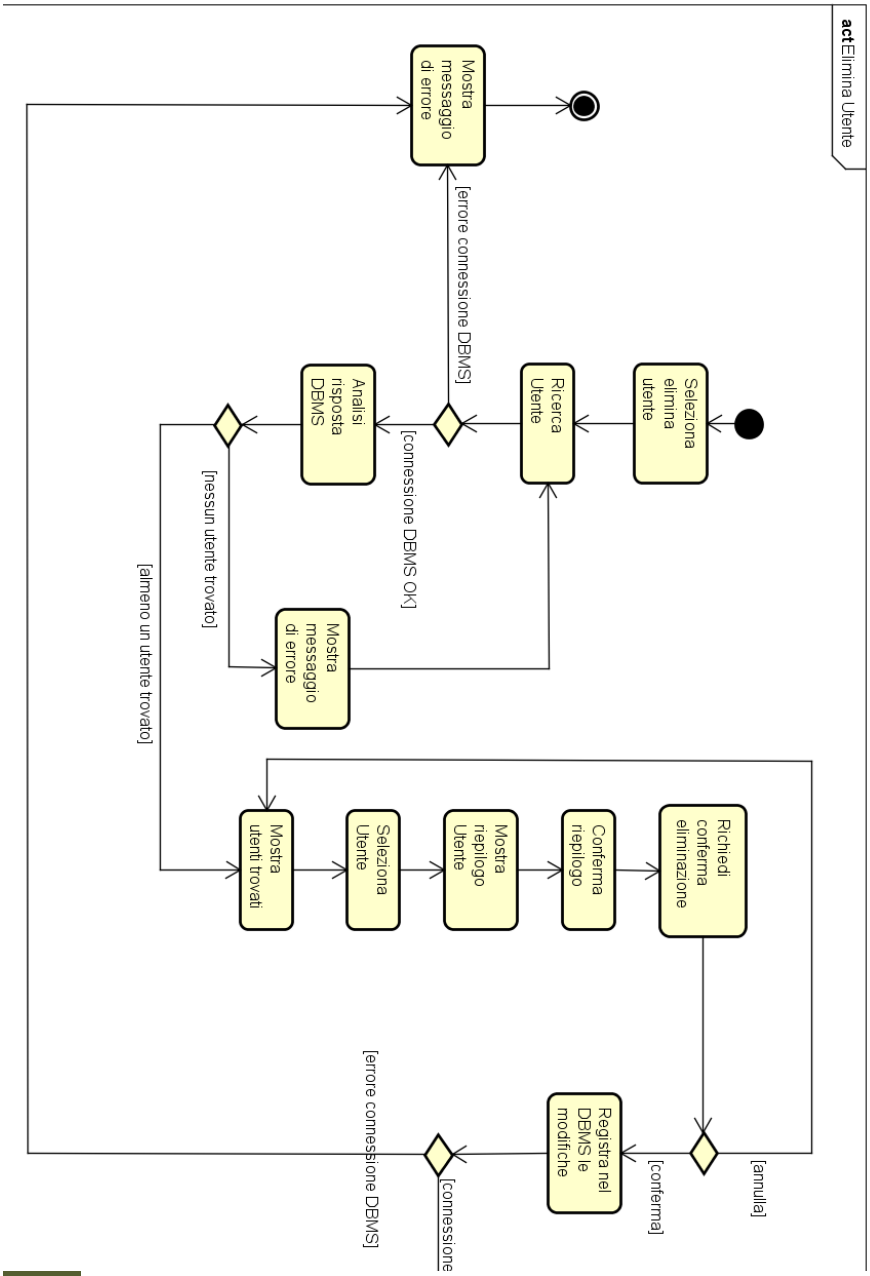


FIGURA 46 - DIAGRAMMA DI ATTIVITÀ ELIMINA UTENTE

3.5.5.2.2 Diagramma di attività Modifica Utente

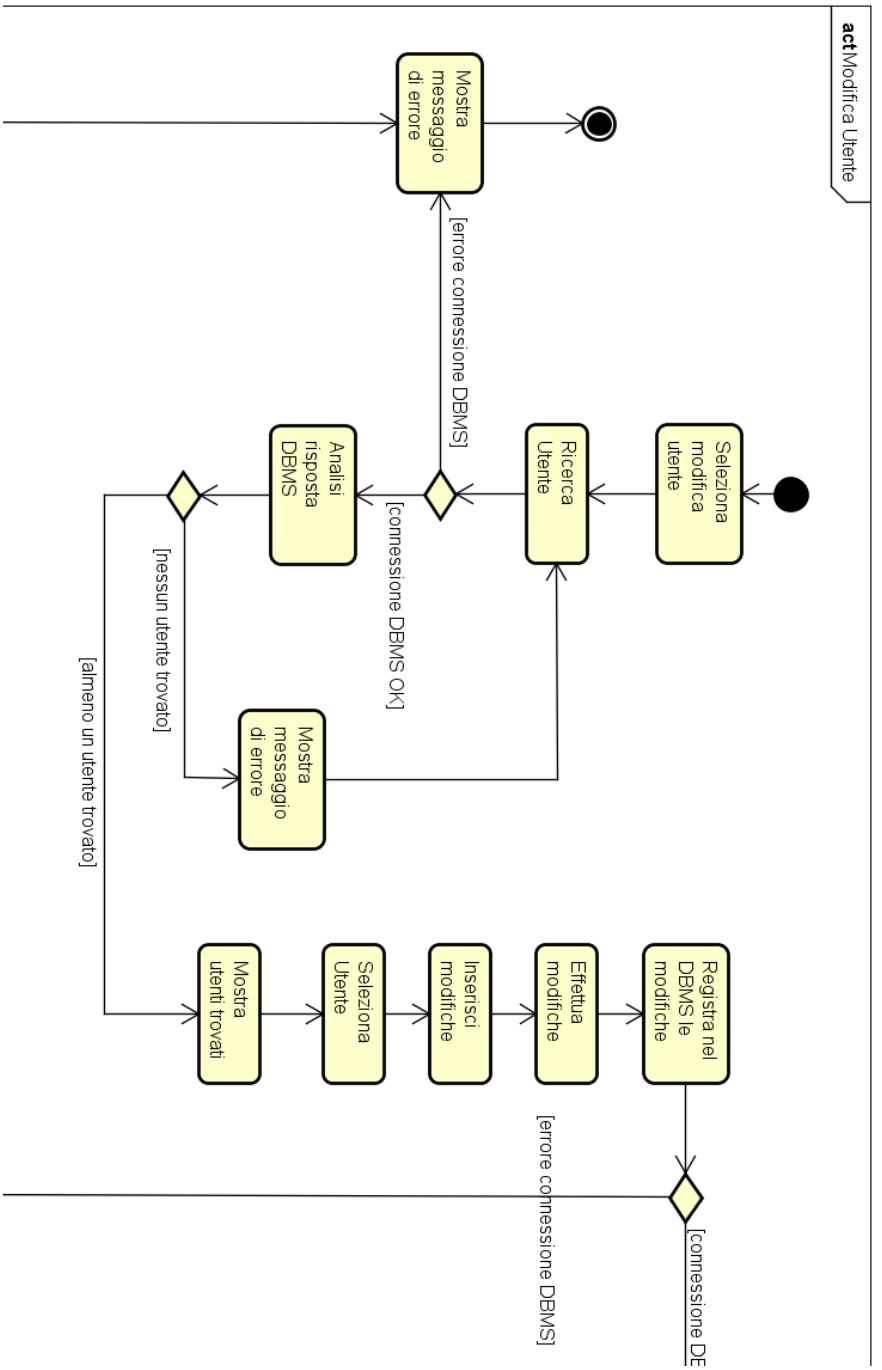


FIGURA 47 - DIAGRAMMA DI ATTIVITÀ MODIFICA UTENTE

3.5.5.2.3 Diagramma di attività Registra Medico

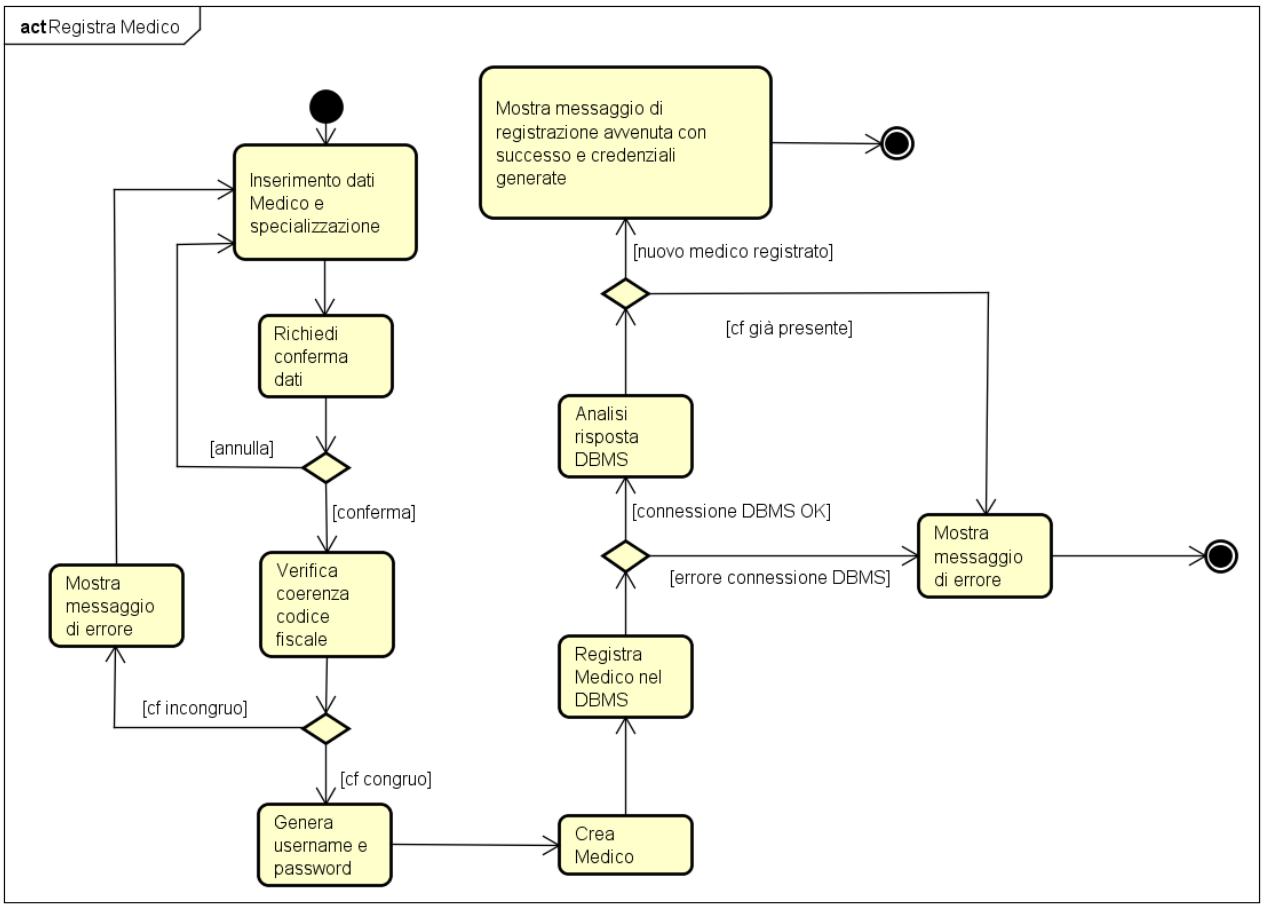


FIGURA 48 - DIAGRAMMA DI ATTIVITÀ REGISTRA MEDICO

3.5.5.2.4 Diagramma di attività Registra Receptionist

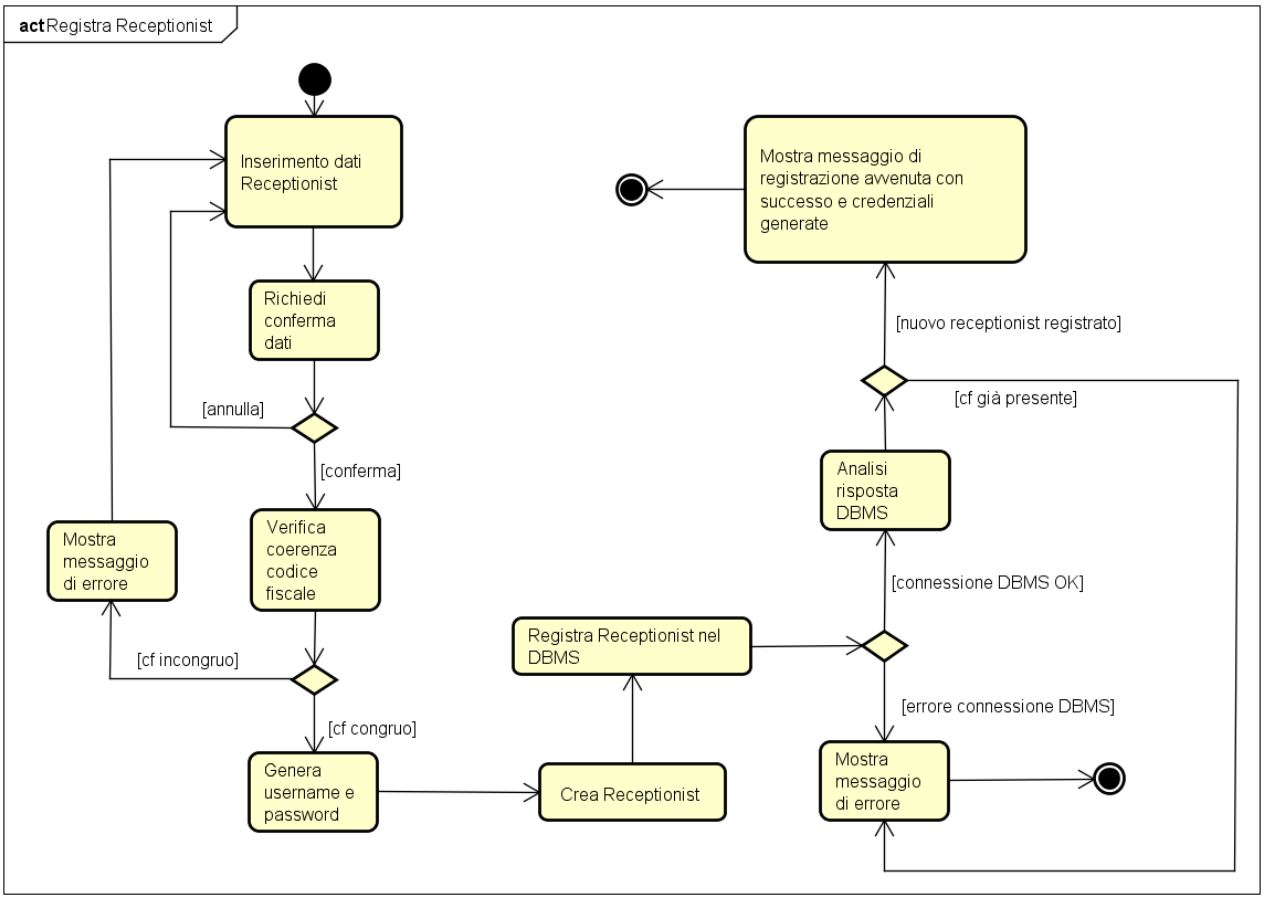


FIGURA 49 - DIAGRAMMA DI ATTIVITÀ REGISTRA RECEPTIONIST

3.5.5.2.5 Diagramma di attività Inserimento Report

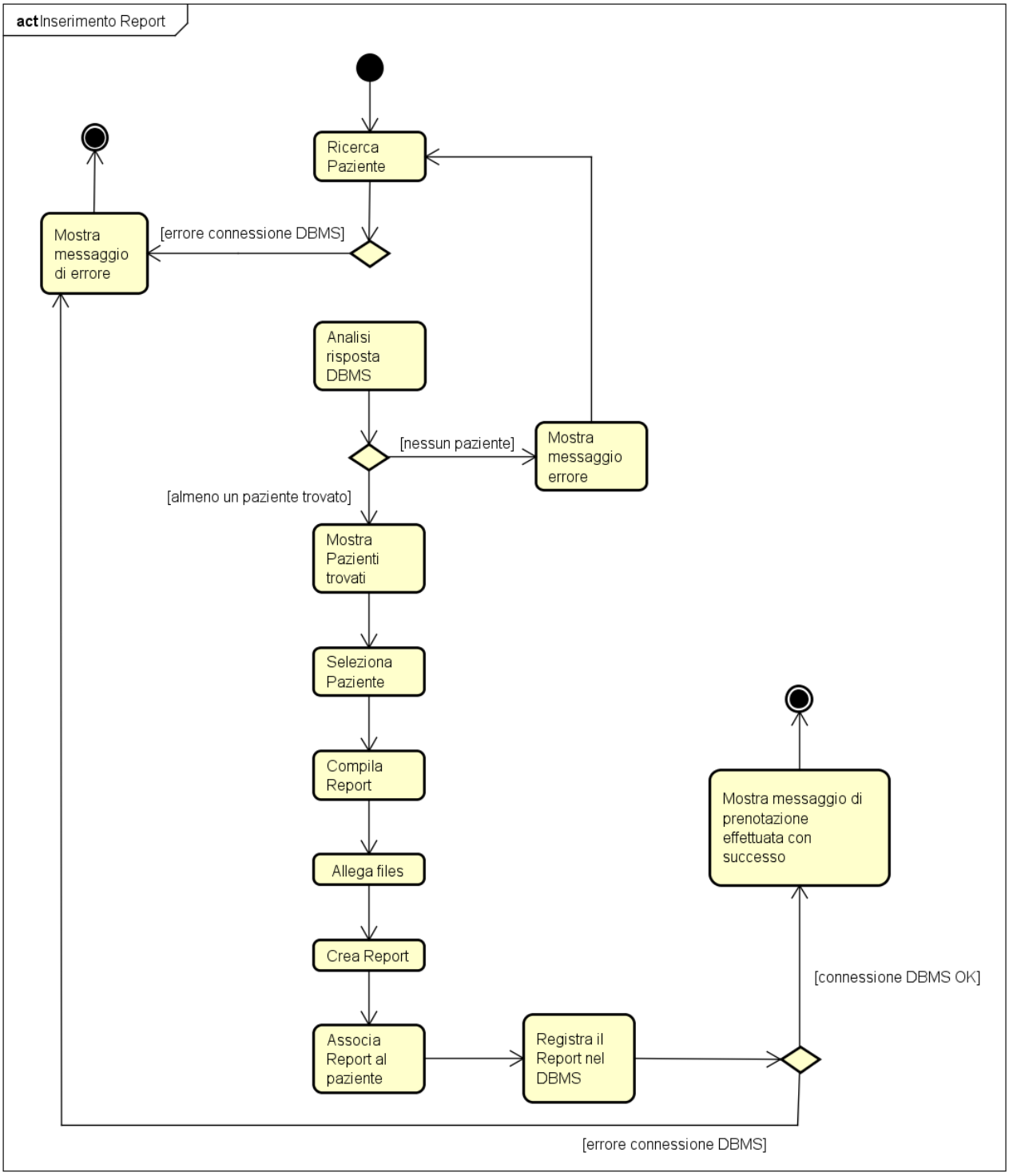


FIGURA 50 - DIAGRAMMA DI ATTIVITÀ INSERIMENTO REPORT

3.5.5.2.6 Diagramma di attività Visualizza Report

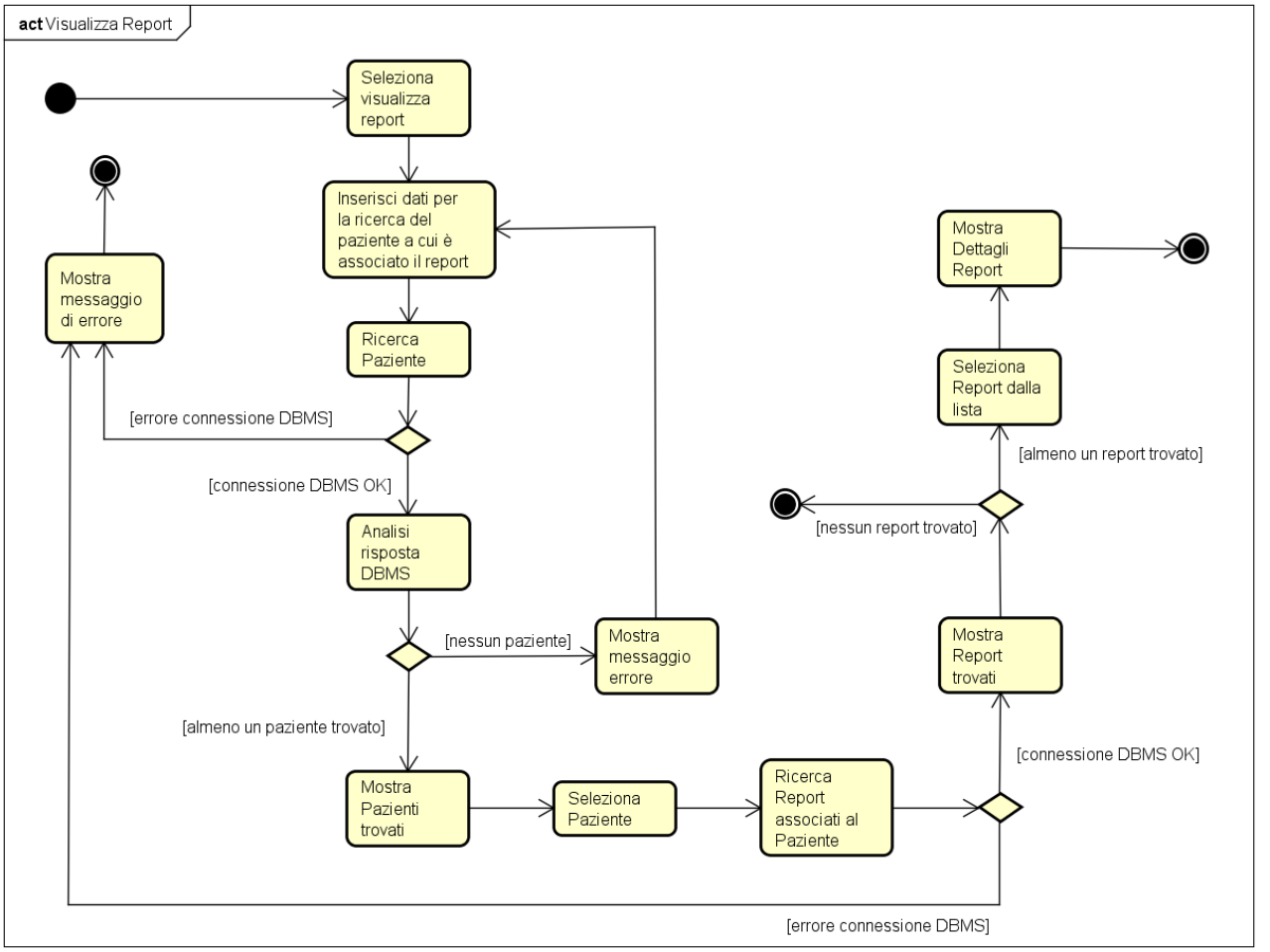


FIGURA 51 - DIAGRAMMA DI ATTIVITÀ VISUALIZZA REPORT

3.5.5.2.7 Diagramma di attività Visualizza Visite

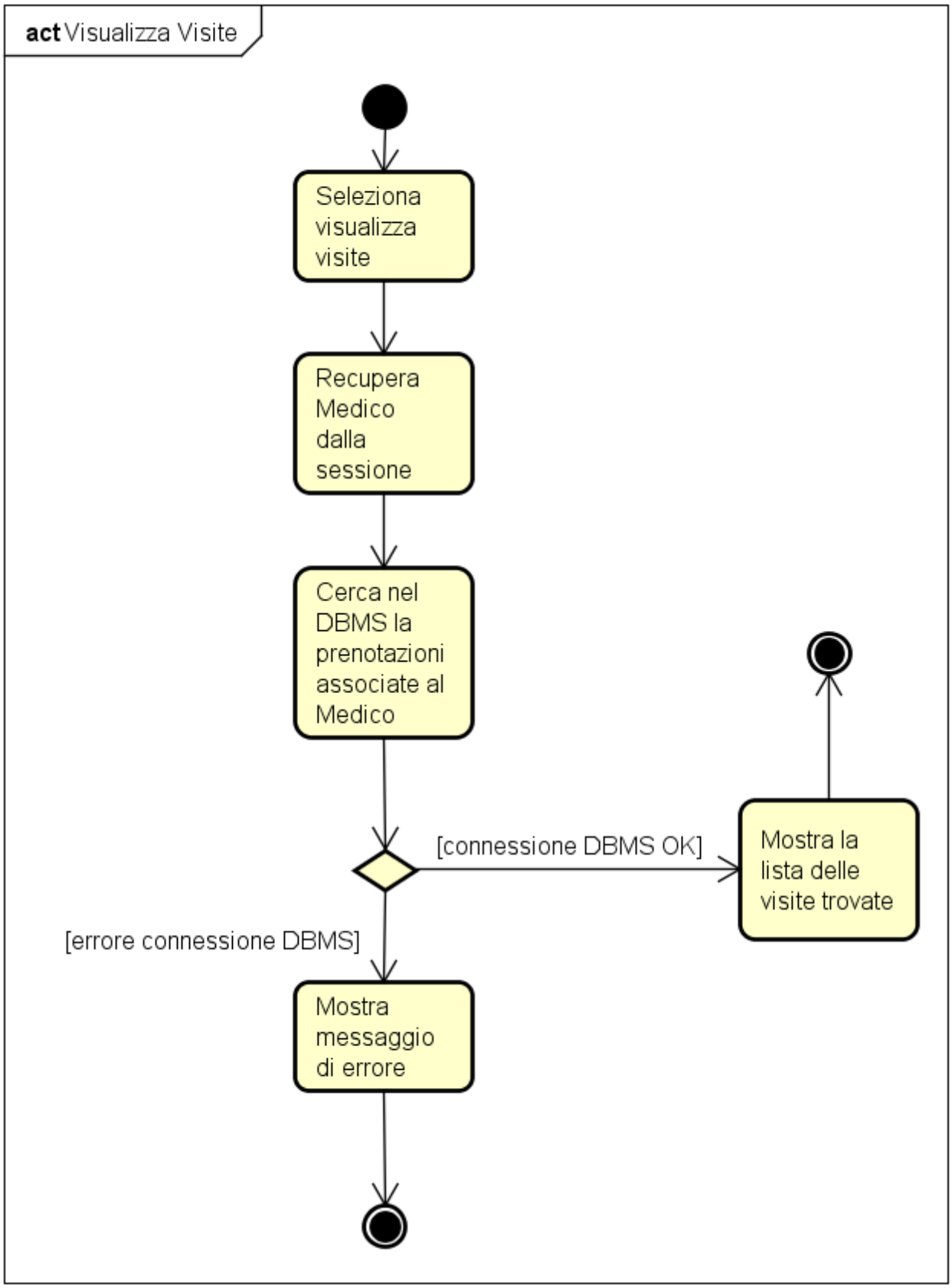


FIGURA 52 - DIAGRAMMA DI ATTIVITÀ VISUALIZZA VISITE

3.5.5.2.8 Diagramma di attività Prenotazione Visita

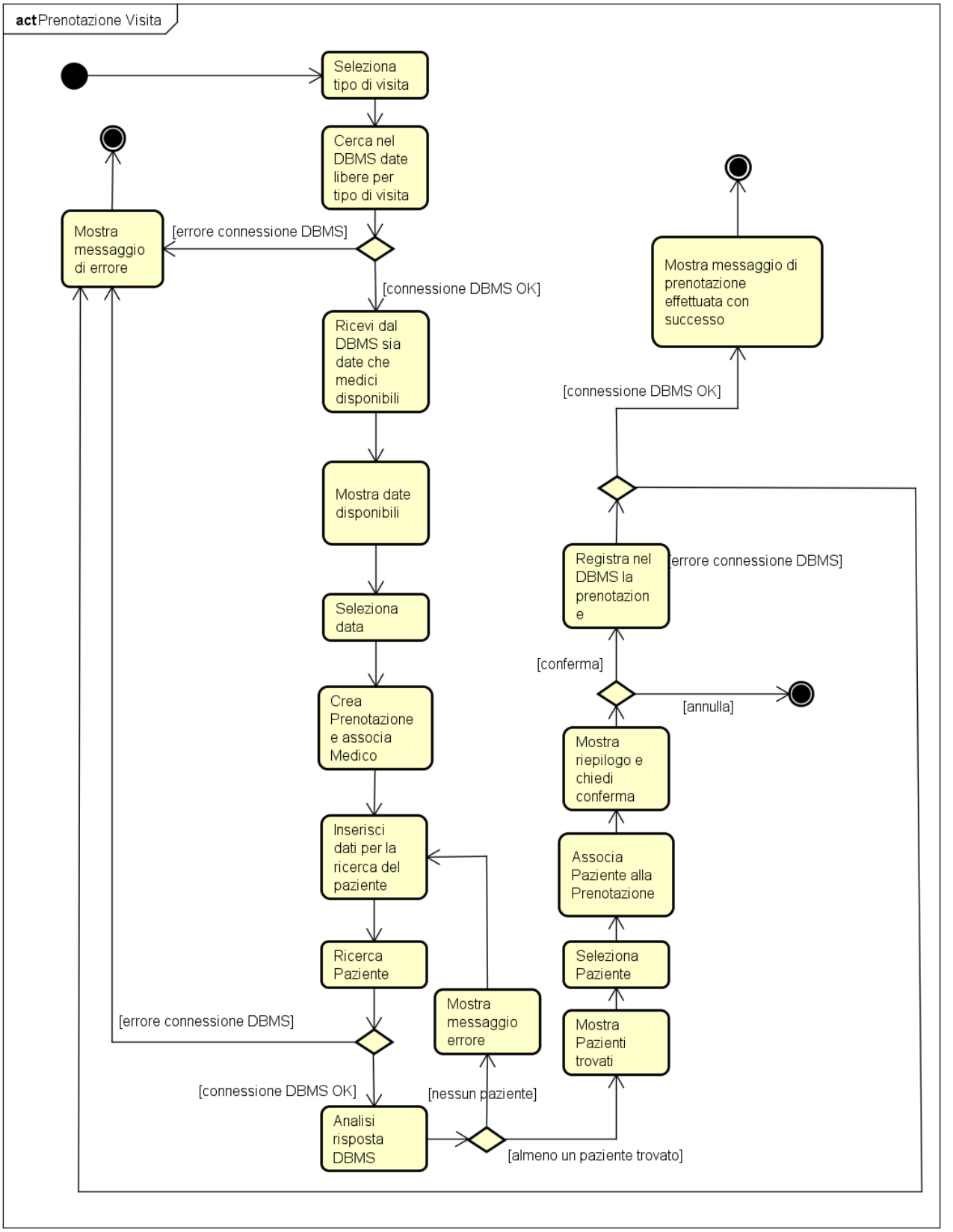


FIGURA 53 - DIAGRAMMA DI ATTIVITÀ PRENOTAZIONE VISITA

3.5.5.2.9 Diagramma di attività Registra Paziente

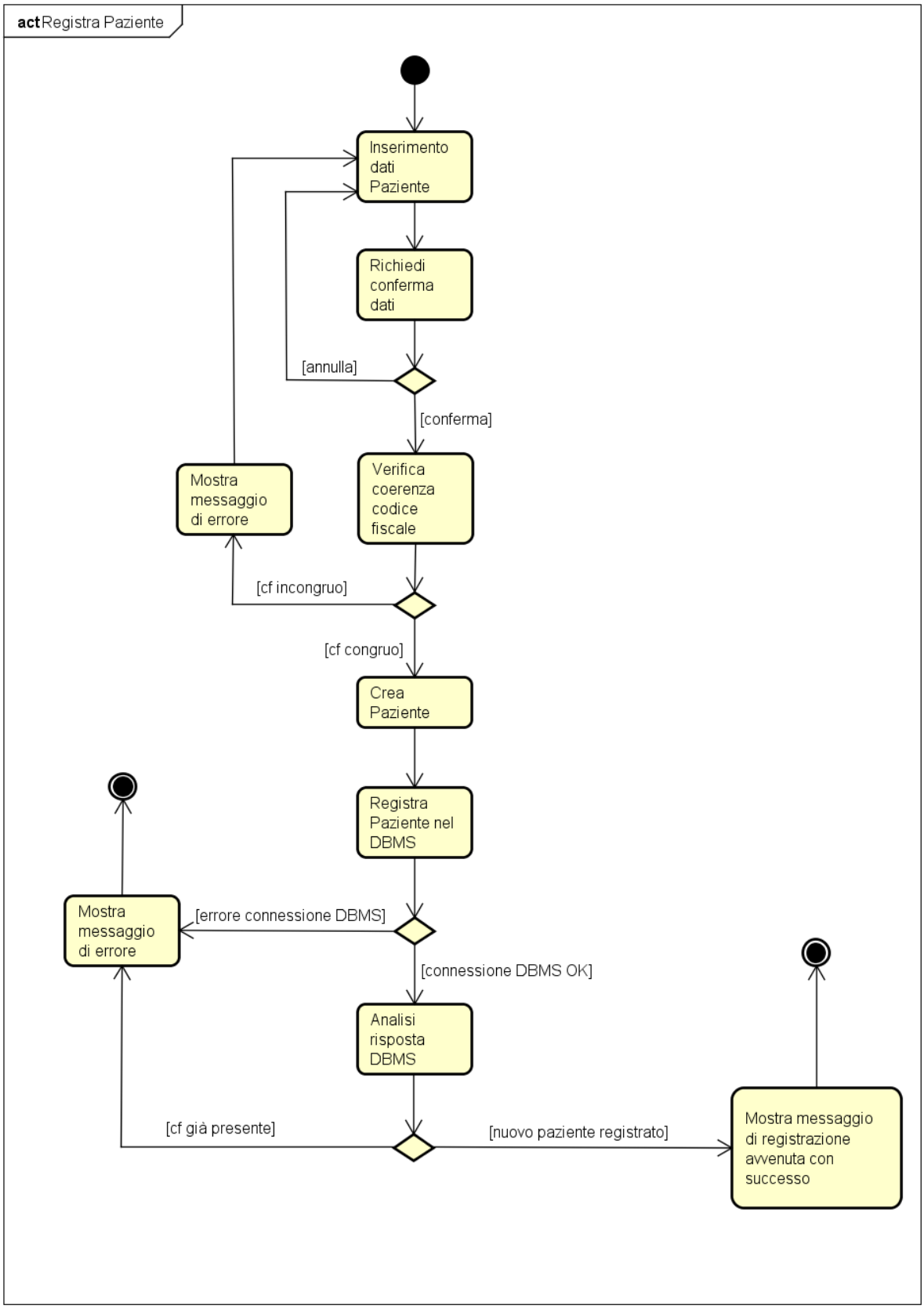


FIGURA 54 - DIAGRAMMA DI ATTIVITÀ REGISTRA PAZIENTE

3.5.5.2.10 Diagramma di attività Login

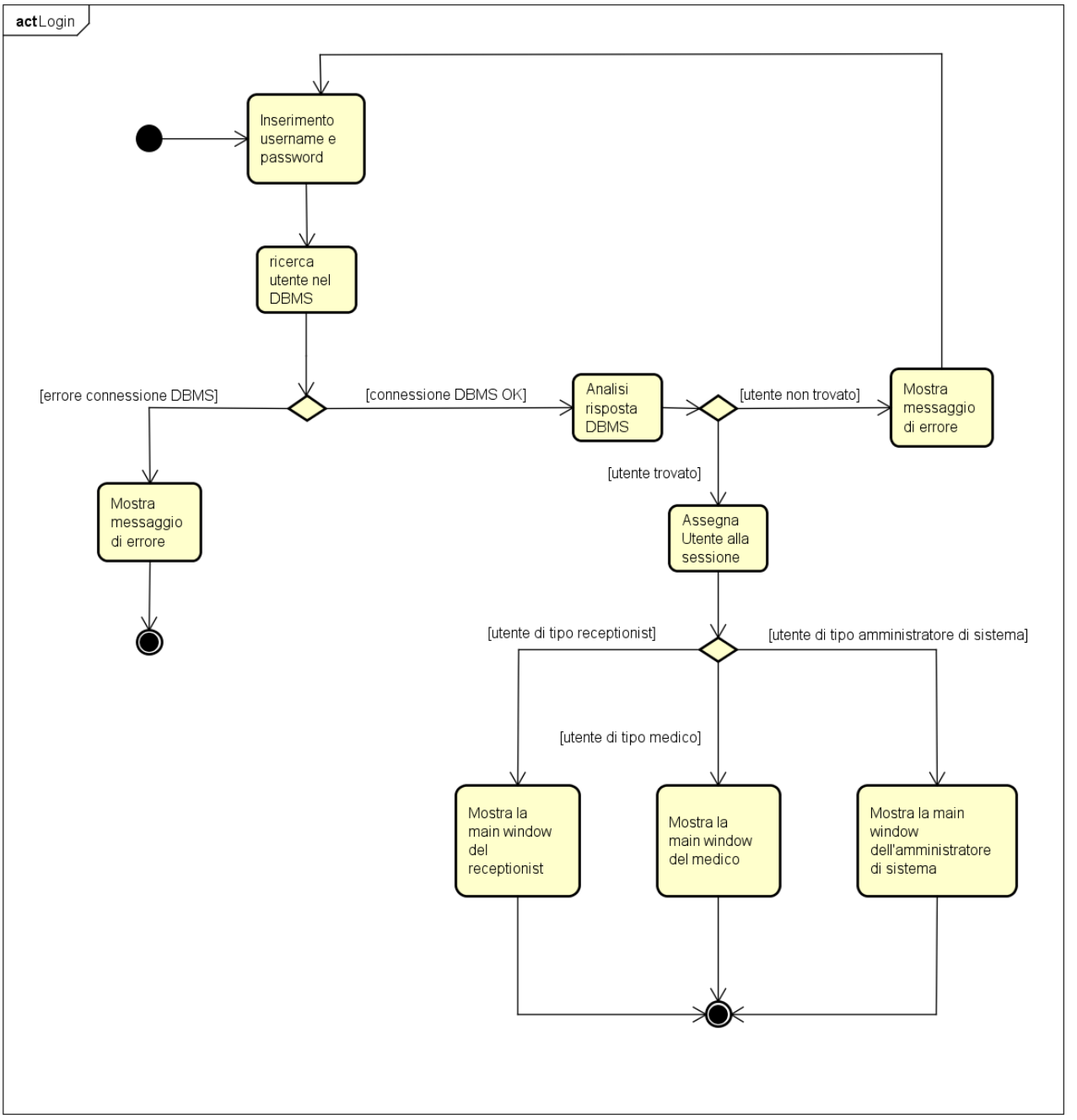


FIGURA 55 - DIAGRAMMA DI ATTIVITÀ LOGIN

3.5.5.2.11 Diagramma di attività Modifica Password

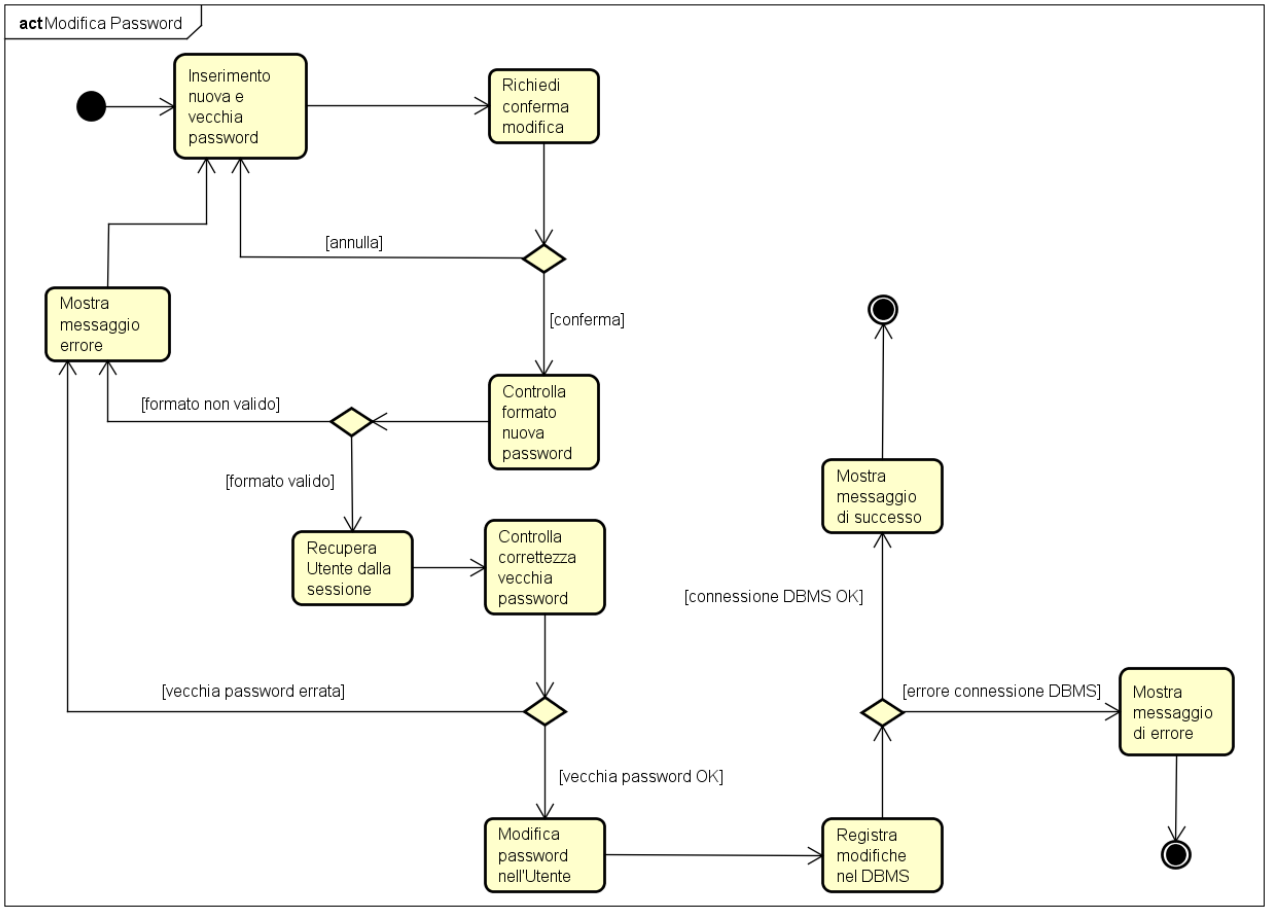


FIGURA 56 - DIAGRAMMA DI ATTIVITÀ MODIFICA PASSWORD

3.5.5.3 Diagrammi di stato

3.5.5.3.1 Diagramma di stato Ricerca Paziente

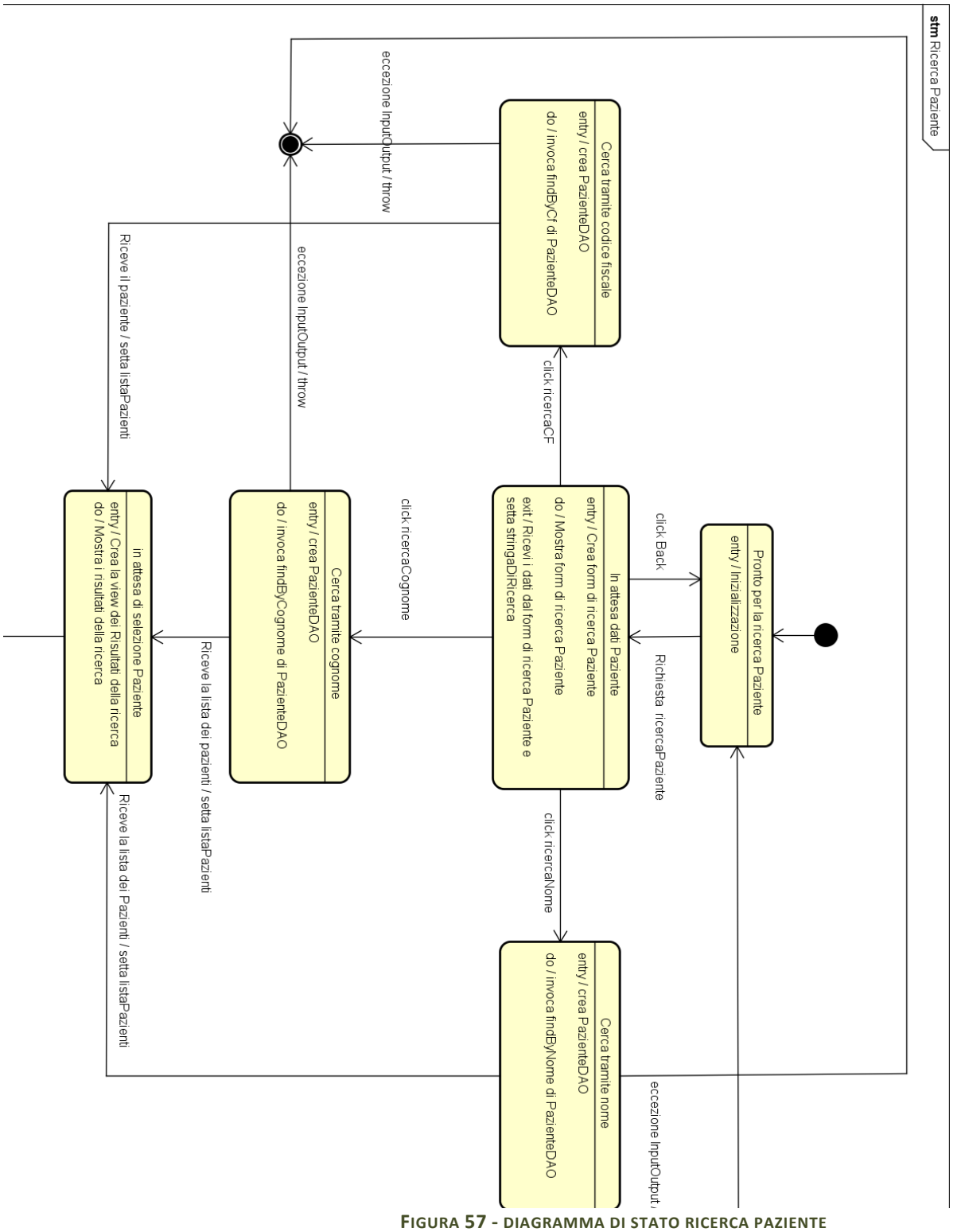


FIGURA 57 - DIAGRAMMA DI STATO RICERCA PAZIENTE

3.5.5.3.2 Diagramma di stato Ricerca Utente

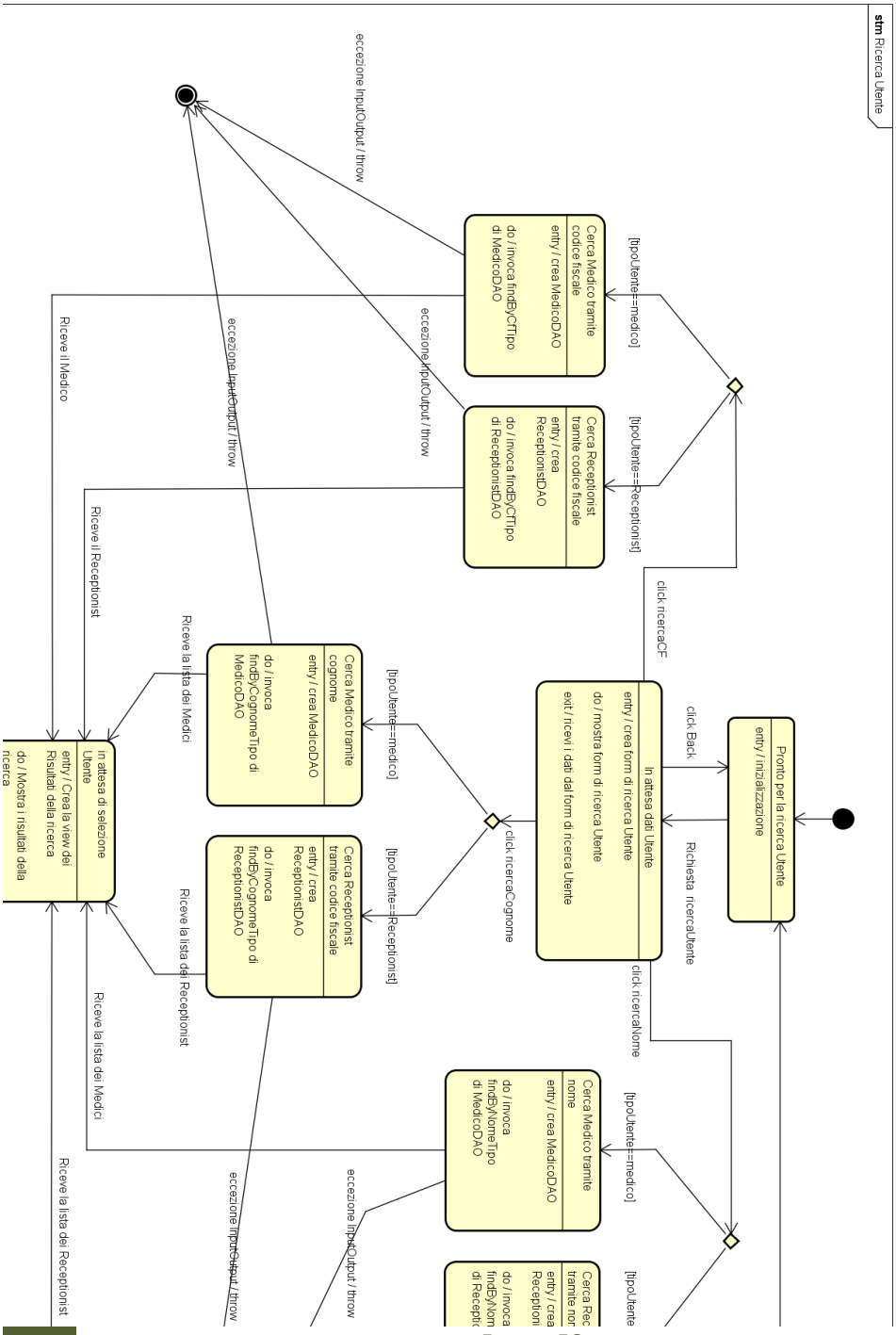


FIGURA 58 - DIAGRAMMA DI STATO RICERCA UTENTE

3.5.6 Interfacce Grafiche

3.5.6.1 Login

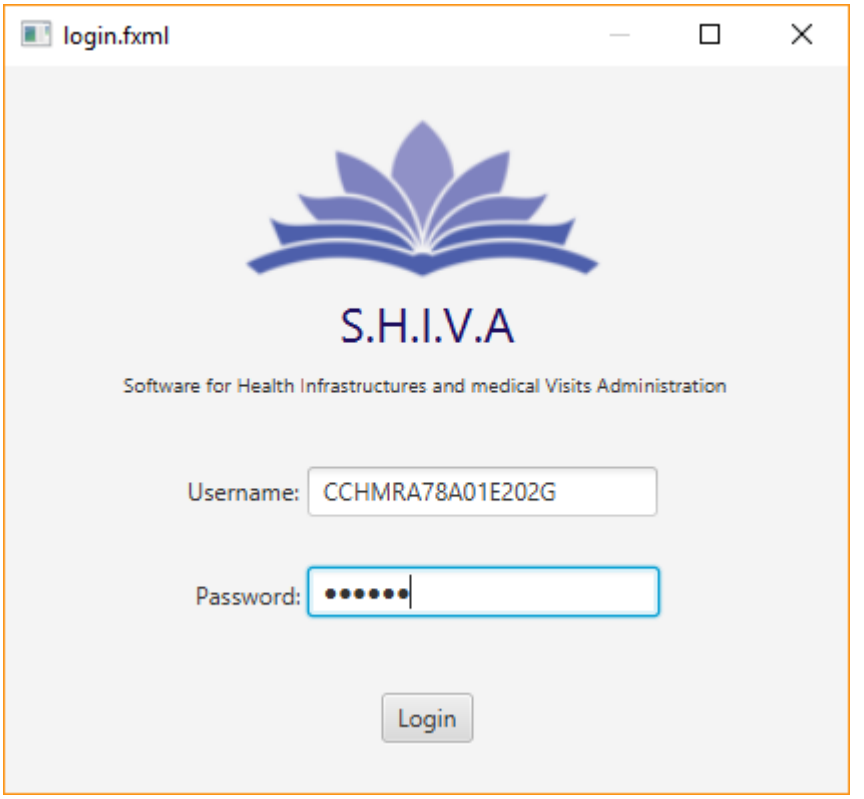


FIGURA 59 - FINESTRA DI LOGIN

3.5.6.2 MainWindowReceptionist

FIGURA 60 - FINESTRA PRINCIPALE DEL RECEPTIONIST

3.5.6.3 MainWindowMedico

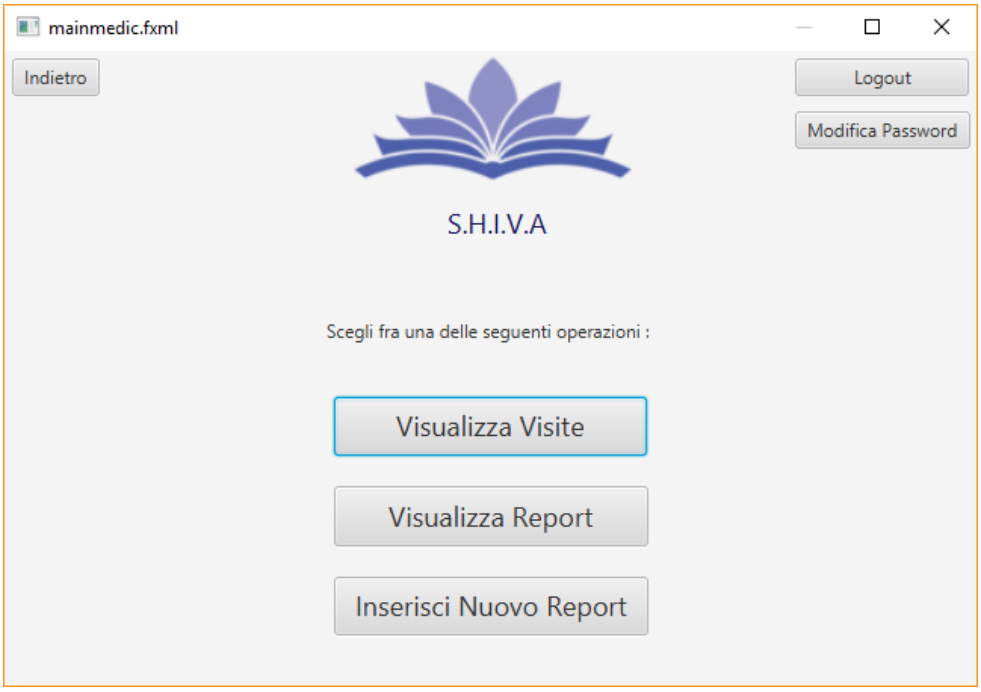


FIGURA 61 - FINESTRA PRINCIPALE DEL MEDICO

3.5.6.4 MainWindowAmministratore

FIGURA 62 - FINESTRA PRINCIPALE AMMINISTRATORE

3.5.6.5 ModificaPassword

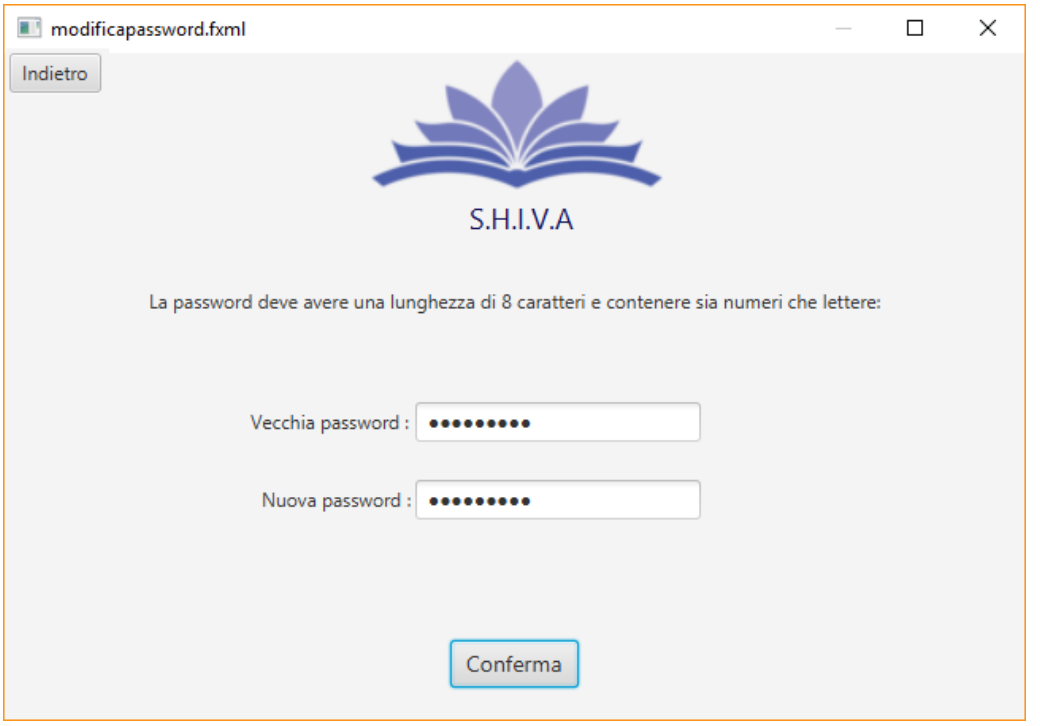


FIGURA 63 - FINESTRA PER LA MODIFICA DELLA PASSWORD

3.5.6.6 RegistraPaziente

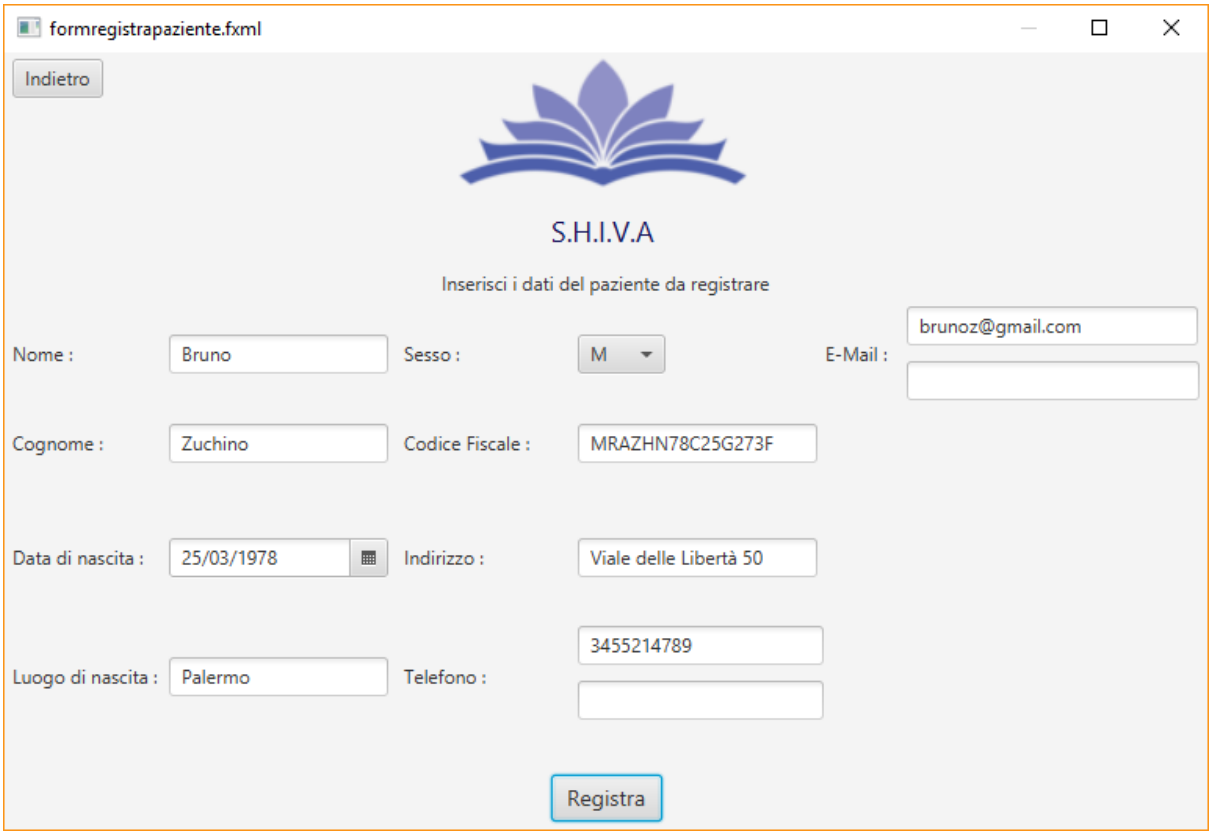


FIGURA 64 - FINESTRA PER LA REGISTRAZIONE DI UN PAZIENTE

3.5.6.7 RicercaPaziente

FIGURA 65 - FINESTRA PER LA RICERCA DI UN PAZIENTE

3.5.6.8 RisultatiRicercaPaziente

FIGURA 66 - FINESTRA CHE CONTIENE I RISULTATI DELLA RICERCA PAZIENTE

3.5.6.9 TipologiaVisita

FIGURA 67 - FINESTRA IN CUI SELEZIONARE LA TIPOLOGIA DI VISITA

3.5.6.10 SelezioneData

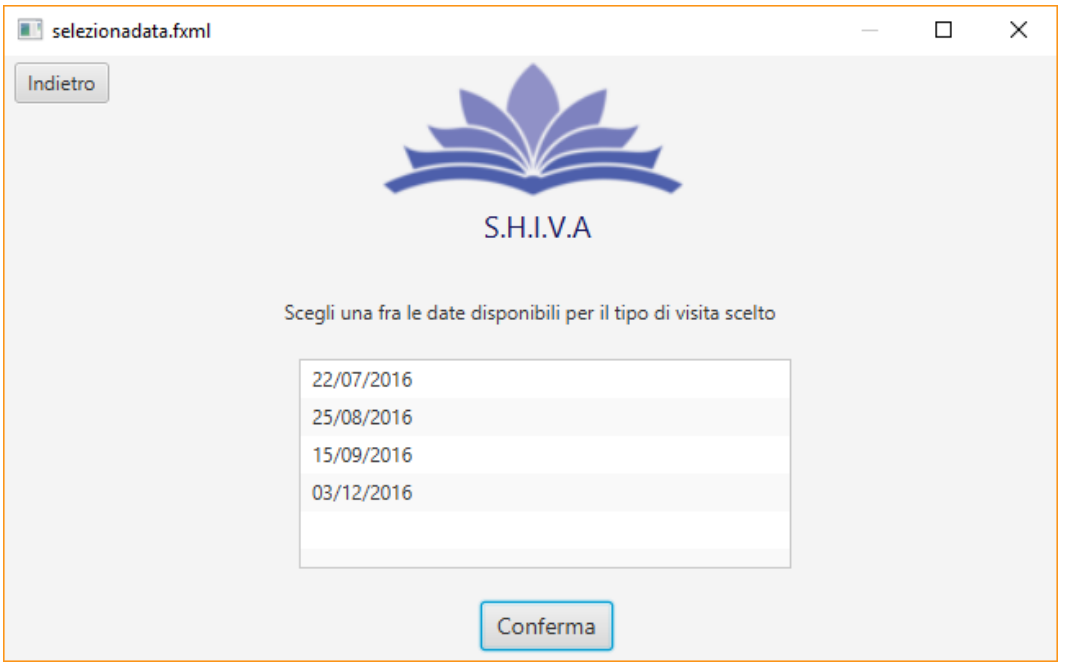


FIGURA 68 - FINESTRA DI SELEZIONE DATA

3.5.6.11 RiepilogoPrenotazione

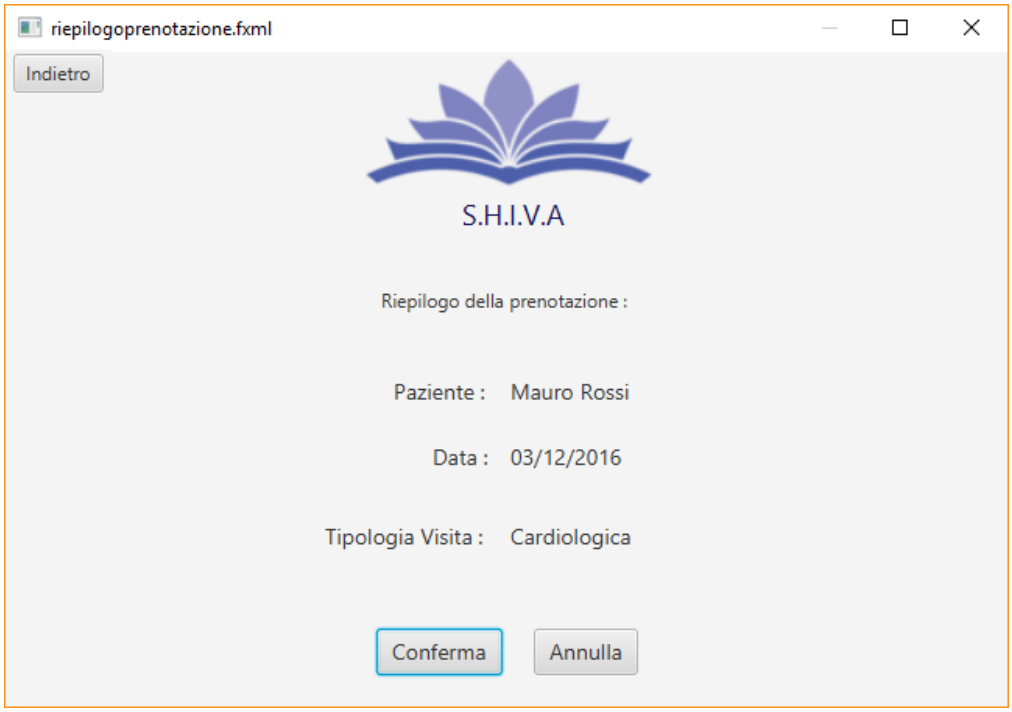


FIGURA 69 - FINESTRA DI RIEPILOGO PRENOTAZIONE

3.5.6.12 ElencoVisite

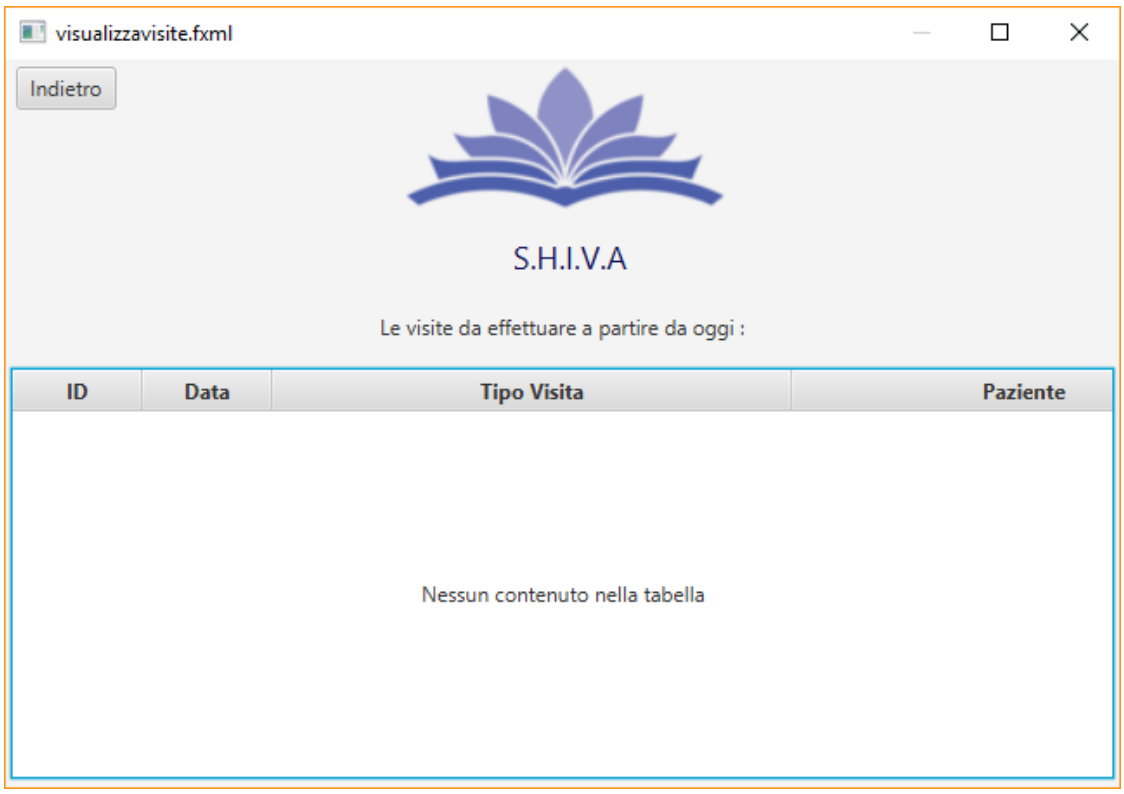


FIGURA 70 - FINESTRA CHE ELENCA LE VISITE DA EFFETTUARE

3.5.6.13 ListaReport

FIGURA 71 - FINESTRA CHE ELENCA I REPORT ASSOCIATI AL PAZIENTE

3.5.6.14 DettagliReport

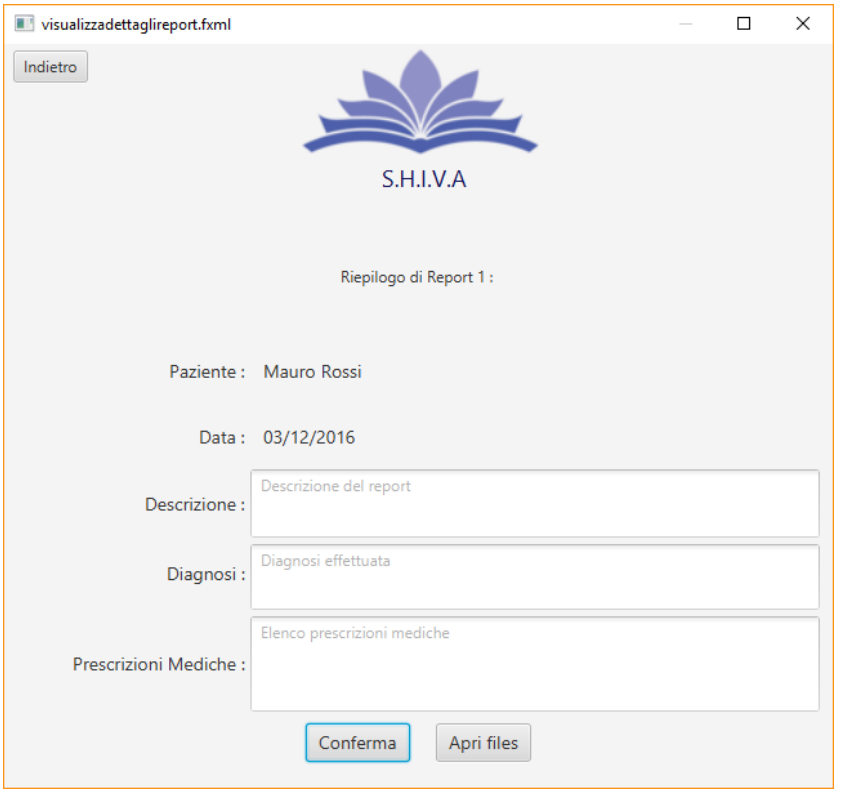


FIGURA 72 - FINESTRA DEI DETTAGLI REPORT

3.5.6.15 CreaReport

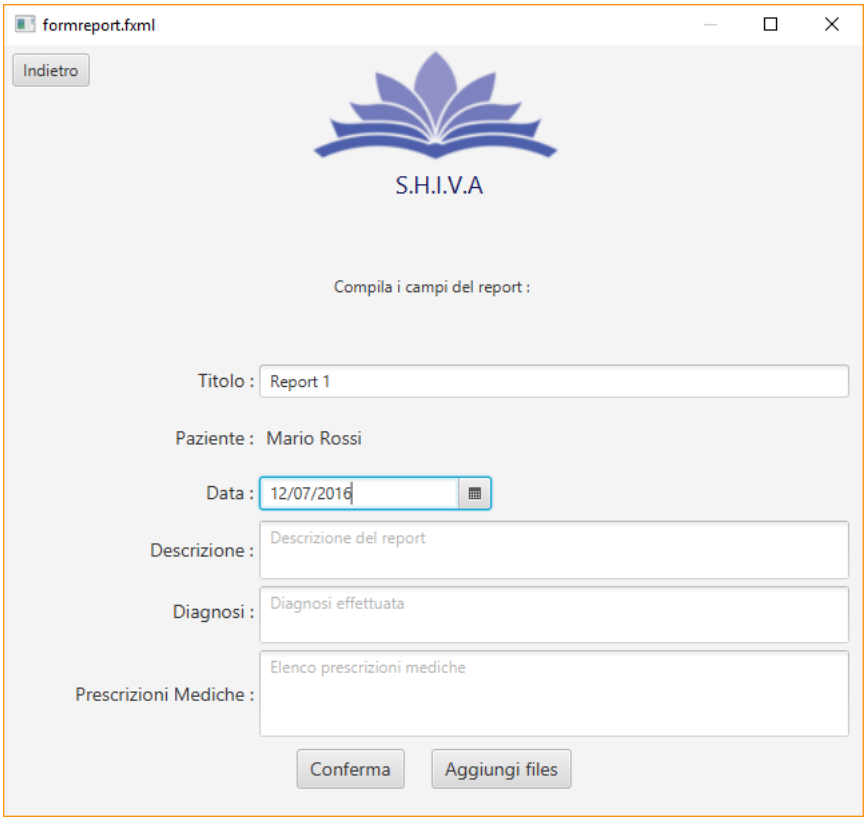


FIGURA 73 - FINESTRA PER CREAZIONE DI UN REPORT

3.5.6.16 RicercaUtente

FIGURA 74 - FINESTRA PER LA RICERCA DI UN UTENTE

3.5.6.17 RegistraMedico

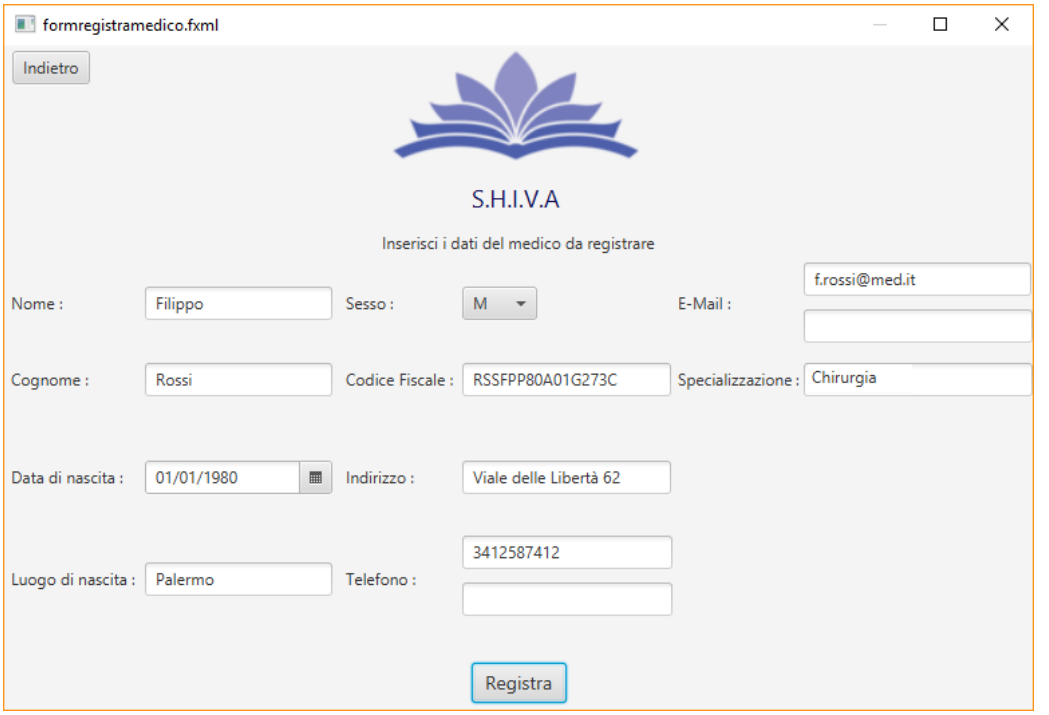


FIGURA 75 - FINESTRA DI REGISTRAZIONE MEDICO

3.5.6.18 RegistraReceptionist

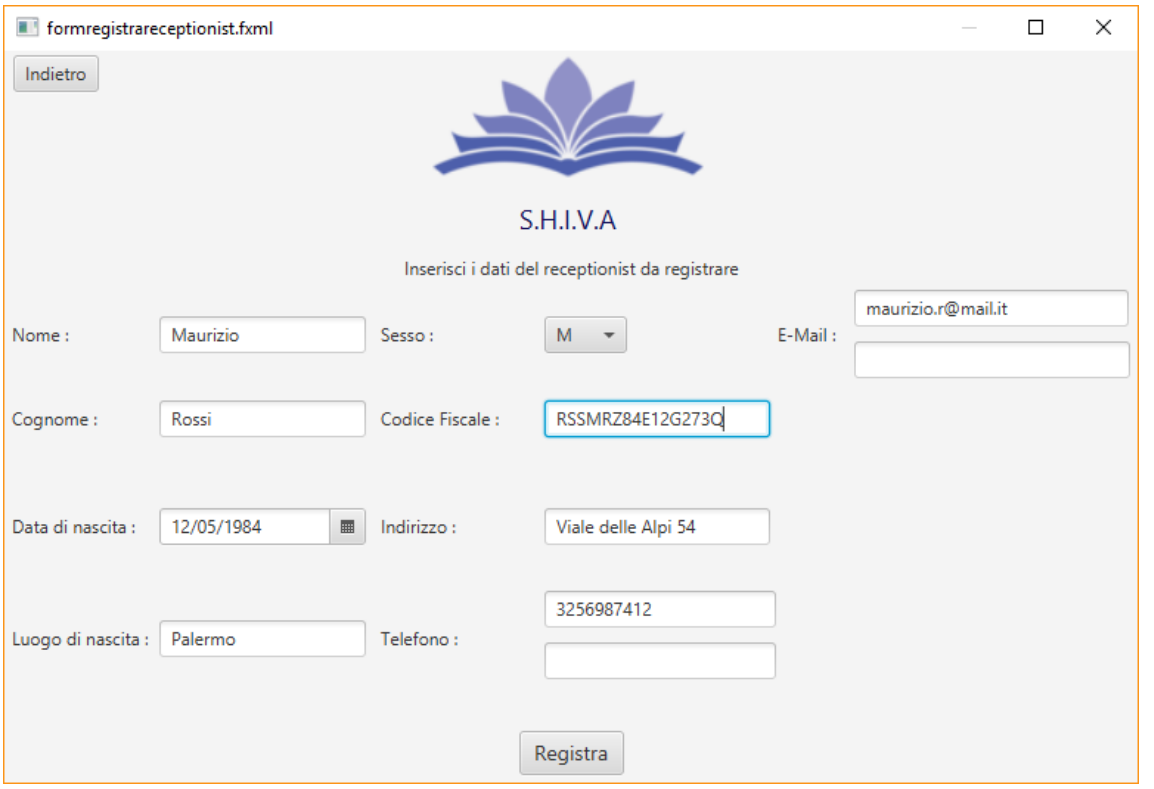


FIGURA 76 - FINESTRA DI REGISTRAZIONE RECEPTIONIST

3.5.6.19 ModificaUtente

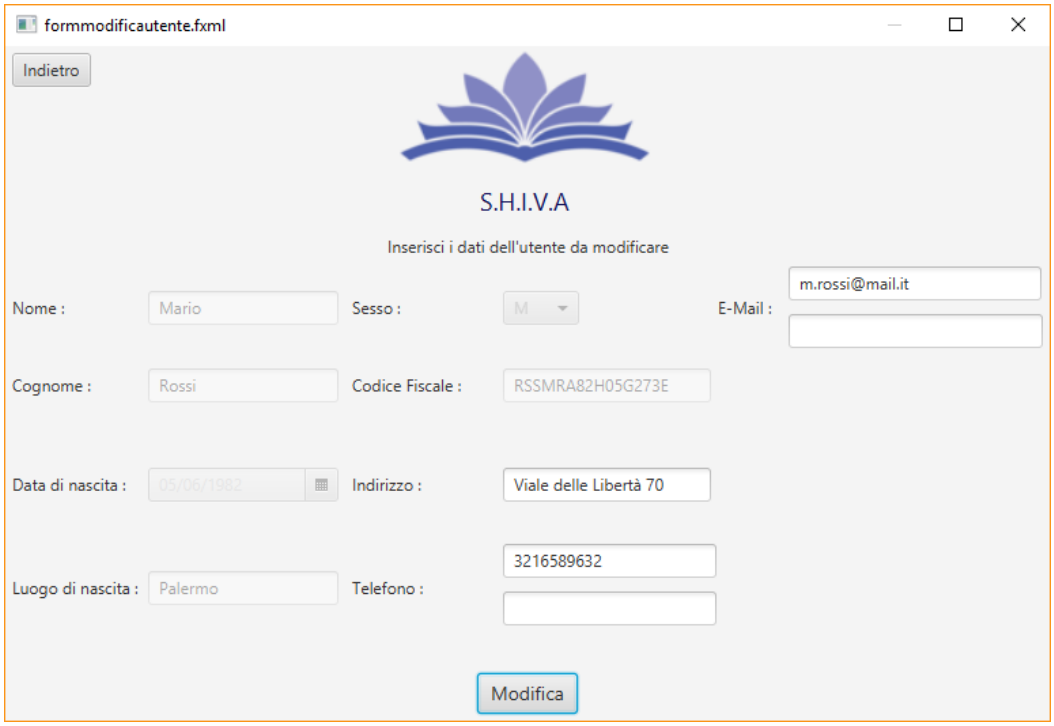


FIGURA 77 - FINESTRA DI MODIFICA UTENTE

3.5.6.20 RiepilogoUtente

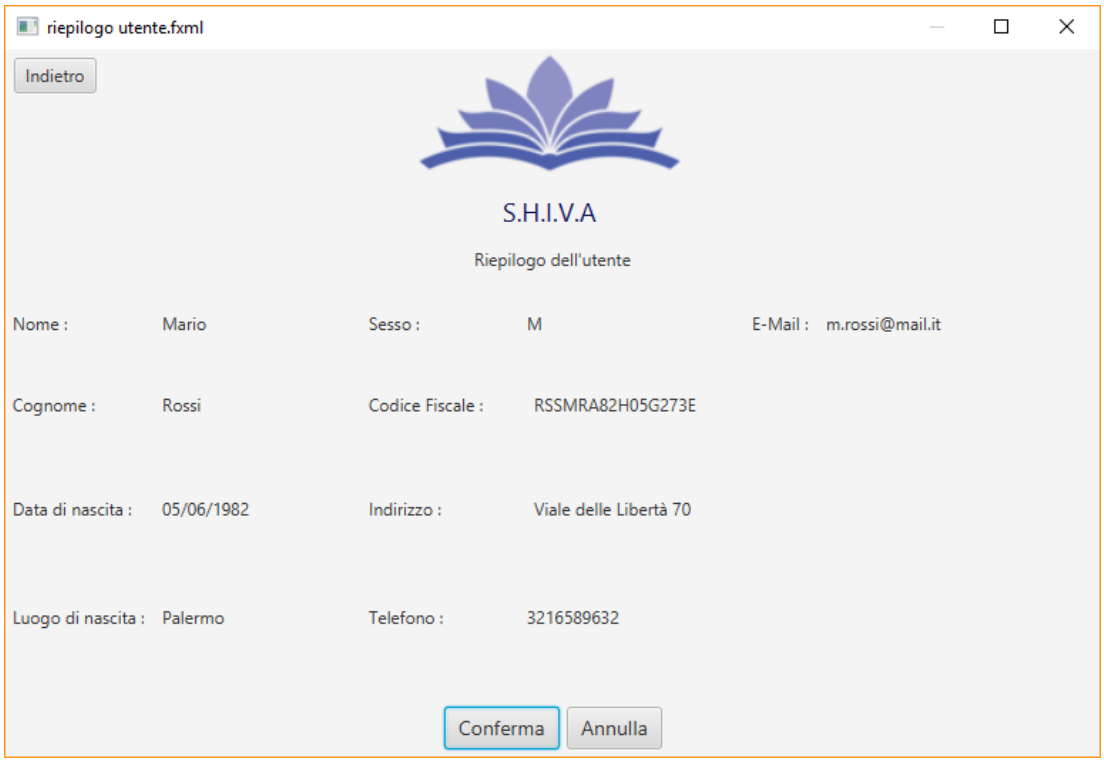


FIGURA 78 - FINESTRA DI RIEPILOGO UTENTE

3.5.6.21 RisultatiRicercaUtente

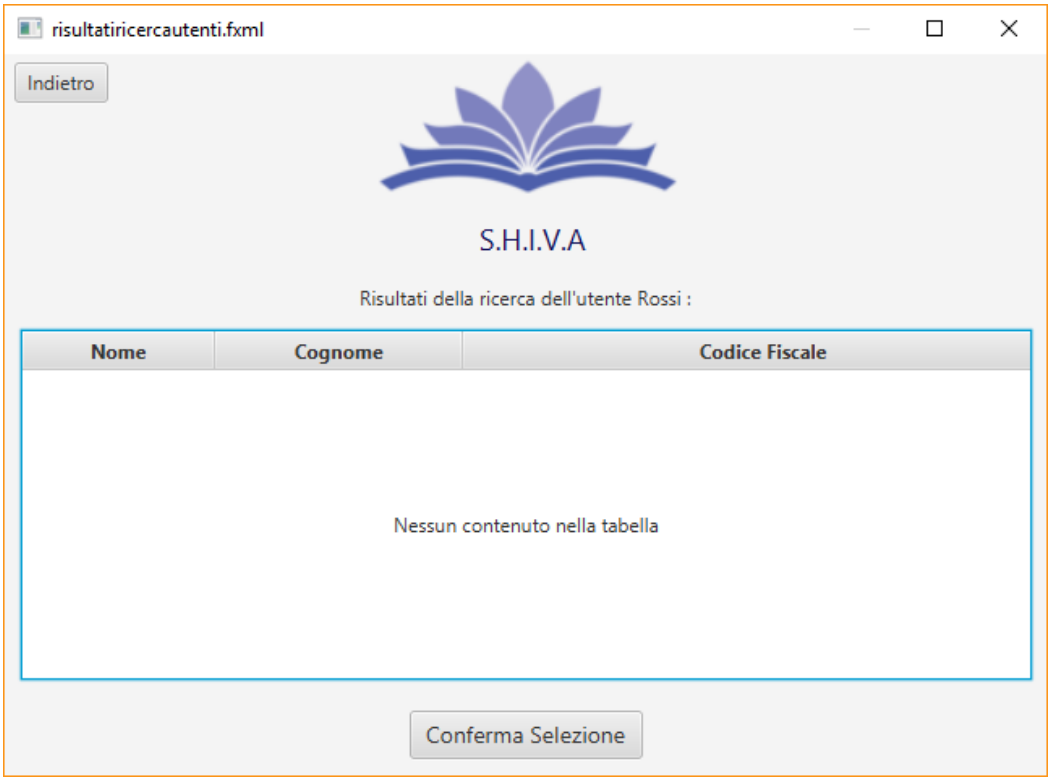


FIGURA 79 - FINESTRA DEI RISULTATI RICERCA UTENTE

3.5.6.22 DialogConferma

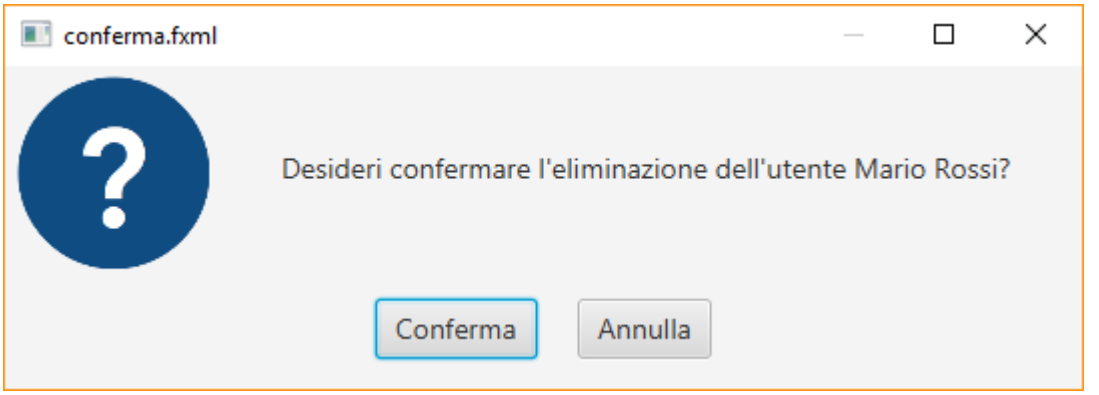


FIGURA 80 - FINESTRA DI CONFERMA

3.5.6.23 DialogErrore

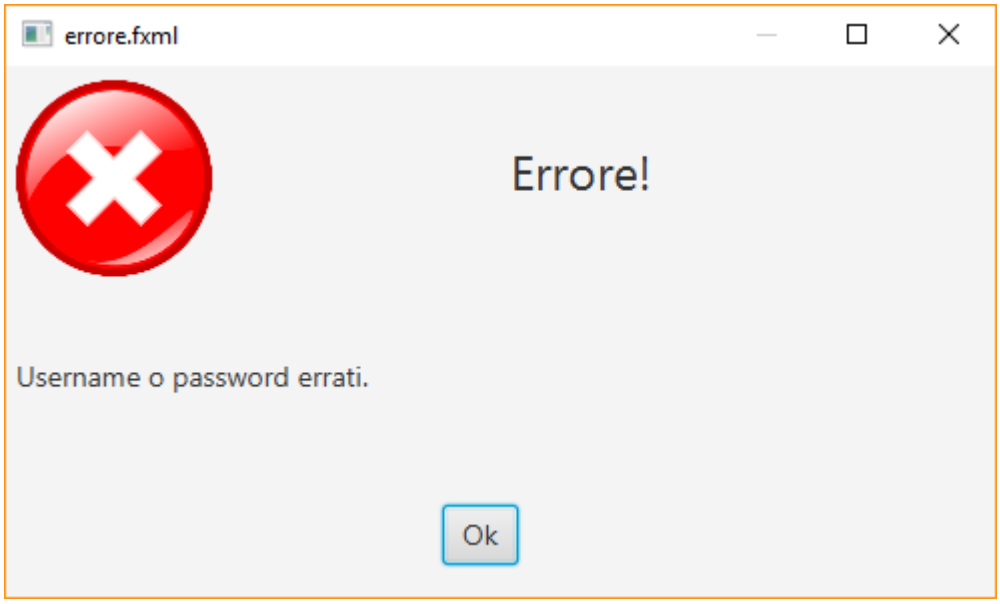


FIGURA 81 - FINESTRA DI ERRORE

3.5.6.24 DialogSuccesso

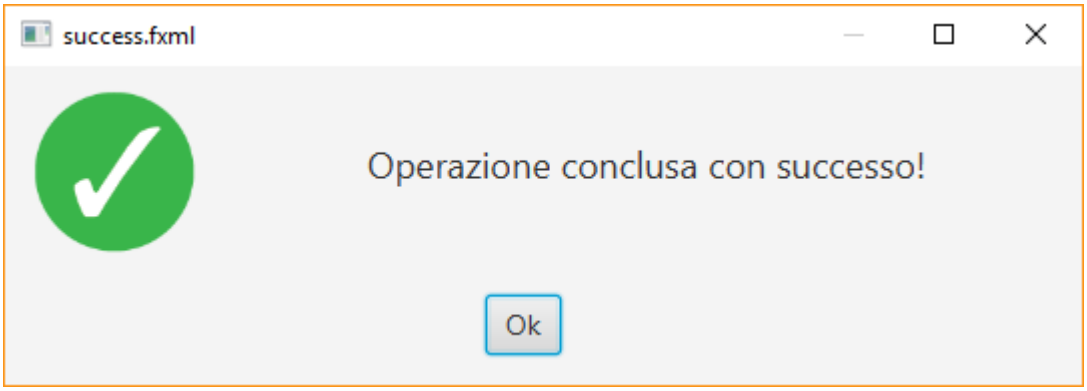


FIGURA 82 - FINESTRA DI SUCCESSO OPERAZIONE

3.5.7 Diagrammi di navigazione fra le schermate

La seguente tabella elenca le finestre del software e il corrispettivo identificatore:

ID	Nome	Descrizione
1	Login	Finestra che consente all'utente di loggarsi.
2	MainWindowReceptionist	Menù principale dell'utente Receptionist.
3	MainWindowMedico	Menù principale dell'utente Medico.
4	MainWindowAmministratore	Menù principale dell'amministratore.
5	ModificaPassword	Finestra usata dall'utente per modificare la propria password.
6	RegistraPaziente	Finestra usata dal Receptionist per inserire i dati del paziente.
7	RicercaPaziente	Finestra di ricerca del paziente.
8	RisultatiRicercaPaziente	Finestra che mostra i risultati della ricerca del paziente.
9	TipologiaVisita	Consente di selezionare il tipo di visita da prenotare.
10	SelezioneData	Permette di selezionare una delle date disponibili per una determinata tipologia di visita.
11	RiepilogoPrenotazione	Mostra il riepilogo della prenotazione.
12	ElencoVisite	Consente al medico di visualizzare la lista delle visite da effettuare.
13	ListaReport	Finestra che mostra la lista dei report per un determinato paziente.

14	DettagliReport	Mostra i dettagli di un determinato report.
15	CreaReport	Consente di inserire i dati del report.
16	RicercaUtente	Finestra con cui l'amministratore ricerca gli utenti.
17	RegistraMedico	Consente di compilare i dati del medico da registrare.
18	RegistraReceptionist	Consente di compilare i dati del receptionist da registrare.
19	ModificaUtente	Permette di modificare i dati di un utente
20	RiepilogoUtente	Finestra contenente un riepilogo dei dati dell'utente.
21	RisultatiRicercaUtente	Mostra i risultati della ricerca dell'utente.
22	DialogConferma	Finestra di dialogo che richiede la conferma per procedere con una determinata operazione.
23	DialogErrore	Finestra di dialogo che informa riguardo un errore nella operazione.
24	DialogSuccesso	Finestra di dialogo che informa riguardo il successo di una operazione.

3.5.7.1 Diagramma di navigazione fra le finestre Amministratore

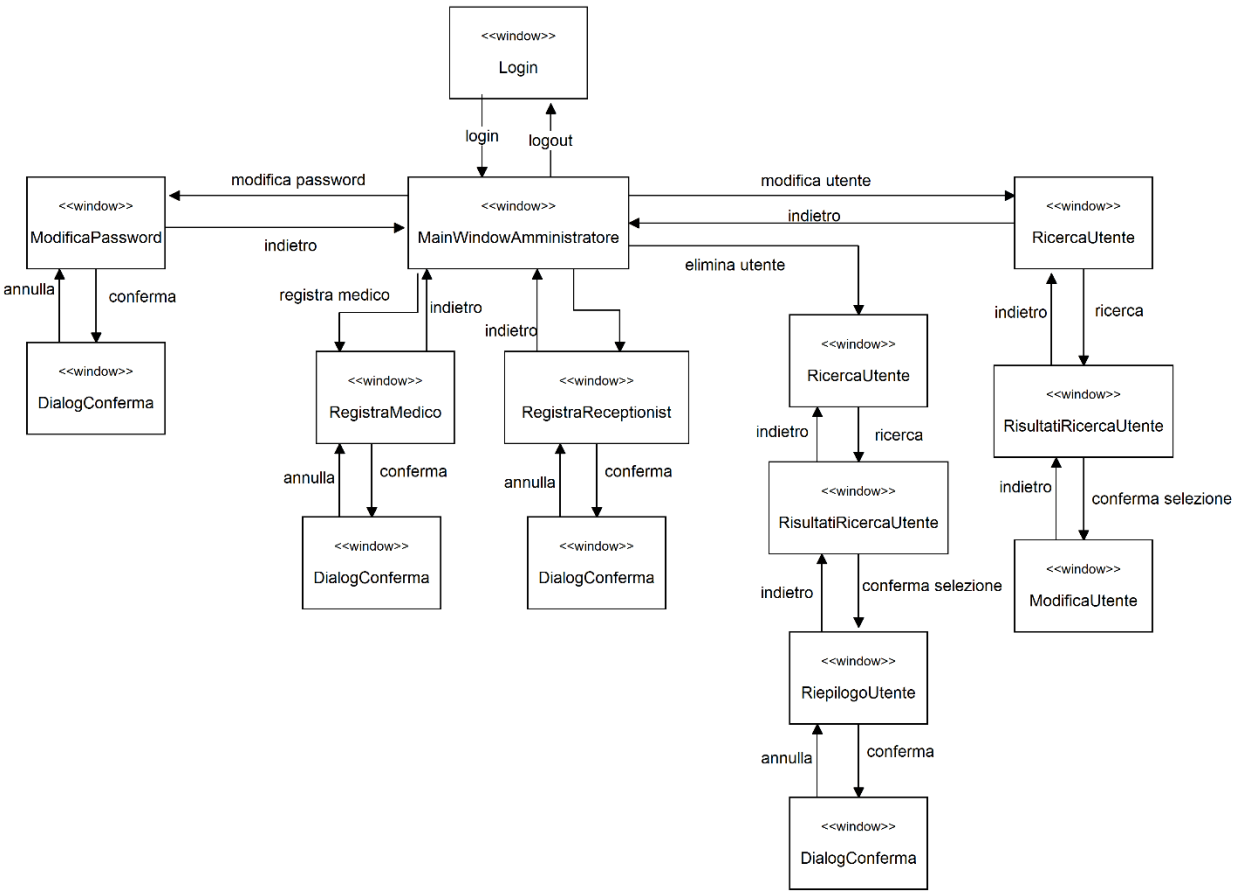


FIGURA 83 - DIAGRAMMA DI NAVIGAZIONE AMMINISTRATORE

3.5.7.2 Diagramma di navigazione fra le finestre Medico

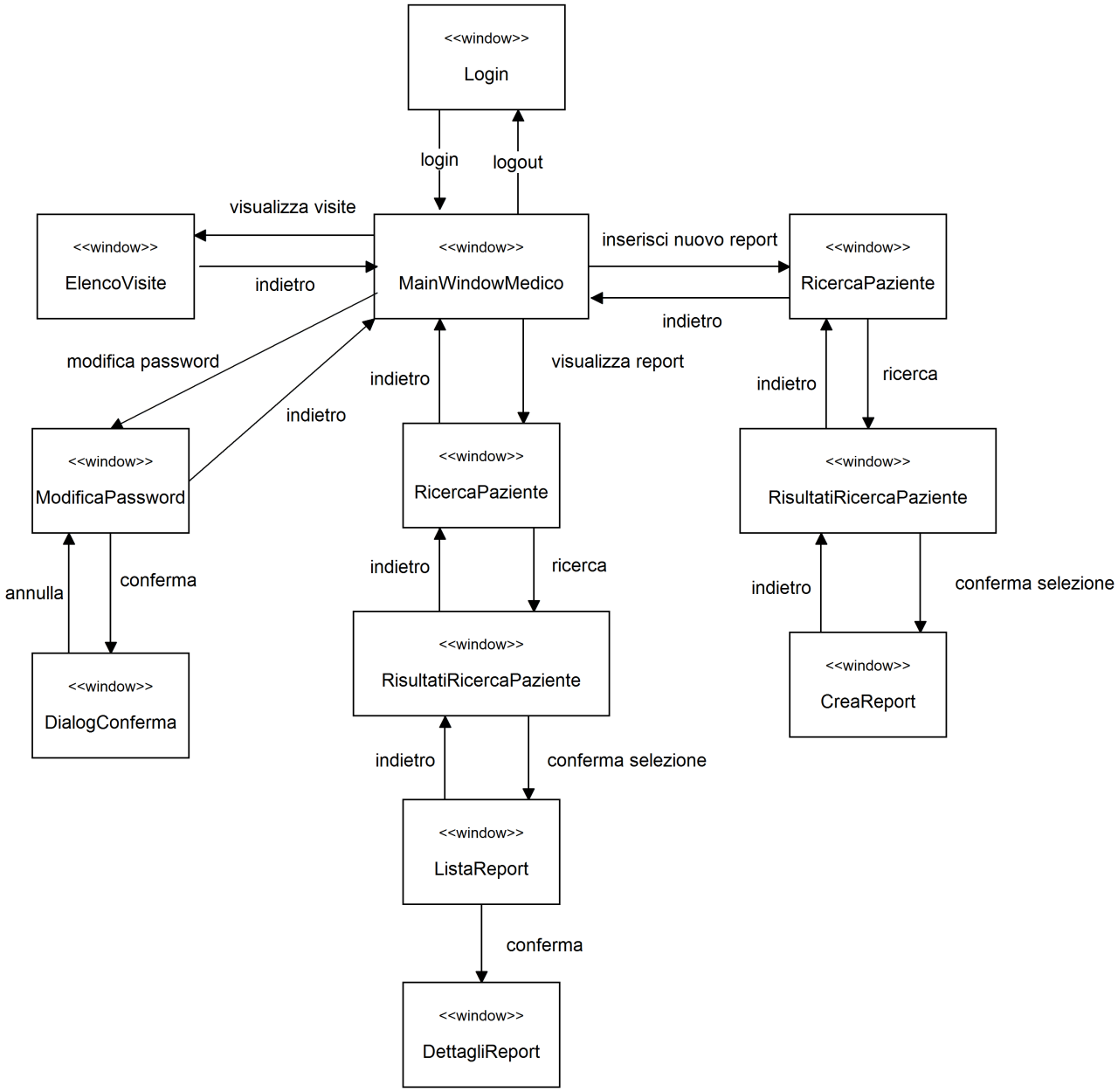


FIGURA 84 - DIAGRAMMA DI NAVIGAZIONE MEDICO

3.5.7.3 Diagramma di navigazione fra le finestre Receptionist

FIGURA 85 - DIAGRAMMA DI NAVIGAZIONE RECEPTIONIST

SDD

System Design Documen

Indice SDD

1. Introduzione

1.1 Scopo del sistema

Il sistema SHIVA ha come obbiettivo la semplificazione di numerose operazioni comunemente svolte all'interno di strutture mediche. Il sistema, che deve essere accessibile da diverse postazioni dislocate all'interno della struttura medica, è per sua natura distribuito, pertanto deve garantire centralizzazione e coerenza dei dati utilizzati.

1.2 Design Goals

Gli obbiettivi progettuali perseguiti durante questa fase sono stati:

- Affidabilità e coerenza dei dati
- Protezione dei dati sensibili
- Flessibilità al cambiamento
- Limitati requisiti hardware

1.3 Architettura software corrente

Al momento la struttura che ha commissionato SHIVA non è provvista di un sistema informatico che soddisfi le esigenze che SHIVA vuole soddisfare.

2. Architettura Software Proposta

Sulla base degli obbiettivi progettuali identificati e in funzione della natura intrinsecamente distribuita di SHIVA è stata selezionata un'architettura *Three-Tier*: Il sistema sarà identificato da un tier per l'interazione con l'utente (il client), un tier per la gestione degli accessi, delle richieste e la manipolazione dei dati (il server), e un tier per l'effettiva persistenza dei dati (il DBMS).

Questa scelta architetturale dà la possibilità di ottenere più facilmente un forte controllo sull'utilizzo e la manipolazione dei dati che fluiscono in SHIVA mantenendo però la natura distribuita e la possibilità di scegliere in futuro sistemi di persistenza dei dati alternativi.

A seguito della scelta di questa architettura sono stati identificati in questa fase di progetto nuovi sottosistemi (Comunicazione, MessaggiComunicazione, InterfacciaServer, OperazioniCentralizzate) necessari alla comunicazione tra client e server, e alla centralizzazione delle operazioni che verranno in seguito descritti.

2.1 Decomposizione in sottosistemi

Di seguito vengono descritti i sottosistemi nei quali è stato suddiviso SHIVA.

2.1.1 Sottosistema Client

Per migliorare la visibilità le relazioni sono state omesse. Verranno descritte più avanti.

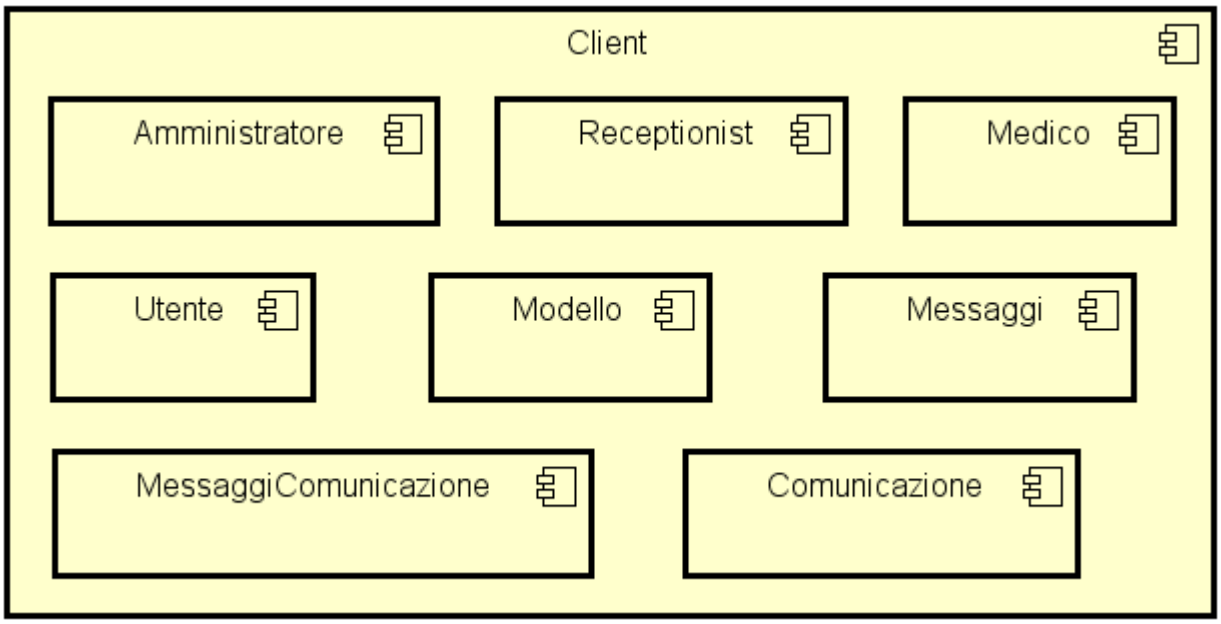


FIGURA 86 - SOTTOSISTEMA CLIENT

2.1.2 Sottosistema Server

Per migliorare la visibilità le relazioni sono state omesse. Verranno descritte più avanti.

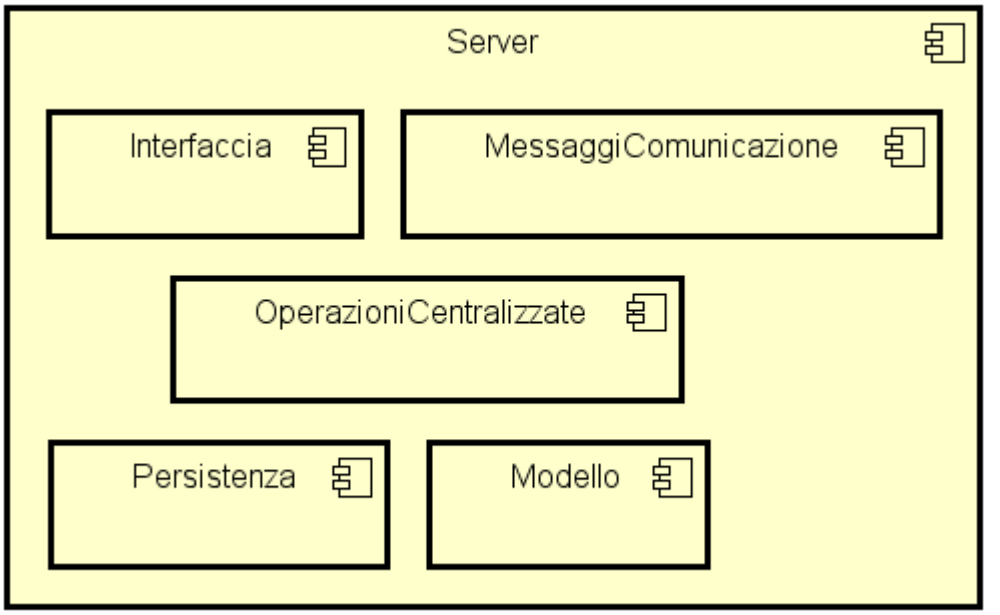


FIGURA 87 - SOTTOSISTEMA SERVER

2.1.1 Package Amministratore

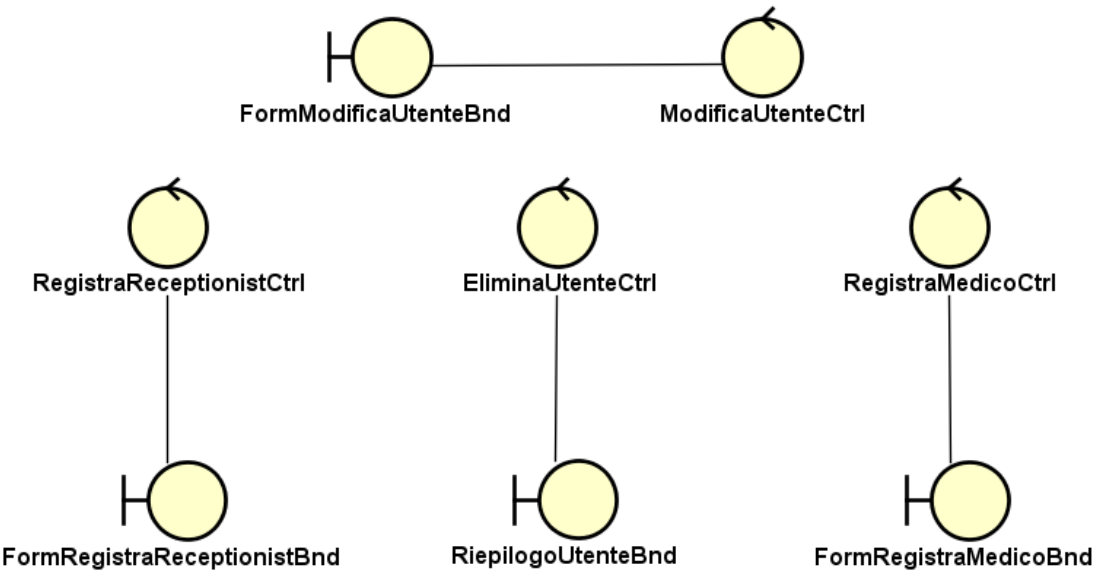


FIGURA 88 - PACKAGE AMMINISTRATORE

2.1.3 Package Comunicazione

Questo package è stato identificato e aggiunto durante questa fase al fine di consentire al client di comunicare con il server. È stato strutturato in maniera da adattarsi in un ipotetico futuro ad una scelta di canale di comunicazione diversa.

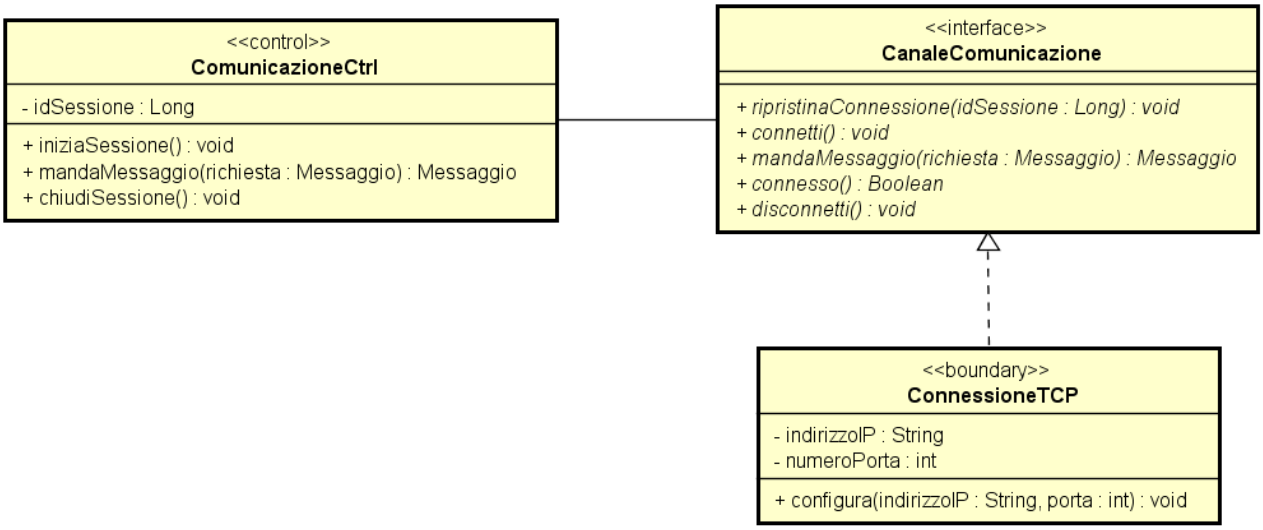


FIGURA 89 - PACKAGE COMUNICAZIONE

2.1.4 Package InterfacciaServer

Questo package è stato identificato e aggiunto durante questa fase al fine di fornire al server un'interfaccia per la comunicazione tra client e server. È stato strutturato in maniera da adattarsi in un ipotetico futuro ad una scelta di canale di comunicazione diversa.

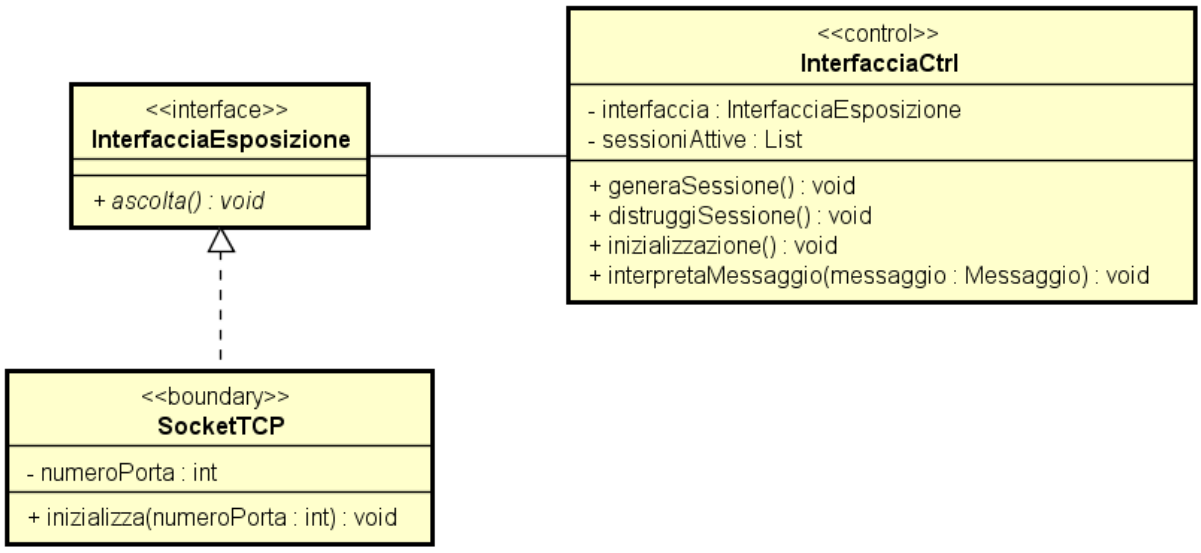


FIGURA 90 - PACKAGE INTERFACCIASERVER

2.1.4 Package Medico

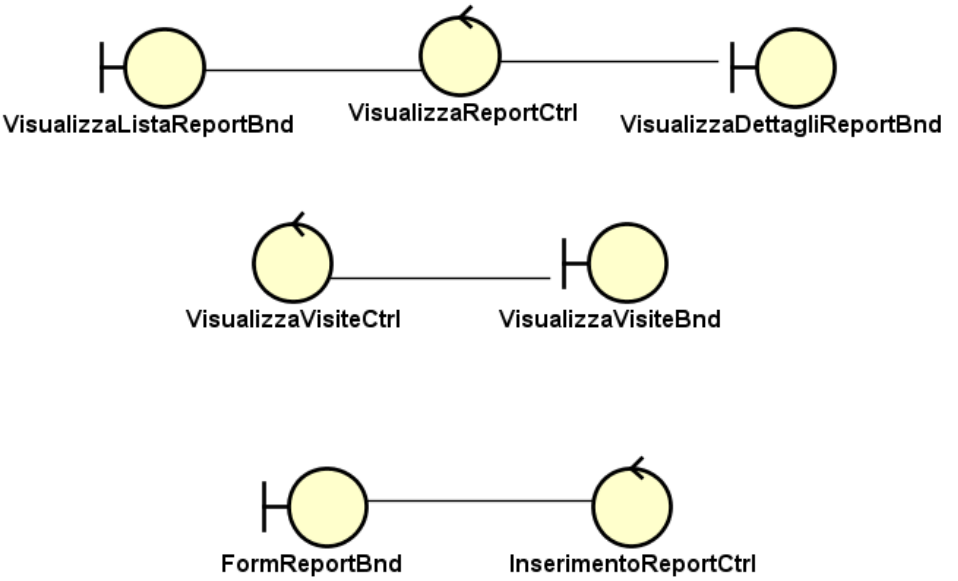


FIGURA 91 - PACKAGE MEDICO

2.1.5 Package Messaggi

FIGURA 92 - PACKAGE MESSAGGI

2.1.5 Package MessaggiComunicazione

Questo package è stato identificato e aggiunto durante questa fase al fine di definire i messaggi utilizzati durante la comunicazione tra client e server.

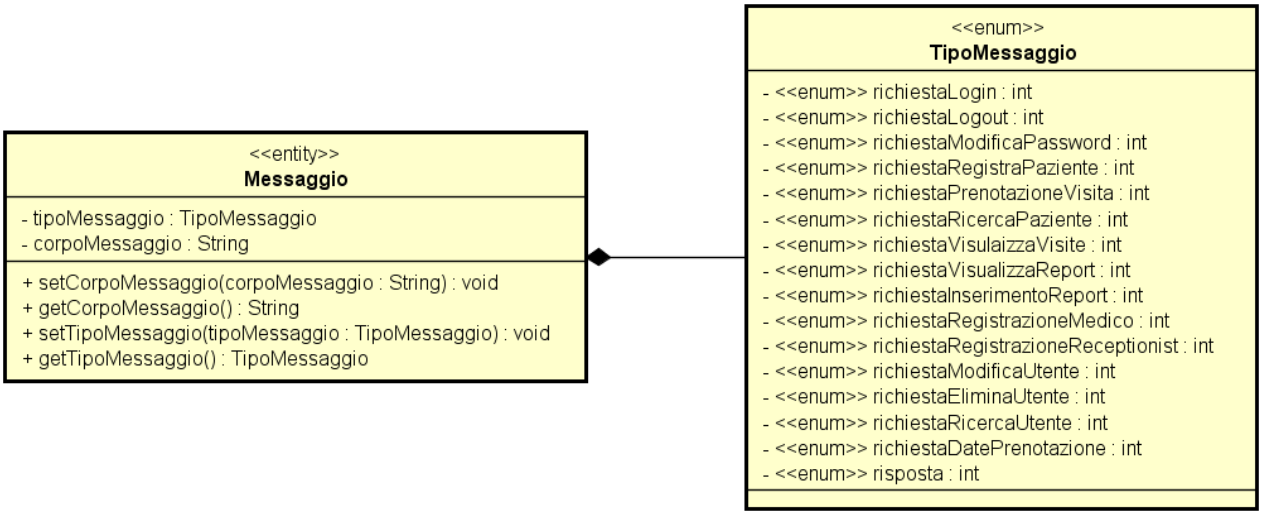


FIGURA 93 - PACKAGE MESSAGGICOMUNICAZIONE

2.1.7 Package Modello

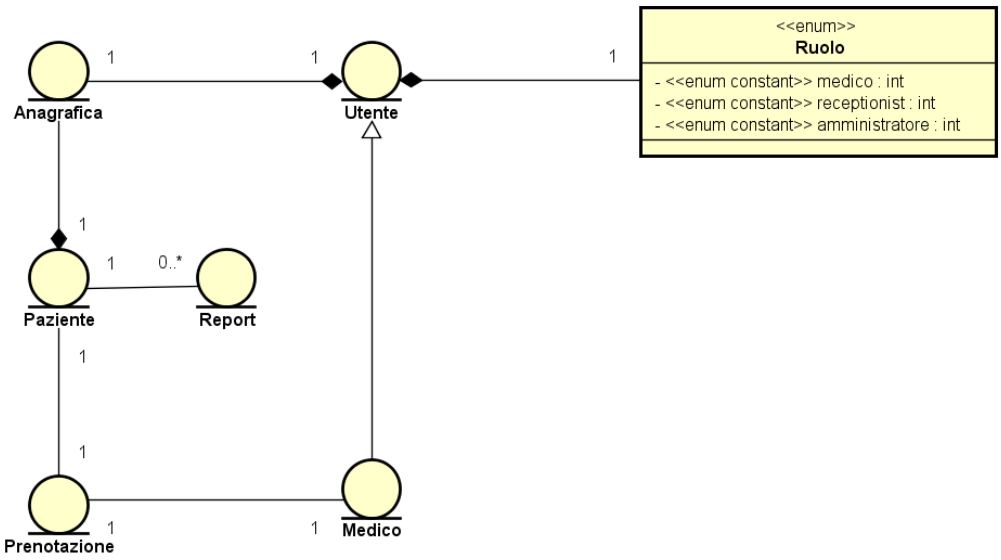


FIGURA 94 - PACKAGE MODELLO

2.1.8 Package OperazioniCentralizzate

Questo package è stato identificato e aggiunto durante questa fase al fine di definire il modulo responsabile di realizzare sul server le operazioni richieste dai client in maniera centralizzata.

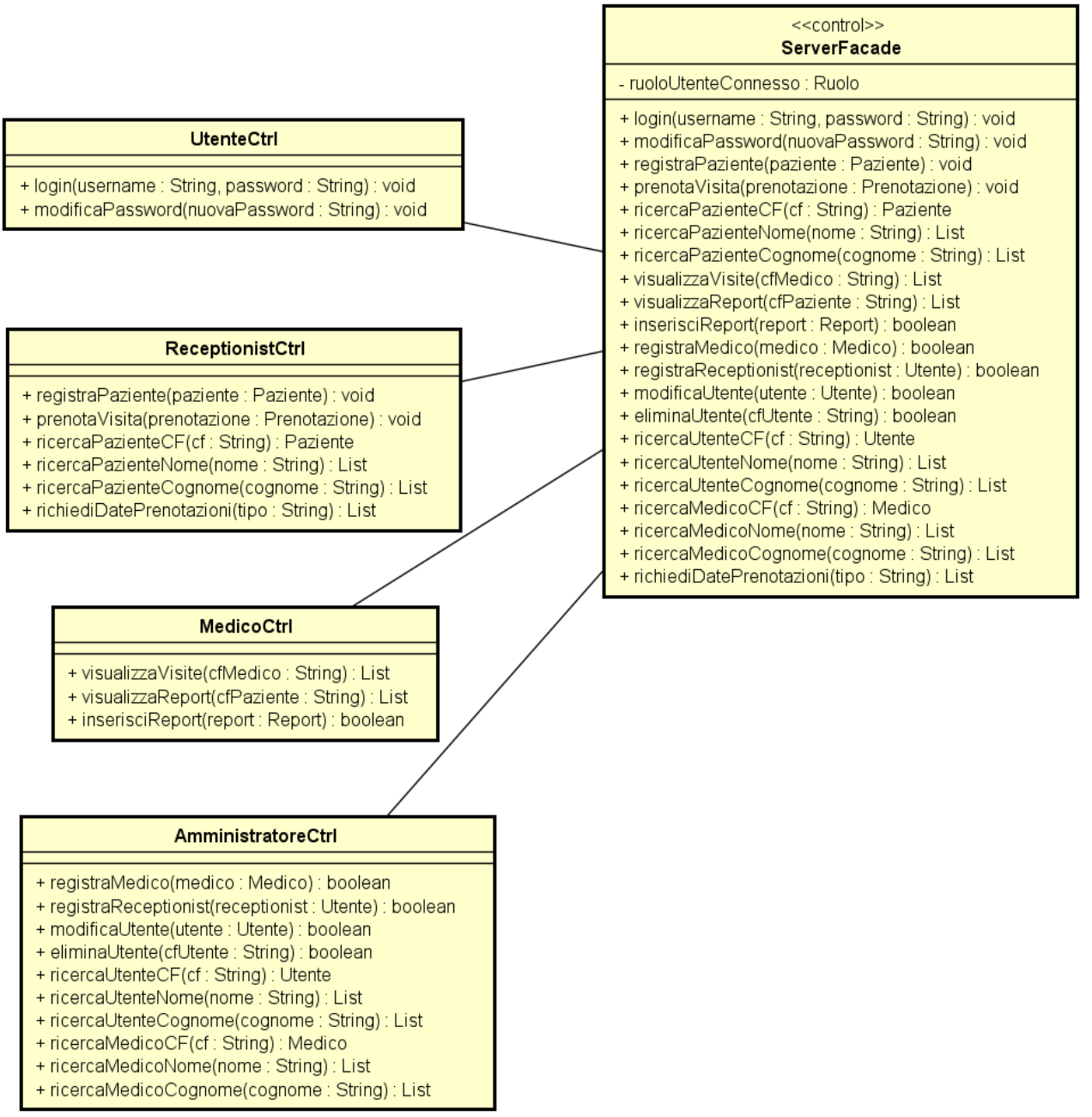


FIGURA 95 - PACKAGE OPERAZIONICENTRALIZZATE

2.1.9 Package Persistenza

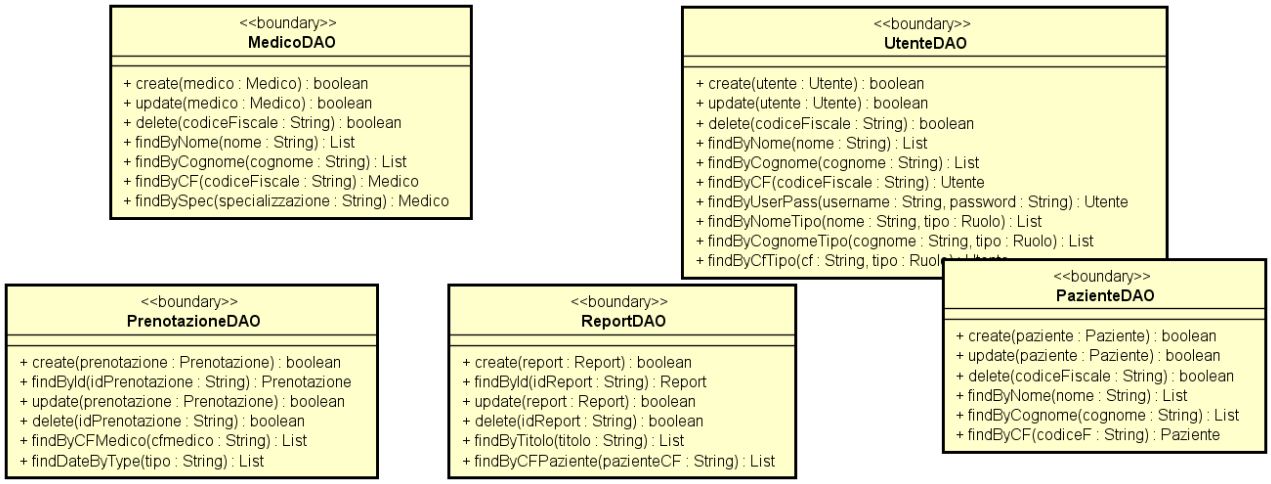


FIGURA 96 - PACKAGE PERSISTENZA

2.1.10 Package Receptionist

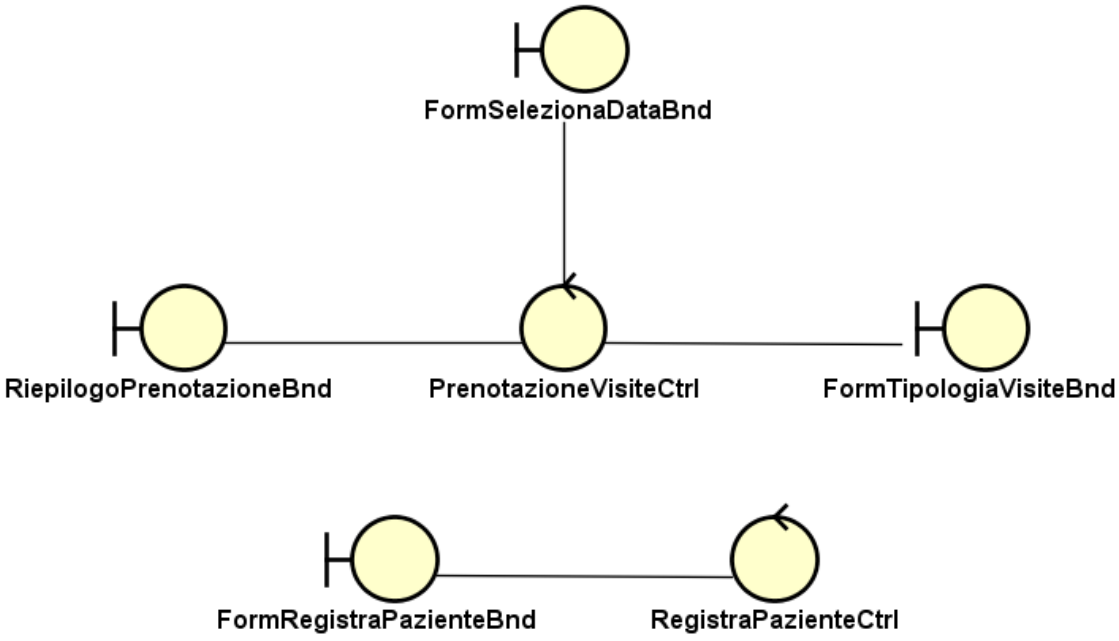


FIGURA 97 - PACKAGE RECEPTIONIST

2.1.11 Package RicercaPaziente

FIGURA 98 - PASCKAGE RICERCAPAZIENTE

2.1.12 Package RicercaUtente

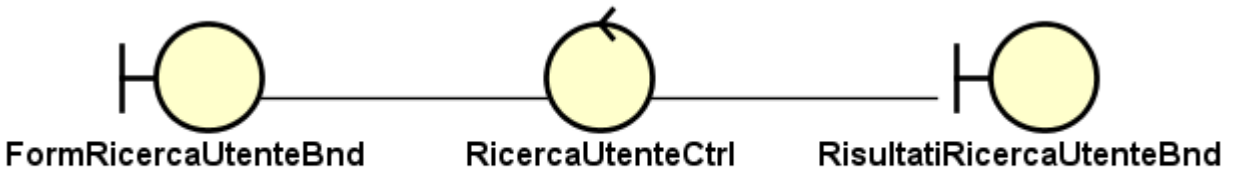


FIGURA 99 - PACKAGE RICERCAUTENTE

2.1.13 Package Utente

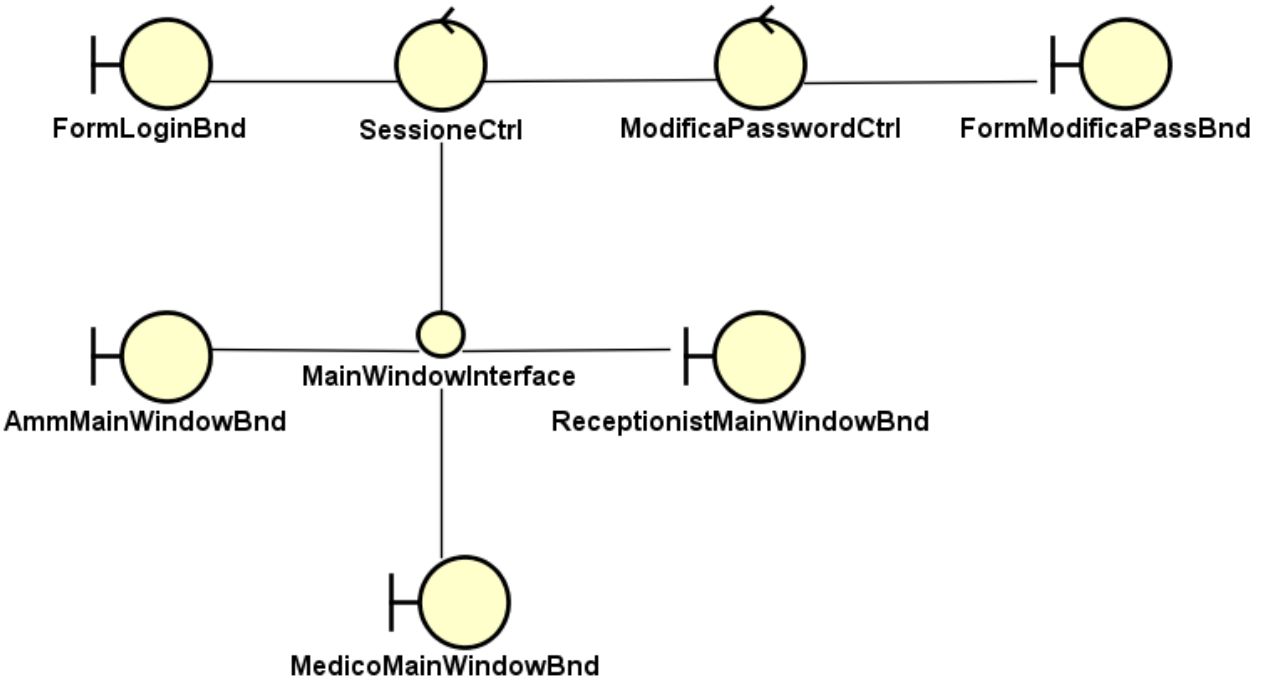


FIGURA 100 - PACKAGE UTENTE

2.1.14 Diagramma dei componenti: Amministratore

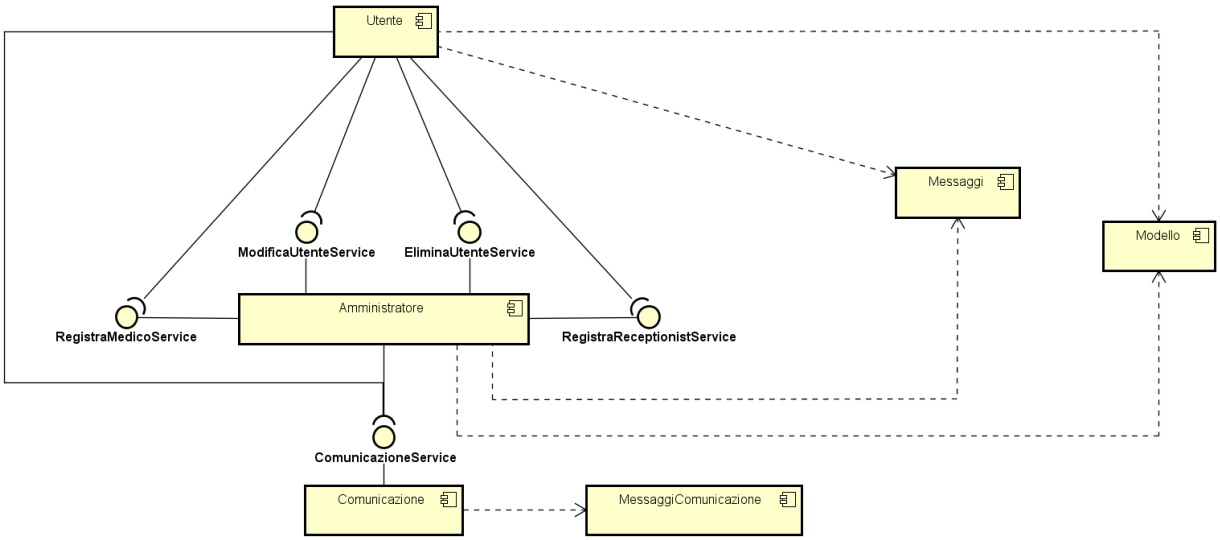


FIGURA 101 - DIAGRAMMA DEI COMPONENTI: AMMINISTRATORE

2.1.15 Diagramma dei componenti: Medico

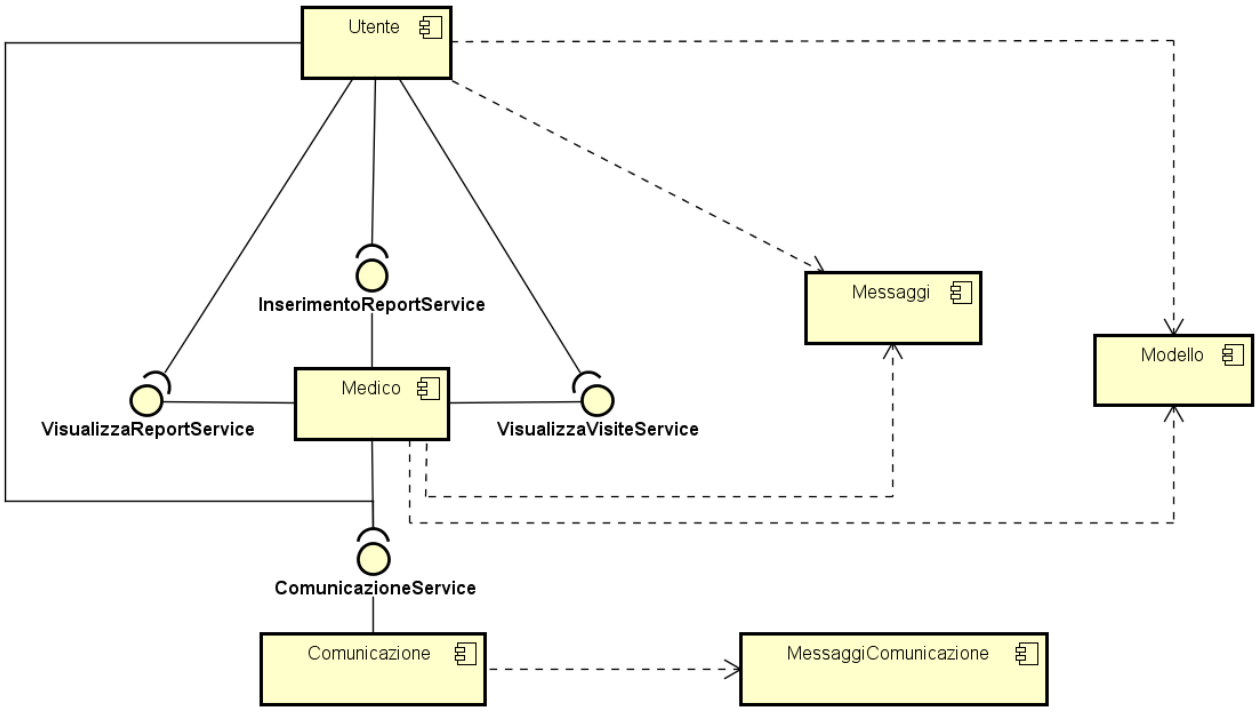


FIGURA 102 - DIAGRAMMA DEI COMPONENTI: MEDICO

2.1.16 Diagramma dei componenti: Receptionist

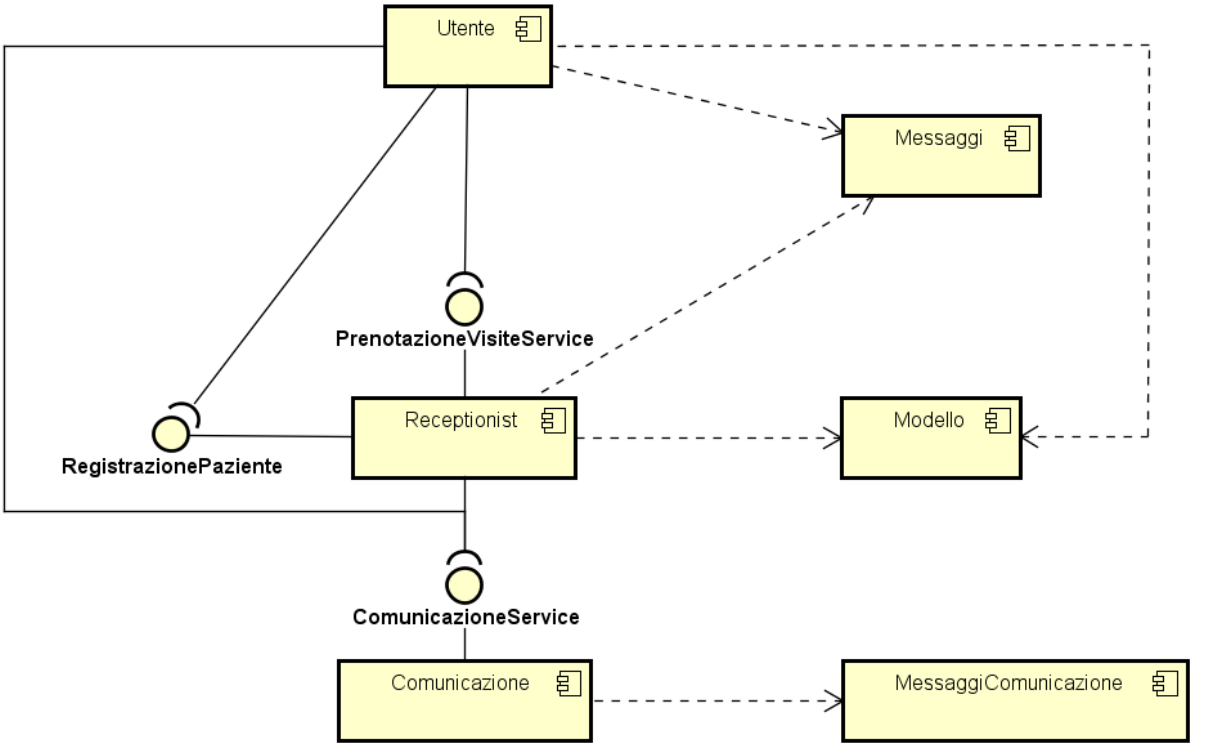


FIGURA 103 - DIAGRAMMA DEI COMPONENTI: RECEPTIONIST

2.1.17 Diagramma dei componenti: Server

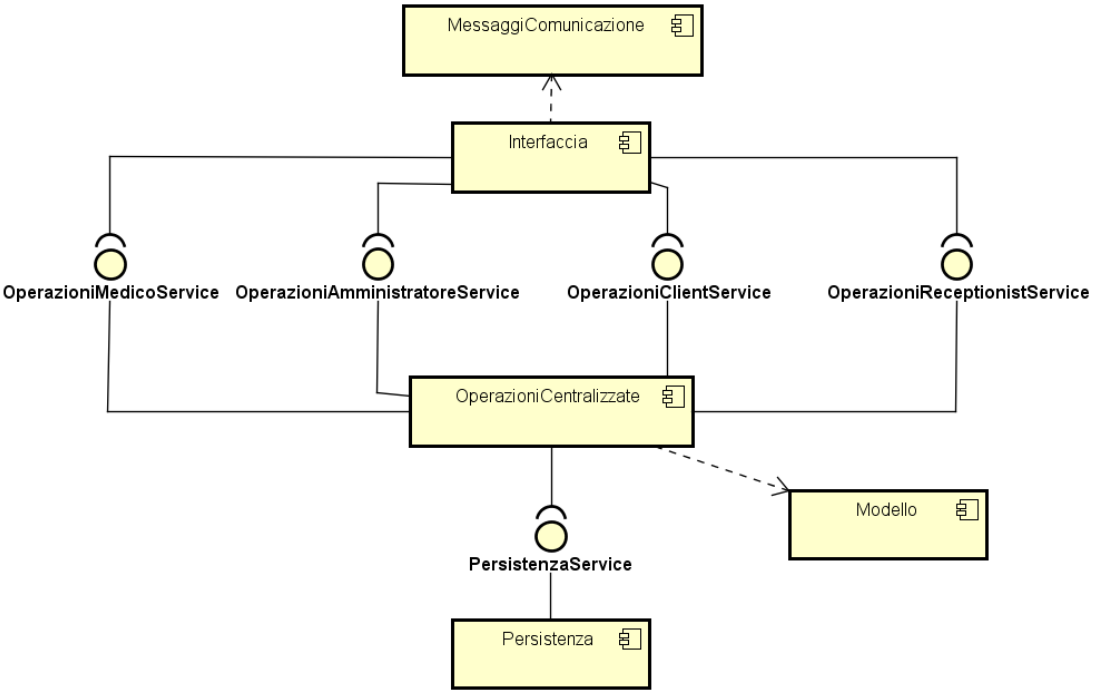


FIGURA 104 - DIAGRAMMA DEI COMPONENTI: SERVER

3. Hardware/software mapping

Il sistema proposto è stato sviluppato in JAVA e quindi risulta indipendente dal sistema operativo presente nelle postazioni utilizzate.

Il software progettato non presenta particolari richieste prestazionali a livello hardware e pertanto qualsiasi postazione a livello consumer risulta adatta. L'unica eccezione può presentarsi per la PostazioneServer nel caso di un numero elevato di client, caso in cui l'hardware della postazione dev'essere valutato opportunamente per evitare un eventuale collo di bottiglia nel sistema.

3.1 Diagramma di Deployment

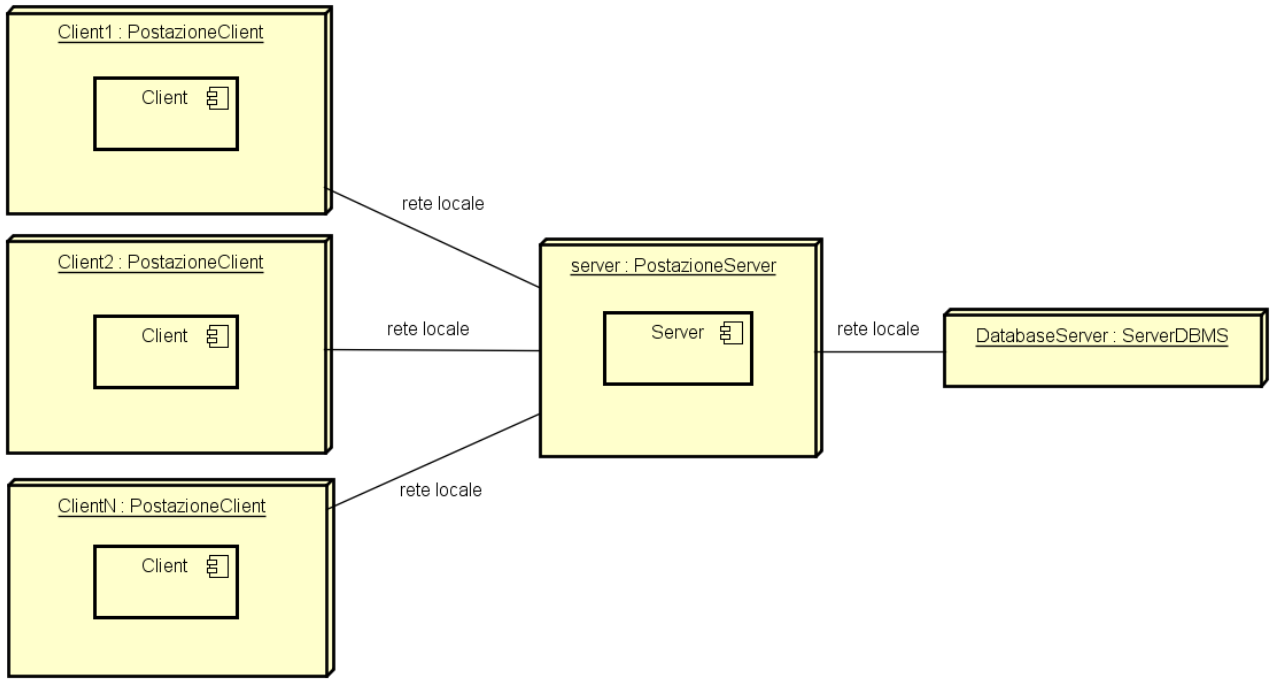


FIGURA 105 - DIAGRAMMA DI DEPLOYMENT

4. Gestione dei dati persistenti

4.1 Progetto logico del Database

Di seguito si riporta la struttura logica del Database, necessario a SHIVA per persistere le informazioni gestite, ottenuta come conseguenza dell'analisi progettuale del Database effettuata nel RAD.

La tipologia di Database scelta è quella relazionale.

Il seguente diagramma è stati ottenuto utilizzando il software MySQL Workbench.

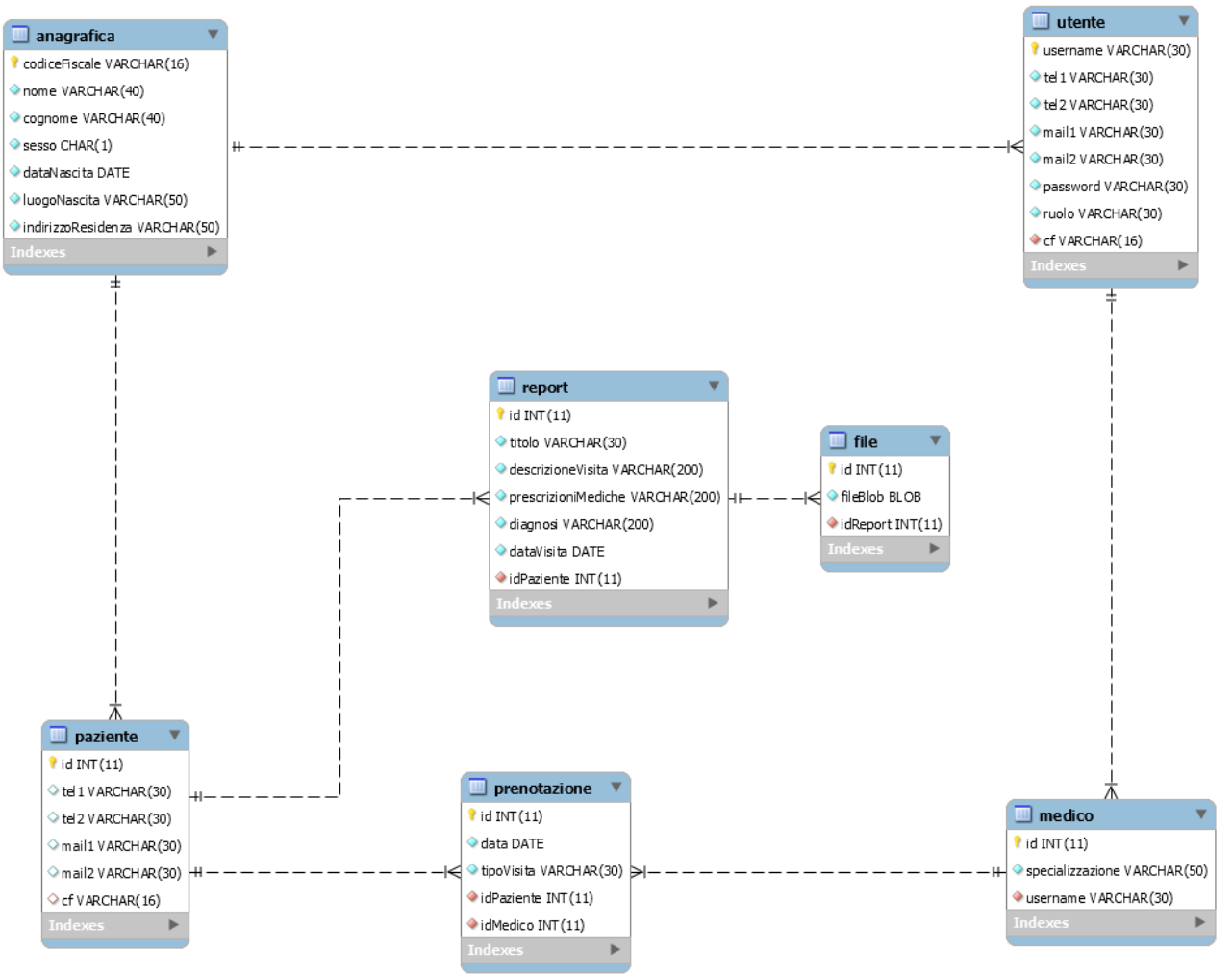


FIGURA 106 - DIAGRAMMA DATABASE

4.2 Script di creazione del Database

Di seguito viene riportato lo script per la creazione del database.

```
CREATE DATABASE  IF NOT EXISTS `shiva` /*!40100 DEFAULT CHARACTER SET utf8 */;
USE `shiva`;
-- MySQL 10.13  Distrib 5.7.9, for Win64 (x86_64)
--
-- Host: localhost    Database: shiva
-- ------------------------------------------------------
-- Server version   5.6.28-log

/*!40101 SET @OLD_CHARACTER_SET_CLIENT=@@CHARACTER_SET_CLIENT */;
/*!40101 SET @OLD_CHARACTER_SET_RESULTS=@@CHARACTER_SET_RESULTS */;
/*!40101 SET @OLD_COLLATION_CONNECTION=@@COLLATION_CONNECTION */;
/*!40101 SET NAMES utf8 */;
/*!40103 SET @OLD_TIME_ZONE=@@TIME_ZONE */;
/*!40103 SET TIME_ZONE='+00:00' */;
/*!40014 SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0 */;
/*!40014 SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS, FOREIGN_KEY_CHECKS=0 */;
/*!40101 SET @OLD_SQL_MODE=@@SQL_MODE, SQL_MODE='NO_AUTO_VALUE_ON_ZERO' */;
/*!40111 SET @OLD_SQL_NOTES=@@SQL_NOTES, SQL_NOTES=0 */;

--
-- Table structure for table `anagrafica`
--

DROP TABLE IF EXISTS `anagrafica`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
/*!40101 SET character_set_client = utf8 */;
CREATE TABLE `anagrafica` (
  `codiceFiscale` varchar(16) NOT NULL,
  `nome` varchar(40) NOT NULL,
  `cognome` varchar(40) NOT NULL,
  `sesso` char(1) NOT NULL,
  `dataNascita` date NOT NULL,
  `luogoNascita` varchar(50) NOT NULL,
  `indirizzoResidenza` varchar(50) NOT NULL,
  PRIMARY KEY (`codiceFiscale`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
/*!40101 SET character_set_client = @saved_cs_client */;

--
-- Table structure for table `file`
--

DROP TABLE IF EXISTS `file`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
/*!40101 SET character_set_client = utf8 */;
CREATE TABLE `file` (
  `id` int(11) NOT NULL,
  `fileBlob` blob NOT NULL,
  `idReport` int(11) NOT NULL,
  PRIMARY KEY (`id`),
  KEY `idReportFile_idx` (`idReport`),
  CONSTRAINT `idReportFile` FOREIGN KEY (`idReport`) REFERENCES `report` (`id`) ON DELETE NO ACTION ON UPDATE NO ACTION
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
/*!40101 SET character_set_client = @saved_cs_client */;

--
-- Table structure for table `medico`
--

DROP TABLE IF EXISTS `medico`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
/*!40101 SET character_set_client = utf8 */;
CREATE TABLE `medico` (
  `id` int(11) NOT NULL,
  `specializzazione` varchar(50) NOT NULL,
  `username` varchar(30) NOT NULL,
  PRIMARY KEY (`id`),
  KEY `usernameMedico_idx` (`username`),
  CONSTRAINT `usernameMedico` FOREIGN KEY (`username`) REFERENCES `utente` (`username`) ON DELETE NO ACTION ON UPDATE NO ACTION
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
/*!40101 SET character_set_client = @saved_cs_client */;
```

```
--
-- Table structure for table `paziente`
--

DROP TABLE IF EXISTS `paziente`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
/*!40101 SET character_set_client = utf8 */;
CREATE TABLE `paziente` (
  `id` int(11) NOT NULL,
  `tel1` varchar(30) DEFAULT NULL,
  `tel2` varchar(30) DEFAULT NULL,
  `mail1` varchar(30) DEFAULT NULL,
  `mail2` varchar(30) DEFAULT NULL,
  `cf` varchar(16) DEFAULT NULL,
  PRIMARY KEY (`id`),
  KEY `cf_idx` (`cf`),
  CONSTRAINT `cf` FOREIGN KEY (`cf`) REFERENCES `anagrafica` (`codiceFiscale`) ON DELETE NO ACTION ON UPDATE NO ACTION
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
/*!40101 SET character_set_client = @saved_cs_client */;

--
-- Table structure for table `prenotazione`
--

DROP TABLE IF EXISTS `prenotazione`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
/*!40101 SET character_set_client = utf8 */;
CREATE TABLE `prenotazione` (
  `id` int(11) NOT NULL,
  `data` date NOT NULL,
  `tipoVisita` varchar(30) NOT NULL,
  `idPaziente` int(11) NOT NULL,
  `idMedico` int(11) NOT NULL,
  PRIMARY KEY (`id`),
  KEY `idPazPren_idx` (`idPaziente`),
  KEY `idMedPren_idx` (`idMedico`),
  CONSTRAINT `idMedPren` FOREIGN KEY (`idMedico`) REFERENCES `medico` (`id`) ON DELETE NO ACTION ON UPDATE NO ACTION,
  CONSTRAINT `idPazPren` FOREIGN KEY (`idPaziente`) REFERENCES `paziente` (`id`) ON DELETE NO ACTION ON UPDATE NO ACTION
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
/*!40101 SET character_set_client = @saved_cs_client */;

--
-- Table structure for table `report`
--

DROP TABLE IF EXISTS `report`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
/*!40101 SET character_set_client = utf8 */;
CREATE TABLE `report` (
  `id` int(11) NOT NULL,
  `titolo` varchar(30) NOT NULL,
  `descrizioneVisita` varchar(200) NOT NULL,
  `prescrizioniMediche` varchar(200) NOT NULL,
  `diagnosi` varchar(200) NOT NULL,
  `dataVisita` date NOT NULL,
  `idPaziente` int(11) NOT NULL,
  PRIMARY KEY (`id`),
  KEY `idPazReport_idx` (`idPaziente`),
  CONSTRAINT `idPazReport` FOREIGN KEY (`idPaziente`) REFERENCES `paziente` (`id`) ON DELETE NO ACTION ON UPDATE NO ACTION
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
/*!40101 SET character_set_client = @saved_cs_client */;

--
-- Table structure for table `utente`
--

DROP TABLE IF EXISTS `utente`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
/*!40101 SET character_set_client = utf8 */;
CREATE TABLE `utente` (
  `username` varchar(30) NOT NULL,
  `tel1` varchar(30) NOT NULL,
  `tel2` varchar(30) NOT NULL,
  `mail1` varchar(30) NOT NULL,
  `mail2` varchar(30) NOT NULL,
  `password` varchar(30) NOT NULL,
  `ruolo` varchar(30) NOT NULL,
  `cf` varchar(16) NOT NULL,
  PRIMARY KEY (`username`),
  KEY `cf_idx` (`cf`),
  CONSTRAINT `cfutente` FOREIGN KEY (`cf`) REFERENCES `anagrafica` (`codiceFiscale`) ON DELETE NO ACTION ON UPDATE NO ACTION
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
/*!40101 SET character_set_client = @saved_cs_client */;


/*!40101 SET SQL_MODE=@OLD_SQL_MODE */;
/*!40014 SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS */;
/*!40014 SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS */;
/*!40101 SET CHARACTER_SET_CLIENT=@OLD_CHARACTER_SET_CLIENT */;
/*!40101 SET CHARACTER_SET_RESULTS=@OLD_CHARACTER_SET_RESULTS */;
/*!40101 SET COLLATION_CONNECTION=@OLD_COLLATION_CONNECTION */;
/*!40111 SET SQL_NOTES=@OLD_SQL_NOTES */;
```

5. Controllo degli accessi e sicurezza

Le funzionalità offerte agli utenti di SHIVA sono differenti a seconda del ruolo che svolgono all'interno della struttura medica. Il sistema SHIVA per altro contiene al suo interno informazioni sensibili che vanno tutelate. Emergono quindi le seguenti problematiche inerenti agli accessi al sistema e alla sicurezza dei dati:

- L'accesso al sistema dev'essere garantito solo ad utenti riconosciuti dalla struttura medica.
- A categorie di utenti diverse il sistema offre funzionalità differenti. Le operazioni garantite ad una particolare categoria di utenti non devono poter essere accessibili alle altre. (Es. un Medico non può avere la possibilità di creare e/o modificare account utente). Nello specifico:
    - Un utente Receptionist può effettuare le seguenti operazioni:
        - Registrazione di un paziente
        - Prenotazione di una visita
    - Un utente Medico può effettuare le seguenti operazioni:
        - Visualizzazione delle visite assegnategli
        - Visualizzazione dei report assegnati ad un paziente
        - Creazione e assegnazione di un nuovo report ad un paziente
    - Un Amministratore di sistema può effettuare le seguenti operazioni:
        - Creare un nuovo account utente di tipo Medico o Receptionist
        - Modificare un account utente di tipo Medico o Receptionist
        - Eliminare un account utente di tipo Medico o Receptionist
    - In più ad ogni utente è garantita la possibilità di modificare la propria password.
- I dati sensibili registrati all'interno del sistema SHIVA devono essere tutelati.

Per risolvere queste problematiche a livello implementativo è stato pensato un sistema di controllo a vari livelli:

- L'accesso al sistema avviene tramite credenziali attraverso le quali il sistema identifica se un utente è autorizzato o no, e nel caso sia autorizzato che ruolo svolge all'interno della struttura medica.
- In funzione del ruolo riconosciuto all'utente viene presentata un'interfaccia grafica diversa (che contiene solo le voci relative alle operazioni consentite a quel particolare ruolo).
- Il server, per ogni richiesta ricevuta da un client, effettua un controllo per vedere se il ruolo dell'utente gli garantisce accesso alla funzionalità richiesta.

- Il server centralizza e gestisce tutte le manipolazioni dei dati persistenti registrati all'interno di SHIVA. Pertanto è l'unico ad avere accesso garantito al DBMS.

Nonostante alcuni dei livelli di controllo siano ridondanti, garantiscono la protezione del sistema anche in caso di manipolazione malevola dei pacchetti inviati al server attraverso la rete e/o di reverse engineering e manipolazione del codice del client

ODD

Object Design Document

Indice ODD

1 Codice Sorgente

1.1 Client

1.1.1 Package Amministratore

FIGURA 107 - PACKAGE AMMINISTRATORE

1.1.1.1 Classe Elimina Utente

```
package SHIVA.client.amministratore;

import SHIVA.client.modello.Utente;
import SHIVA.server.persistenza.UtenteDAO;
import SHIVA.client.messaggi.MsgErroreOperazioneBnd;
import SHIVA.client.messaggi.MsgSuccessoOperazioneBnd;

/**
 * Gestisce l'eliminazione di un utente dal sistema.
 *
 * @author Autore 2.
 */
public class EliminaUtenteCtrl {

	private Utente utente;

	private RiepilogoUtenteBnd viewRiepilogo;

	private UtenteDAO utenteDao;

	private MsgErroreOperazioneBnd messErrore;

	private MsgSuccessoOperazioneBnd messOk;

	public void setUtente(Utente utente) {

	}

	public Utente getUtente() {
		return null;
	}

	public void setViewRiepilogo(RiepilogoUtenteBnd viewRiepilogo) {

	}

	public RiepilogoUtenteBnd getViewRiepilogo() {
		return null;
	}

	public void setUtenteDao(UtenteDAO utenteDao) {

	}

	public UtenteDAO getUtenteDao() {
		return null;
	}

	public void setMessErrore(MsgErroreOperazioneBnd messErrore) {

	}

	public MsgErroreOperazioneBnd getMessErrore() {
		return null;
	}

	public void setMessOk(MsgSuccessoOperazioneBnd messOk) {

	}

	public MsgSuccessoOperazioneBnd getMessOk() {
```

```
            return null;
      }

      public void confermaRiepilogo() {

      }

      public void confermaElimina() {

      }

      public void eliminaUtente() {

      }

}
```

1.1.1.2 Classe FormModificaUtenteBnd

```
package SHIVA.client.amministratore;

import java.util.Date;

/**
 * Form utilizzato dall'amministratore di sistema per la modifica di un utente.
 *
 * @author Autore 2
 */
public class FormModificaUtenteBnd {

      private String nome;

      private String cognome;

      private Date dataNascita;

      private String luogoNascita;

      private char sesso;

      private String codiceFiscale;

      private String indirizzoResidenza;

      private String recapitoTel1;

      private String recapitoTel2;

      private String recapitoMail1;

      private String recapitoMail2;

      public void fillForm(String indirizzoResidenza, String recapitoTel1,
String recapitoTel2, String recapitoMail1, String recapitoMail2) {

      }

      public void setNome(String nome) {

      }
```

```
public String getNome() {
	return null;
}

public void setCognome(String cognome) {

}

public String getCognome() {
	return null;
}

public void setDataNascita(Date dataNascita) {

}

public Date getDataNascita() {
	return null;
}

public void setLuogoNascita(String luogoNascita) {

}

public String getLuogoNascita() {
	return null;
}

public void setSesso(char sesso) {

}

public char getSesso() {
	return 0;
}

public void setCodiceFiscale(String codiceFiscale) {

}

public String getCodiceFiscale() {
	return null;
}

public void setIndirizzoResidenza(String indirizzoResidenza) {

}

public String getIndirizzoResidenza() {
	return null;
}

public void setRecapitoTel1(String recapitoTel1) {

}

public String getRecapitoTel1() {
	return null;
}

public void setRecapitoTel2(String recapitoTel2) {

}
```

```
	public String getRecapitoTel2() {
		return null;
	}

	public void setRecapitoMail1(String recapitoMail1) {

	}

	public String getRecapitoMail1() {
		return null;
	}

	public void setRecapitoMail2(String recapitoMail2) {

	}

	public String getRecapitoMail2() {
		return null;
	}

	public void modifica() {

	}

}
```

1.1.1.3 Classe FormRegistraMedicoBnd

```
package SHIVA.client.amministratore;

import java.util.Date;

/**
 * Form utilizzato dall'amministratore di sistema per la modifica di un utente.
 *
 * @author Autore 2
 */
public class FormModificaUtenteBnd {

	private String nome;

	private String cognome;

	private Date dataNascita;

	private String luogoNascita;

	private char sesso;

	private String codiceFiscale;

	private String indirizzoResidenza;

	private String recapitoTel1;

	private String recapitoTel2;

	private String recapitoMail1;

	private String recapitoMail2;
```

```
	public void fillForm(String indirizzoResidenza, String recapitoTel1,
String recapitoTel2, String recapitoMail1, String recapitoMail2) {

	}

	public void setNome(String nome) {

	}

	public String getNome() {
		return null;
	}

	public void setCognome(String cognome) {

	}

	public String getCognome() {
		return null;
	}

	public void setDataNascita(Date dataNascita) {

	}

	public Date getDataNascita() {
		return null;
	}

	public void setLuogoNascita(String luogoNascita) {

	}

	public String getLuogoNascita() {
		return null;
	}

	public void setSesso(char sesso) {

	}

	public char getSesso() {
		return 0;
	}

	public void setCodiceFiscale(String codiceFiscale) {

	}

	public String getCodiceFiscale() {
		return null;
	}

	public void setIndirizzoResidenza(String indirizzoResidenza) {

	}

	public String getIndirizzoResidenza() {
		return null;
	}

	public void setRecapitoTel1(String recapitoTel1) {
```

```
	}

	public String getRecapitoTel1() {
		return null;
	}

	public void setRecapitoTel2(String recapitoTel2) {

	}

	public String getRecapitoTel2() {
		return null;
	}

	public void setRecapitoMail1(String recapitoMail1) {

	}

	public String getRecapitoMail1() {
		return null;
	}

	public void setRecapitoMail2(String recapitoMail2) {

	}

	public String getRecapitoMail2() {
		return null;
	}

	public void modifica() {

	}

}
```

1.1.1.4 Classe FormRegistraReceptionistBnd

```
package SHIVA.client.amministratore;

import java.util.Date;

/**
 * Form utilizzato dall'amministratore di sistema per la modifica di un utente.
 *
 * @author Autore 2
 */
public class FormModificaUtenteBnd {

	private String nome;

	private String cognome;

	private Date dataNascita;

	private String luogoNascita;

	private char sesso;

	private String codiceFiscale;
```

```
	private String indirizzoResidenza;

	private String recapitoTel1;

	private String recapitoTel2;

	private String recapitoMail1;

	private String recapitoMail2;

	public void fillForm(String indirizzoResidenza, String recapitoTel1,
String recapitoTel2, String recapitoMail1, String recapitoMail2) {

	}

	public void setNome(String nome) {

	}

	public String getNome() {
		return null;
	}

	public void setCognome(String cognome) {

	}

	public String getCognome() {
		return null;
	}

	public void setDataNascita(Date dataNascita) {

	}

	public Date getDataNascita() {
		return null;
	}

	public void setLuogoNascita(String luogoNascita) {

	}

	public String getLuogoNascita() {
		return null;
	}

	public void setSesso(char sesso) {

	}

	public char getSesso() {
		return 0;
	}

	public void setCodiceFiscale(String codiceFiscale) {

	}

	public String getCodiceFiscale() {
		return null;
	}
```

```
	public void setIndirizzoResidenza(String indirizzoResidenza) {

	}

	public String getIndirizzoResidenza() {
		return null;
	}

	public void setRecapitoTel1(String recapitoTel1) {

	}

	public String getRecapitoTel1() {
		return null;
	}

	public void setRecapitoTel2(String recapitoTel2) {

	}

	public String getRecapitoTel2() {
		return null;
	}

	public void setRecapitoMail1(String recapitoMail1) {

	}

	public String getRecapitoMail1() {
		return null;
	}

	public void setRecapitoMail2(String recapitoMail2) {

	}

	public String getRecapitoMail2() {
		return null;
	}

	public void modifica() {

	}

}
```

1.1.1.5 Classe ModificaUtenteCtrl

```
package SHIVA.client.amministratore;
```

```java
import SHIVA.client.modello.Utente;
import SHIVA.server.persistenza.UtenteDAO;
import java.util.List;
import SHIVA.client.messaggi.MsgErroreOperazioneBnd;
import SHIVA.client.messaggi.MsgSuccessoOperazioneBnd;

/**
 * Gestisce la modifca dei dati di un utente.
 *
 * @author Autore 1
 */
public class ModificaUtenteCtrl {

	/**
	 * l'utente autenticato.
	 */
	private Utente utente;

	/**
	 * form tramite cui vengono richiesti i dati da modificare.
	 */
	private FormModificaUtenteBnd formModUtente;

	private UtenteDAO utenteDao;

	private String nuovoIndirizzo;

	private String nuovaCitta;

	private List nuoviRecapiti;

	private MsgErroreOperazioneBnd messErrore;

	private MsgSuccessoOperazioneBnd messOk;

	public void setUtente(Utente utente) {

	}

	public Utente getUtente() {
		return null;
	}

	public void setFormModUtente(FormModificaUtenteBnd formModUtente) {

	}

	public FormModificaUtenteBnd getFormModUtente() {
		return null;
	}

	public void setUtenteDao(UtenteDAO utenteDao) {

	}

	public UtenteDAO getUtenteDao() {
		return null;
	}

	public void setNuovoIndirizzo(String nuovoIndirizzo) {

	}
```

```
	public String getNuovoIndirizzo() {
		return null;
	}

	public void setNuovaCitta(String nuovaCitta) {

	}

	public String getNuovaCitta() {
		return null;
	}

	public void setNuoviRecapiti(List nuoviRecapiti) {

	}

	public List getNuoviRecapiti() {
		return null;
	}

	public void setMessErrore(MsgErroreOperazioneBnd messErrore) {

	}

	public MsgErroreOperazioneBnd getMessErrore() {
		return null;
	}

	public void setMessOk(MsgSuccessoOperazioneBnd messOk) {

	}

	public MsgSuccessoOperazioneBnd getMessOk() {
		return null;
	}

	public void modificaDati(List listaDati) {

	}

	public void modificaUtente() {

	}

}
```

1.1.1.6 Classe RegistraMedicoCtrl

```
package SHIVA.client.amministratore;

import SHIVA.server.persistenza.MedicoDAO;
```

```java
import SHIVA.client.messaggi.MsgErroreOperazioneBnd;
import SHIVA.client.messaggi.MsgSuccessoOperazioneBnd;
import SHIVA.client.modello.Medico;
import java.util.Date;
import java.util.List;

/**
 * Gestisce la registrazione di un nuovo medico.
 *
 * @author Autore 2
 */
public class RegistraMedicoCtrl {

	/**
	 * il form tramite cui vengono inseriti i dati del medico.
	 */
	private FormRegistraMedicoBnd formRegMedico;

	private MedicoDAO medicoDao;

	private MsgErroreOperazioneBnd messErrore;

	private MsgSuccessoOperazioneBnd messOk;

	private Medico medico;

	private String password;

	public void setFormRegMedico(FormRegistraMedicoBnd formRegMedico) {

	}

	public FormRegistraMedicoBnd getFormRegMedico() {
		return null;
	}

	public void setMedicoDao(MedicoDAO medicoDao) {

	}

	public MedicoDAO getMedicoDao() {
		return null;
	}

	public void setMessErrore(MsgErroreOperazioneBnd messErrore) {

	}

	public MsgErroreOperazioneBnd getMessErrore() {
		return null;
	}

	public void setMessOk(MsgSuccessoOperazioneBnd messOk) {

	}

	public MsgSuccessoOperazioneBnd getMessOk() {
		return null;
	}

	public void setMedico(Medico medico) {

	}
```

```
	public Medico getMedico() {
		return null;
	}

	public void setPassword(String password) {

	}

	public String getPassword() {
		return null;
	}

	/**
	 * genera la password per il nuovo medico in maniera casuale.
	 */
	public void generatePassword() {

	}

	public void registraMedico(String nome, String cognome, Date dataNascita,
String luogoNascita, char sesso, String codiceFiscale, String recapitoTel1,
String recapitoTel2, String recapitoMail1, String recapitoMail2, List
specializzazioni) {

	}

	public void confermaRegistrazione() {

	}

}
```

1.1.1.7 Classe RegistraReceptionistCtrl

```
package SHIVA.client.amministratore;

import SHIVA.client.messaggi.MsgErroreOperazioneBnd;
import SHIVA.client.messaggi.MsgSuccessoOperazioneBnd;
import java.util.Date;

/**
 * Gestisce la registrazione di un nuovo receptionist.
 *
 * @author Autore 1
 */
public class RegistraReceptionistCtrl {

	/**
	 * il form tramite cui vengono inseriti i dati del nuovo receptionist.
	 */
	private FormRegistraReceptionistBnd formRegRec;

	private int receptionistDao;

	private int receptionist;

	private String password;

	private MsgErroreOperazioneBnd messErrore;
```

```
	private MsgSuccessoOperazioneBnd messOk;

	public void setFormRegRec(FormRegistraReceptionistBnd formRegRec) {

	}

	public FormRegistraReceptionistBnd getFormRegRec() {
		return null;
	}

	public void setReceptionistDao(int receptionistDao) {

	}

	public int getReceptionistDao() {
		return 0;
	}

	public void setReceptionist(int receptionist) {

	}

	public int getReceptionist() {
		return 0;
	}

	public void setMessErrore(MsgErroreOperazioneBnd messErrore) {

	}

	public MsgErroreOperazioneBnd getMessErrore() {
		return null;
	}

	public void setMessOk(MsgSuccessoOperazioneBnd messOk) {

	}

	public MsgSuccessoOperazioneBnd getMessOk() {
		return null;
	}

	public void setPassword(String password) {

	}

	public String getPassword() {
		return null;
	}

	/**
	 * metodo che genera la password per il nuovo receptionist in maniera
casuale.
	 */
	public void generatePassword() {

	}

	public void registraReceptionist(String nome, String cognome, Date
dataNascita, String luogoNascita, char sesso, String codiceFiscale, String
recapitoTel1, String recapitoTel2, String recapitoMail1, String recapitoMail2) {

	}
```

```
	public void confermaRegistrazione() {

	}

}
```

1.1.1.8 Classe RiepilogoUtenteBnd

```
package SHIVA.client.amministratore;

import java.util.Date;

/**
 * Vista con cui SHIVA mostra il riepilogo di un utente.
 *
 * @author Autore 1
 */
public class RiepilogoUtenteBnd {

	private String nome;

	private String cognome;

	private Date dataNascita;

	private String luogoNascita;

	private char sesso;

	private String codiceFiscale;

	private String indirizzoResidenza;

	private String recapitoTel1;

	private String recapitoTel2;

	private String recapitoMail1;

	private String recapitoMail2;

	public void conferma() {

	}

	public void annulla() {

	}

	public void setNome(String nome) {

	}

	public String getNome() {
		return null;
	}

	public void setCognome(String cognome) {

	}
```

```
public String getCognome() {
	return null;
}

public void setDataNascita(Date dataNascita) {

}

public Date getDataNascita() {
	return null;
}

public void setLuogoNascita(String luogoNascita) {

}

public String getLuogoNascita() {
	return null;
}

public void setSesso(char sesso) {

}

public char getSesso() {
	return 0;
}

public void setCodiceFiscale(String codiceFiscale) {

}

public String getCodiceFiscale() {
	return null;
}

public void setIndirizzoResidenza(String indirizzoResidenza) {

}

public String getIndirizzoResidenza() {
	return null;
}

public void setRecapitoTel1(String recapitoTel1) {

}

public String getRecapitoTel1() {
	return null;
}

public void setRecapitoTel2(String recapitoTel2) {

}

public String getRecapitoTel2() {
	return null;
}

public void setRecapitoMail1(String recapitoMail1) {
```

```
	}

	public String getRecapitoMail1() {
		return null;
	}

	public void setRecapitoMail2(String recapitoMail2) {

	}

	public String getRecapitoMail2() {
		return null;
	}

}
```

1.1.2 Package Comunicazione

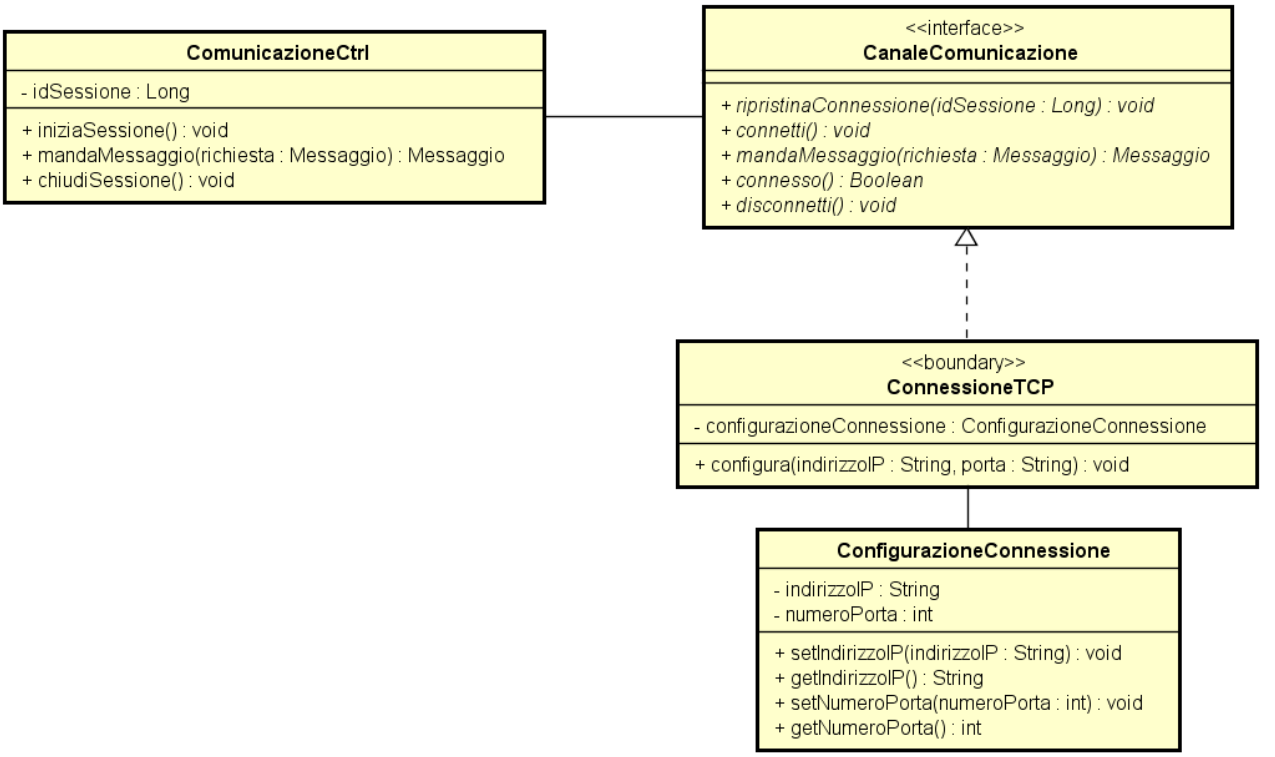


FIGURA 108 - PACKAGE COMUNICAZIONE

1.1.2.1 Interfaccia CanaleComunicazione

```
package SHIVA.client.comunicazione;

import SHIVA.client.messaggicomunicazione.Messaggio;
```

```java
/**
 * Interfaccia per la comunicazione
 *
 * @author Autore 1
 */
public interface CanaleComunicazione {

	/**
	 * Metodo che si occupa di ripristinare la connessione
	 */
	public abstract void ripristinaConnessione(Long idSessione);

	/**
	 * Si occupa di effettuare la connessione
	 */
	public abstract void connetti();

	/**
	 * Trasmette il messaggio nel canale
	 */
	public abstract Messaggio mandaMessaggio(Messaggio richiesta);

	/**
	 * Restituisce un valore booleano che indica lo stato della connessione
	 */
	public abstract Boolean connesso();

	/**
	 * Termina la connessione
	 */
	public abstract void disconnetti();

}
```

1.1.2.2 Classe ComunicazioneCtrl

```java
package SHIVA.client.comunicazione;

import SHIVA.client.messaggicomunicazione.Messaggio;

/**
 * Controller di connessione
 *
 * @author Autore 2
 */
public class ComunicazioneCtrl {

	private Long idSessione;

	/**
	 * Inizializza la sessione
	 */
	public void iniziaSessione() {

	}

	public Messaggio mandaMessaggio(Messaggio richiesta) {
		return null;
	}
```

```
	/**
	 * Chiude la sessione corrente
	 */
	public void chiudiSessione() {

	}

}
```

1.1.2.3 Classe ConfigurazioneConnessione

```
package SHIVA.client.comunicazione;

/**
 * Contiene i parametri di inizializzazione della connessione verso il server
 *
 * @author Autore 1
 */
public class ConfigurazioneConnessione {

	/**
	 * Indirizzo IP del server
	 */
	private String indirizzoIP;

	/**
	 * Porta di connessione per il server
	 */
	private int numeroPorta;

	public void setIndirizzoIP(String indirizzoIP) {

	}

	public String getIndirizzoIP() {
		return null;
	}

	public void setNumeroPorta(int numeroPorta) {

	}

	public int getNumeroPorta() {
		return 0;
	}

}
```

1.1.2.4 Classe ConnessioneTCP

```
package SHIVA.client.comunicazione;

import SHIVA.client.messaggicomunicazione.Messaggio;

/**
```

```
 * Boundary per la connessione TCP
 *
 * @author Autore 2
 */
public class ConnessioneTCP implements CanaleComunicazione {

	private ConfigurazioneConnessione configurazioneConnessione;

	public void configura(String indirizzoIP, String porta) {

	}


	/**
	 * @see
SHIVA.client.comunicazione.CanaleComunicazione#ripristinaConnessione(java.lang.L
ong)
	 */
	public void ripristinaConnessione(Long idSessione) {

	}


	/**
	 * @see SHIVA.client.comunicazione.CanaleComunicazione#connetti()
	 */
	public void connetti() {

	}


	/**
	 * @see
SHIVA.client.comunicazione.CanaleComunicazione#mandaMessaggio(SHIVA.client.messa
ggicomunicazione.Messaggio)
	 */
	public Messaggio mandaMessaggio(Messaggio richiesta) {
		return null;
	}


	/**
	 * @see SHIVA.client.comunicazione.CanaleComunicazione#connesso()
	 */
	public Boolean connesso() {
		return null;
	}


	/**
	 * @see SHIVA.client.comunicazione.CanaleComunicazione#disconnetti()
	 */
	public void disconnetti() {

	}
}
```

1.1.3 Package Medico

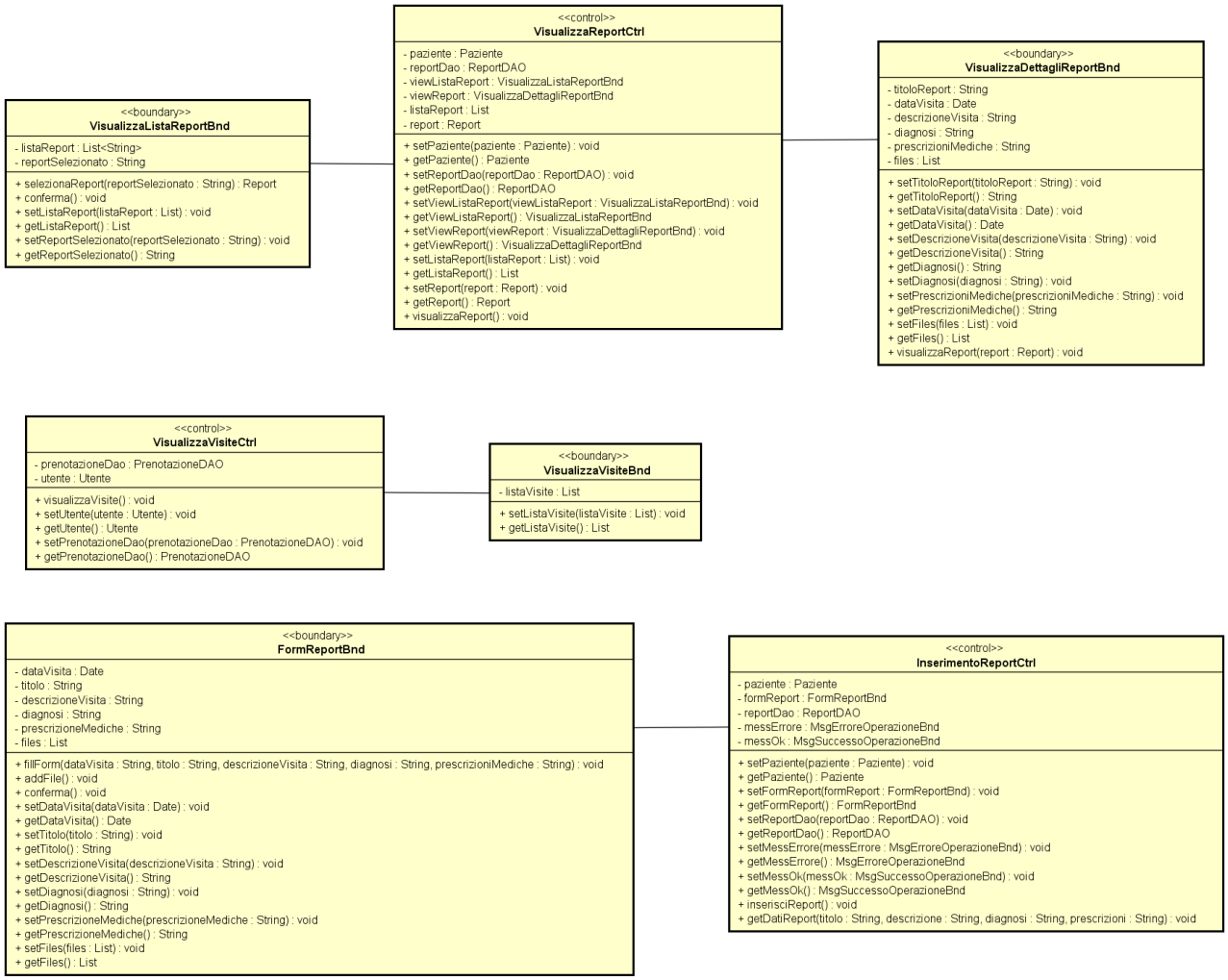


FIGURA 109 - PACKAGE MEDICO

1.1.3.1 Classe FormReportBnd

```
package SHIVA.client.medico;

import java.util.Date;
import java.util.List;

/**
 * Form utilizzato dal medico per la compilazione del report.
 *
 * @author Autore 2
 */
public class FormReportBnd {

	private Date dataVisita;

	private String titolo;

	private String descrizioneVisita;

	private String diagnosi;

	private String prescrizioneMediche;
```

```
	private List files;

	public void fillForm(String dataVisita, String titolo, String
descrizioneVisita, String diagnosi, String prescrizioniMediche) {

	}

	public void addFile() {

	}

	public void conferma() {

	}

	public void setDataVisita(Date dataVisita) {

	}

	public Date getDataVisita() {
		return null;
	}

	public void setTitolo(String titolo) {

	}

	public String getTitolo() {
		return null;
	}

	public void setDescrizioneVisita(String descrizioneVisita) {

	}

	public String getDescrizioneVisita() {
		return null;
	}

	public void setDiagnosi(String diagnosi) {

	}

	public String getDiagnosi() {
		return null;
	}

	public void setPrescrizioneMediche(String prescrizioneMediche) {

	}

	public String getPrescrizioneMediche() {
		return null;
	}

	public void setFiles(List files) {

	}

	public List getFiles() {
		return null;
	}
}
```

1.1.3.2 Classe InserimentoReportCtrl

```
package SHIVA.client.medico;

import SHIVA.client.modello.Paziente;
import SHIVA.server.persistenza.ReportDAO;
import SHIVA.client.messaggi.MsgErroreOperazioneBnd;
import SHIVA.client.messaggi.MsgSuccessoOperazioneBnd;

/**
 * Gestisce l'inserimento di un nuovo report e lo associa al paziente.
 *
 * @author Autore 1
 */
public class InserimentoReportCtrl {

	/**
	 * il paziente da associare al report.
	 */
	private Paziente paziente;

	/**
	 * il form tramite cui viene composto il report.
	 */
	private FormReportBnd formReport;

	private ReportDAO reportDao;

	private MsgErroreOperazioneBnd messErrore;

	private MsgSuccessoOperazioneBnd messOk;

	public void setPaziente(Paziente paziente) {

	}

	public Paziente getPaziente() {
		return null;
	}

	public void setFormReport(FormReportBnd formReport) {

	}

	public FormReportBnd getFormReport() {
		return null;
	}

	public void setReportDao(ReportDAO reportDao) {

	}

	public ReportDAO getReportDao() {
		return null;
	}

	public void setMessErrore(MsgErroreOperazioneBnd messErrore) {

	}

	public MsgErroreOperazioneBnd getMessErrore() {
		return null;
	}
```

```
	public void setMessOk(MsgSuccessoOperazioneBnd messOk) {

	}

	public MsgSuccessoOperazioneBnd getMessOk() {
		return null;
	}

	public void inserisciReport() {

	}

	/**
	 * metodo tramite cui vengono ricevuti i campi del record da comporre.
	 */
	public void getDatiReport(String titolo, String descrizione, String
diagnosi, String prescrizioni) {

	}

}
```

1.1.3.3 Classe VisualizzaDettagliReportBnd

```
package SHIVA.client.medico;

import java.util.Date;
import java.util.List;
import SHIVA.client.modello.Report;

/**
 * Vista con cui SHIVA mostra al medico i dettagli del report selezionato.
 *
 * @author Autore 2
 */
public class VisualizzaDettagliReportBnd {

	private String titoloReport;

	private Date dataVisita;

	private String descrizioneVisita;

	private String diagnosi;

	private String prescrizioniMediche;

	private List files;

	public void setTitoloReport(String titoloReport) {

	}

	public String getTitoloReport() {
		return null;
	}

	public void setDataVisita(Date dataVisita) {

	}
```

```
	public Date getDataVisita() {
		return null;
	}

	public void setDescrizioneVisita(String descrizioneVisita) {

	}

	public String getDescrizioneVisita() {
		return null;
	}

	public String getDiagnosi() {
		return null;
	}

	public void setDiagnosi(String diagnosi) {

	}

	public void setPrescrizioniMediche(String prescrizioniMediche) {

	}

	public String getPrescrizioniMediche() {
		return null;
	}

	public void setFiles(List files) {

	}

	public List getFiles() {
		return null;
	}

	public void visualizzaReport(Report report) {

	}

}
```

1.1.3.4 Classe VisualizzaListaReportBnd

```
package SHIVA.client.medico;

import java.util.List;
import SHIVA.client.modello.Report;

/**
 * Vista con cui SHIVA mostra la medico la lista dei report associati al paziente precedentemente cercato e gli permette di selezionare il report a cui è interessato.
 *
 * @author Autore 1
 */
public class VisualizzaListaReportBnd {

	private List listaReport;
```

```
	private String reportSelezionato;

	public Report selezionaReport(String reportSelezionato) {
		return null;
	}

	public void conferma() {

	}

	public void setListaReport(List listaReport) {

	}

	public List getListaReport() {
		return null;
	}

	public void setReportSelezionato(String reportSelezionato) {

	}

	public String getReportSelezionato() {
		return null;
	}

}
```

1.1.3.5 Classe VisualizzaReportCtrl

```
package SHIVA.client.medico;

import java.util.List;
import SHIVA.client.modello.Report;

/**
 * Vista con cui SHIVA mostra la medico la lista dei report associati al
paziente precedentemente cercato e gli permette di selezionare il report a cui è
interessato.
 *
 * @author Autore 1
 */
public class VisualizzaListaReportBnd {

	private List listaReport;

	private String reportSelezionato;

	public Report selezionaReport(String reportSelezionato) {
		return null;
	}

	public void conferma() {

	}

	public void setListaReport(List listaReport) {

	}

	public List getListaReport() {
```

```
		return null;
	}

	public void setReportSelezionato(String reportSelezionato) {

	}

	public String getReportSelezionato() {
		return null;
	}

}
```

1.1.3.6 Classe VisualizzaVisiteBnd

```
package SHIVA.client.medico;

import java.util.List;

/**
 * Vista con cui SHIVA comunica al medico le visite da effettuare.
 *
 * @author Autore 1
 */
public class VisualizzaVisiteBnd {

	private List listaVisite;

	public void setListaVisite(List listaVisite) {

	}

	public List getListaVisite() {
		return null;
	}

}
```

1.1.3.7 Classe VisualizzaVisiteCtrl

```
package SHIVA.client.medico;

import SHIVA.server.persistenza.PrenotazioneDAO;
import SHIVA.client.modello.Utente;

/**
 * Gestisce la visualizzazione delle visite.
 *
 * @author Autore 2
 */
public class VisualizzaVisiteCtrl {

	private PrenotazioneDAO prenotazioneDao;

	/**
	 * l'utente autenticato
	 */
	private Utente utente;

	public void visualizzaVisite() {
```

```
	}

	public void setUtente(Utente utente) {

	}

	public Utente getUtente() {
		return null;
	}

	public void setPrenotazioneDao(PrenotazioneDAO prenotazioneDao) {

	}

	public PrenotazioneDAO getPrenotazioneDao() {
		return null;
	}

}
```

1.1.4 Package Messaggi

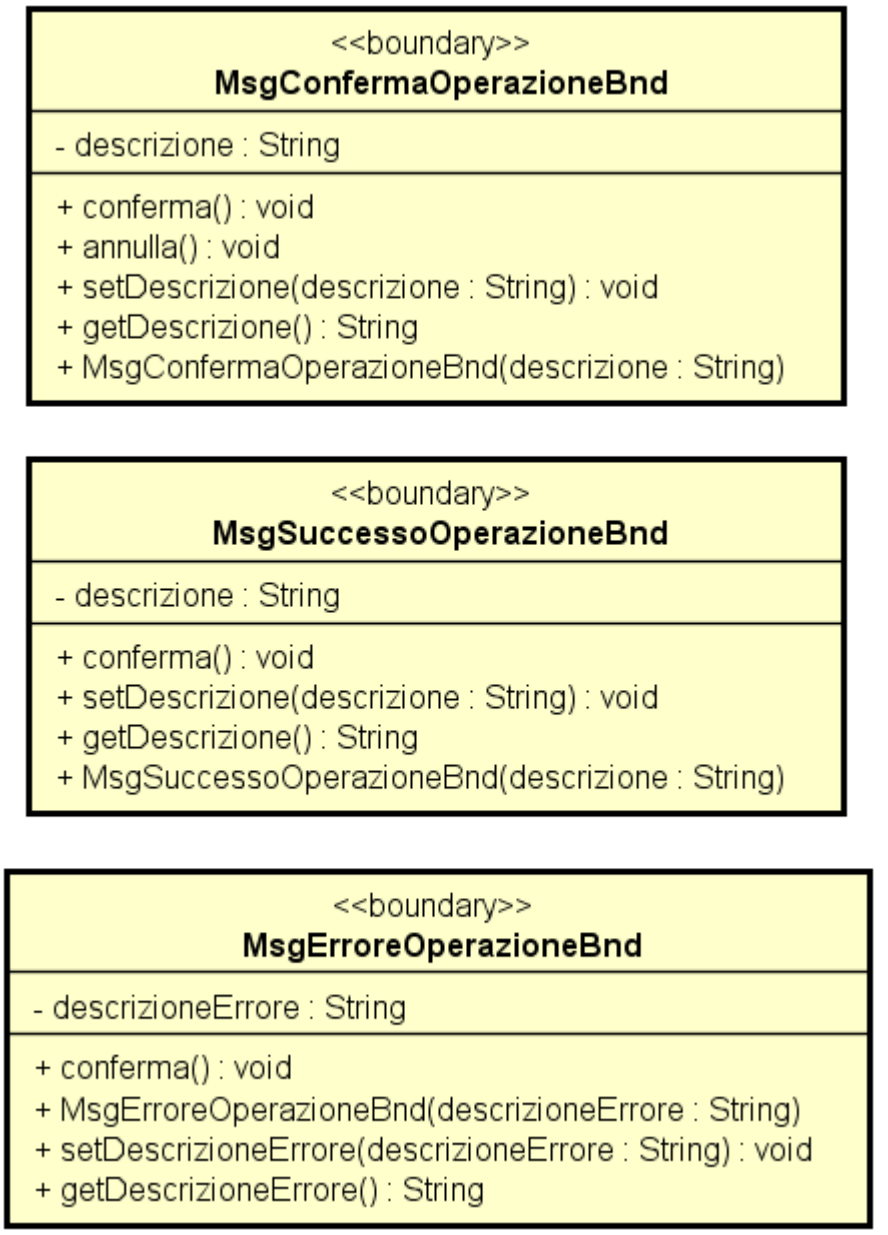


FIGURA 110 - PACKAGE MESSAGGI

1.1.4.1 Classe MsgConfermaOperazioneBnd

```
package SHIVA.client.messaggi;

/**
 * Messaggio con cui SHIVA richiede la conferma dell'operazione da effettuare
 *
 * @author Autore 2
 */
public class MsgConfermaOperazioneBnd {

	private String descrizione;

	public void conferma() {

	}

	public void annulla() {

	}

	public void setDescrizione(String descrizione) {

	}

	public String getDescrizione() {
		return null;
	}

	public MsgConfermaOperazioneBnd(String descrizione) {

	}

}
```

1.1.4.2 Classe MsgErroreOperazioneBnd

```
package SHIVA.client.messaggi;

/**
 * Messaggio di errore con cui SHIVA comunica all'utente che l'operazione non è
avvenuta con successo e specifica le cause del fallimento.
 *
 * @author Autore 1
 */
public class MsgErroreOperazioneBnd {

	private String descrizioneErrore;

	/**
	 * conferma la lettura del messaggio
	 */
	public void conferma() {

	}

	public MsgErroreOperazioneBnd(String descrizioneErrore) {

	}

	public void setDescrizioneErrore(String descrizioneErrore) {
```

```
	}

	public String getDescrizioneErrore() {
		return null;
	}

}
```

1.1.4.3 Classe MsgSuccessoOperazione

```
package SHIVA.client.messaggi;

/**
 * Messaggio utilizzato da SHIVA per comunicare il successo dell'operazione.
 *
 * @author Autore 1
 */
public class MsgSuccessoOperazioneBnd {

	private String descrizione;

	/**
	 * conferma la lettura del messaggio
	 */
	public void conferma() {

	}

	public void setDescrizione(String descrizione) {

	}

	public String getDescrizione() {
		return null;
	}

	public MsgSuccessoOperazioneBnd(String descrizione) {

	}

}
```

1.1.5 Package MessaggiComunicazione

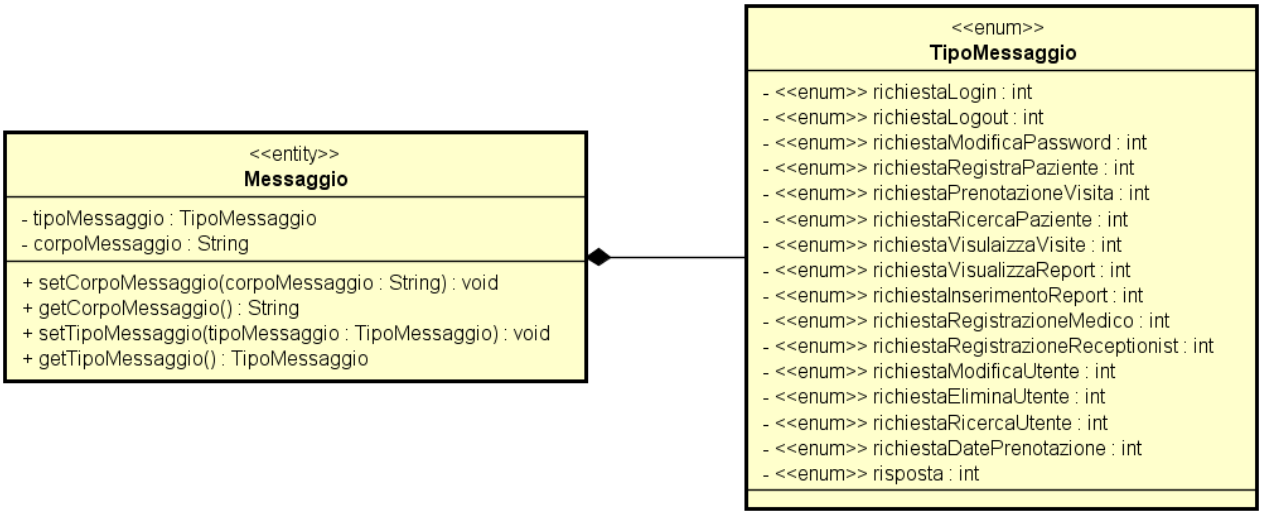


FIGURA 111 - PACKAGE MESSAGGICOMUNICAZIONE

1.1.5.1 Classe Messaggio

```
package SHIVA.client.messaggicomunicazione;

/**
 * Entità che rappresenta il generico messaggio inviato fra client e server
 *
 * @author Autore 1
 */
public class Messaggio {

	private TipoMessaggio tipoMessaggio;

	private String corpoMessaggio;

	public void setCorpoMessaggio(String corpoMessaggio) {

	}

	public String getCorpoMessaggio() {
		return null;
	}

	public void setTipoMessaggio(TipoMessaggio tipoMessaggio) {

	}

	public TipoMessaggio getTipoMessaggio() {
		return null;
	}

}
```

1.1.5.2 Enumerazione TipoMessaggio

```
package SHIVA.client.messaggicomunicazione;

/**
 * Elenca i tipi di messaggio che client e server possono trasmettersi. Ogni tipo indica un particolare operazione da svolgere
 *
 * @author Autore 2
 */
public enum TipoMessaggio {

	;

	private int richiestaLogin;

	private int richiestaLogout;

	private int richiestaModificaPassword;

	private int richiestaRegistraPaziente;

	private int richiestaPrenotazioneVisita;

	private int richiestaRicercaPaziente;

	private int richiestaVisulaizzaVisite;

	private int richiestaVisualizzaReport;

	private int richiestaInserimentoReport;

	private int richiestaRegistrazioneMedico;

	private int richiestaRegistrazioneReceptionist;

	private int richiestaModificaUtente;

	private int richiestaEliminaUtente;

	private int richiestaRicercaUtente;

	private int richiestaDatePrenotazione;

	private int risposta;}
```

1.1.6 Package Modello

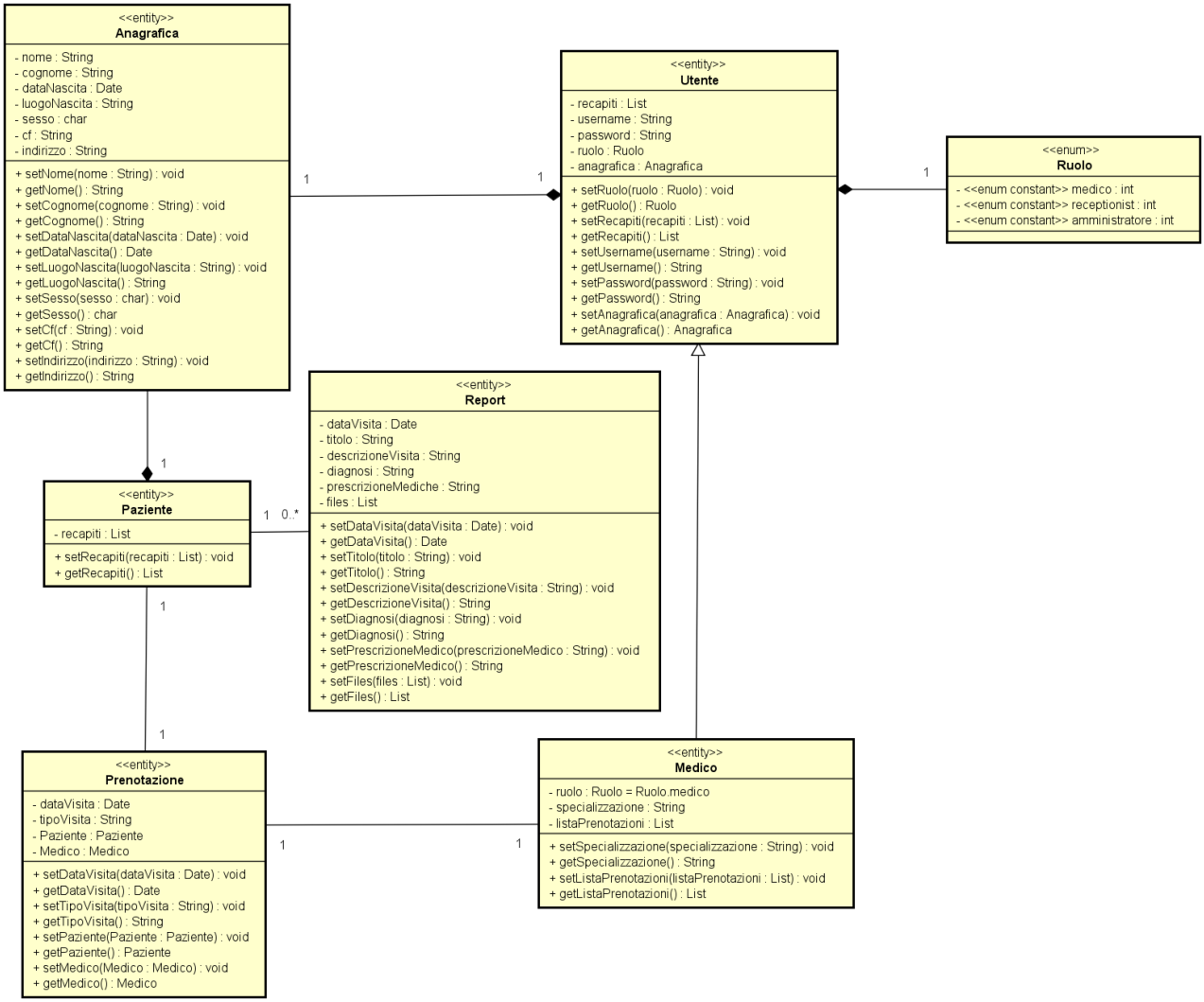


FIGURA 112 - PACKAGE MODELLO

1.1.6.1 Classe Anagrafica

```
package SHIVA.client.modello;

import java.util.Date;

/**
 * Rappresenta i dati anagrafici di un utente o di un paziente.
 *
 * @author Autore 2
 */
public class Anagrafica {

	private String nome;

	private String cognome;

	private Date dataNascita;

	private String luogoNascita;

	private char sesso;
```

```java
/**
 * Il codice fiscale.
 */
private String cf;

/**
 * L'indirizzo di residenza.
 */
private String indirizzo;

public void setNome(String nome) {

}

public String getNome() {
	return null;
}

public void setCognome(String cognome) {

}

public String getCognome() {
	return null;
}

public void setDataNascita(Date dataNascita) {

}

public Date getDataNascita() {
	return null;
}

public void setLuogoNascita(String luogoNascita) {

}

public String getLuogoNascita() {
	return null;
}

public void setSesso(char sesso) {

}

public char getSesso() {
	return 0;
}

public void setCf(String cf) {

}

public String getCf() {
	return null;
}

public void setIndirizzo(String indirizzo) {

}

public String getIndirizzo() {
```

```
            return null;
      }

}
```

1.1.6.2 Classe Medico

```
package SHIVA.client.modello;

import java.util.List;

/**
 * Rappresenta un utente di tipo medico. Eredita da Utente.
 *
 * @author Autore 1
 */
public class Medico extends Utente {

      private final Ruolo ruolo = Ruolo.medico;

      private String specializzazione;

      private List listaPrenotazioni;

      public void setSpecializzazione(String specializzazione) {

      }

      public String getSpecializzazione() {
            return null;
      }

      public void setListaPrenotazioni(List listaPrenotazioni) {

      }

      public List getListaPrenotazioni() {
            return null;
      }

}
```

1.1.6.3 Classe Paziente

```
package SHIVA.client.modello;

import java.util.List;

/**
 * Rappresenta un Paziente della struttura medica.
 *
 * @author Autore 2
 */
public class Paziente {

      /**
       * lista dei recapiti (sia telefonici che email)
       */
      private List recapiti;
```

```
	public void setRecapiti(List recapiti) {

	}

	public List getRecapiti() {
		return null;
	}

}
```

1.1.6.4 Classe Prenotazione

```
package SHIVA.client.modello;

import java.util.Date;

/**
 * Rappresenta le prenotazioni delle visite effettuate all'interno della struttura medica.
 *
 * @author Autore 2
 */
public class Prenotazione {

	private Date dataVisita;

	/**
	 * Il tipo della visita prenotata. Deve corrispondere ad una delle specializzazioni dei medici presenti nella struttura.
	 */
	private String tipoVisita;

	/**
	 * il Paziente associato alla visita
	 */
	private Paziente Paziente;

	/**
	 * il Medico associato alla visita
	 */
	private Medico Medico;

	public void setDataVisita(Date dataVisita) {

	}

	public Date getDataVisita() {
		return null;
	}

	public void setTipoVisita(String tipoVisita) {

	}

	public String getTipoVisita() {
		return null;
	}

	public void setPaziente(Paziente Paziente) {

	}
```

```
	public Paziente getPaziente() {
		return null;
	}

	public void setMedico(Medico Medico) {

	}

	public Medico getMedico() {
		return null;
	}

}
```

1.1.6.5 Classe Report

```
package SHIVA.client.modello;

import java.util.Date;
import java.util.List;

/**
 * Rappresenta il report associato ad una visita e compilato dal medico.
 *
 * @author Autore 2
 */
public class Report {

	private Date dataVisita;

	private String titolo;

	private String descrizioneVisita;

	private String diagnosi;

	private String prescrizioneMediche;

	/**
	 * Lista di file serializzati in formato array di byte.
	 */
	private List files;

	public void setDataVisita(Date dataVisita) {

	}

	public Date getDataVisita() {
		return null;
	}

	public void setTitolo(String titolo) {

	}

	public String getTitolo() {
		return null;
	}

	public void setDescrizioneVisita(String descrizioneVisita) {
```

```
	}

	public String getDescrizioneVisita() {
		return null;
	}

	public void setDiagnosi(String diagnosi) {

	}

	public String getDiagnosi() {
		return null;
	}

	public void setPrescrizioneMedico(String prescrizioneMedico) {

	}

	public String getPrescrizioneMedico() {
		return null;
	}

	public void setFiles(List files) {

	}

	public List getFiles() {
		return null;
	}

}
```

1.1.6.6 Enumerazione Ruolo

```
package SHIVA.client.modello;

/**
 * L'enum che definisce i ruoli possibili.
 *
 * @author Autore 1
 */
public enum Ruolo {

	medico,

	receptionist,

	amministratore;

}
```

1.1.6.7 Classe Utente

```
package SHIVA.client.modello;

import java.util.List;

/**
 * Rappresenta un utente con un ruolo qualsiasi tra amministratore di sistema,
receptionist e medico.
 *
 * @author Autore 1
 *
 */
public class Utente {

	private List recapiti;

	private String username;

	private String password;

	private Ruolo ruolo;

	private Anagrafica anagrafica;

	public void setRuolo(Ruolo ruolo) {

	}

	public Ruolo getRuolo() {
		return null;
	}

	public void setRecapiti(List recapiti) {

	}

	public List getRecapiti() {
		return null;
	}

	public void setUsername(String username) {

	}

	public String getUsername() {
		return null;
	}

	public void setPassword(String password) {

	}

	public String getPassword() {
		return null;
	}

	public void setAnagrafica(Anagrafica anagrafica) {

	}
```

```
	public Anagrafica getAnagrafica() {
		return null;
	}

}
```

1.1.7 Package Receptionist

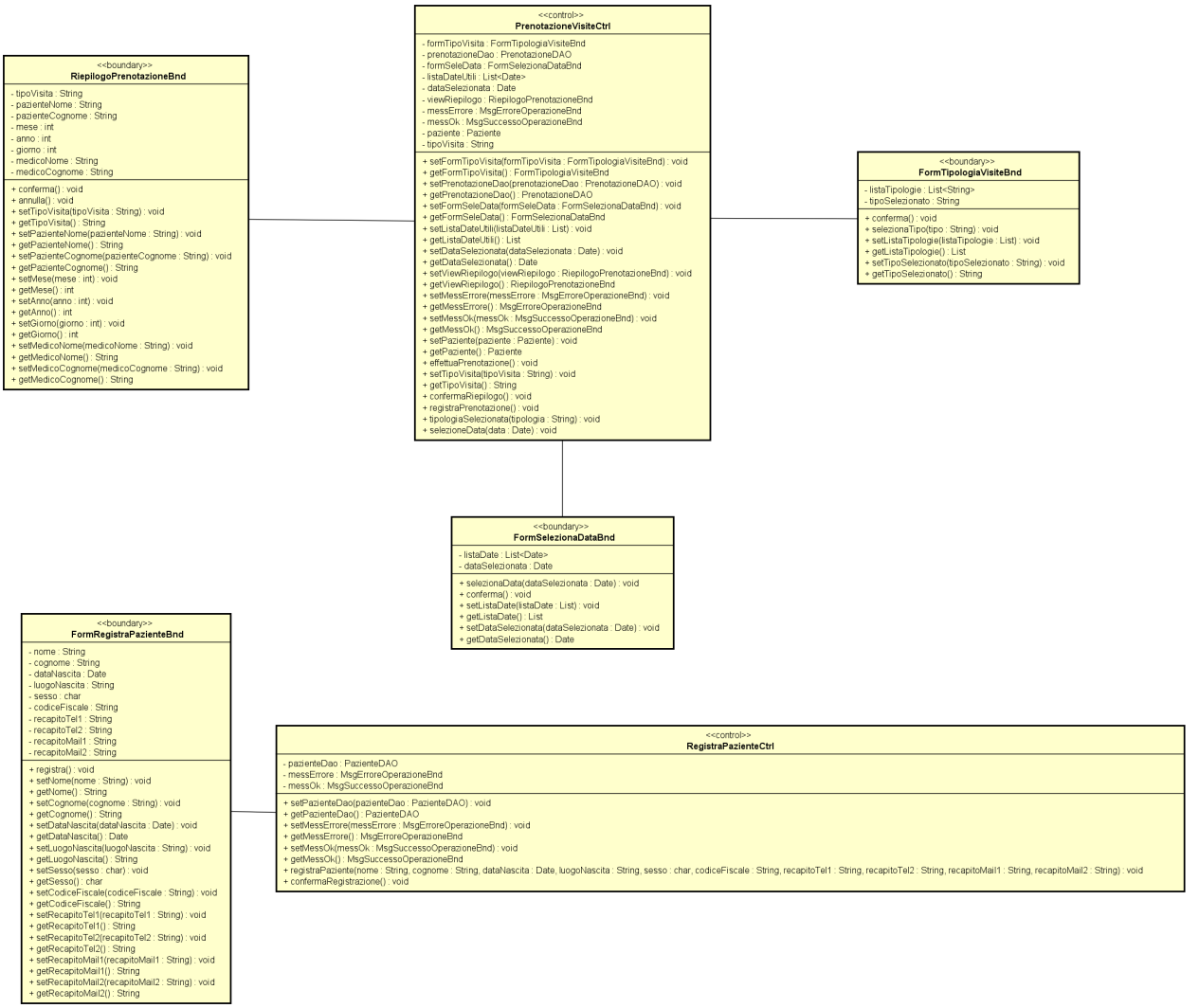


FIGURA 113 - PACKAGE RECEPTIONIST

1.1.7.1 Classe FormRegistraPazienteBnd

```
package SHIVA.client.receptionist;

import java.util.Date;

/**
 * Form utilizzato dal receptionist per registrare il paziente.
 *
 * @author Autore 2
 */
public class FormRegistraPazienteBnd {

	private String nome;

	private String cognome;
```

```
private Date dataNascita;

private String luogoNascita;

private char sesso;

private String codiceFiscale;

private String recapitoTel1;

private String recapitoTel2;

private String recapitoMail1;

private String recapitoMail2;

/**
 * permette di inserire i dati del nuovo paziente.
 */
public void registra() {

}

public void setNome(String nome) {

}

public String getNome() {
	return null;
}

public void setCognome(String cognome) {

}

public String getCognome() {
	return null;
}

public void setDataNascita(Date dataNascita) {

}

public Date getDataNascita() {
	return null;
}

public void setLuogoNascita(String luogoNascita) {

}

public String getLuogoNascita() {
	return null;
}

public void setSesso(char sesso) {

}

public char getSesso() {
	return 0;
}
```

```
	public void setCodiceFiscale(String codiceFiscale) {

	}

	public String getCodiceFiscale() {
		return null;
	}

	public void setRecapitoTel1(String recapitoTel1) {

	}

	public String getRecapitoTel1() {
		return null;
	}

	public void setRecapitoTel2(String recapitoTel2) {

	}

	public String getRecapitoTel2() {
		return null;
	}

	public void setRecapitoMail1(String recapitoMail1) {

	}

	public String getRecapitoMail1() {
		return null;
	}

	public void setRecapitoMail2(String recapitoMail2) {

	}

	public String getRecapitoMail2() {
		return null;
	}

}
```

1.1.7.2 Classe FormSelezionaDataBnd

```
package SHIVA.client.receptionist;

import java.util.List;
import java.util.Date;

/**
 * Form utilizzato dal receptionist per selezionare la data per la prenotazione.
 *
 * @author Autore 2
 */
public class FormSelezionaDataBnd {

	private List listaDate;

	private Date dataSelezionata;
```

```
	public void selezionaData(Date dataSelezionata) {

	}

	public void conferma() {

	}

	public void setListaDate(List listaDate) {

	}

	public List getListaDate() {
		return null;
	}

	public void setDataSelezionata(Date dataSelezionata) {

	}

	public Date getDataSelezionata() {
		return null;
	}

}
```

1.1.7.3 Classe FormTipologiaVisiteBnd

```
package SHIVA.client.receptionist;

import java.util.List;

/**
 * Form utilizzato dal receptionist per selezionare il tipo di visita da prenotare.
 *
 * @author Autore 2
 */
public class FormTipologiaVisiteBnd {

	private List listaTipologie;

	private String tipoSelezionato;

	public void conferma() {

	}

	public void selezionaTipo(String tipo) {

	}

	public void setListaTipologie(List listaTipologie) {

	}

	public List getListaTipologie() {
		return null;
	}

	public void setTipoSelezionato(String tipoSelezionato) {
```

```
	}

	public String getTipoSelezionato() {
		return null;
	}

}
```

1.1.7.4 Classe PrenotazioneVisiteCtrl

```
package SHIVA.client.receptionist;

import SHIVA.server.persistenza.PrenotazioneDAO;
import java.util.List;
import java.util.Date;
import SHIVA.client.messaggi.MsgErroreOperazioneBnd;
import SHIVA.client.messaggi.MsgSuccessoOperazioneBnd;
import SHIVA.client.modello.Paziente;

/**
 * Gestisce la prenotazione della visita.
 *
 * @author Autore 2
 */
public class PrenotazioneVisiteCtrl {

	/**
	 * il form tramite cui viene selezionato il tipo della visita da
prenotare.
	 */
	private FormTipologiaVisiteBnd formTipoVisita;

	private PrenotazioneDAO prenotazioneDao;

	/**
	 * il form tramite cui viene selezionata la data per la prenotazione.
	 */
	private FormSelezionaDataBnd formSeleData;

	private List listaDateUtili;

	private Date dataSelezionata;

	/**
	 * vista tramite cui viene mostrato il riepilogo della prenotazione.
	 */
	private RiepilogoPrenotazioneBnd viewRiepilogo;

	private MsgErroreOperazioneBnd messErrore;

	private MsgSuccessoOperazioneBnd messOk;

	/**
	 * paziente associato alla visita.
	 */
	private Paziente paziente;

	private String tipoVisita;

	public void setFormTipoVisita(FormTipologiaVisiteBnd formTipoVisita) {
```

```
}

public FormTipologiaVisiteBnd getFormTipoVisita() {
	return null;
}

public void setPrenotazioneDao(PrenotazioneDAO prenotazioneDao) {

}

public PrenotazioneDAO getPrenotazioneDao() {
	return null;
}

public void setFormSeleData(FormSelezionaDataBnd formSeleData) {

}

public FormSelezionaDataBnd getFormSeleData() {
	return null;
}

public void setListaDateUtili(List listaDateUtili) {

}

public List getListaDateUtili() {
	return null;
}

public void setDataSelezionata(Date dataSelezionata) {

}

public Date getDataSelezionata() {
	return null;
}

public void setViewRiepilogo(RiepilogoPrenotazioneBnd viewRiepilogo) {

}

public RiepilogoPrenotazioneBnd getViewRiepilogo() {
	return null;
}

public void setMessErrore(MsgErroreOperazioneBnd messErrore) {

}

public MsgErroreOperazioneBnd getMessErrore() {
	return null;
}

public void setMessOk(MsgSuccessoOperazioneBnd messOk) {

}

public MsgSuccessoOperazioneBnd getMessOk() {
	return null;
}
```

```
	public void setPaziente(Paziente paziente) {

	}

	public Paziente getPaziente() {
		return null;
	}

	public void effettuaPrenotazione() {

	}

	public void setTipoVisita(String tipoVisita) {

	}

	public String getTipoVisita() {
		return null;
	}

	public void confermaRiepilogo() {

	}

	public void registraPrenotazione() {

	}

	public void tipologiaSelezionata(String tipologia) {

	}

	public void selezioneData(Date data) {

	}

}
```

1.1.7.5 Classe RegistraPazienteCtrl

```
package SHIVA.client.receptionist;

import SHIVA.server.persistenza.PazienteDAO;
import SHIVA.client.messaggi.MsgErroreOperazioneBnd;
import SHIVA.client.messaggi.MsgSuccessoOperazioneBnd;
import java.util.Date;

/**
 * Gestisce la registrazione di un nuovo paziente.
 *
 * @author Autore 2
 */
public class RegistraPazienteCtrl {

	private PazienteDAO pazienteDao;

	private MsgErroreOperazioneBnd messErrore;

	private MsgSuccessoOperazioneBnd messOk;

	public void setPazienteDao(PazienteDAO pazienteDao) {
```

```
	}

	public PazienteDAO getPazienteDao() {
		return null;
	}

	public void setMessErrore(MsgErroreOperazioneBnd messErrore) {

	}

	public MsgErroreOperazioneBnd getMessErrore() {
		return null;
	}

	public void setMessOk(MsgSuccessoOperazioneBnd messOk) {

	}

	public MsgSuccessoOperazioneBnd getMessOk() {
		return null;
	}

	public void registraPaziente(String nome, String cognome, Date
dataNascita, String luogoNascita, char sesso, String codiceFiscale, String
recapitoTel1, String recapitoTel2, String recapitoMail1, String recapitoMail2) {

	}

	public void confermaRegistrazione() {

	}

}
```

1.1.7.6 Classe RiepilogoPrenotazioneBnd

```
package SHIVA.client.receptionist;

/**
 * Vista con cui SHIVA mostra la Receptionist il riepilogo della prenotazione e
ne chiede conferma.
 *
 * @author Autore 1
 */
public class RiepilogoPrenotazioneBnd {

	private String tipoVisita;

	private String pazienteNome;

	private String pazienteCognome;

	private int mese;

	private int anno;

	private int giorno;

	private String medicoNome;
```

```
private String medicoCognome;

public void conferma() {

}

public void annulla() {

}

public void setTipoVisita(String tipoVisita) {

}

public String getTipoVisita() {
	return null;
}

public void setPazienteNome(String pazienteNome) {

}

public String getPazienteNome() {
	return null;
}

public void setPazienteCognome(String pazienteCognome) {

}

public String getPazienteCognome() {
	return null;
}

public void setMese(int mese) {

}

public int getMese() {
	return 0;
}

public void setAnno(int anno) {

}

public int getAnno() {
	return 0;
}

public void setGiorno(int giorno) {

}

public int getGiorno() {
	return 0;
}

public void setMedicoNome(String medicoNome) {

}

public String getMedicoNome() {
```

```
            return null;
        }

        public void setMedicoCognome(String medicoCognome) {

        }

        public String getMedicoCognome() {
            return null;
        }

}
```

1.1.8 Package RicercaPaziente

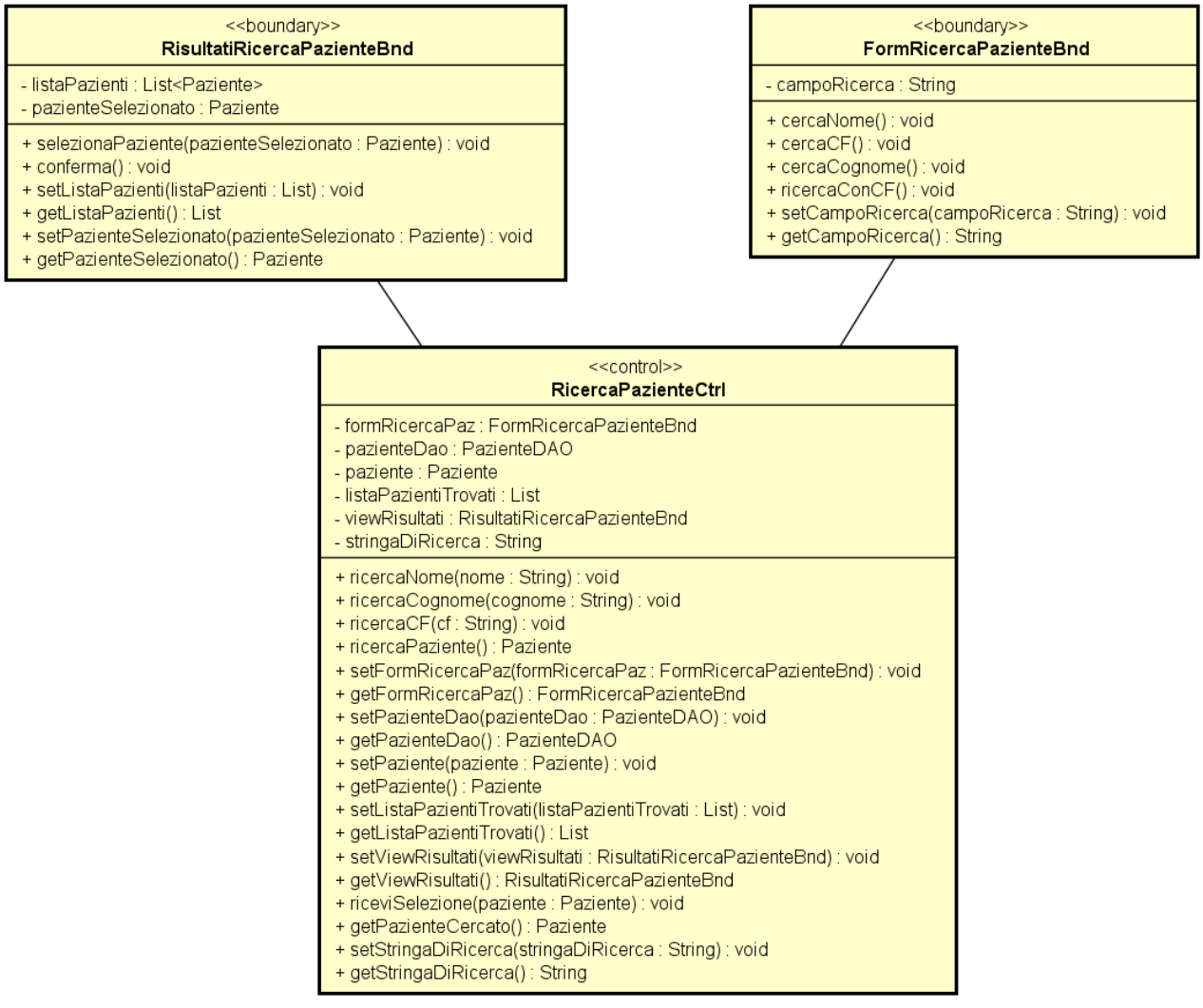


FIGURA 114 - PACKAGE RICERCAPAZIENTE

1.1.8.1 Classe FormRicercaPazienteBnd

```
package SHIVA.client.ricercapaziente;

/**
 * Form utilizzato dal receptionist/medico per effettuare la ricerca di un Paziente
 *
 * @author Autore 2
 */
public class FormRicercaPazienteBnd {

	private String campoRicerca;

	/**
	 * richiede la ricerca per nome
	 */
	public void cercaNome() {

	}

	/**
	 * richiede la ricerca per codice fiscale
	 */
	public void cercaCF() {

	}

	/**
	 * richiede la ricerca per cognome
	 */
	public void cercaCognome() {

	}

	public void ricercaConCF() {

	}

	public void setCampoRicerca(String campoRicerca) {

	}

	public String getCampoRicerca() {
		return null;
	}

}
```

1.1.8.2 Classe RicercaPazienteCtrl

```
package SHIVA.client.ricercapaziente;

import SHIVA.server.persistenza.PazienteDAO;
import SHIVA.client.modello.Paziente;
import java.util.List;

/**
 * Gestisce la ricerca di un paziente.
 *
 * @author Autore 1
 */
public class RicercaPazienteCtrl {

	private FormRicercaPazienteBnd formRicercaPaz;

	private PazienteDAO pazienteDao;

	private Paziente paziente;

	private List listaPazientiTrovati;

	/**
	 * la vista con cui mostrerà i risultati.
	 */
	private RisultatiRicercaPazienteBnd viewRisultati;

	/**
	 * la stringa che verrà utilizzata per la ricerca.
	 */
	private String stringaDiRicerca;

	public void ricercaNome(String nome) {

	}

	public void ricercaCognome(String cognome) {

	}

	public void ricercaCF(String cf) {

	}

	/**
	 * metodo che permette di offrire il servizio di ricerca ad altre control.
	 */
	public Paziente ricercaPaziente() {
		return null;
	}

	public void setFormRicercaPaz(FormRicercaPazienteBnd formRicercaPaz) {

	}

	public FormRicercaPazienteBnd getFormRicercaPaz() {
		return null;
	}

	public void setPazienteDao(PazienteDAO pazienteDao) {

	}
```

```
	public PazienteDAO getPazienteDao() {
		return null;
	}

	public void setPaziente(Paziente paziente) {

	}

	public Paziente getPaziente() {
		return null;
	}

	public void setListaPazientiTrovati(List listaPazientiTrovati) {

	}

	public List getListaPazientiTrovati() {
		return null;
	}

	public void setViewRisultati(RisultatiRicercaPazienteBnd viewRisultati) {

	}

	public RisultatiRicercaPazienteBnd getViewRisultati() {
		return null;
	}

	/**
	 * metodo tramite cui riceve la selezione dalla lista dei risultati.
	 */
	public void riceviSelezione(Paziente paziente) {

	}

	public Paziente getPazienteCercato() {
		return null;
	}

	public void setStringaDiRicerca(String stringaDiRicerca) {

	}

	public String getStringaDiRicerca() {
		return null;
	}

}
```

1.1.8.3 Classe RisultatiRicercaPazienteBnd

```
package SHIVA.client.ricercapaziente;

import java.util.List;
import SHIVA.client.modello.Paziente;

/**
 * Vista con cui SHIVA comunica i risultati della ricerca sui pazienti e
permette la selezione del paziente cercato.
 *
 * @author Autore 2
 */
public class RisultatiRicercaPazienteBnd {

	private List listaPazienti;

	private Paziente pazienteSelezionato;

	public void selezionaPaziente(Paziente pazienteSelezionato) {

	}

	public void conferma() {

	}

	public void setListaPazienti(List listaPazienti) {

	}

	public List getListaPazienti() {
		return null;
	}

	public void setPazienteSelezionato(Paziente pazienteSelezionato) {

	}

	public Paziente getPazienteSelezionato() {
		return null;
	}

}
```

1.1.9 Package RicercaUtente

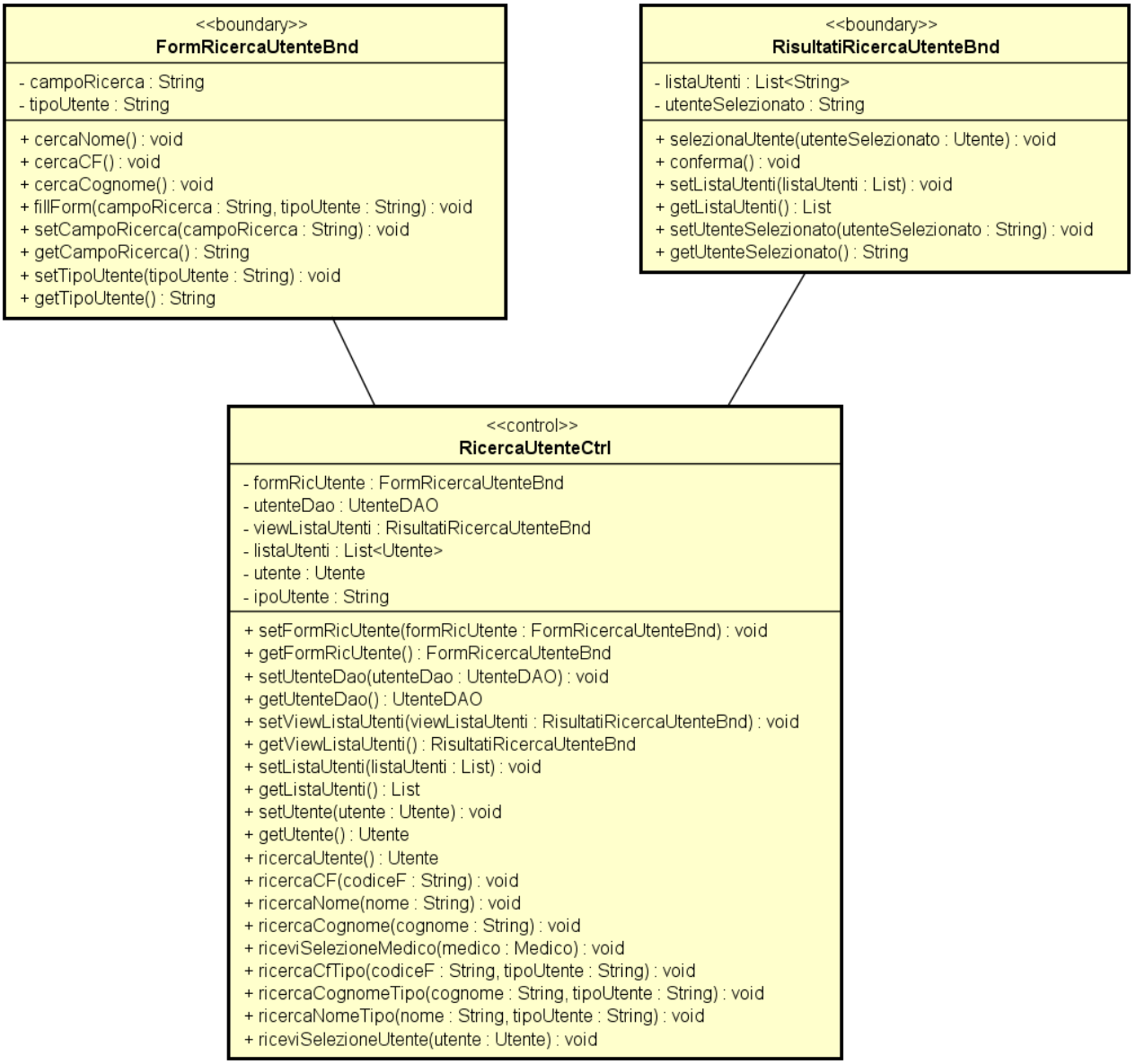


FIGURA 115 - PACKAGE RICERCAUTENTE

1.1.9.1 Classe FormRicercaUtenteBnd

```
package SHIVA.client.ricercautente;

/**
 * Form utilizzato dall'amministratore di sistema per effettuare la ricerca di
un receptionist o di un medico.
 *
 * @author Autore 2
 */
public class FormRicercaUtenteBnd {

	private String campoRicerca;

	private String tipoUtente;

	/**
	 * permette la ricerca per nome
	 */
	public void cercaNome() {

	}

	/**
	 * permette la ricerca per codice fiscale
	 */
	public void cercaCF() {

	}

	/**
	 * permette la ricerca per cognome
	 */
	public void cercaCognome() {

	}

	/**
	 * permette di inserire i dati di ricerca
	 */
	public void fillForm(String campoRicerca, String tipoUtente) {

	}

	public void setCampoRicerca(String campoRicerca) {

	}

	public String getCampoRicerca() {
		return null;
	}

	public void setTipoUtente(String tipoUtente) {

	}

	public String getTipoUtente() {
		return null;
	}

}
```

1.1.9.2 Classe RicercaUtenteCtrl

```
package SHIVA.client.ricercautente;

import SHIVA.server.persistenza.UtenteDAO;
import java.util.List;
import SHIVA.client.modello.Utente;
import SHIVA.client.modello.Medico;

/**
 * Gestisce la ricerca degli utenti.
 *
 * @author Autore 2
 */
public class RicercaUtenteCtrl {

	/**
	 * form per l'inserimento dei dati per la ricerca.
	 */
	private FormRicercaUtenteBnd formRicUtente;

	private UtenteDAO utenteDao;

	/**
	 * vista per la visualizzazione dei risultati della ricerca.
	 */
	private RisultatiRicercaUtenteBnd viewListaUtenti;

	private List listaUtenti;

	private Utente utente;

	private String ipoUtente;

	public void setFormRicUtente(FormRicercaUtenteBnd formRicUtente) {

	}

	public FormRicercaUtenteBnd getFormRicUtente() {
		return null;
	}

	public void setUtenteDao(UtenteDAO utenteDao) {

	}

	public UtenteDAO getUtenteDao() {
		return null;
	}

	public void setViewListaUtenti(RisultatiRicercaUtenteBnd viewListaUtenti)
{

	}

	public RisultatiRicercaUtenteBnd getViewListaUtenti() {
		return null;
	}

	public void setListaUtenti(List listaUtenti) {

	}
```

```
	public List getListaUtenti() {
		return null;
	}

	public void setUtente(Utente utente) {

	}

	public Utente getUtente() {
		return null;
	}

	public Utente ricercaUtente() {
		return null;
	}

	/**
	 * ricerca di tutti gli utenti che hanno un dato codice fiscale.
	 */
	public void ricercaCF(String codiceF) {

	}

	/**
	 * ricerca di tutti gli utenti che hanno un dato nome.
	 */
	public void ricercaNome(String nome) {

	}

	/**
	 * ricerca di tutti gli utenti che hanno un dato cognome.
	 */
	public void ricercaCognome(String cognome) {

	}

	/**
	 * metodo tramite cui la control riceve l'utente selezionato.
	 */
	public void riceviSelezioneMedico(Medico medico) {

	}

	/**
	 * ricerca degli tutti gli utenti di un paricolare tipo che hanno un dato
codice fiscale.
	 */
	public void ricercaCfTipo(String codiceF, String tipoUtente) {

	}

	/**
	 * ricerca degli tutti gli utenti di un paricolare tipo che hanno un dato
cognome.
	 */
	public void ricercaCognomeTipo(String cognome, String tipoUtente) {

	}

	/**
	 * ricerca degli tutti gli utenti di un paricolare tipo che hanno un dato
nome.
```

```
	 */
	public void ricercaNomeTipo(String nome, String tipoUtente) {

	}

	public void riceviSelezioneUtente(Utente utente) {

	}

}
```

1.1.9.3 Classe RisultatiRicercaUtenteBnd

```
package SHIVA.client.ricercautente;

import java.util.List;
import SHIVA.client.modello.Utente;

/**
 * Vista con cui SHIVA comunica i risultati della ricerca utente e permette la selezione di un utente.
 *
 * @author Autore 2
 */
public class RisultatiRicercaUtenteBnd {

	private List listaUtenti;

	private String utenteSelezionato;

	public void selezionaUtente(Utente utenteSelezionato) {

	}

	public void conferma() {

	}

	public void setListaUtenti(List listaUtenti) {

	}

	public List getListaUtenti() {
		return null;
	}

	public void setUtenteSelezionato(String utenteSelezionato) {

	}

	public String getUtenteSelezionato() {
		return null;
	}

}
```

1.1.10 Package Utente

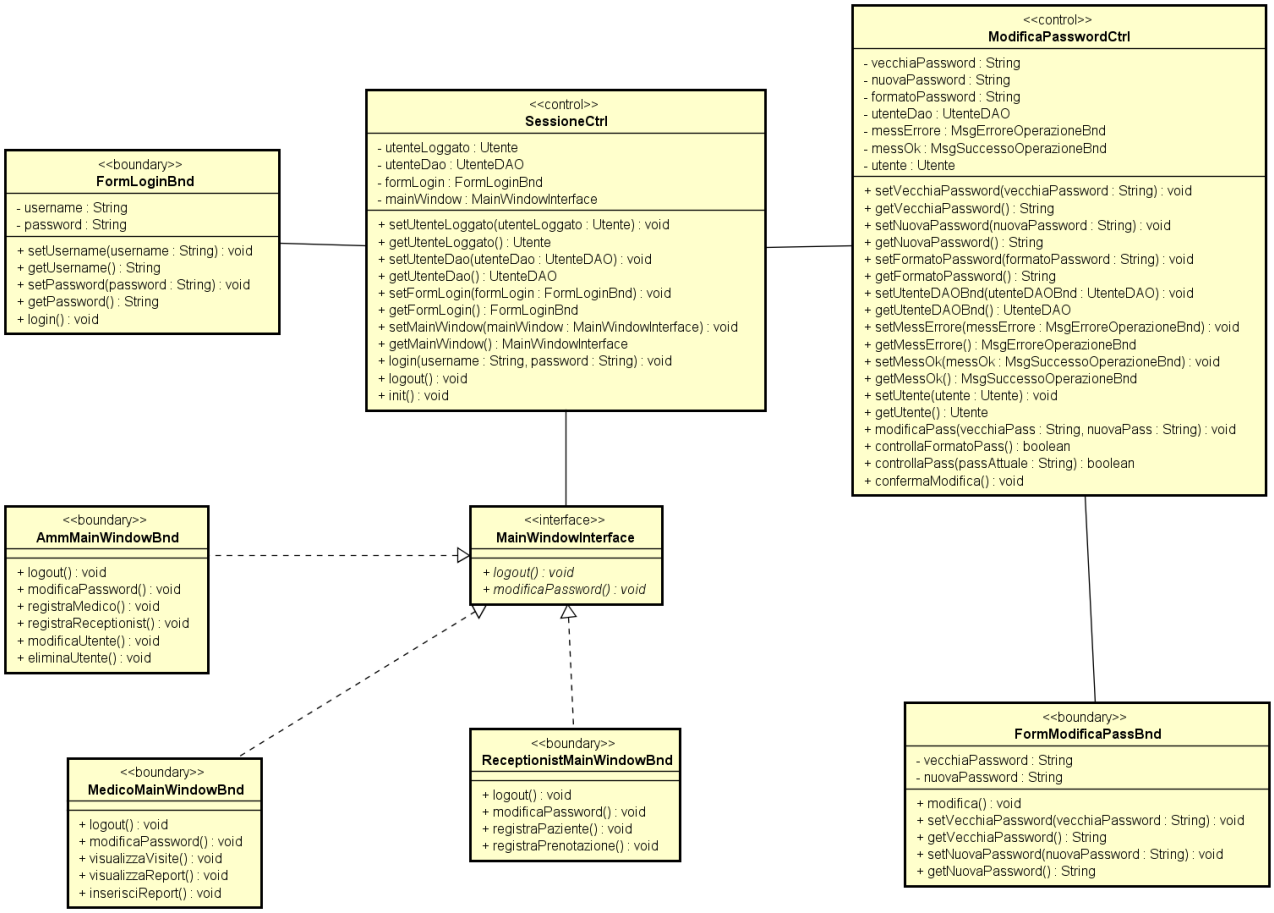


FIGURA 116 - PACKAGE UTENTE

1.1.10.1 Classe AmmMainWindowBnd

```
package SHIVA.client.utente;

/**
 * Vista principale presentata all'amministratore di sistema dopo il login. Gli permette di avviare le principali funzionalità.
 *
 * @author Autore 2
 */
public class AmmMainWindowBnd implements MainWindowInterface {

	public void logout() {

	}

	/**
	 * permette di avviare l'operazione di modifica password.
	 */
	public void modificaPassword() {

	}

	/**
	 * permette di avviare l'operazione di registra medico.
```

```
	 */
	public void registraMedico() {

	}

	/**
	 * permette di avviare l'operazione di registra receptionist.
	 */
	public void registraReceptionist() {

	}

	/**
	 * permette di avviare l'operazione di modifica utente.
	 */
	public void modificaUtente() {

	}

	/**
	 * permette di avviare l'operazione di elimina utente.
	 */
	public void eliminaUtente() {

	}

}
```

1.1.10.2 Classe FormLoginBnd

```
package SHIVA.client.utente;

/**
 * Il form utiilizzato dall'utente per inserire i dati di login.
 *
 * @author Autore 1
 */
public class FormLoginBnd {

	private String username;

	private String password;

	public void setUsername(String username) {

	}

	public String getUsername() {
		return null;
	}

	public void setPassword(String password) {

	}

	public String getPassword() {
		return null;
	}

	public void login() {

	}
}
```

1.1.10.3 Classe FormModificaPasswordBnd

```
package SHIVA.client.utente;

/**
 * Form utilizzato dall'utente per modificare la propria password.
 *
 * @author Autore 1
 */
public class FormModificaPassBnd {

	private String vecchiaPassword;

	private String nuovaPassword;

	/**
	 * permette di inserire la vecchia password e la nuova password.
	 */
	public void modifica() {

	}

	public void setVecchiaPassword(String vecchiaPassword) {

	}

	public String getVecchiaPassword() {
		return null;
	}

	public void setNuovaPassword(String nuovaPassword) {

	}

	public String getNuovaPassword() {
		return null;
	}

}
```

1.1.10.4 Interfaccia MainWindowInterface

```
package SHIVA.client.utente;

/**
 * Interfaccia per la finestra principale dell'utente
 *
 * @author Autore 1
 */
public interface MainWindowInterface {

	public abstract void logout();

	public abstract void modificaPassword();

}
```

1.1.10.5 Classe MedicoMainWindowBnd

```
package SHIVA.client.utente;

/**
 * Vista principale presentata al medico dopo il login. Gli permette di avviare
le principali funzionalità.
 *
 * @author Autore 1
 */
public class MedicoMainWindowBnd implements MainWindowInterface {

	/**
	 * permette di avviare l'operazione di logout
	 */
	public void logout() {

	}

	/**
	 * permette di avviare l'operazione di modifica password
	 */
	public void modificaPassword() {

	}

	/**
	 * permette di avviare l'operazione di visualizza visite
	 */
	public void visualizzaVisite() {

	}

	/**
	 * permette di avviare l'operazione di visualizza report
	 */
	public void visualizzaReport() {

	}

	/**
	 * permette di avviare l'operazione di inserisci report.
	 */
	public void inserisciReport() {

	}

}
```

1.1.10.6 Classe ModificaPasswordCtrl

```
package SHIVA.client.utente;

import SHIVA.server.persistenza.UtenteDAO;
import SHIVA.client.messaggi.MsgErroreOperazioneBnd;
import SHIVA.client.messaggi.MsgSuccessoOperazioneBnd;
import SHIVA.client.modello.Utente;

/**
 * Gestisce la modifica della password dell'utente.
 *
 * @author Autore 2
 */
public class ModificaPasswordCtrl {

	private String vecchiaPassword;

	private String nuovaPassword;

	private String formatoPassword;

	private UtenteDAO utenteDao;

	private MsgErroreOperazioneBnd messErrore;

	private MsgSuccessoOperazioneBnd messOk;

	private Utente utente;

	public void setVecchiaPassword(String vecchiaPassword) {

	}

	public String getVecchiaPassword() {
		return null;
	}

	public void setNuovaPassword(String nuovaPassword) {

	}

	public String getNuovaPassword() {
		return null;
	}

	public void setFormatoPassword(String formatoPassword) {

	}

	public String getFormatoPassword() {
		return null;
	}

	public void setUtenteDAOBnd(UtenteDAO utenteDAOBnd) {

	}

	public UtenteDAO getUtenteDAOBnd() {
		return null;
	}

	public void setMessErrore(MsgErroreOperazioneBnd messErrore) {
```

```
	}

	public MsgErroreOperazioneBnd getMessErrore() {
		return null;
	}

	public void setMessOk(MsgSuccessoOperazioneBnd messOk) {

	}

	public MsgSuccessoOperazioneBnd getMessOk() {
		return null;
	}

	public void setUtente(Utente utente) {

	}

	public Utente getUtente() {
		return null;
	}

	public void modificaPass(String vecchiaPass, String nuovaPass) {

	}

	/**
	 * controlla che il formato della nuova password sia corretto.
	 */
	public boolean controllaFormatoPass() {
		return false;
	}

	/**
	 * controlla che la vecchia password sia corretta.
	 */
	public boolean controllaPass(String passAttuale) {
		return false;
	}

	public void confermaModifica() {

	}

}
```

1.1.10.7 Classe ReceptionistMainWindowBnd

```
package SHIVA.client.utente;

/**
 * Vista principale presentata al receptionist dopo il login. Gli permette di
avviare le principali funzionalità.
 *
 * @author Autore 2
 */
public class ReceptionistMainWindowBnd implements MainWindowInterface {

	/**
	 * permette di avviare l'operazione di logout
	 */
	public void logout() {

	}

	/**
	 * permette di avviare l'operazione di modifica password
	 */
	public void modificaPassword() {

	}

	/**
	 * permette di avviare l'operazione di registra paziente
	 */
	public void registraPaziente() {

	}

	/**
	 * permette di avviare l'operazione di registra prenotazione
	 */
	public void registraPrenotazione() {

	}

}
```

1.1.10.8 Classe SessioneCtrl

```
package SHIVA.client.utente;

import SHIVA.client.modello.Utente;
import SHIVA.server.persistenza.UtenteDAO;

/**
 * Gestisce la sessione dell'utente, inclusi il processo di login e il processo
di logout.
 *
 * @author Autore 1
 */
public class SessioneCtrl {

	private Utente utenteLoggato;

	private UtenteDAO utenteDao;
```

```
	private FormLoginBnd formLogin;

	/**
	 * la finestra principale dell'utente autenticato
	 */
	private MainWindowInterface mainWindow;

	public void setUtenteLoggato(Utente utenteLoggato) {

	}

	public Utente getUtenteLoggato() {
		return null;
	}

	public void setUtenteDao(UtenteDAO utenteDao) {

	}

	public UtenteDAO getUtenteDao() {
		return null;
	}

	public void setFormLogin(FormLoginBnd formLogin) {

	}

	public FormLoginBnd getFormLogin() {
		return null;
	}

	public void setMainWindow(MainWindowInterface mainWindow) {

	}

	public MainWindowInterface getMainWindow() {
		return null;
	}

	public void login(String username, String password) {

	}

	public void logout() {

	}

	/**
	 * inizializzazione sessione
	 */
	public void init() {

	}

}
```

1.2 Server

1.2.1 Package InterfacciaServer

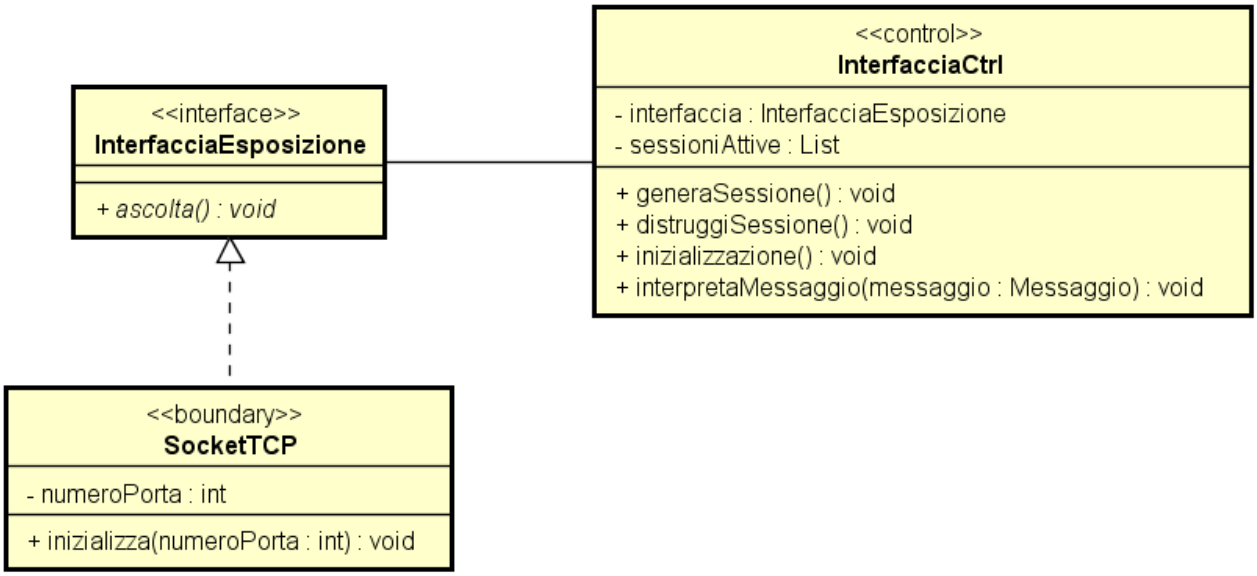


FIGURA 117 - PACKAGE INTERFACCIASERVER

1.2.1.1 Classe InterfacciaCtrl

```
package SHIVA.server.interfacciaserver;

import java.util.List;
import SHIVA.client.messaggicomunicazione.Messaggio;

/**
 * Controller dell'interfaccia server
 *
 * @author Autore 1
 */
public class InterfacciaCtrl {

	private InterfacciaEsposizione interfaccia;

	private List sessioniAttive;

	public void generaSessione() {

	}

	public void distruggiSessione() {

	}

	public void inizializzazione() {

	}
```

```
	public void interpretaMessaggio(Messaggio messaggio) {

	}

}
```

1.2.1.2 Interfaccia InterfacciaEsposizione

```
package SHIVA.server.interfacciaserver;

/**
 * Interfaccia con cui il server si espone ai client
 *
 * @author Autore 2
 */
public interface InterfacciaEsposizione {

	public abstract void ascolta();

}
```

1.2.1.3 Classe SocketTCP

```
package SHIVA.server.interfacciaserver;

/**
 * Boundary per la connessione TCP del server
 *
 * @author Autore 2
 */
public class SocketTCP implements InterfacciaEsposizione {

	private int numeroPorta;

	public void inizializza(int numeroPorta) {

	}


	/**
	 * @see SHIVA.server.interfacciaserver.InterfacciaEsposizione#ascolta()
	 */
	public void ascolta() {

	}

}
```

1.2.2 Package MessaggiComunicazione

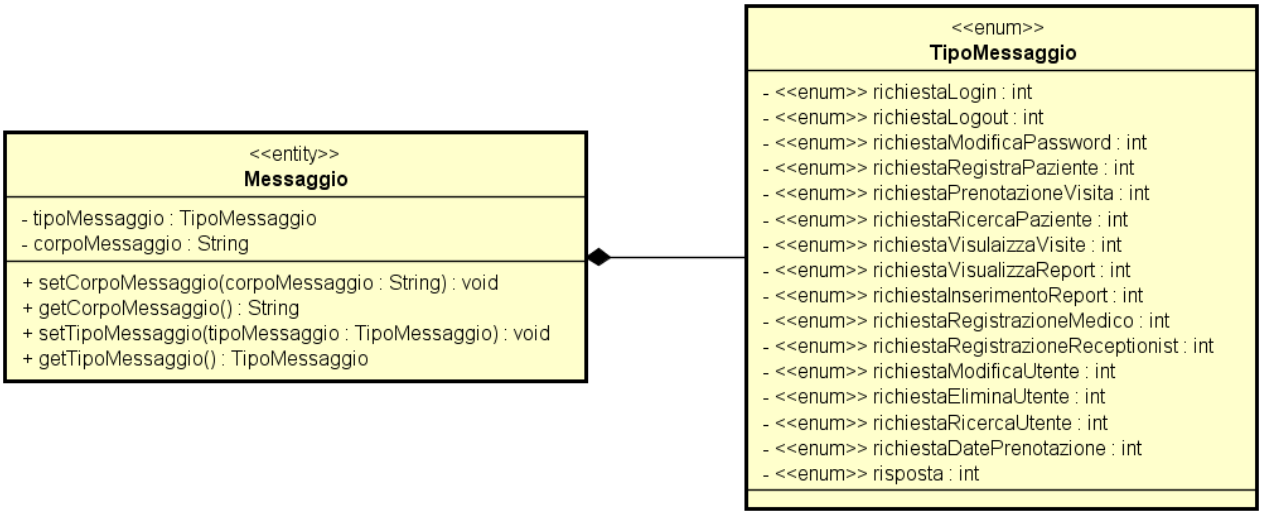


FIGURA 118 - PACKAGE MESSAGGICOMUNICAZIONE

1.2.2.1 Classe Messaggio

```
package SHIVA.server.messaggicomunicazione;

/**
 * Entità che rappresenta il generico messaggio inviato fra client e server
 *
 * @author Autore 1
 */
public class Messaggio {

	private TipoMessaggio tipoMessaggio;

	private String corpoMessaggio;

	public void setCorpoMessaggio(String corpoMessaggio) {

	}

	public String getCorpoMessaggio() {
		return null;
	}

	public void setTipoMessaggio(TipoMessaggio tipoMessaggio) {

	}

	public TipoMessaggio getTipoMessaggio() {
		return null;
	}

}
```

1.2.2.2 Enumerazione TipoMessaggio

```
package SHIVA.server.messaggicomunicazione;

/**
 * Elenca i tipi di messaggio che client e server possono trasmettersi. Ogni
tipo indica un particolare operazione da svolgere
 *
 * @author Autore 2
 */
public enum TipoMessaggio {

	;

	private int richiestaLogin;

	private int richiestaLogout;

	private int richiestaModificaPassword;

	private int richiestaRegistraPaziente;

	private int richiestaPrenotazioneVisita;

	private int richiestaRicercaPaziente;

	private int richiestaVisulaizzaVisite;

	private int richiestaVisualizzaReport;

	private int richiestaInserimentoReport;

	private int richiestaRegistrazioneMedico;

	private int richiestaRegistrazioneReceptionist;

	private int richiestaModificaUtente;

	private int richiestaEliminaUtente;

	private int richiestaRicercaUtente;

	private int richiestaDatePrenotazione;

	private int risposta;

}
```

1.2.3 Package Modello

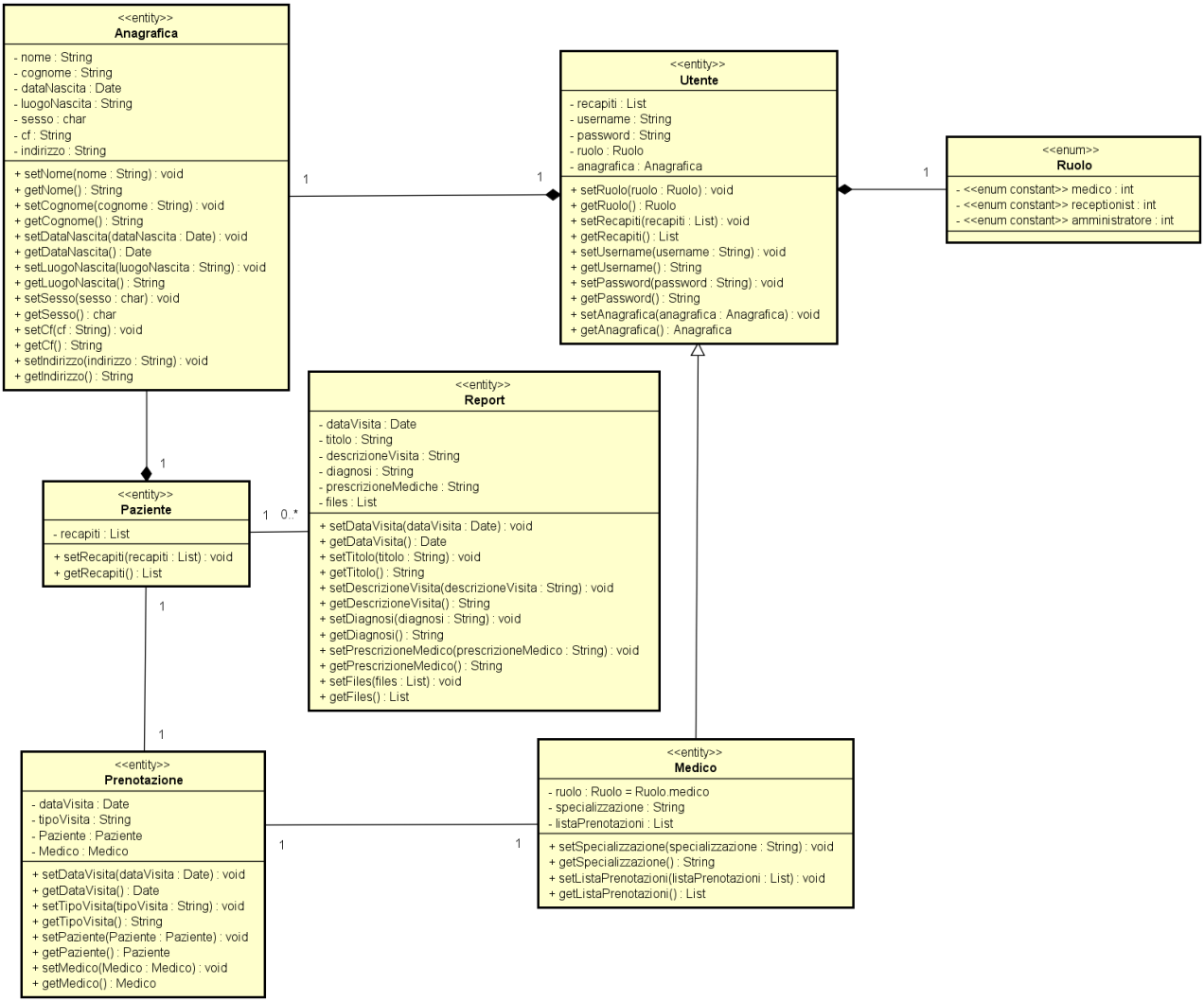


FIGURA 119 - PACKAGE MODELLO

1.2.3.1 Classe Anagrafica

```
package SHIVA.server.modello;

import java.util.Date;

/**
 * Rappresenta i dati anagrafici di un utente o di un paziente.
 *
 * @author Autore 2
 */
public class Anagrafica {

	private String nome;

	private String cognome;

	private Date dataNascita;

	private String luogoNascita;

	private char sesso;

	/**
	 * Il codice fiscale.
	 */
	private String cf;

	/**
	 * L'indirizzo di residenza.
	 */
	private String indirizzo;

	public void setNome(String nome) {

	}

	public String getNome() {
		return null;
	}

	public void setCognome(String cognome) {

	}

	public String getCognome() {
		return null;
	}

	public void setDataNascita(Date dataNascita) {

	}

	public Date getDataNascita() {
		return null;
	}

	public void setLuogoNascita(String luogoNascita) {

	}

	public String getLuogoNascita() {
		return null;
```

```
	}

	public void setSesso(char sesso) {

	}

	public char getSesso() {
		return 0;
	}

	public void setCf(String cf) {

	}

	public String getCf() {
		return null;
	}

	public void setIndirizzo(String indirizzo) {

	}

	public String getIndirizzo() {
		return null;
	}

}
```

1.2.3.2 Classe Medico

```
package SHIVA.server.modello;

import java.util.List;

/**
 * Rappresenta un utente di tipo medico. Eredita da Utente.
 *
 * @author Autore 1
 */
public class Medico extends Utente {

	private final Ruolo ruolo = Ruolo.medico;

	private String specializzazione;

	private List listaPrenotazioni;

	public void setSpecializzazione(String specializzazione) {

	}

	public String getSpecializzazione() {
		return null;
	}

	public void setListaPrenotazioni(List listaPrenotazioni) {

	}

	public List getListaPrenotazioni() {
```

```
		return null;
	}

}
```

1.2.3.3 Classe Paziente

```
package SHIVA.server.modello;

import java.util.List;

/**
 * Rappresenta un Paziente della struttura medica.
 *
 * @author Autore 2
 */
public class Paziente {

	/**
	 * lista dei recapiti (sia telefonici che email)
	 */
	private List recapiti;

	public void setRecapiti(List recapiti) {

	}

	public List getRecapiti() {
		return null;
	}

}
```

1.2.3.4 Classe Prenotazione

```
package SHIVA.server.modello;

import java.util.Date;

/**
 * Rappresenta le prenotazioni delle visite effettuate all'interno della
struttura medica.
 *
 * @author Autore 2
 */
public class Prenotazione {

	private Date dataVisita;

	/**
	 * Il tipo della visita prenotata. Deve corrispondere ad una delle
specializzazioni dei medici presenti nella struttura.
	 */
	private String tipoVisita;

	/**
	 * il Paziente associato alla visita
	 */
	private Paziente Paziente;
```

```
	/**
	 * il Medico associato alla visita
	 */
	private Medico Medico;

	public void setDataVisita(Date dataVisita) {

	}

	public Date getDataVisita() {
		return null;
	}

	public void setTipoVisita(String tipoVisita) {

	}

	public String getTipoVisita() {
		return null;
	}

	public void setPaziente(Paziente Paziente) {

	}

	public Paziente getPaziente() {
		return null;
	}

	public void setMedico(Medico Medico) {

	}

	public Medico getMedico() {
		return null;
	}

}
```

1.2.3.5 Classe Report

```
package SHIVA.server.modello;

import java.util.Date;
import java.util.List;

/**
 * Rappresenta il report associato ad una visita e compilato dal medico.
 *
 * @author Autore 2
 */
public class Report {

	private Date dataVisita;

	private String titolo;

	private String descrizioneVisita;

	private String diagnosi;
```

```
	private String prescrizioneMediche;

	/**
	 * Lista di file serializzati in formato array di byte.
	 */
	private List files;

	public void setDataVisita(Date dataVisita) {

	}

	public Date getDataVisita() {
		return null;
	}

	public void setTitolo(String titolo) {

	}

	public String getTitolo() {
		return null;
	}

	public void setDescrizioneVisita(String descrizioneVisita) {

	}

	public String getDescrizioneVisita() {
		return null;
	}

	public void setDiagnosi(String diagnosi) {

	}

	public String getDiagnosi() {
		return null;
	}

	public void setPrescrizioneMedico(String prescrizioneMedico) {

	}

	public String getPrescrizioneMedico() {
		return null;
	}

	public void setFiles(List files) {

	}

	public List getFiles() {
		return null;
	}

}
```

1.2.3.6 Enumerazione Ruolo

```
package SHIVA.server.modello;

/**
 * L'enum che definisce i ruoli possibili.
 *
 * @author Autore 1
 */
public enum Ruolo {

	medico,

	receptionist,

	amministratore;

}
```

1.2.3.7 Classe Utente

```
package SHIVA.server.modello;

import java.util.List;

/**
 * Rappresenta un utente con un ruolo qualsiasi tra amministratore di sistema,
receptionist e medico.
 *
 * @author Autore 1
 *
 */
public class Utente {

	private List recapiti;

	private String username;

	private String password;

	private Ruolo ruolo;

	private Anagrafica anagrafica;

	public void setRuolo(Ruolo ruolo) {

	}

	public Ruolo getRuolo() {
		return null;
	}

	public void setRecapiti(List recapiti) {

	}

	public List getRecapiti() {
		return null;
	}
```

```
	public void setUsername(String username) {

	}

	public String getUsername() {
		return null;
	}

	public void setPassword(String password) {

	}

	public String getPassword() {
		return null;
	}

	public void setAnagrafica(Anagrafica anagrafica) {

	}

	public Anagrafica getAnagrafica() {
		return null;
	}
}
```

1.2.4 Package OperazioniCentralizzate

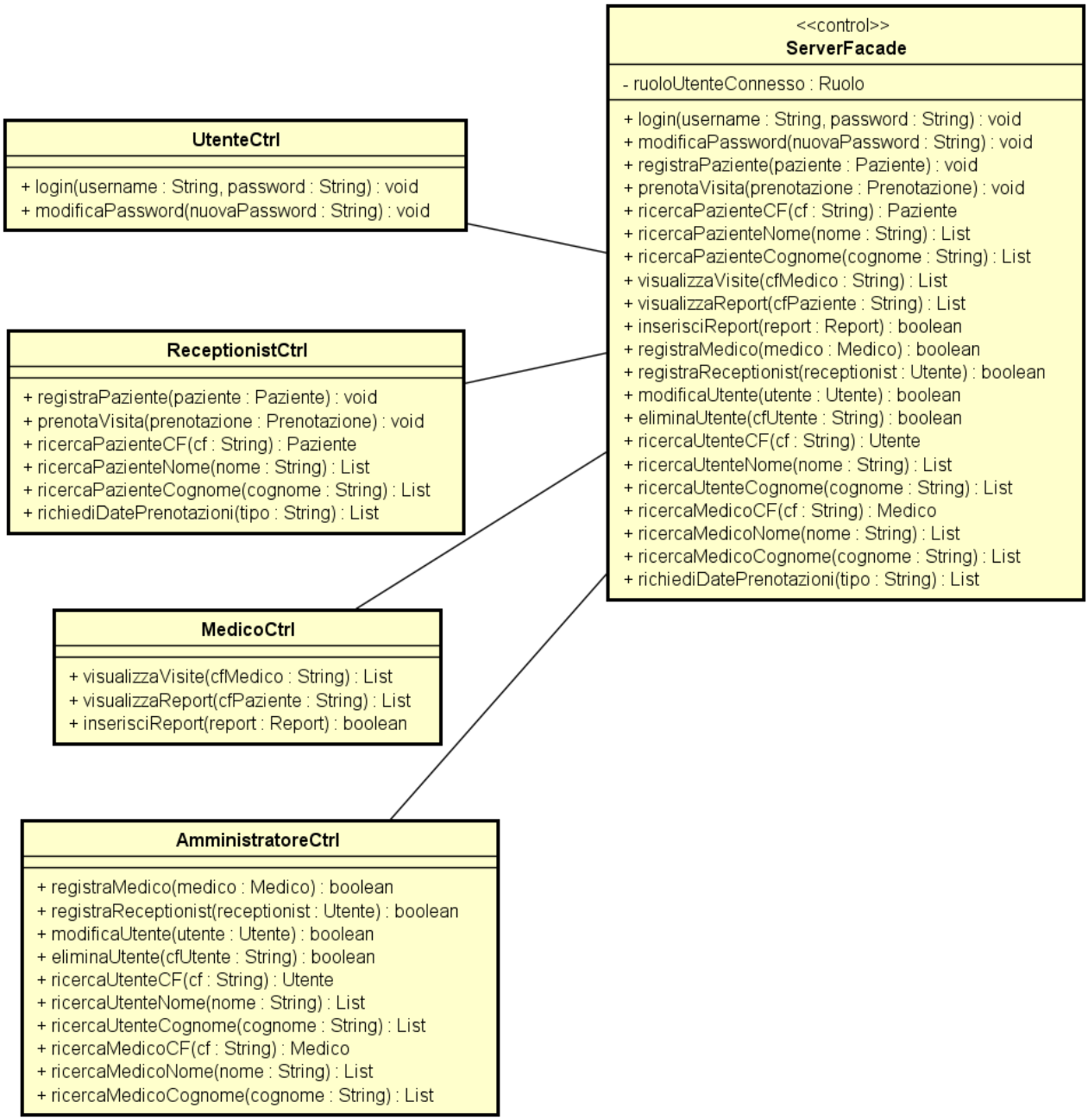


FIGURA 120 - PACKAGE OPERAZIONICENTRALIZZATE

1.2.4.1 Classe AmministratoreCtrl

```
package SHIVA.server.operazionicentralizzate;

import SHIVA.client.modello.Medico;
import SHIVA.server.modello.Utente;
import java.util.List;

/**
 * Controller del server che gestisce le richieste  dell'amministratore di sistema
 *
 * @author Autore 2
 */
public class AmministratoreCtrl {

	public boolean registraMedico(Medico medico) {
		return false;
	}

	public boolean registraReceptionist(Utente receptionist) {
		return false;
	}

	public boolean modificaUtente(Utente utente) {
		return false;
	}

	public boolean eliminaUtente(String cfUtente) {
		return false;
	}

	public Utente ricercaUtenteCF(String cf) {
		return null;
	}

	public List ricercaUtenteNome(String nome) {
		return null;
	}

	public List ricercaUtenteCognome(String cognome) {
		return null;
	}

	public Medico ricercaMedicoCF(String cf) {
		return null;
	}

	public List ricercaMedicoNome(String nome) {
		return null;
	}

	public List ricercaMedicoCognome(String cognome) {
		return null;
	}

}
```

1.2.4.2 Classe MedicoCtrl

```
package SHIVA.server.operazionicentralizzate;

import java.util.List;
import SHIVA.client.modello.Report;

/**
 * Controller del server che gestisce le richieste  del medico
 *
 * @author Autore 1
 */
public class MedicoCtrl {

	public List visualizzaVisite(String cfMedico) {
		return null;
	}

	public List visualizzaReport(String cfPaziente) {
		return null;
	}

	public boolean inserisciReport(Report report) {
		return false;
	}

}
```

1.2.4.3 Classe ReceptionistCtrl

```
package SHIVA.server.operazionicentralizzate;

import SHIVA.client.modello.Paziente;
import SHIVA.client.modello.Prenotazione;
import java.util.List;

/**
 * Controller del server che gestisce le richieste  del receptionist
 *
 * @author Autore 2
 */
public class ReceptionistCtrl {

	public void registraPaziente(Paziente paziente) {

	}

	public void prenotaVisita(Prenotazione prenotazione) {

	}

	public Paziente ricercaPazienteCF(String cf) {
		return null;
	}

	public List ricercaPazienteNome(String nome) {
		return null;
	}

	public List ricercaPazienteCognome(String cognome) {
		return null;
```

```
	}

	public List richiediDatePrenotazioni(String tipo) {
		return null;
	}

}
```

1.2.4.4 Classe ServerFacade

```
package SHIVA.server.operazionicentralizzate;

import SHIVA.client.modello.Ruolo;
import SHIVA.client.modello.Paziente;
import SHIVA.client.modello.Prenotazione;
import java.util.List;
import SHIVA.client.modello.Report;
import SHIVA.client.modello.Medico;
import SHIVA.client.modello.Utente;

/**
 * Control che unisce tutte le funzionalità fornite all'utente
 *
 * @author Autore 1
 */
public class ServerFacade {

	private Ruolo ruoloUtenteConnesso;

	public void login(String username, String password) {

	}

	public void modificaPassword(String nuovaPassword) {

	}

	public void registraPaziente(Paziente paziente) {

	}

	public void prenotaVisita(Prenotazione prenotazione) {

	}

	public Paziente ricercaPazienteCF(String cf) {
		return null;
	}

	public List ricercaPazienteNome(String nome) {
		return null;
	}

	public List ricercaPazienteCognome(String cognome) {
		return null;
	}

	public List visualizzaVisite(String cfMedico) {
		return null;
	}
```

```
	public List visualizzaReport(String cfPaziente) {
		return null;
	}

	public boolean inserisciReport(Report report) {
		return false;
	}

	public boolean registraMedico(Medico medico) {
		return false;
	}

	public boolean registraReceptionist(Utente receptionist) {
		return false;
	}

	public boolean modificaUtente(Utente utente) {
		return false;
	}

	public boolean eliminaUtente(String cfUtente) {
		return false;
	}

	public Utente ricercaUtenteCF(String cf) {
		return null;
	}

	public List ricercaUtenteNome(String nome) {
		return null;
	}

	public List ricercaUtenteCognome(String cognome) {
		return null;
	}

	public Medico ricercaMedicoCF(String cf) {
		return null;
	}

	/**
	 *
	 */
	public List ricercaMedicoNome(String nome) {
		return null;
	}

	public List ricercaMedicoCognome(String cognome) {
		return null;
	}

	public List richiediDatePrenotazioni(String tipo) {
		return null;
	}

}
```

1.2.4.5 Classe UtenteCtrl

```
package SHIVA.server.operazionicentralizzate;

/**
 * Controller del server che gestisce le richieste  dell'utente
 *
 * @author Autore 2
 */
public class UtenteCtrl {

	public void login(String username, String password) {

	}

	public void modificaPassword(String nuovaPassword) {

	}

}
```

1.2.5 Package Persistenza

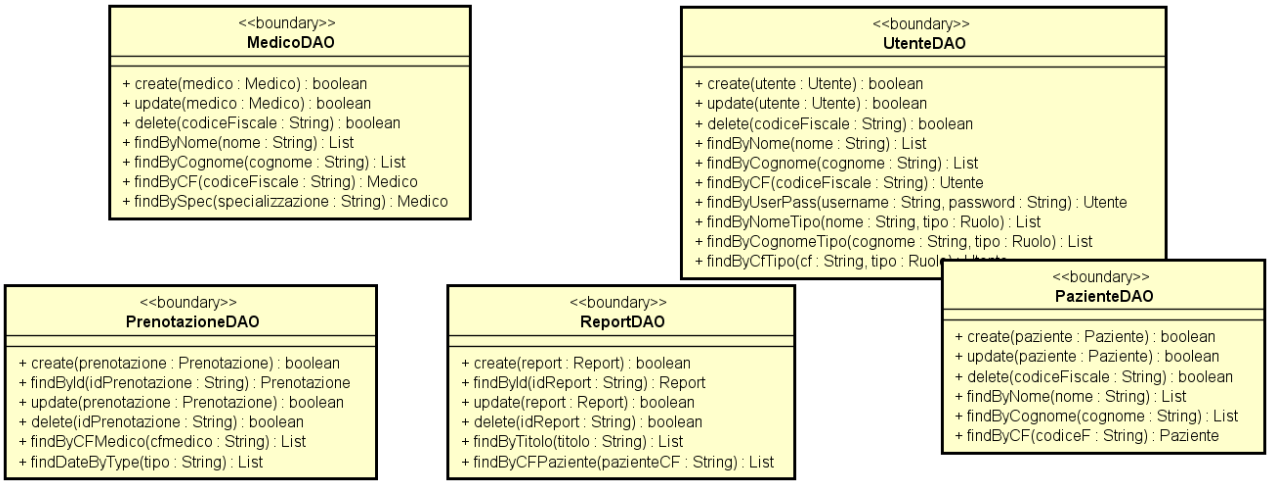


FIGURA 121 - PACKAGE PERSISTENZA

1.2.5.1 Classe MedicoDAO

```
package SHIVA.server.persistenza;

import SHIVA.client.modello.Medico;
import java.util.List;

/**
 * Si occupa di comunicare con il sistema che si occupa della persistenza effettiva degli oggetti Medico.
 *
 * @author Autore 1
 */
public class MedicoDAO {

	public boolean create(Medico medico) {
		return false;
	}
```

```
	public boolean update(Medico medico) {
		return false;
	}

	public boolean delete(String codiceFiscale) {
		return false;
	}

	public List findByNome(String nome) {
		return null;
	}

	public List findByCognome(String cognome) {
		return null;
	}

	public Medico findByCF(String codiceFiscale) {
		return null;
	}

	public Medico findBySpec(String specializzazione) {
		return null;
	}

}
```

1.2.5.2 Classe PazienteDAO

```
package SHIVA.server.persistenza;

import SHIVA.client.modello.Paziente;
import java.util.List;

/**
 * Si occupa di comunicare con il sistema che si occupa della persistenza effettiva degli oggetti Paziente.
 *
 * @author Autore 1
 */
public class PazienteDAO {

	public boolean create(Paziente paziente) {
		return false;
	}

	public boolean update(Paziente paziente) {
		return false;
	}

	public boolean delete(String codiceFiscale) {
		return false;
	}

	public List findByNome(String nome) {
		return null;
	}

	public List findByCognome(String cognome) {
		return null;
	}
```

```
	public Paziente findByCF(String codiceF) {
		return null;
	}

}
```

1.2.5.3 Classe PrenotazioneDAO

```
package SHIVA.server.persistenza;

import SHIVA.client.modello.Prenotazione;
import java.util.List;

/**
 * Si occupa di comunicare con il sistema che si occupa della persistenza
effettiva degli oggetti Prenotazione.
 *
 * @author Autore 1
 */
public class PrenotazioneDAO {

	public boolean create(Prenotazione prenotazione) {
		return false;
	}

	public Prenotazione findById(String idPrenotazione) {
		return null;
	}

	public boolean update(Prenotazione prenotazione) {
		return false;
	}

	public boolean delete(String idPrenotazione) {
		return false;
	}

	public List findByCFMedico(String cfmedico) {
		return null;
	}

	/**
	 * trova le date libere dato un tipo di visita.
	 */
	public List findDateByType(String tipo) {
		return null;
	}

}
```

1.2.5.4 Classe ReportDAO

```
package SHIVA.server.persistenza;

import SHIVA.client.modello.Report;
import java.util.List;

/**
 * Si occupa di comunicare con il sistema che si occupa della persistenza
effettiva degli oggetti Report.
 *
 * @author Autore 2
 */
public class ReportDAO {

	public boolean create(Report report) {
		return false;
	}

	public Report findById(String idReport) {
		return null;
	}

	public boolean update(Report report) {
		return false;
	}

	public boolean delete(String idReport) {
		return false;
	}

	public List findByTitolo(String titolo) {
		return null;
	}

	public List findByCFPaziente(String pazienteCF) {
		return null;
	}

}
```

1.2.5.5 Classe UtenteDAO

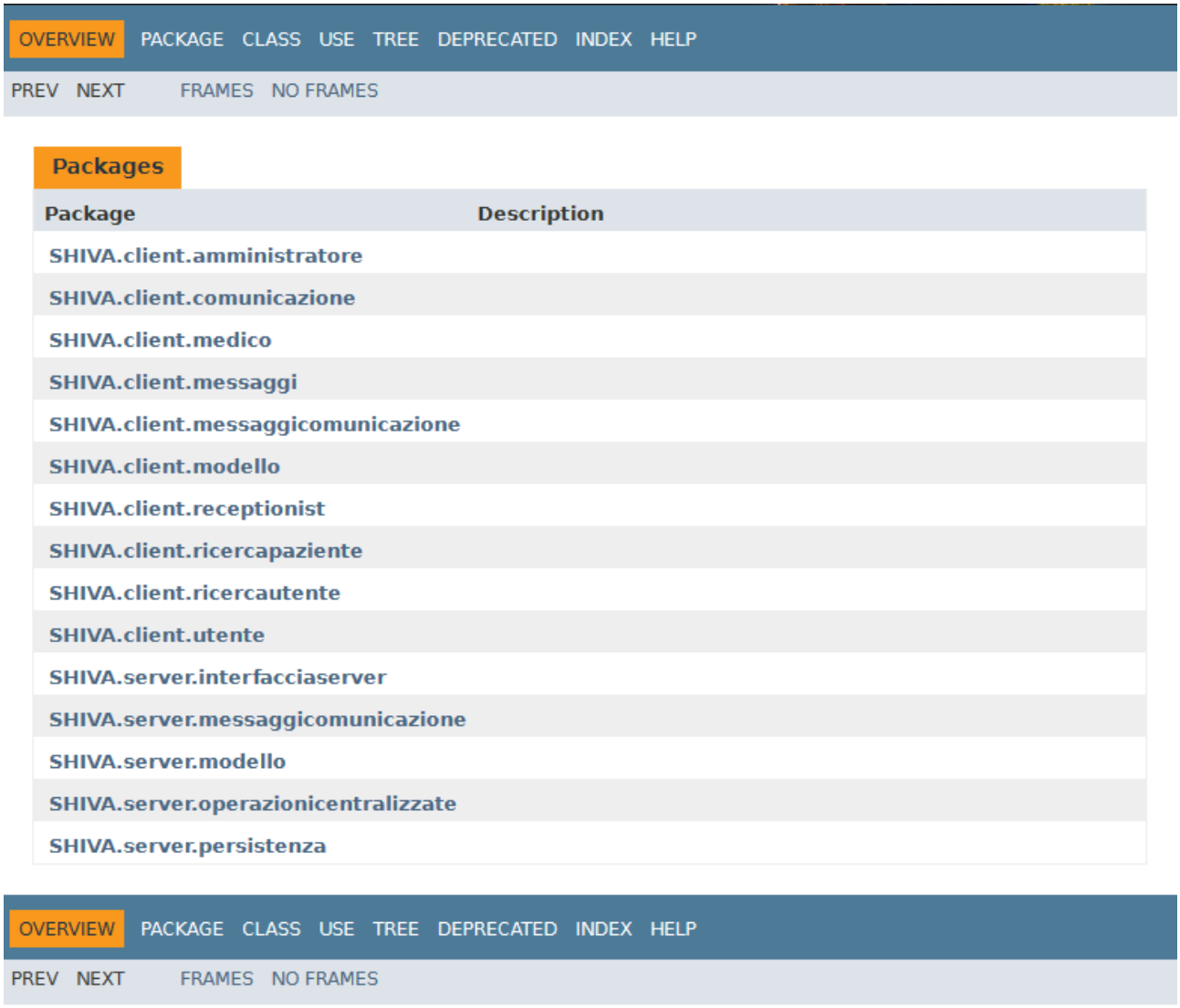
```
package SHIVA.server.persistenza;

import SHIVA.client.modello.Utente;
import java.util.List;

/**
 * Si occupa di comunicare con il sistema che si occupa della persistenza
effettiva degli oggetti Utente.
 *
 * @author Autore 1
 */
public class UtenteDAO {

	public boolean create(Utente utente) {
		return false;
	}

	public boolean update(Utente utente) {
```

```
		return false;
	}

	public boolean delete(String codiceFiscale) {
		return false;
	}

	public List findByNome(String nome) {
		return null;
	}

	public List findByCognome(String cognome) {
		return null;
	}

	public Utente findByCF(String codiceFiscale) {
		return null;
	}

	public Utente findByUserPass(String username, String password) {
		return null;
	}

}
```

2 Javadoc

In seguito alcuni Javadoc di SHIVA

2.1 Index



OVERVIEW PACKAGE CLASS USE TREE DEPRECATED INDEX HELP

PREV NEXT FRAMES NO FRAMES

Packages

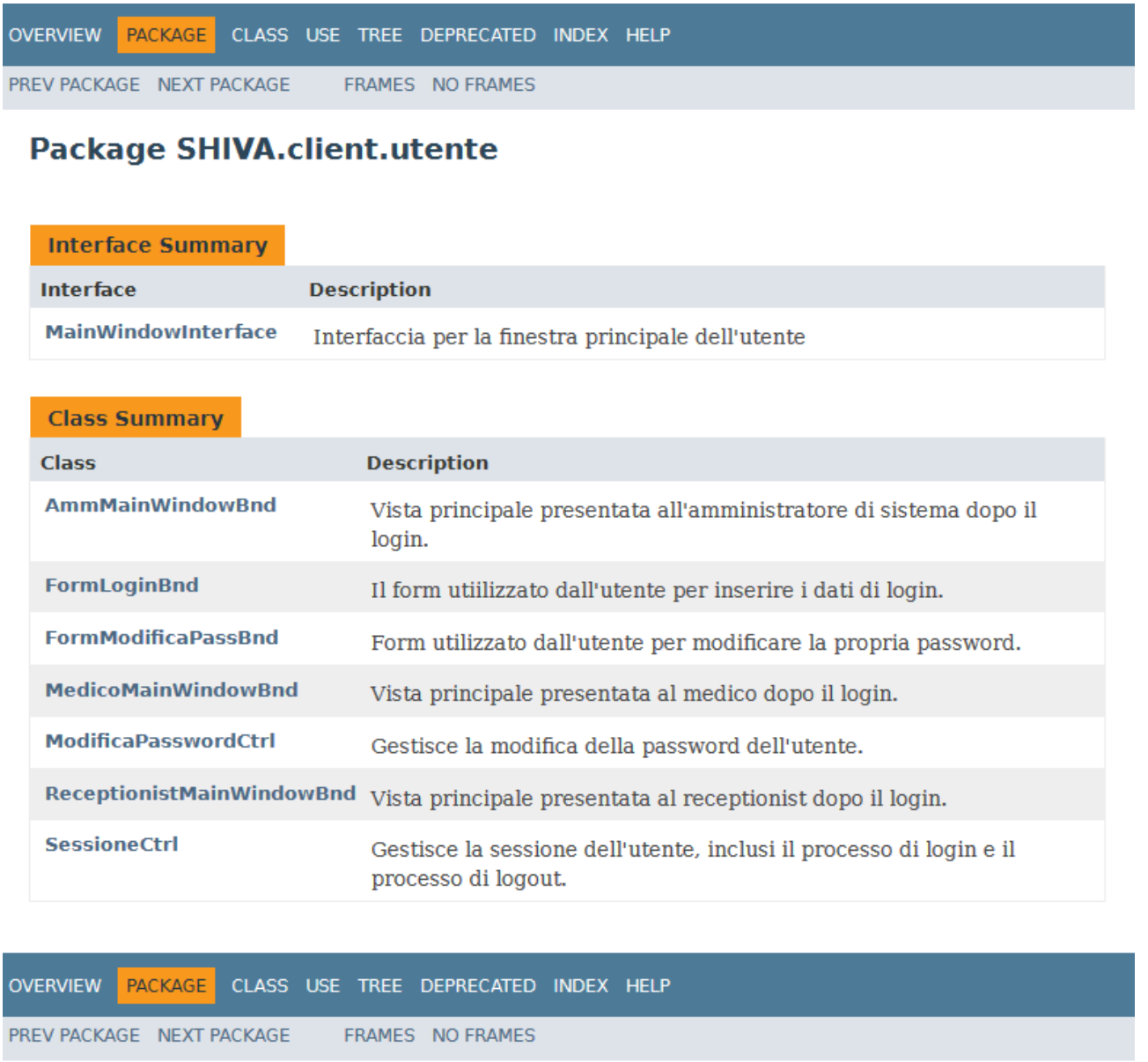
Package	Description
SHIVA.client.amministratore	
SHIVA.client.comunicazione	
SHIVA.client.medico	
SHIVA.client.messaggi	
SHIVA.client.messaggicomunicazione	
SHIVA.client.modello	
SHIVA.client.receptionist	
SHIVA.client.ricercapaziente	
SHIVA.client.ricercautente	
SHIVA.client.utente	
SHIVA.server.interfacciaserver	
SHIVA.server.messaggicomunicazione	
SHIVA.server.modello	
SHIVA.server.operazionicentralizzate	
SHIVA.server.persistenza	

OVERVIEW PACKAGE CLASS USE TREE DEPRECATED INDEX HELP

PREV NEXT FRAMES NO FRAMES

FIGURA 122 - PAGINA PRINCIPALE DEL JAVADOC

2.2 Package SHIVA.client.utente



OVERVIEW PACKAGE CLASS USE TREE DEPRECATED INDEX HELP

PREV PACKAGE NEXT PACKAGE FRAMES NO FRAMES

Package SHIVA.client.utente

Interface Summary

Interface	Description
MainWindowInterface	Interfaccia per la finestra principale dell'utente

Class Summary

Class	Description
AmmMainWindowBnd	Vista principale presentata all'amministratore di sistema dopo il login.
FormLoginBnd	Il form utiilizzato dall'utente per inserire i dati di login.
FormModificaPassBnd	Form utilizzato dall'utente per modificare la propria password.
MedicoMainWindowBnd	Vista principale presentata al medico dopo il login.
ModificaPasswordCtrl	Gestisce la modifica della password dell'utente.
ReceptionistMainWindowBnd	Vista principale presentata al receptionist dopo il login.
SessioneCtrl	Gestisce la sessione dell'utente, inclusi il processo di login e il processo di logout.

OVERVIEW PACKAGE CLASS USE TREE DEPRECATED INDEX HELP

PREV PACKAGE NEXT PACKAGE FRAMES NO FRAMES

FIGURA 123 - PAGINA DEL PACKAGE UTENTE

2.3 SHIVA.client.utente.AmmMainWindowBnd

OVERVIEW PACKAGE CLASS USE TREE DEPRECATED INDEX HELP

PREV CLASS NEXT CLASS FRAMES NO FRAMES

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

SHIVA.client.utente

Class AmmMainWindowBnd

java.lang.Object
SHIVA.client.utente.AmmMainWindowBnd

All Implemented Interfaces:

MainWindowInterface

```
public class AmmMainWindowBnd
extends java.lang.Object
implements MainWindowInterface
```

Vista principale presentata all'amministratore di sistema dopo il login. Gli permette di avviare le principali funzionalità.

Author:

Francesco Lo Monaco

Constructor Summary

Constructors

Constructor and Description
AmmMainWindowBnd()

Method Summary

All Methods | **Instance Methods** | **Concrete Methods**

Modifier and Type	Method and Description
void	eliminaUtente() permette di avviare l'operazione di elimina utente.
void	logout()
void	modificaPassword() permette di avviare l'operazione di modifica password.
void	modificaUtente() permette di avviare l'operazione di modifica utente.
void	registraMedico() permette di avviare l'operazione di registra medico.
void	registraReceptionist() permette di avviare l'operazione di registra receptionist.

Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

AmmMainWindowBnd

```
public AmmMainWindowBnd()
```

Method Detail

logout

```
public void logout()
```

Specified by:
`logout` in interface `MainWindowInterface`

modificaPassword

```
public void modificaPassword()
```

permette di avviare l'operazione di modifica password.

Specified by:
`modificaPassword` in interface `MainWindowInterface`

registraMedico

```
public void registraMedico()
```

permette di avviare l'operazione di registra medico.

registraReceptionist

```
public void registraReceptionist()
```

permette di avviare l'operazione di registra receptionist.

modificaUtente

```
public void modificaUtente()
```

permette di avviare l'operazione di modifica utente.

eliminaUtente

```
public void eliminaUtente()
```

permette di avviare l'operazione di elimina utente.

FIGURA 124 - JAVADOC DELLA CLASSE AMMMAINWINDOWBND

2.4 SHIVA.client.utente.MedicoMainWindowBnd

OVERVIEW PACKAGE CLASS USE TREE DEPRECATED INDEX HELP

PREV CLASS NEXT CLASS FRAMES NO FRAMES

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

SHIVA.client.utente

Class MedicoMainWindowBnd

java.lang.Object
SHIVA.client.utente.MedicoMainWindowBnd

All Implemented Interfaces:

MainWindowInterface

```
public class MedicoMainWindowBnd
extends java.lang.Object
implements MainWindowInterface
```

Vista principale presentata al medico dopo il login. Gli permette di avviare le principali funzionalità.

Author:

Salvatore Perna

Constructor Summary

Constructors

Constructor and Description
MedicoMainWindowBnd()

Method Summary

All Methods | **Instance Methods** | **Concrete Methods**

Modifier and Type	Method and Description
void	inserisciReport() permette di avviare l'operazione di inserisci report.
void	logout() permette di avviare l'operazione di logout
void	modificaPassword() permette di avviare l'operazione di modifica password
void	visualizzaReport() permette di avviare l'operazione di visualizza report
void	visualizzaVisite() permette di avviare l'operazione di visualizza visite

Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

MedicoMainWindowBnd

```
public MedicoMainWindowBnd()
```

Method Detail

logout

```
public void logout()
```

permette di avviare l'operazione di logout

Specified by:

`logout` in interface `MainWindowInterface`

modificaPassword

```
public void modificaPassword()
```

permette di avviare l'operazione di modifica password

Specified by:

`modificaPassword` in interface `MainWindowInterface`

visualizzaVisite

```
public void visualizzaVisite()
```

permette di avviare l'operazione di visualizza visite

visualizzaReport

```
public void visualizzaReport()
```

permette di avviare l'operazione di visualizza report

inserisciReport

```
public void inserisciReport()
```

permette di avviare l'operazione di inserisci report.

FIGURA 125 - JAVADOC DELLA CLASSE RECEPTIONISTMAINWINDOWBND

3 Contratti fra le classi

Qui elencati i contratti fra alcune classi scritti in OCL

3.1 SessioneCtrl

context SessioneCtrl::setUtenteLoggato(utente:Utente) **pre:**

utente<>null

context SessioneCtrl::logout() **post:**

utenteLoggato=null

3.2 ComunicazioneCtrl

context ComunicazioneCtrl::iniziaSessione() **post:**

idSessione<>0

context ComunicazioneCtrl::mandaMessaggio(richiesta:Messaggio) **pre:**

richiesta<>null

context ComunicazioneCtrl::chiudiSessione() **post:**

idSessione=0

3.3 RicercaUtenteCtrl

context RicercaUtenteCtrl::setListaUtenti(listaUtenti:List) **pre:**

listaUtenti<>null

context RicercaUtenteCtrl::ricercaCF(codiceF:String) **pre:**

codiceF<>null and codiceF.size()=16

context RicercaUtenteCtrl::ricercaNome(nome:String) **pre:**

nome<>null and nome.size()>0

context RicercaUtenteCtrl::ricercaCognome(cognome:String) **pre:**

cognome<>null and cognome.size()>0

context RicercaUtenteCtrl::riceviSelezioneMedico(medico:Medico) **pre:**

medico<>null

context RicercaUtenteCtrl::ricercaCfTipo(codiceF:String, tipoUtente:String) **pre:**

codiceF<>null and codiceF.size()=16 and (tipoUtente='medico' or tipoUtente='receptionist')

context RicercaUtenteCtrl::ricercaNomeTipo(nome:String, tipoUtente:String) **pre:**

nome<>null and nome.size()>0 and (tipoUtente='medico' or tipoUtente='receptionist')

context RicercaUtenteCtrl::ricercaCognomeTipo(cognome:String, tipoUtente:String) **pre:**

cognome<>null and cognome.size()>0 and (tipoUtente='medico' or tipoUtente='receptionist')

context RicercaUtenteCtrl::riceviSelezioneUtente(utente:Utente) **pre:**

utente<>null

3.4 ModificaPasswordCtrl

context ModificaPasswordCtrl::modificaPass(vecchiaPass:String,nuovaPass:String) **pre:**

vecchiaPass<>null and nuovaPass<>null and vecchiaPass<>nuovaPass and controllaFormatoPass(nuovaPass)=true

context ModificaPasswordCtrl::modificaPass(vecchiaPass:String,nuovaPass:String) **post:**

if (vecchiaPass<>null and nuovaPass<>null and vecchiaPass<>nuovaPass and controllaFormatoPass(nuovaPass)=true) **then**

utente.password = nuovaPass

3.5 InterfacciaCtrl

context InterfacciaCtrl::inizializzazione() **post:**

sessioniAttive.size() = 0

context InterfacciaCtrl::interpretaMessaggio(messaggio:Messaggio) **pre:**

messaggio<>null

3.6 UtenteCtrl

context UtenteCtrl::login(username:String, password:String) **pre:**

username<>null and username.size()>0 and password<>null and password.size()>0

context UtenteCtrl::modificaPassword(nuovaPassword:String) **pre:**

nuovaPassword<>null and nuovaPasswod.size()>0

4 Esito Compilazione

Si genera un file (sources.txt) contenente i path di tutti i sorgenti Java presenti all'interno della directory, al fine di poter fornire al compilatore Java la lista dei files da compilare.

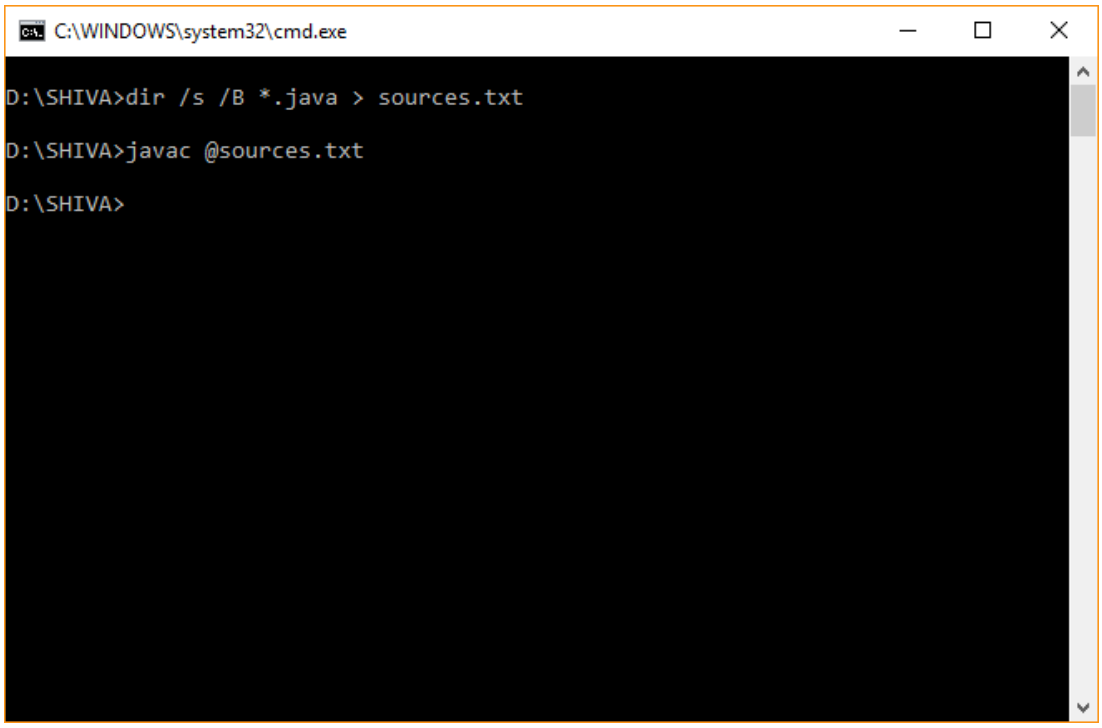


FIGURA 126 - SCREENSHOT DEL RISULTATO DI COMPILAZIONE