

Università degli Studi di Verona
Dipartimento di Informatica

RELAZIONE FINALE

PROGETTO DI INGEGNERIA DEL SOFTWARE

Prof. del Corso: Combi Carlo

Studenti: Vantini Edoardo VR457710
Darie Alexandru VR500116
Ariton Maria Gemma VR500028

INDICE

INTRODUZIONE	3
PROCESSO DI SVILUPPO DEL SOFTWARE	4
CAPITOLO PRIMO: Attori del sistema	6
CAPITOLO SECONDO: Casi d'uso e i relativi Diagrammi Di Sequenza	7
CAPITOLO TERZO: Class Diagram	18
CAPITOLO QUARTO: Activity Diagram	19
CAPITOLO QUINTO: Interfaccia grafica	22
CAPITOLO SESTO: Test del software	26

INTRODUZIONE

Si vuole progettare un sistema di telemedicina di un servizio clinico per la gestione di pazienti diabetici (diabete di tipo 2).

Il sistema deve permettere l'interazione di due attori principali, il diabetologo e il paziente.

Il paziente, dopo essersi autenticato, potrà memorizzare le rilevazioni giornaliere di glicemia (prima e dopo ogni pasto). I livelli normali prima dei pasti dovrebbero essere compresi tra 80 e 130 mg/dL, mentre due ore dopo i pasti non dovrebbero superare i 180 mg/dL.

Il paziente può, inoltre, aggiungere eventuali sintomi (spossatezza, nausea, mal di testa, e così via), e le assunzioni di insulina e/o di farmaci antidiabetici orali, come da prescrizione dello specialista (giorno, ora, farmaco e quantità assunta). Eventuali sintomi, patologie e/o terapie concomitanti vanno opportunamente segnalati da parte del paziente, con l'indicazione del sintomo, della terapia e/o della patologia e del periodo associato.

Il diabetologo, dopo essersi autenticato, deve poter specificare le terapie che i pazienti devono seguire. Per ogni terapia il medico specifica il farmaco, il numero di assunzioni giornaliere, la quantità di farmaco per ogni assunzione, ed eventuali indicazioni (ad es., dopo i pasti, lontano dai pasti, e così via). Il medico potrà vedere i dati dei pazienti, anche in forma sintetica (ad es., andamento della glicemia settimana per settimana o mese per mese). Il medico potrà aggiungere o modificare la terapia a seconda dell'evoluzione dello stato del paziente. Il medico può, inoltre, aggiornare una breve sezione di informazioni sul paziente, contenente fattori di rischio (fumatore, ex-fumatore, problemi di dipendenza da alcol, o da stupefacenti, obesità), pregresse patologie, comorbidità presenti quali ipertensione, e così via.

Il sistema deve verificare che le assunzioni di farmaci da parte dei pazienti siano coerenti con le terapie prescritte. Il sistema deve invitare il paziente a completare gli inserimenti relativi alle assunzioni di farmaci, in modo da poter gestire sia alert verso il paziente, nel caso si dimenticasse di assumere i farmaci, sia verso il medico, nel caso il paziente non segua per più di 3 giorni consecutivi le prescrizioni. Il sistema, inoltre, segnala ai medici tutti i pazienti che registrano glicemie oltre le soglie indicate con diverse modalità a seconda della gravità.

I responsabili del servizio inseriscono i dati iniziali di pazienti e medici, necessari per l'autenticazione.

Per ogni paziente è specificato un medico di riferimento, al quale il paziente può inviare e-mail per richieste e domande varie. Ogni medico può vedere e aggiornare i dati di ogni paziente. Il sistema provvederà a tenere traccia di quale medico ha effettuato le varie operazioni.

PROCESSO DI SVILUPPO DEL SOFTWARE

Il processo di sviluppo è stato condotto in modo agile e incrementale: inizialmente, sono state definite le tabelle che compongono il database, e successivamente sono state progettate le classi principali associate a ciascun attore del sistema (paziente, medico e segreteria).

Il lavoro si è concentrato su un attore alla volta, definendo le relative classi e implementando le funzioni necessarie. Ogni funzione, una volta completata, veniva testata per assicurarne il corretto funzionamento. Una volta ultimato lo sviluppo di una classe, venivano eseguite ulteriori verifiche prima di procedere con la classe successiva.

Quando si è presentata la necessità di sviluppare una funzione o una classe per un attore con funzionalità simili a quelle già esistenti, è stata seguita la stessa linea di progettazione per mantenere la coerenza nello svolgimento delle nuove funzioni richieste. Ad esempio, la gestione del login è stata realizzata tramite una superclasse comune a tutti e tre gli attori.

Per lo sviluppo software è stata adottata una tecnica di pair-programming, ovvero uno sviluppo agile in cui due programmatori lavorano insieme, collaborando in tempo reale sullo stesso codice. Uno scrive mentre l'altro osserva, rivede il codice e suggerisce miglioramenti: i ruoli si alternano frequentemente, così entrambi mantengono una visione completa del codice e delle logiche implementate.

Alcuni vantaggi sono:

- **Meno bug in fase di sviluppo**, perché il codice viene controllato in tempo reale da entrambi.
- **Decisioni di design/sviluppo migliori**: essendoci un confronto costante, si tende a scrivere codice più chiaro.

Per la realizzazione del sistema è stato scelto il pattern di architettura MVC. Nel dettaglio:

- **Modello**: comprende le classi e le funzioni dedicate alla gestione dei dati e della logica, oltre che all'interazione con il database tramite chiamate alle classi DAO (Data Access Object).
- **Vista**: riguarda l'interfaccia utente, creata con SceneBuilder. Le classi che la gestiscono si trovano nei relativi controller.
- **Controller**: gestisce l'input degli utenti e fa da tramite tra la Vista e il Modello. Ogni schermata ha un controller dedicato.

Si è scelto di sviluppare il sistema seguendo questo pattern di architettura per la sua solidità e flessibilità. La suddivisione tra Modello, Vista e Controllore si è rivelata essenziale per lo sviluppo e l'implementazione delle classi e delle funzioni. Questa struttura ha permesso di modificare le componenti relative a un modulo senza influenzare le altre due, migliorando così la manutenibilità e la possibilità di ottimizzare il codice.

Inoltre, ha facilitato la verifica delle funzionalità del sistema, in particolare quelle relative all'input e alla gestione dei dati, e ha semplificato l'aggiunta di nuove funzioni con compiti simili a quelli già presenti, consentendo un'integrazione più agevole all'interno delle classi esistenti.

Dato l'uso dell'architettura MVC, si è rivelato utile l'utilizzo del pattern Data Transfer Object (DTO): è stato usato principalmente per trasferire le informazioni tra il Modello e la Vista, in particolare per le classi e le funzioni che interagiscono con il database per il recupero e l'aggiornamento di informazioni relative ai pazienti e ai medici.

Insieme al DTO è stato adottato anche il pattern DAO (Data Access Object), che ha permesso di astrarre e incapsulare l'accesso ai dati, permettendo l'iterazione col database. Ogni entità principale (come paziente o medico) è infatti associata a una classe DAO dedicata, responsabile della comunicazione con il database. Così facendo si è mantenuta la logica applicativa separata dalla logica di persistenza, migliorando la manutenibilità e la testabilità del codice.

Lo sviluppo del sistema è stato reso possibile grazie a XAMPP, un pacchetto software che include componenti come il server web Apache e il servizio di database MySQL.

Dopo aver avviato XAMPP e questi due componenti, è sufficiente aprire un browser e navigare all'indirizzo <http://localhost/> per accedere alla dashboard di XAMPP.

Da qui è stato possibile accedere a phpMyAdmin, l'applicazione web utilizzata per creare il database necessario allo sviluppo del sistema.

Design Pattern

Sono stati utilizzati dei design pattern per strutturare meglio il codice e facilitarne la manutenzione: Singleton, Factory e Observer.

- Il pattern **Singleton** è stato applicato alla classe `UserSession`, che gestisce le informazioni dell'utente attualmente loggato.
Questo garantisce che ci sia una sola istanza condivisa nel sistema, accessibile da qualsiasi punto dell'applicazione. In questo modo, è possibile recuperare dati senza doverli passare tra controller o altri componenti.
Anche `ServiceFactory` è stata resa Singleton, così da evitare di creare più istanze dei servizi e aprire più connessioni sparse per il codice.
- Il pattern **Factory** viene utilizzato nella classe `NotificaFactory`: invece di scrivere ogni volta la logica per inizializzare una notifica con i suoi parametri, basta chiamare un metodo statico della factory.
Questo approccio evita la duplicazione di codice e rende più facile aggiornare la struttura delle notifiche in un solo punto.
- Il pattern **Observer** si trova per esempio nel controller `PrescrizioniController`.
la proprietà assunta di una prescrizione è osservata da un listener, ovvero quando l'utente seleziona o deseleziona la checkbox associata, il listener aggiorna automaticamente il database chiamando il servizio corrispondente.
Lo stesso vale per i controller che gestiscono gli slider dei valori glicemici: al variare del valore, le etichette si aggiornano in tempo reale.

CAPITOLO PRIMO

Gli attori del sistema per la gestione della clinica per persone diabetiche sono tre: Medico, Paziente e Segreteria.

Essi accedono alle funzioni tramite un login (CF e password).

- I *Medici* possono visualizzare i pazienti presi in cura e le relative notifiche. Cliccando sul paziente riuscirà a vedere tutte le relative informazioni, il record delle misurazioni dei valori chiave e potrà inserire prescrizioni di farmaci specifici.
- I *Pazienti* possono registrare le misurazioni dei valori chiave, inviare notifiche al medico curante in caso i valori siano sballati e visualizzare il record di tutte le misurazioni passate e assunzioni di farmaci specifici e non.
- La *Segreteria* può inserire ed eliminare pazienti e medici ed eventuali modifiche dei medici curanti.

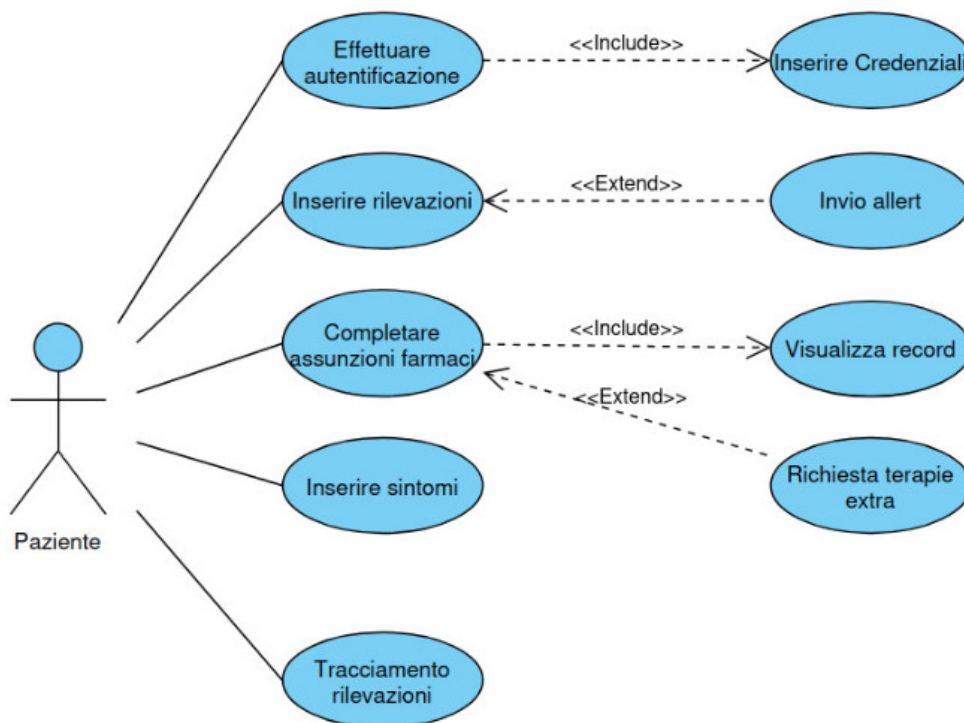
CAPITOLO SECONDO

Casi d'uso con i relativi diagrammi di sequenza

2.1 Gestione Pazienti:

1. Il sistema deve permettere ai pazienti di potersi registrare ed inserire le rilevazioni di glicemia. In caso di valori fuori norma, il sistema invia automaticamente un avviso al medico curante.
2. I pazienti, oltre a poter registrare le assunzioni di insulina/farmaci, possono decidere se avviare richieste di terapie extra che dovranno essere approvate dai medici.
3. Il sistema deve invitare il paziente a completare gli inserimenti relativi alle assunzioni di farmaci, in modo da poter gestire sia alert verso il paziente, nel caso si dimenticasse di assumere i farmaci, sia verso il medico, nel caso il paziente non segua per più di tre giorni consecutivi le prescrizioni.
4. Tiene traccia di ogni notifica e rilevazione fornendo un record completo giorno per giorno.

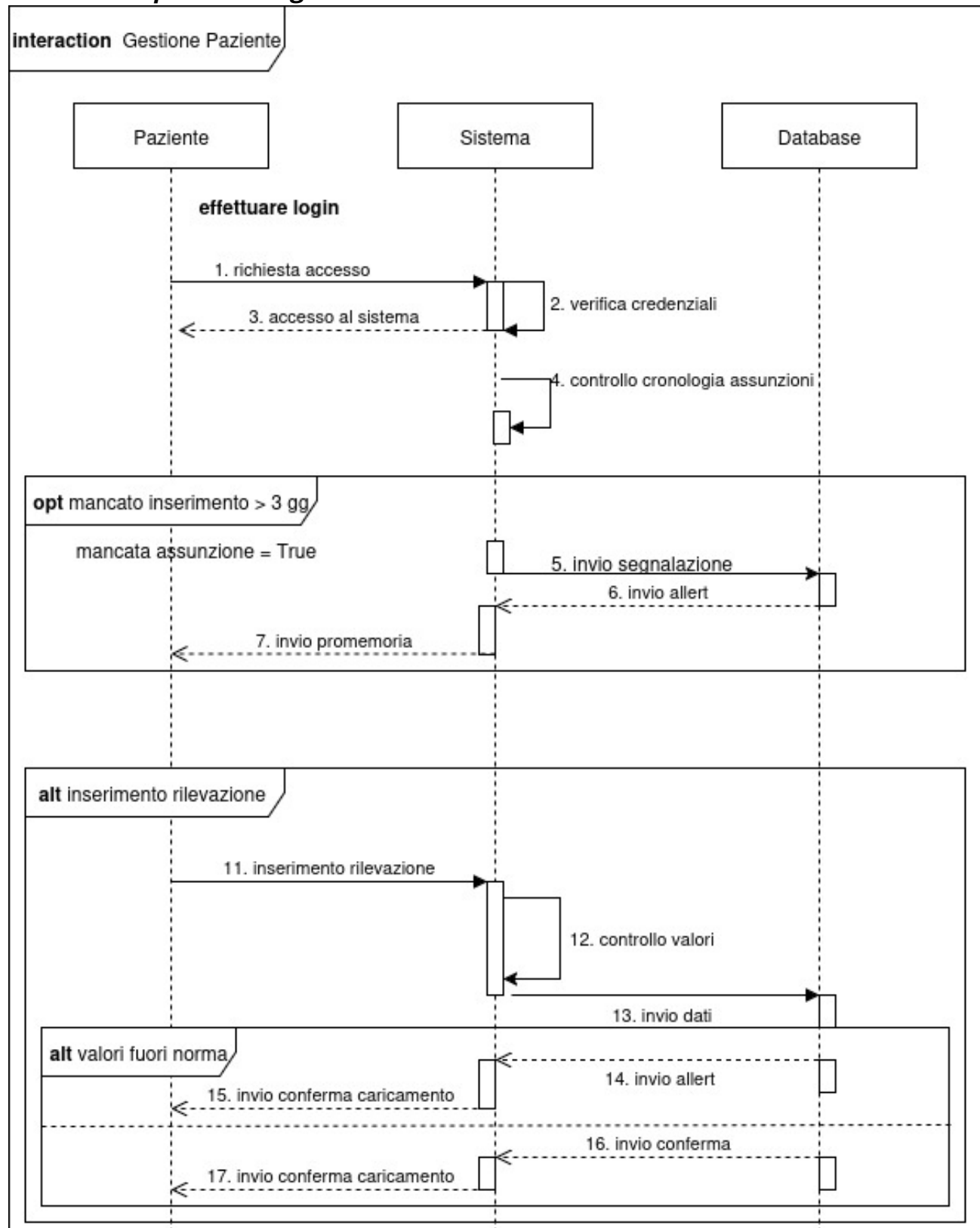
2.1.1 Use Case

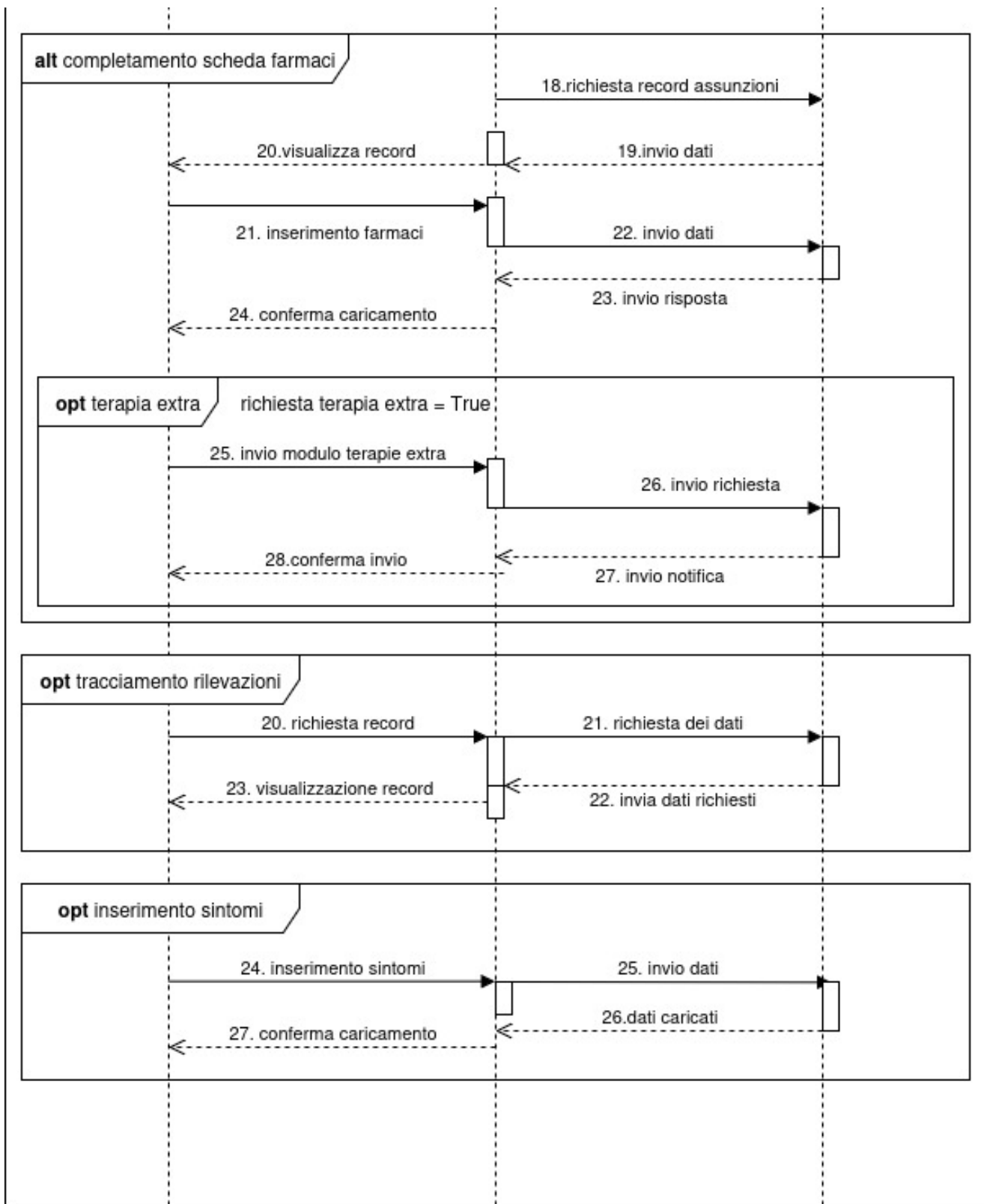


2.1.2 Scheda UC

UC: PAZIENTE
id: UC1
Attore: Paziente
Precondizione: utente è un paziente
Sequenza: 1. Effettuare login 2. Inserire rilevazioni 3. Completare scheda assunzioni farmaci 4. Richiedere terapie extra 5. Inserire sintomi 6. Visualizzare tracciamento rilevazioni
Postcondizioni: paziente aggiornato

2.1.3 Sequence Diagram

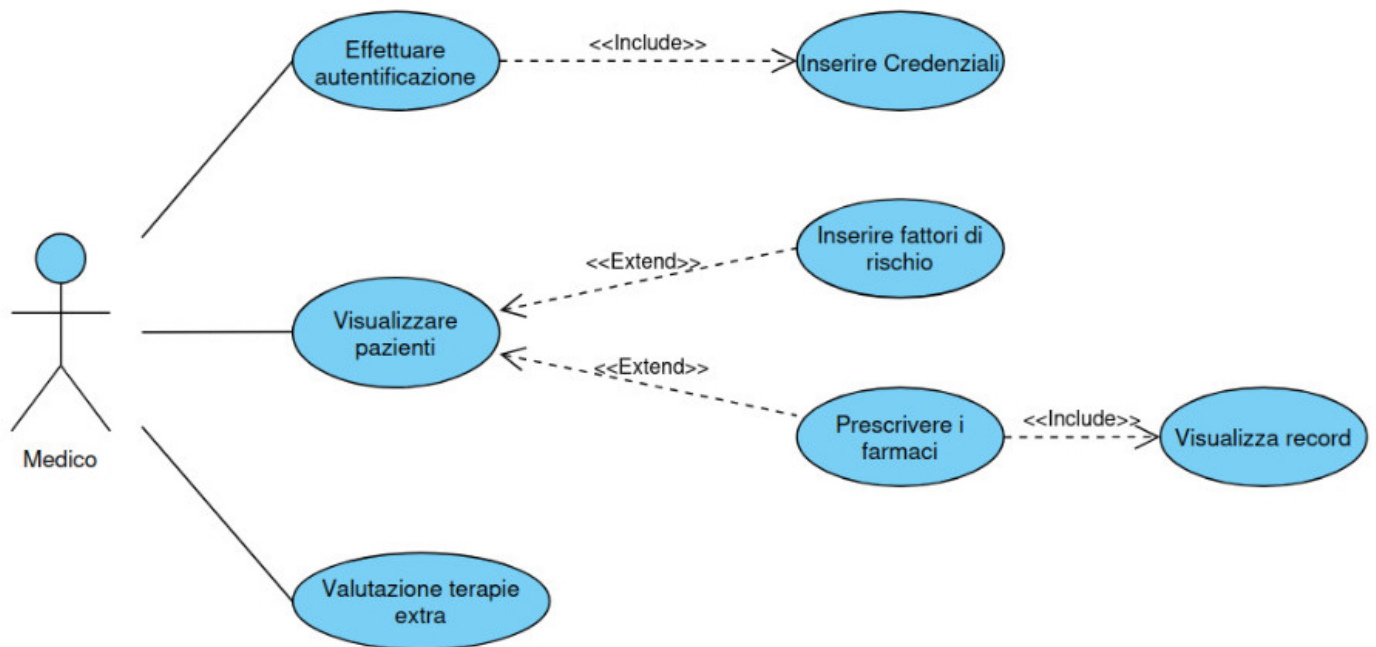




2.2 Gestione Medici

1. I medici possono visualizzare le notifiche dei propri pazienti e i dati di tutti i pazienti registrati. Possono inviare specifiche prescrizioni di farmaci con dosaggi, relativi periodi di assunzione ed eventuali indicazioni.
2. I medici possono inoltre compilare una scheda specifica per ogni paziente dove può inserire fattori di rischio (fumatore, ex-fumatore, problemi di dipendenza da alcol, o da stupefacenti, obesità), pregresse patologie e comorbidità presenti.

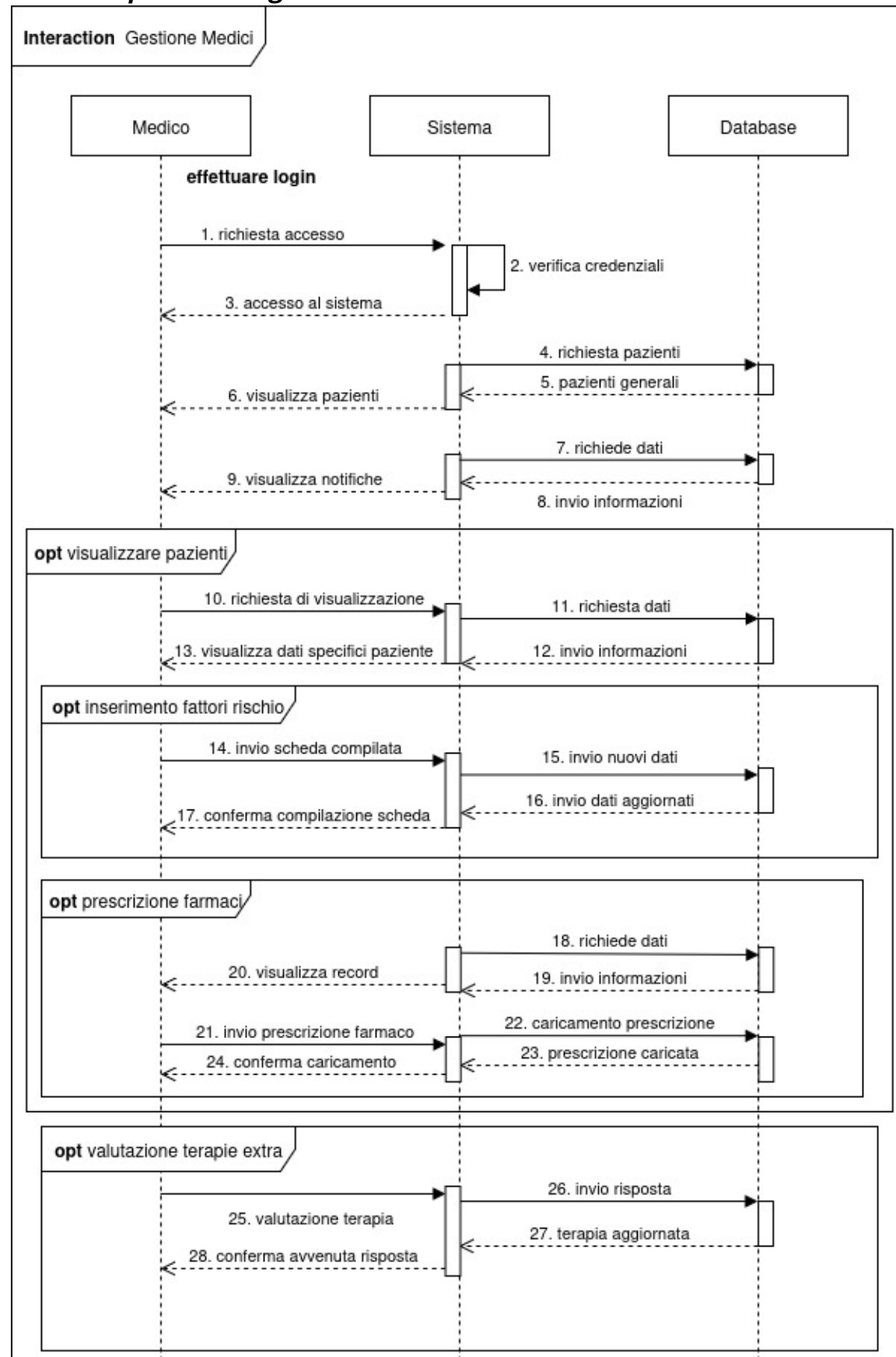
2.2.1 Use Case



2.2.2 Scheda UC

UC. MEDICO
id: UC2
Attore: Medico
Precondizione: utente è un medico
Sequenza: 1. Effettuare login 2. Visualizzare pazienti 3. Inserire fattori di rischio 4. Prescrivere i farmaci 5. Valutare eventuali terapie extra
Postcondizione: Scheda del paziente aggiornata con relative assunzioni e terapie extra

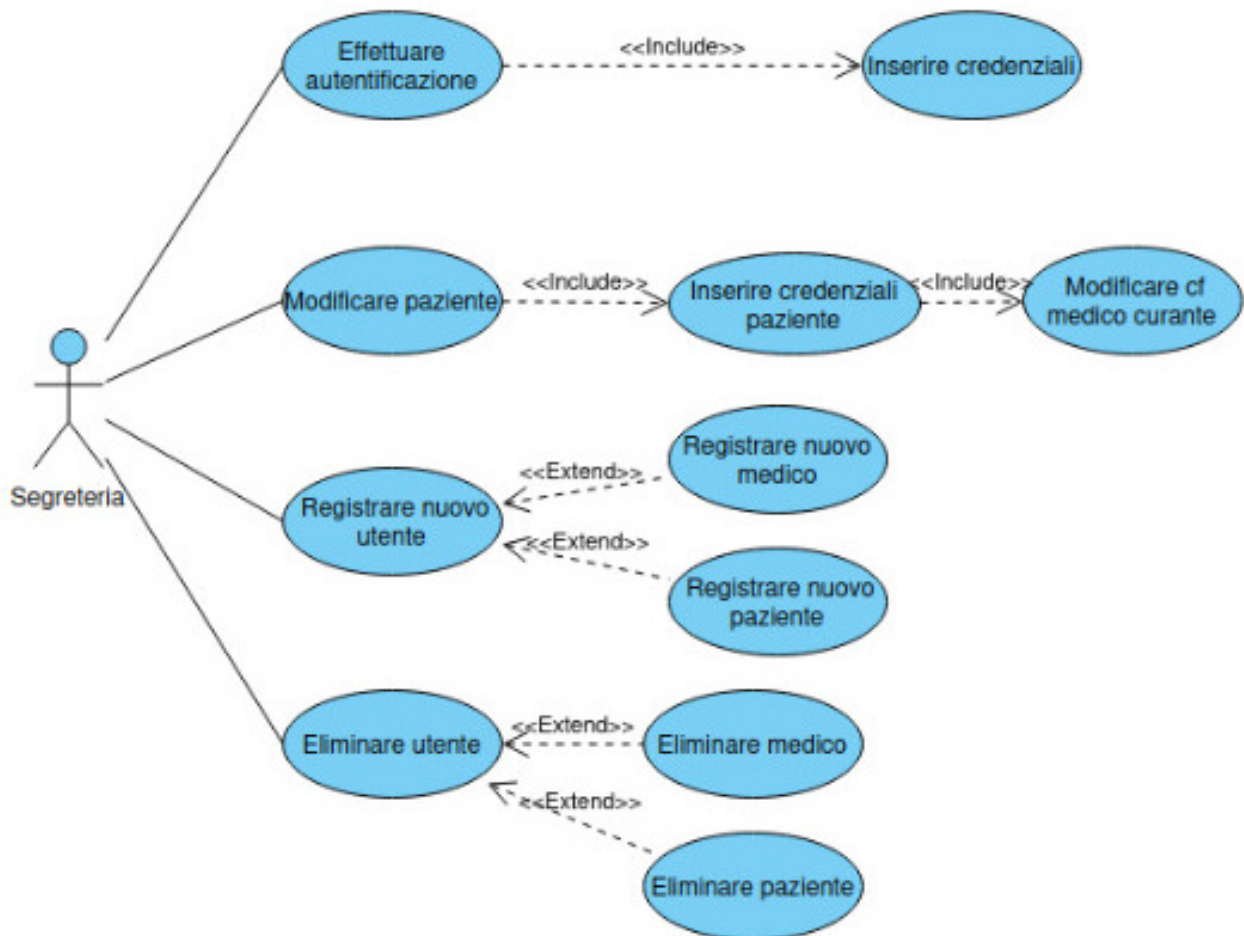
2.2.3 Sequence Diagram



2.3 Gestione Segreteria:

1. La segreteria gestisce le eventuali registrazioni di nuovi pazienti o medici, oltre a poter modificare il medico di riferimento di un paziente. Inoltre, può decidere se eliminare eventuali dottori che non lavorano più nella clinica o pazienti dimessi permanentemente.

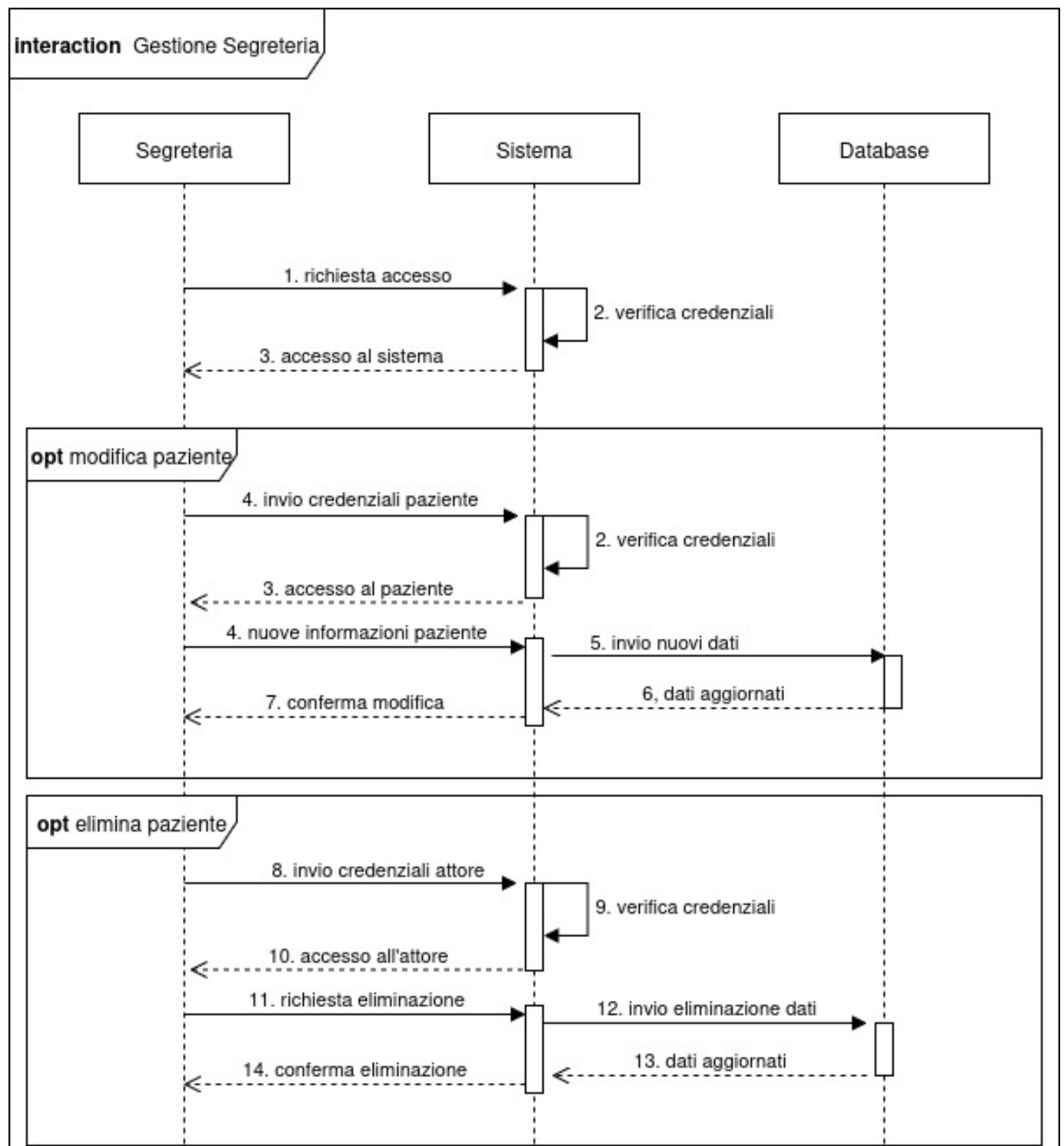
2.3.1 Use Case:

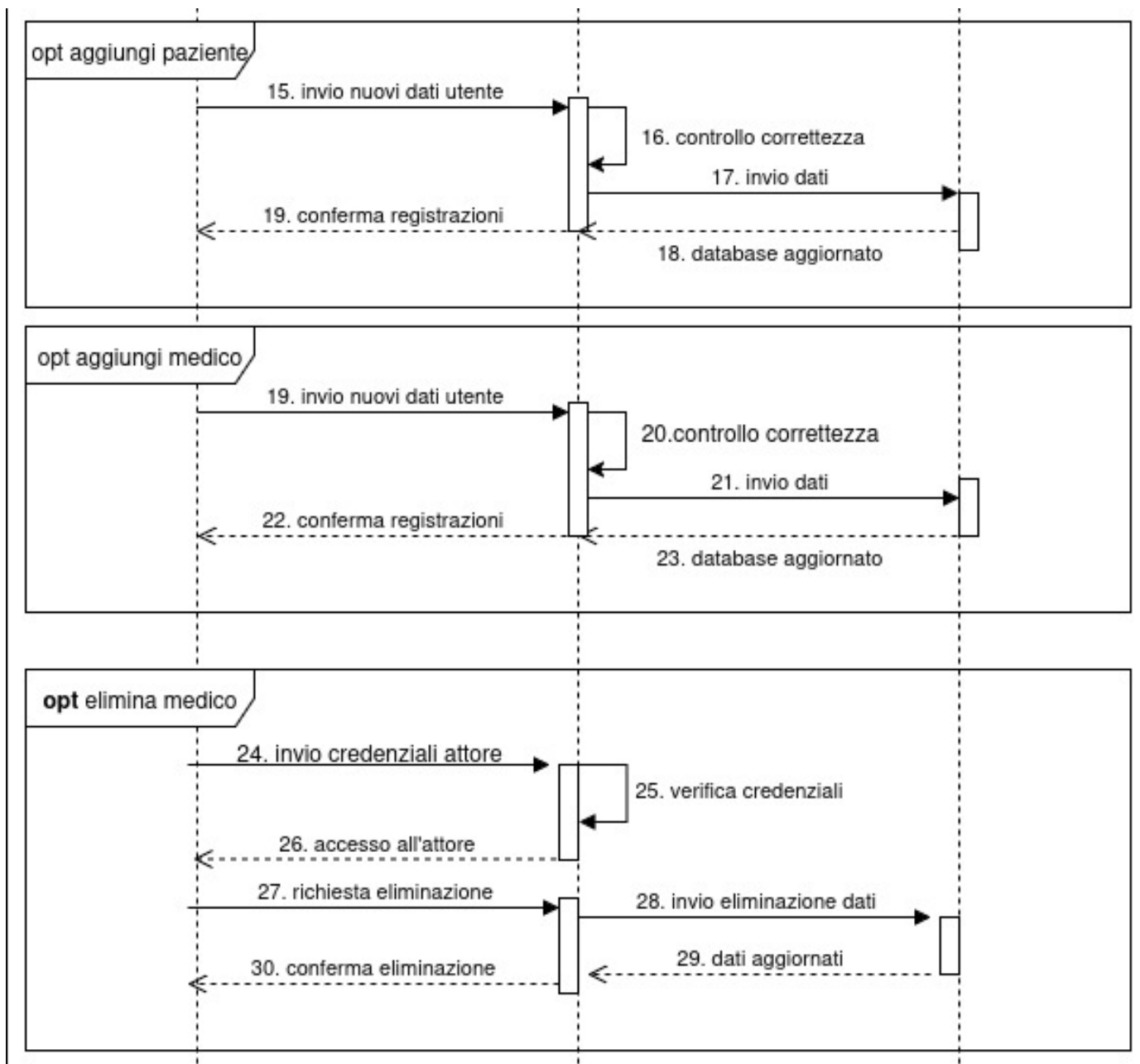


2.3.2 Scheda UC

UC: SEGRETERIA
id: UC3
Attore: Segreteria
precondizione: utente fa parte della segreteria
Sequenza: 1. Effettuare login 2. Aggiungere Paziente 3. Aggiungere Medico 4. Eliminare Paziente 5. Eliminare Utente 6. Modificare Paziente
Postcondizioni: sistema aggiornando

2.3.3 Sequence Diagram

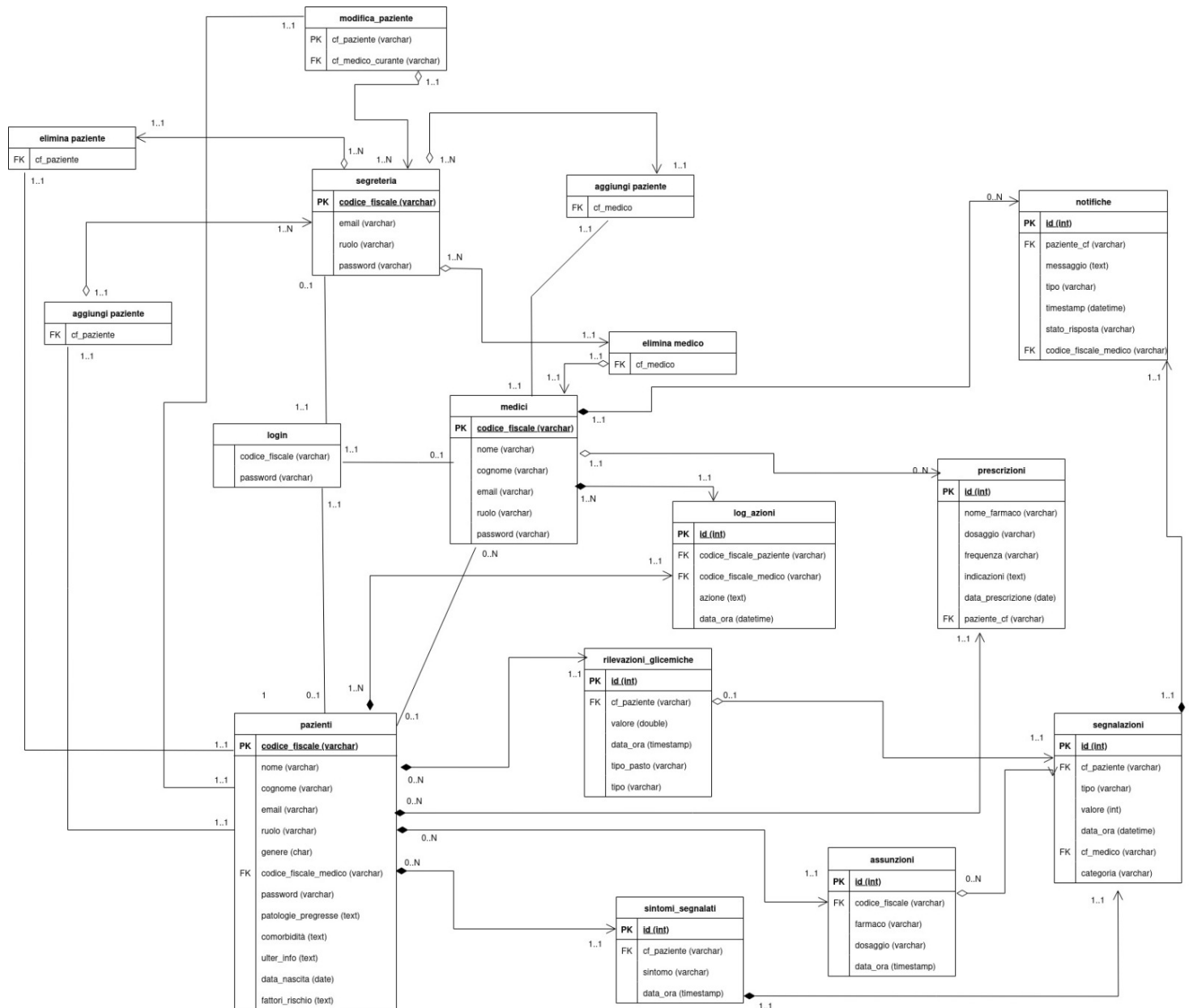




CAPITOLO TERZO

Class Diagram

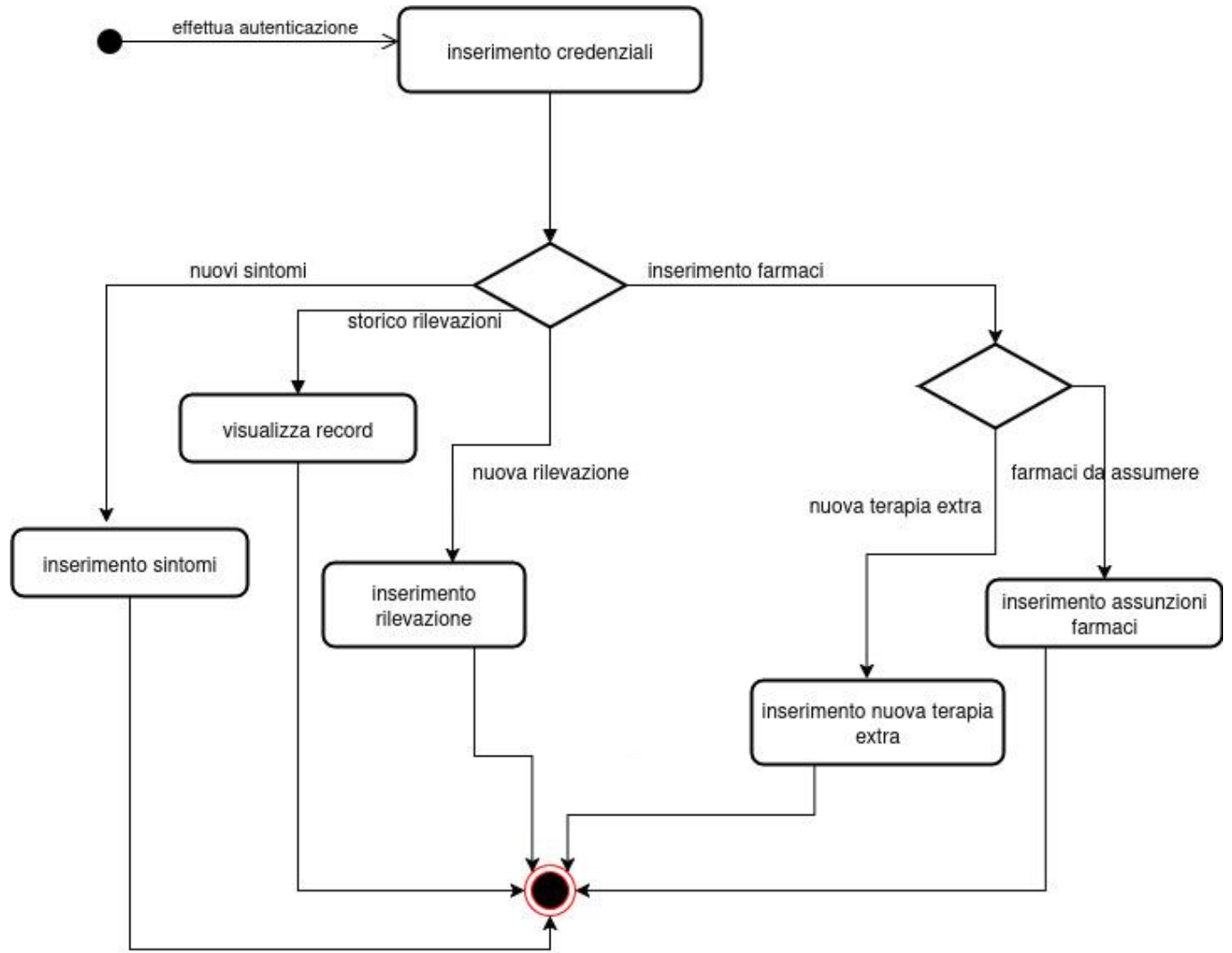
Abbiamo progettato le entità principali **Pazienti**, **Medici** e **Segreteria** ognuna con le relative informazioni e relazioni. A esse collegate ci sono tabelle per la gestione delle **notifiche**, **prescrizioni**, **assunzioni**, **rilevazioni glicemiche** e dei **sintomi segnalati**, permettendo una gestione completa del paziente. Inoltre viene fatto un log di tutte le azioni svolte dal medico nella specifica tabella **log_azioni**.



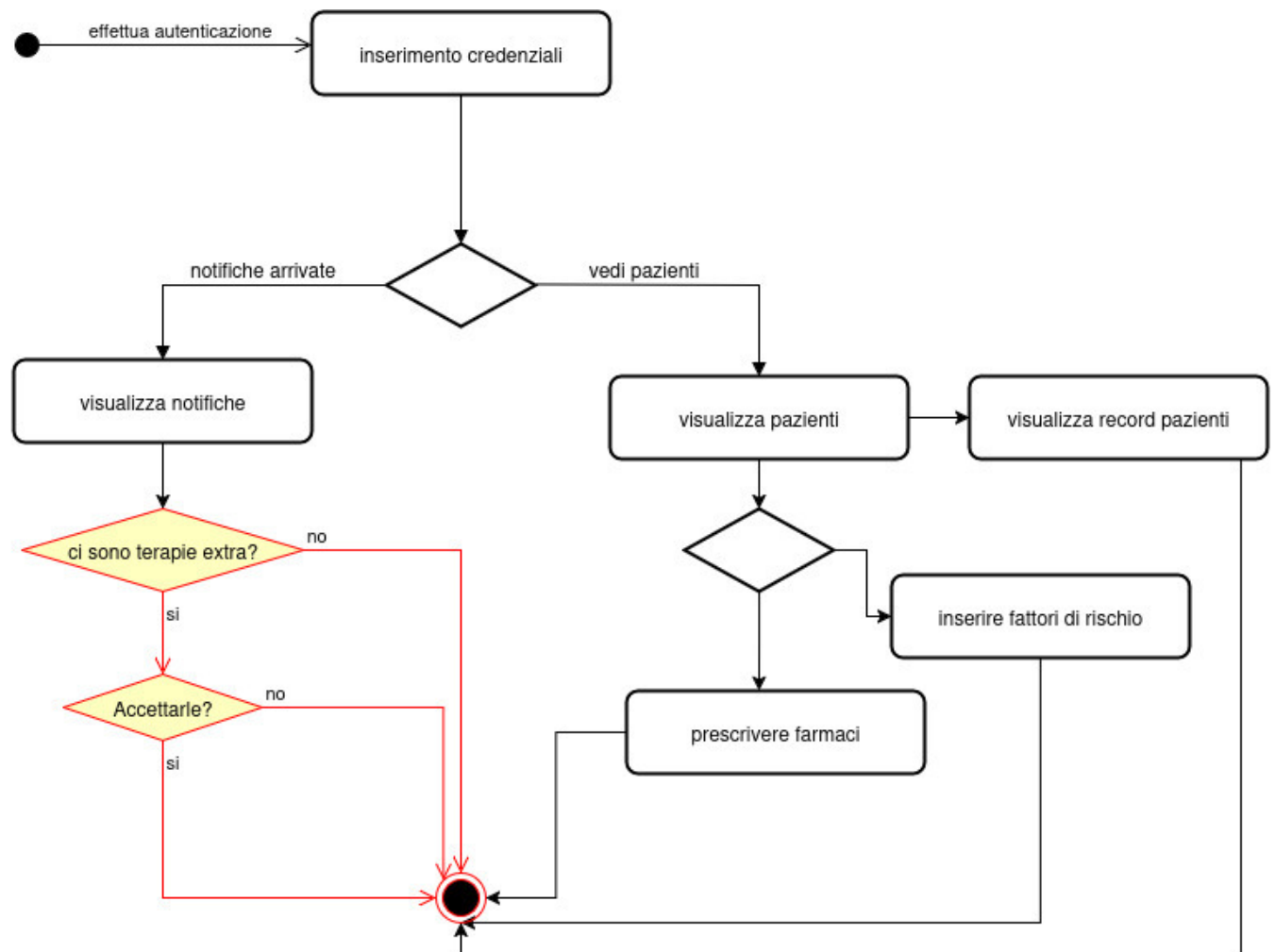
CAPITOLO QUARTO

Activity Diagram

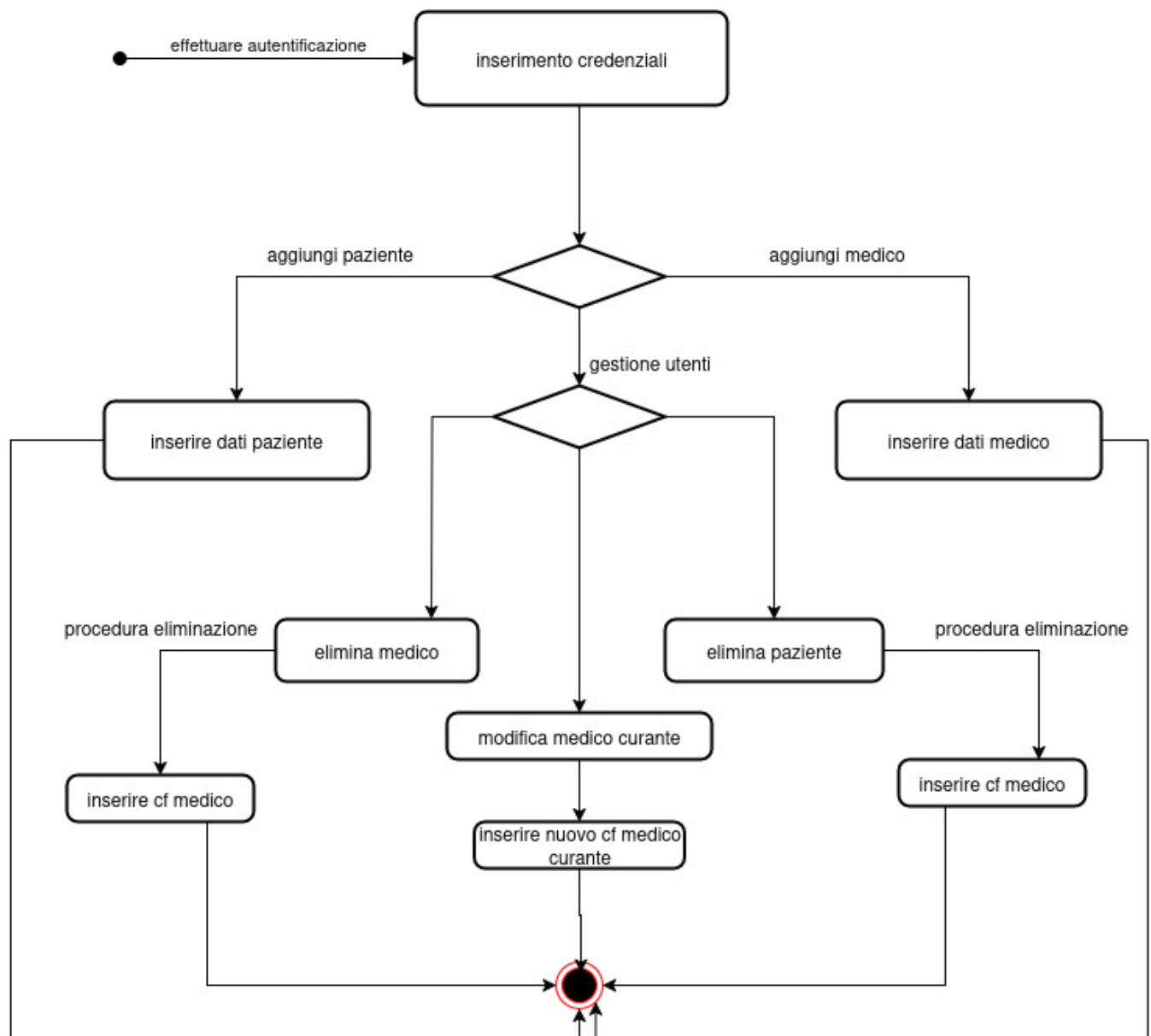
4.1 Activity Paziente



4.2 Activity Medico



4.3 Activity Segreteria




CAPITOLO QUINTO

Interfaccia Grafica

Seguono le catture della pagina di login e delle schermate principali dei tre attori: *Paziente*, *Medico* e *Segreteria*.

6.1 Login Page

Diabetici

 MediGlyk

Inserire le credenziali

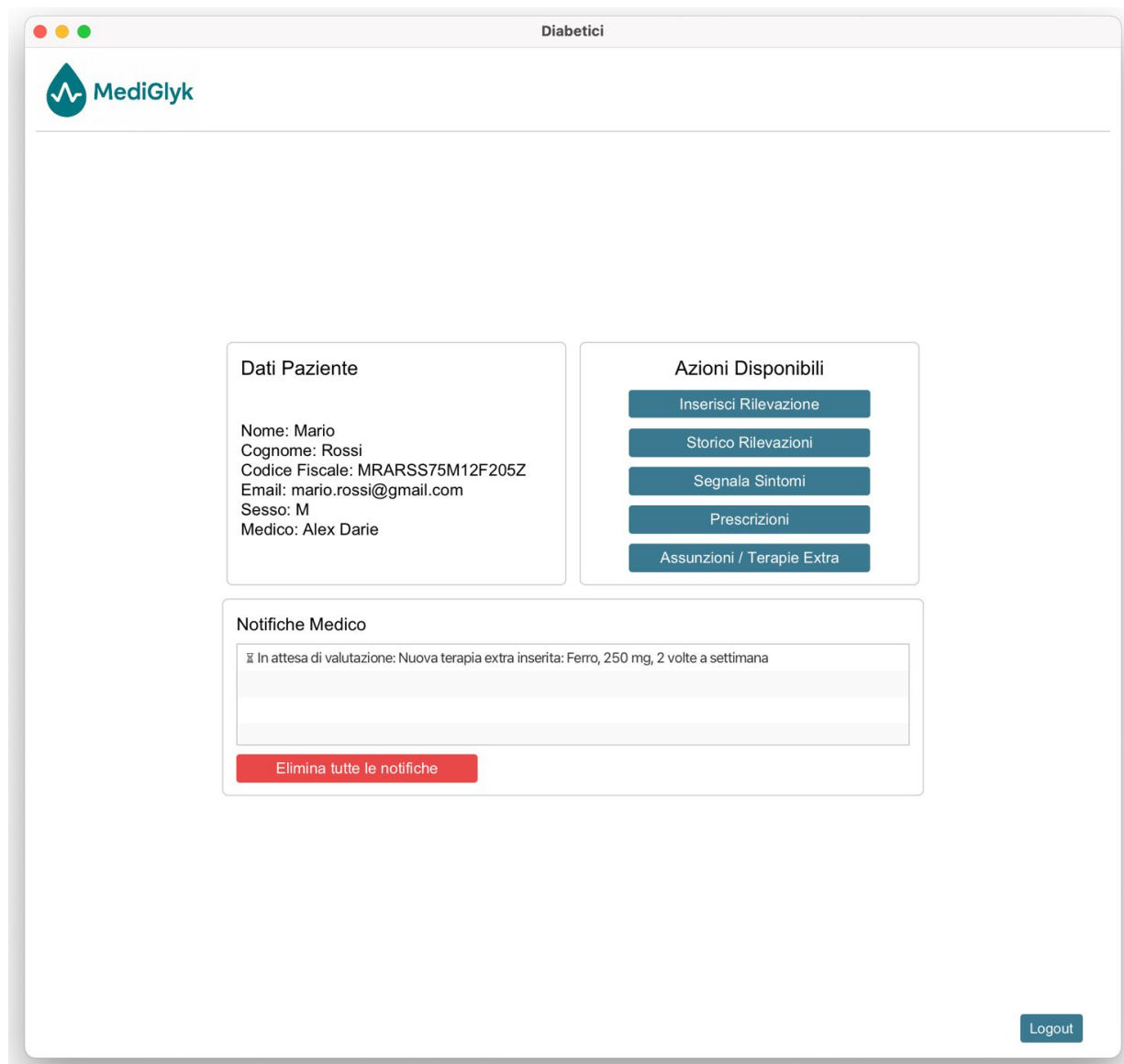
Codice Fiscale

Password

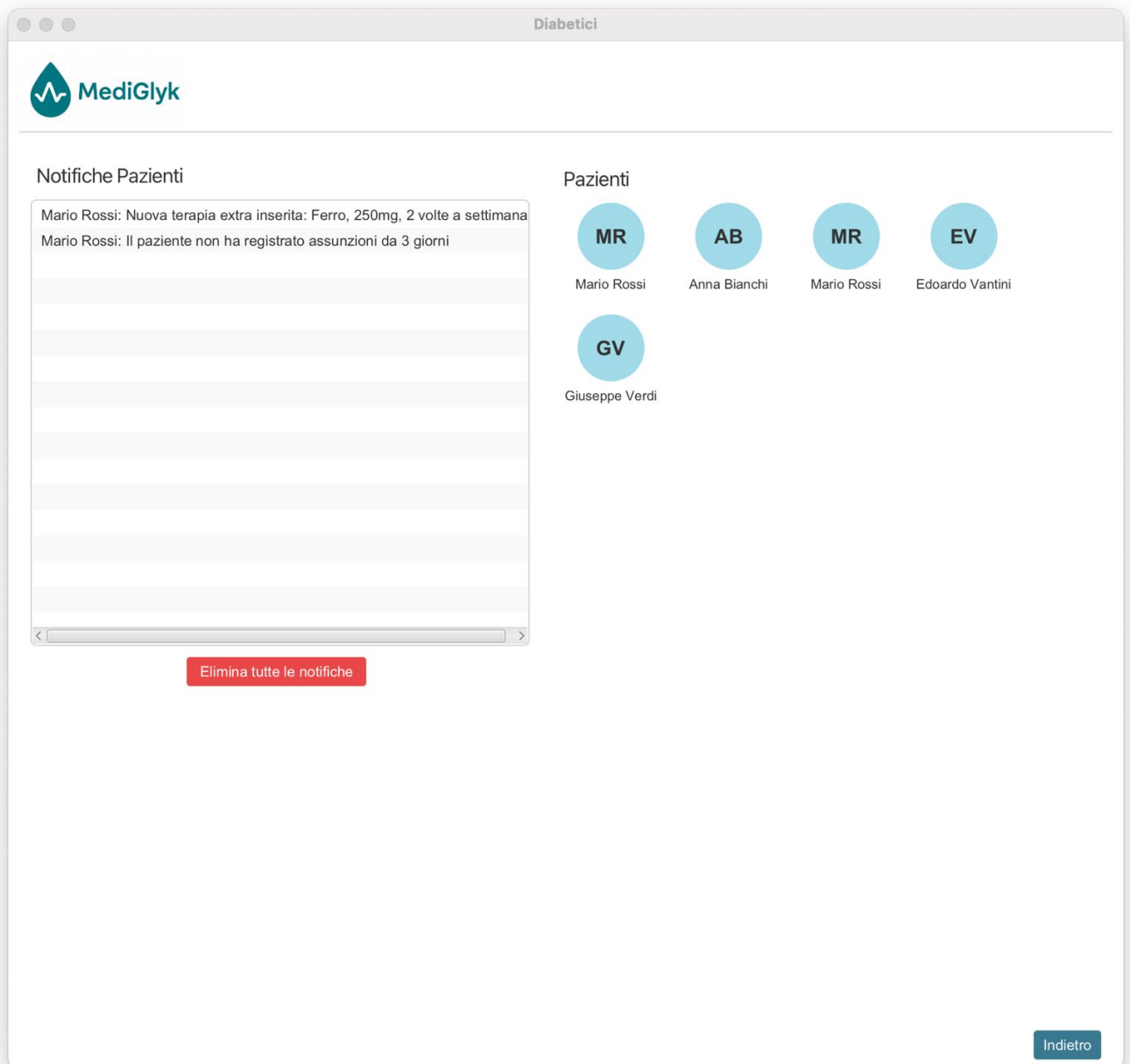
Accedi

Indietro

6.2 Home del Paziente




6.3 Home del Medico



6.4 Home della Segreteria

Diabetici



Aggiungi Paziente

Codice Fiscale:

Nome:

Cognome:

Email:

Password:

Genere:

Ulteriori Info:

Data Nascita:

Fattori Rischio:

Patologie Pregresse:

Comorbidità:

CF Medico Associato:

Aggiungi Paziente

Aggiungi Medico

Codice Fiscale:

Nome:

Cognome:

Email:

Password:

Aggiungi Medico

Gestione Utenti

Elimina Medico

Elimina Paziente

Modifica Medico Curante

Indietro

CAPITOLO SESTO

Test del Software

7.1 Unit Test

Sono stati fatti gli Unit Test implementando alcuni mock come per il database o gli utenti per testare i metodi chiave della classe Bll/model. Alcuni esempi sono:

- *Login Handler (Test 1)*

Si testa: Il comportamento della classe *LoginHandler*, che si occupa di autenticare gli utenti:

- Il successo del login per i 3 attori – paziente, medico, segreteria – controllando che i dati inseriti siano conformi con quelli nel database (**testLoginAttoreSuccess**).
- Gestione del login fallito: in caso di credenziali non conformi o utente non esistente, si ritorni null (**testLoginFallito**).

Superato: sì

- *Assunzione Service (Test 2)*

Si testa: Se l'assunzione fittizia creata venga inserita con successo all'interno del database di mock. (**testSalvaAssunzione**)

In seguito, si controlla se le assunzioni dei pazienti vengono prelevate correttamente dal database mock. (**testGetAssunzioniPerPaziente**)

Se viene inviata la notifica prevista in casa di mancata assunzione per tre giorni. (**testControllaInattivitaAssunzioni**)

Si testa se la prelevazione della glicemia dall'assunzione vada a buon fine. (**testGetAssunzioniInsulina**)

Superato: sì.

- *Check Rilevazione (Test 3)*

Si testa: Il comportamento della classe *CheckRilevazione*, che si occupa della validazione e gestione delle rilevazioni glicemiche:

- Controllo dei limiti glicemici minimi e massimi in PRE e POST pasto. (**testIsPre/PostPastoValido**).
- Nel inserire un POST pasto, che venga rispettato il limite temporale di 2h (**testControllooraPostPasto**).
- Invio della notifica al medico di riferimento, in caso di valori glicemici fuori dai limiti. (**testInviaSegnalazioneMedico**).
- Recupero delle rilevazioni dal database (**testInsertRilevazione**).

Superato: sì

- *Notifica Handler (Test 4)*

Si testa: La classe *NotificaHandler* che si occupa di gestire le notifiche scambiate tra paziente e medico. Viene inoltre utilizzato *UserSession* per simulare un utente attivo.

Vengono testate:

- Salvataggio di nuove notifiche nel database (**testAddNotifica**)
- Recupero delle notifiche filtrate per medico o senza filtro, dal database (**testGetAllNotifiche**), (**getAllNotificheByMedico**).
- Aggiornamento dello stato di una notifica (per es: per le terapie extra "IN_ATTESA", "COMPATIBILE", "NON_COMPATIBILE") (**testRispondiNotifica**)
- Eliminazione delle notifiche filtrate per il medico loggato o per uno specifico paziente, dal database (**eliminaNotifichePerMedico-Paziente**)

Superato: sì

- *Paziente (Test 5)*

Si testa: la classe *PazienteManager* che si occupa di gestire le operazioni riguardante i pazienti, interfacciandosi con *PazienteDAO* e *LogDAO*

Vengono testate:

- Recupero dei pazienti filtrati per CF e non, dal database (**testGetAllPazientiSuccess**), (**testgetByCodiceFiscaleSuccess**)
- Successo (o insuccesso) del recupero o aggiornamento delle informazioni sanitarie del paziente (**testAggiornaInformazioniSanitarie**) e inserimento del log nel database, in caso di aggiornamento delle informazioni del paziente da parte del medico (**testAggiornaInformazioniSanitarieSuccess-Fail**)

Superato: sì

7.1 Test D'integrazione

Il test prevede l'inserimento di utenti reali nel database (non mock) in fase di setup e li rimuove dopo aver eseguito i test. Si utilizza un database reale affinché venga testata l'interazione tra logica di business DAO e il database stesso. Alcuni esempi sono:

- *Login Integration (Test 1)*

Si testa:

- Il login corretto per ogni tipologia di utente (paziente, medico, segreteria) con credenziali valide così da verificare anche che venga salvata la sessione utente
- Il login fallito in caso di credenziali errate
- Funzionalità del singleton UserSession, cosicché vengano gestiti bene login e logout.

Superato: sì

- *Notifica Integration (Test 2)*

Si testa:

- L'aggiunta e la ricerca della notifica: si inserisce una notifica e si verifica che venga recuperata correttamente filtrando per il medico.
- Aggiornamento dello stato della notifica e che persista poi nel database
- Eliminazione delle notifiche filtrate per paziente e che non ci siano più dopo nel database.

Superato: sì

- *Prescrizione Integration (Test 3)*

Si testa:

- Aggiunta e recupero della prescrizione dal database
- Aggiornamento della prescrizione e che l'aggiornamento si sia salvato
- Eliminazione della prescrizione dal database

Superato: sì

- *Sintomi Integration (Test 4)*

Si testa:

- Inserimento del sintomo per il paziente e che si sia salvato nel database
- Restituzione dei sintomi segnalati nelle ultime 24h (limite massimo di durata di uno sintomo nel database) così da controllare l'efficacia della funzione del filtro temporale

Superato: sì

- *Segreteria Integration (Test 5)*

Si testa:

- Inserimento e recupero del medico nel/dal database
- Inserimento e recupero del paziente nel/dal database
- Aggiornamento del medico curante di un paziente nel database
- Eliminazione del medico o del paziente dal database

- **Superato:** sì