Ingegneria del software (2024/2025)

VR501062: BEGHINI LORENZO

VR500376: BATTIATO MATTEO

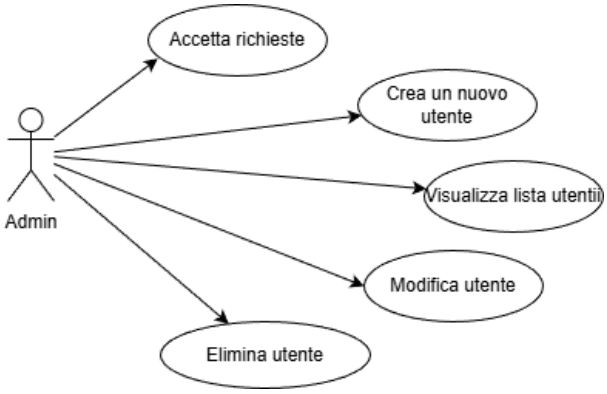
Contents

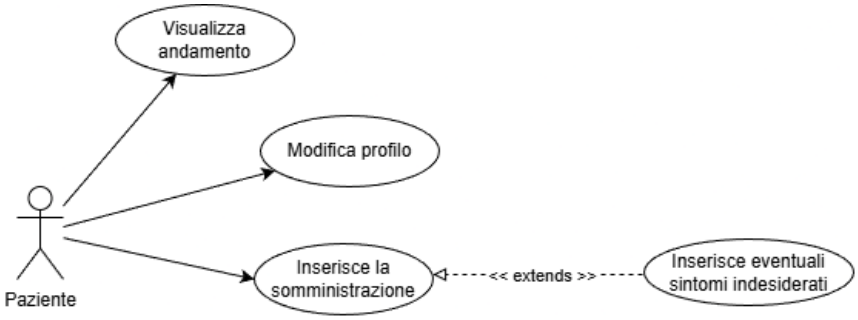
Requisiti ed interazioni utente-sistema

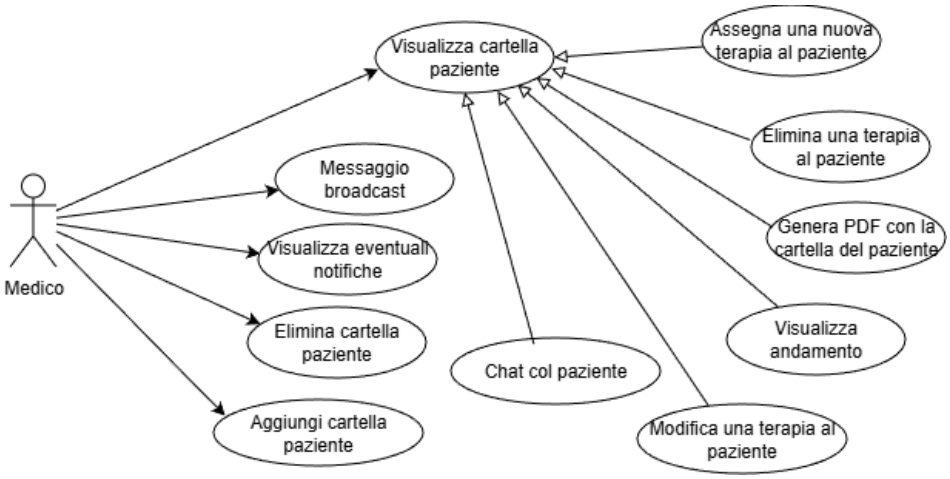
Specifiche casi d’uso

Note generali :

Il sistema creato può essere utilizzato sia dal medico che dai pazienti , inoltre da un addetto , amministratore , per la gestione utenti . Ogni tipo di utente ha le proprie credenziali , fornite dall'amministratore del sistema , con cui potranno effettuare l'autenticazione . Dopo aver effettuato l'autenticazione , nel caso quest'ultima vada a buon fine , l'utente verrà indirizzato alla rispettiva schermata iniziale , a seconda del tipo di utente che effettua il log in .







Casi d’uso relativi ai Medici

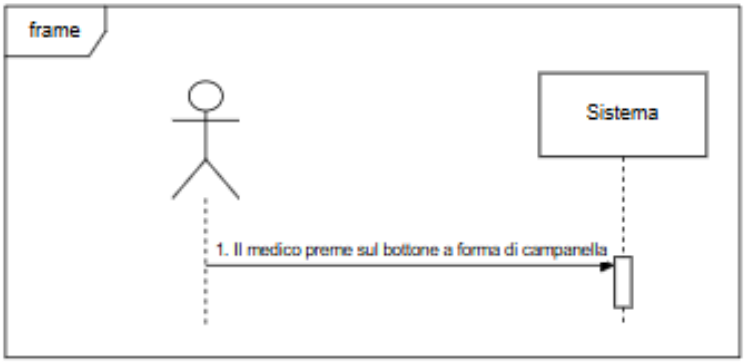
Dopo aver effettuato l'autenticazione, il medico viene indirizzato alla pagina dedicata alla gestione e supervisione dei suoi pazienti .

Visualizza eventuali notifiche

I medici potranno visionare , con l'utilizzo di una campanella con un eventuale numero di notifiche, se hanno ricevuto delle notifiche o devono ancora visualizzare dei messaggi .
Il medico premendo sulla campanella potrà visionare in un box dedicato quali e da chi ha ricevuto gli eventuali messaggi .

Attori : Medico Precondizioni : il medico dev’essersi autenticato . Passi : 1. Il medico preme sul bottone a forma di campanella . Postcondizioni : I messaggi devono essere contrassegnati come letti.

In seguito ulteriori dettagli riguardano visualizza cartella paziente .



Aggiungi/elimina cartella paziente

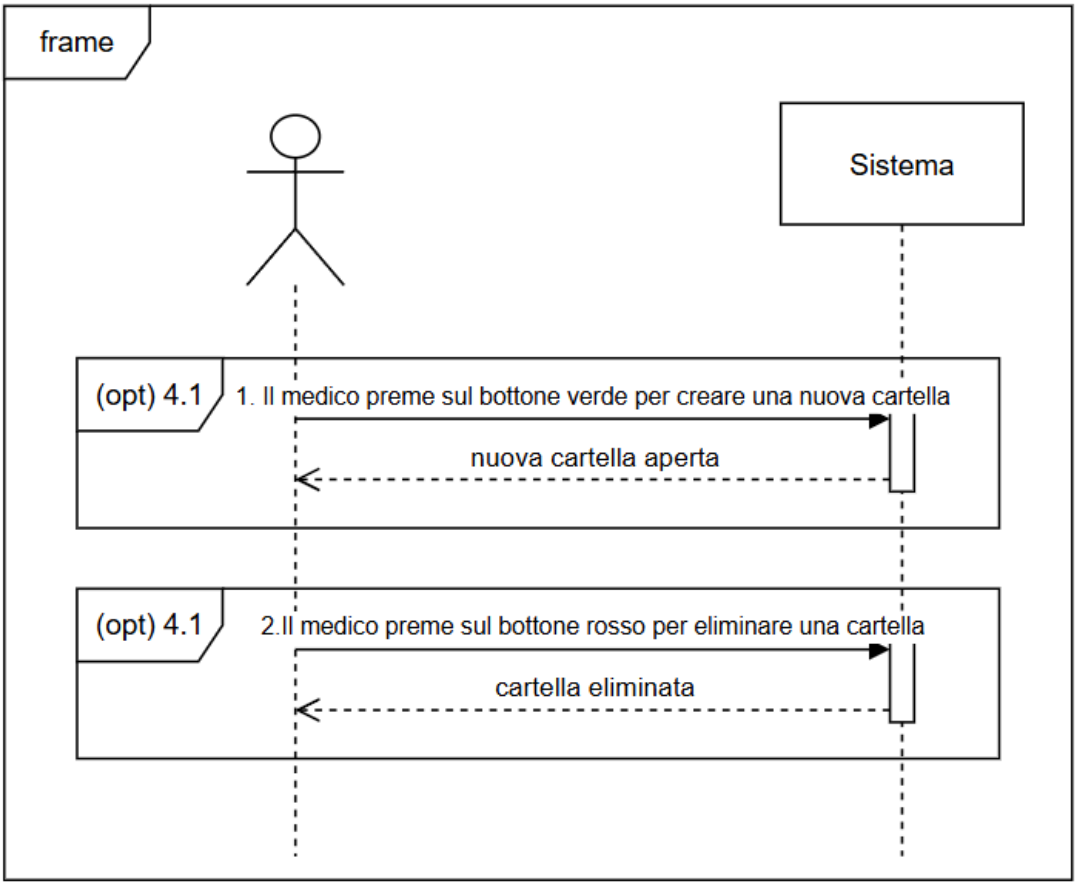
In questa fase il medico può effettuare due azioni :

- Aggiungere una nuova cartella paziente
- Eliminare una cartella paziente

Con cartella paziente si intende una sezione che contiene i dati del paziente , come una tabella con le informazioni delle terapie che sono state assegnate a quel determinato paziente , un grafico col suo andamento ed una sezione con eventuali informazioni del paziente .

Attori : Medico Precondizioni : il medico dev’essersi autenticato . Passi : 1. Il medico preme sul bottone verde per creare una nouova cartella . 2. Il medico preme sul bottone rosso per eliminare una cartella . Postcondizioni : nessuna .

In seguito ulteriori dettagli riguardano visualizza cartella paziente .



Visualizza cartella paziente

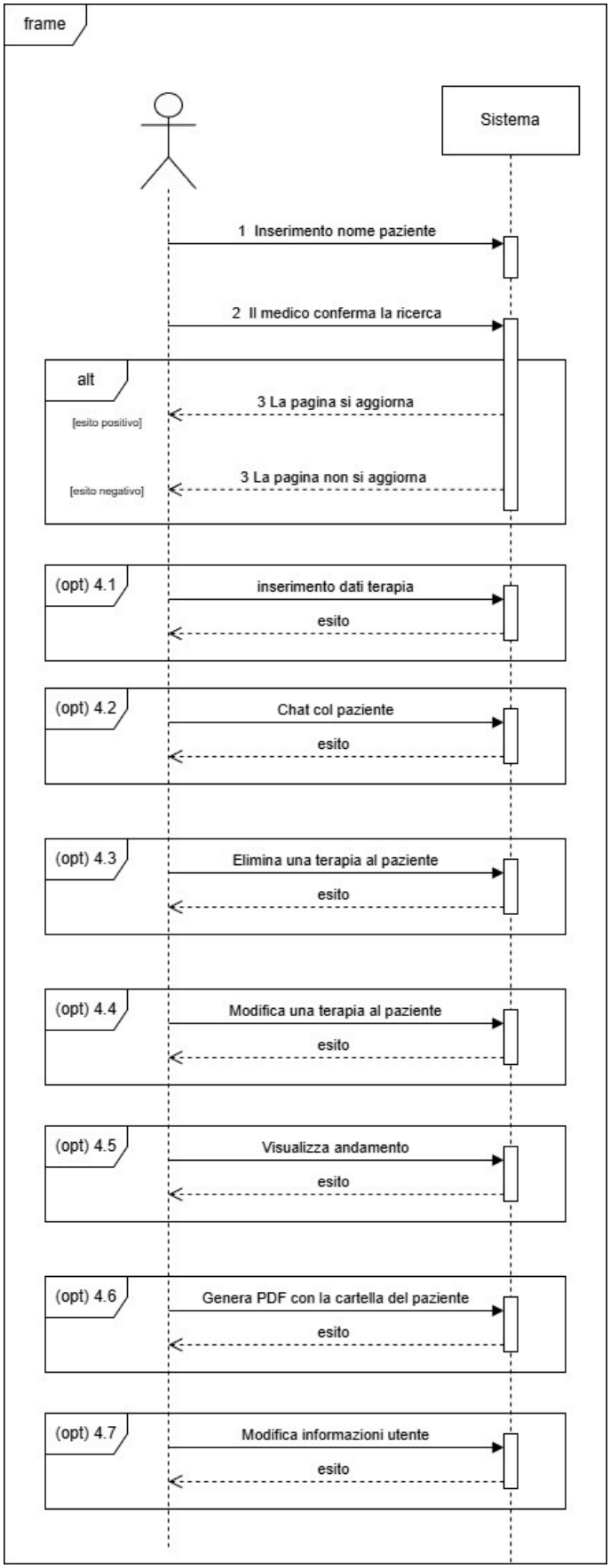
In questa fase il medico potrà visionare la cartella del paziente e svolgere diverse azioni :

- Assegna una nuova terapia al paziente
- Chat col paziente
- Elimina una terapia al paziente
- Modifica una terapia al paziente
- Visualizza andamento
- Genera PDF con la cartella del paziente
- Modifica informazioni utente

Il paziente che il medico desidera osservare verrà visualizzato dopo che il medico avrà inserito l'username del paziente in un apposito input , mentre la chat si potrà osservare una volta che la cartella del paziente verrà caricata correttamente con l'utilizzo di un'apposito bottone che aprirà la chat .

Attori : Medico
Precondizioni : il medico dev'essersi autenticato .
Passi :
1. Il medico inserisce username del paziente .
2. Avvia la ricerca ed il caricamento della cartella .
3. Il medico potrà svolgere diverse azioni all'interno della cartella come :
   3.1 Assegna una nuova terapia al paziente .
   3.2 Chat col paziente .
   3.3 Elimina una terapia al paziente .
   3.4 Modifica una terapia al paziente .
   3.5 Visualizza andamento .
   3.6 Genera PDF con la cartella del paziente .
   3.7 Modifica informazioni utente .

Postcondizioni : nessuna .

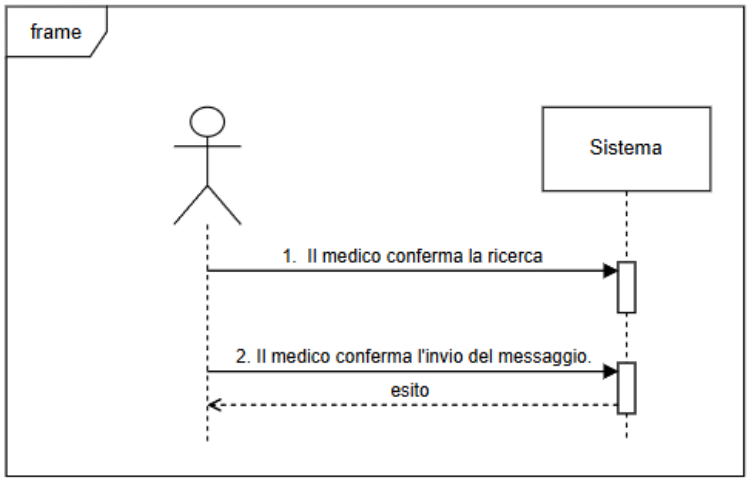

frame
Sistema
1 Inserimento nome paziente
2 Il medico conferma la ricerca
alt
3 La pagina si aggiorna
[esito positivo]
3 La pagina non si aggiorna
[esito negativo]
(opt) 4.1
inserimento dati terapia
esito
(opt) 4.2
Chat col paziente
esito
(opt) 4.3
Elimina una terapia al paziente
esito
(opt) 4.4
Modifica una terapia al paziente
esito
(opt) 4.5
Visualizza andamento
esito
(opt) 4.6
Genera PDF con la cartella del paziente
esito
(opt) 4.7
Modifica informazioni utente
esito

Messaggio broadcast

Il medico può inoltre inviare un messaggio nella chat dei suoi pazienti grazie ad un box dedicato , utile per messaggi informativi , come ferie , orari ecc...

Attori : Medico
Precondizioni : il medico dev'essersi autenticato .
Passi :
1. Il medico scrive il messaggio nell'aposito box .
2. Il medico preme sul bottone invio per inoltrare a tutti i suoi pazienti il messsaggio .

Postcondizioni : inserimento messaggio database .



Il paziente potrà vedere il messaggio una volta aperta la chat col medico .

Casi d’uso relativi ai Pazienti

Casi d’uso relativi ai Pazienti

Dopo aver effettuato l'autenticazione, il paziente viene indirizzato alla pagina dedicata all'inserimento delle sue somministrazioni quotidiane, dove potrà visionare anche lui il suo andamento con l'uso di un grafico ed in una tabella presenti nella pagina dedicata .

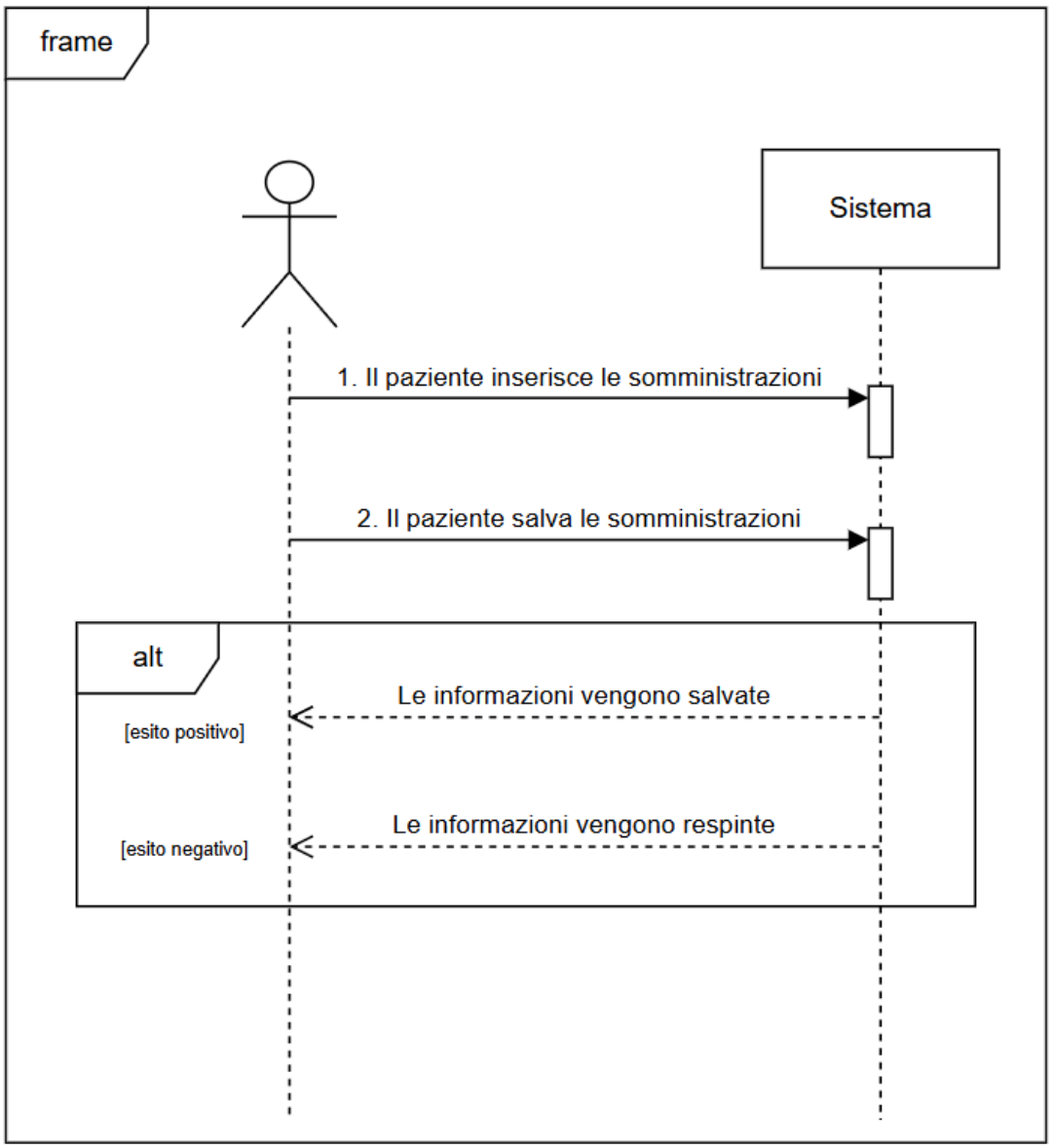
Inserisce la somministrazione

Il paziente potrà inserire la somministrazione che ha appena fatto , ha sei somministrazioni da fare durante il giorno , la prima al mattino prima e dopo la colazione , la seconda prima e dopo pranzo e la terza prima e dopo cena . Se durante il giorno dovesse dimenticarsi di inserirle il sistema invierà un messaggio , sotto forma di alert , dove gli ricorda le somministrazioni dei pasti che ha saltato , il sistema invierà al medico un messaggio qualora il paziente dovesse inserire dei valori al di fuori del range stabilito o se quest'ultimo non inserisce somministrazioni da almeno 3 giorni .

Attori : Paziente
Precondizioni : il paziente dev’essersi autenticato .
Passi :
1. Il paziente inserisce le somministrazioni .
2. Il paziente salva le somministrazioni .

Postcondizioni : salvataggio nel database .

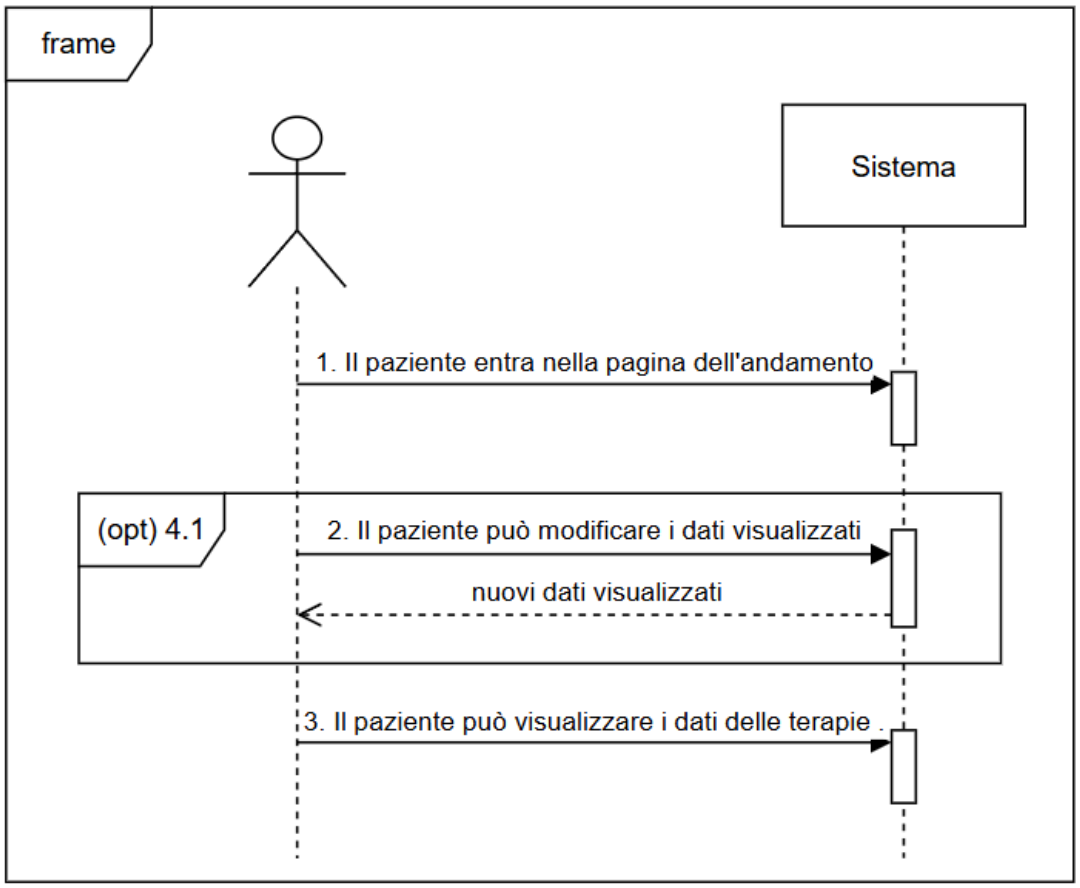
In seguito ulteriori dettagli riguardano inserisce eventuali sintomi indesiderati o in chat col medico.



Visualizza andamento

Il paziente può osservare l'andamento delle sue somministrazioni con l'utilizzo di un grafico che può essere modellato nel tempo (a seconda del periodo che vuole visionare il paziente) .

Attori : Paziente Precondizioni : il paziente dev'essersi autenticato . Passi : 1. Il paziente osserva il grafico andamenti . 2. Il paziente preme i pulsanti per spostarsi nel tempo . Postcondizioni : nessuna .

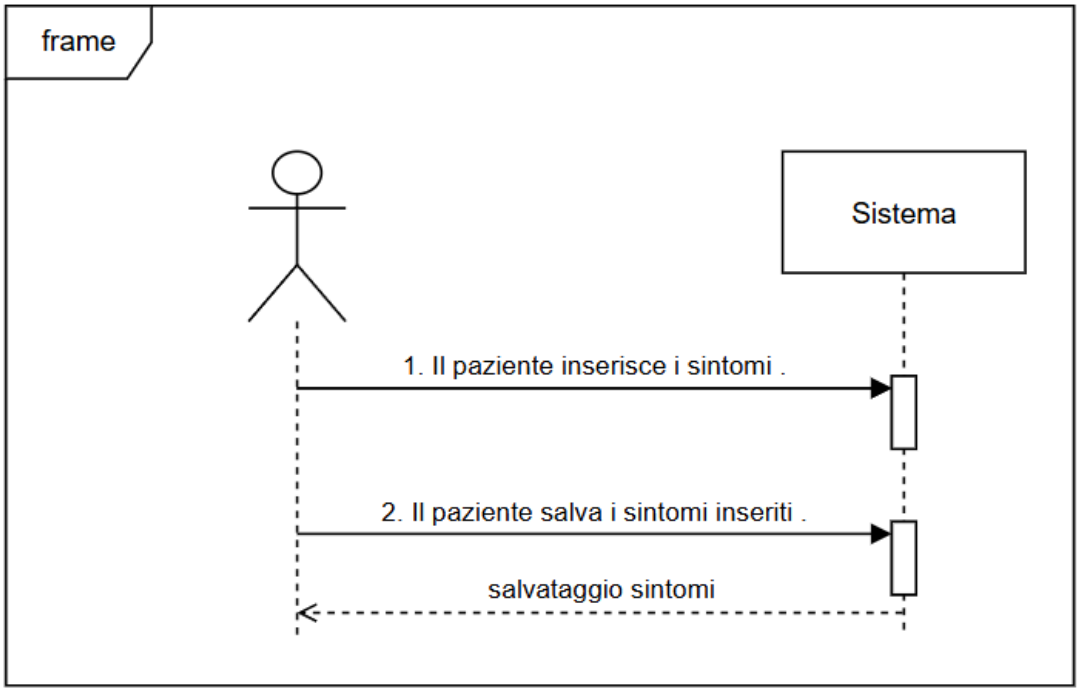


Inserisce eventuali sintomi indesiderati

Il paziente può inserire eventuali sintomi causati dalle somministrazioni .

Attori : Paziente
Precondizioni : il paziente dev'essersi autenticato .
Passi :
1. Il paziente inserisce i sintomi .
2. Il paziente salva i sintomi inseriti .

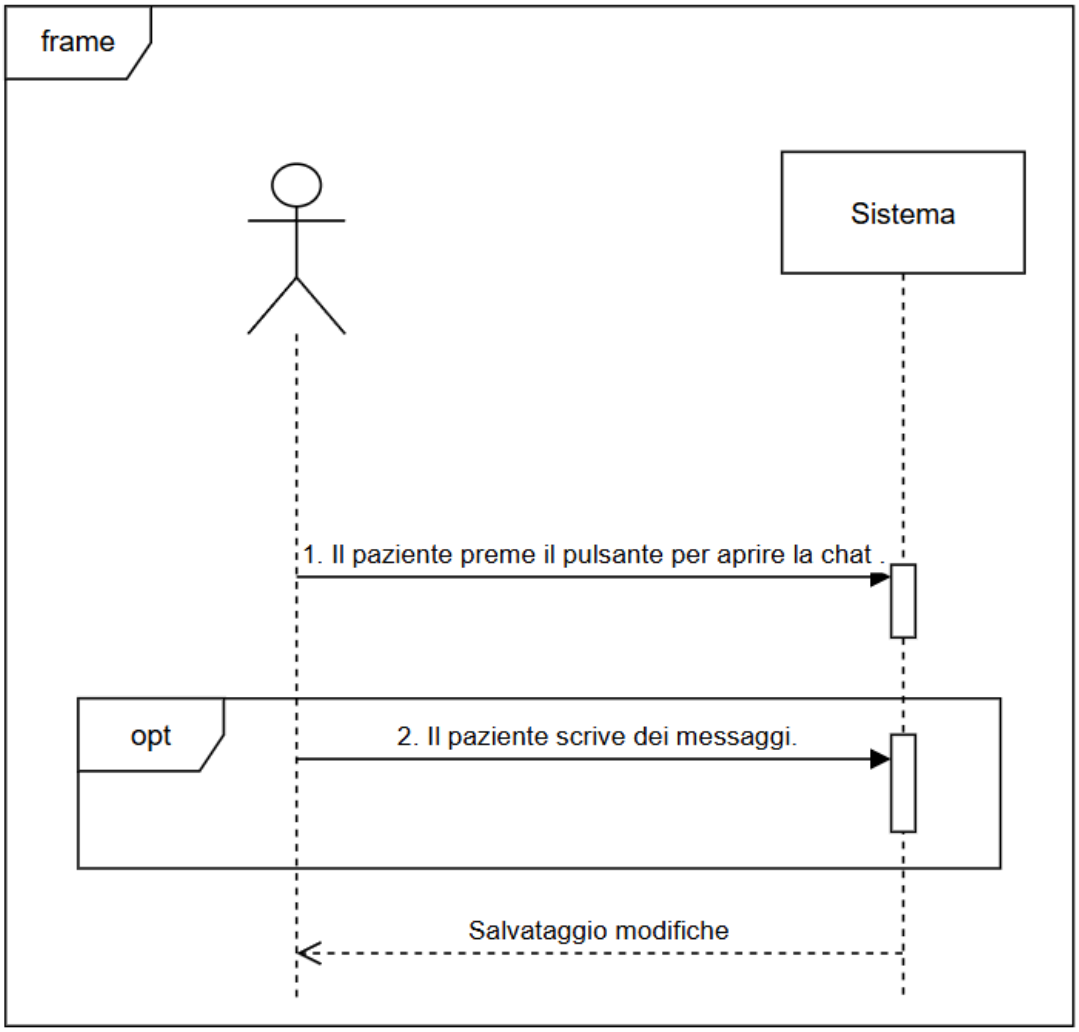
Postcondizioni : salvataggio nel database .



Chat col medico

Il paziente , con l'utilizzo di una chat , può inviare messaggi col medico , il sistema può inoltrare messaggi al medico in questa chat come spiegato in inserisce la somministrazione . Il paziente saprà di avere dei nuovi messaggi da leggere grazie ad una notifica sul bottone per l'apertura della chat .

Attori : Paziente
Precondizioni : il paziente dev'essersi autenticato .
Passi :
1. Il paziente preme il pulsante per aprire la chat .
2. Il paziente legge o scrive eventuali messaggi e li invia .
Postcondizioni : recapito/salvataggio messaggi .

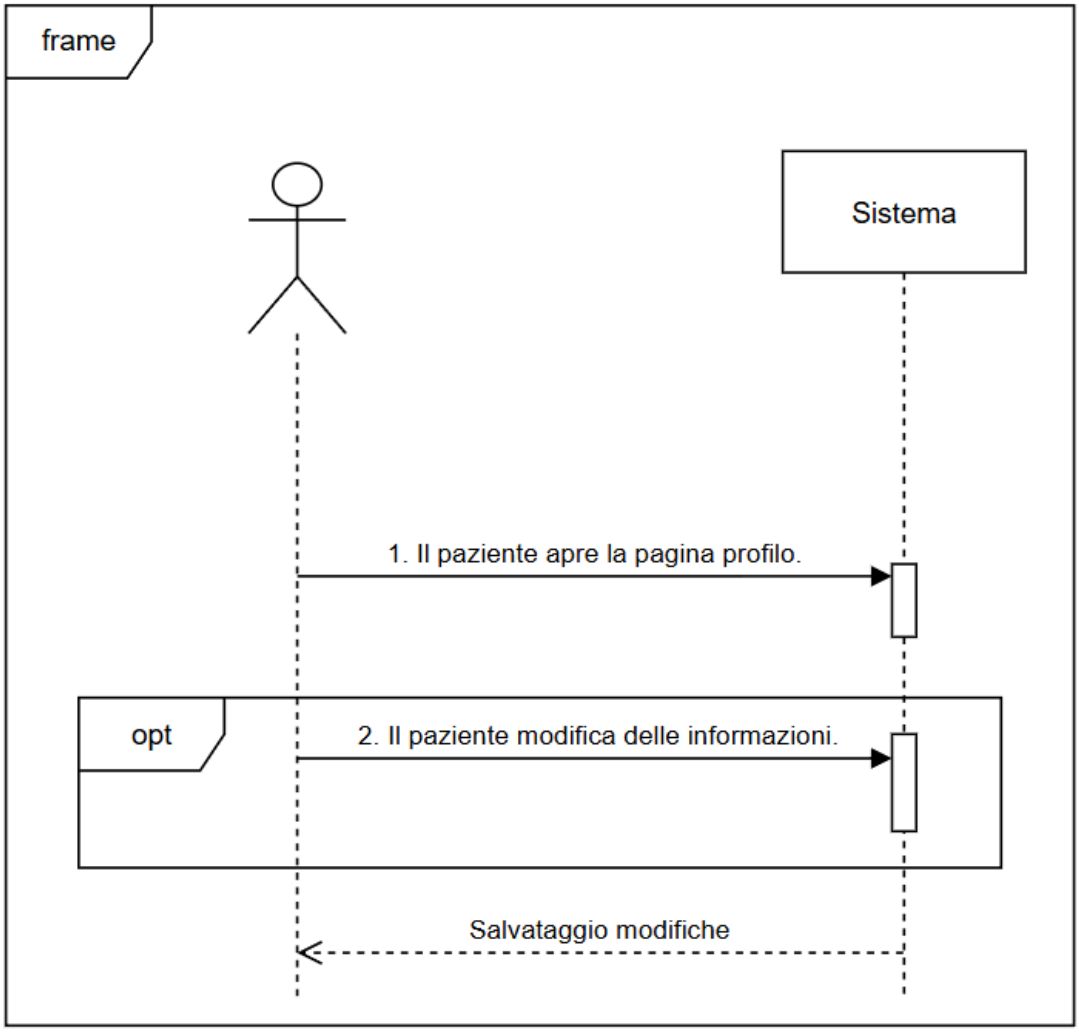


Modifica profilo

Il paziente se desidera potrà modificare i suoi dati in un apposita pagina , potrà inserire la propria mail , nome , cognome , immagine profilo ed ulteriori informazioni , inoltre se desidera potrà richiedere un cambio password all'amministratore .

Attori : Paziente
Precondizioni : il paziente dev'essersi autenticato .
Passi :
1. Il paziente preme il pulsante per aprire la pagina profilo .
2. Il paziente legge o modifica eventuali dati e li salva .

Postcondizioni : salvataggio modifiche .


frame
Sistema
1. Il paziente apre la pagina profilo.
opt
2. Il paziente modifica delle informazioni.
Salvataggio modifiche

Casi d’uso relativi Admin

Casi d’uso relativi Admin

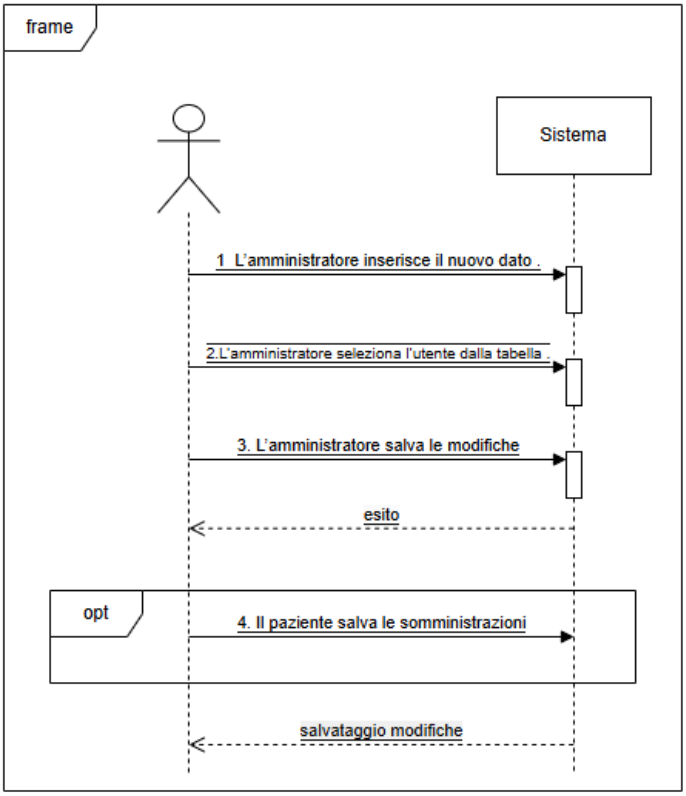
Dopo aver effettuato l'autenticazione, l'amministratore viene indirizzato alla pagina dedicata alla gestione degli utenti .

Crea un nuovo utente

L'amministratore può creare un nuovo utente , dandogli username , tipo utente , password e nel caso fosse un paziente gli viene assegnato il suo medico .

Attori : amministratore
Precondizioni : l’amministratore dev’essersi autenticato .
Passi :
1. L’amministratore inserisce i dati del nuovo utente .
2. L’amministratore salva ill nuovo utente se non ci sono evenntuali problemi .

Postcondizioni : salvataggio nel database .

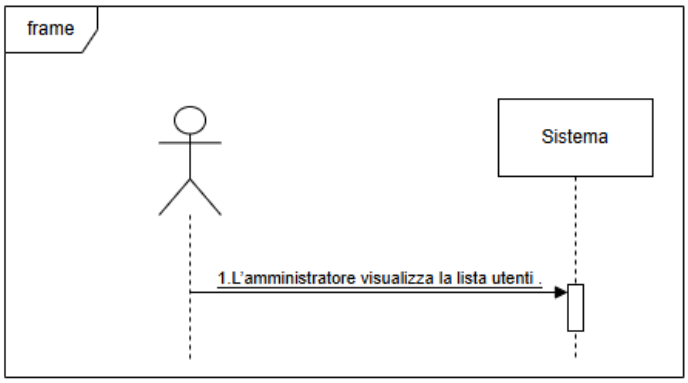


Visualizza lista utenti

L'amministratore può visionare tutta la lista degli utenti , coi loro dati .

Attori : amministratore
Precondizioni : l’amministratore dev’essersi autenticato .
Passi :
1. L’amministratore visualliza la lista utenti .

Postcondizioni : nessuna .

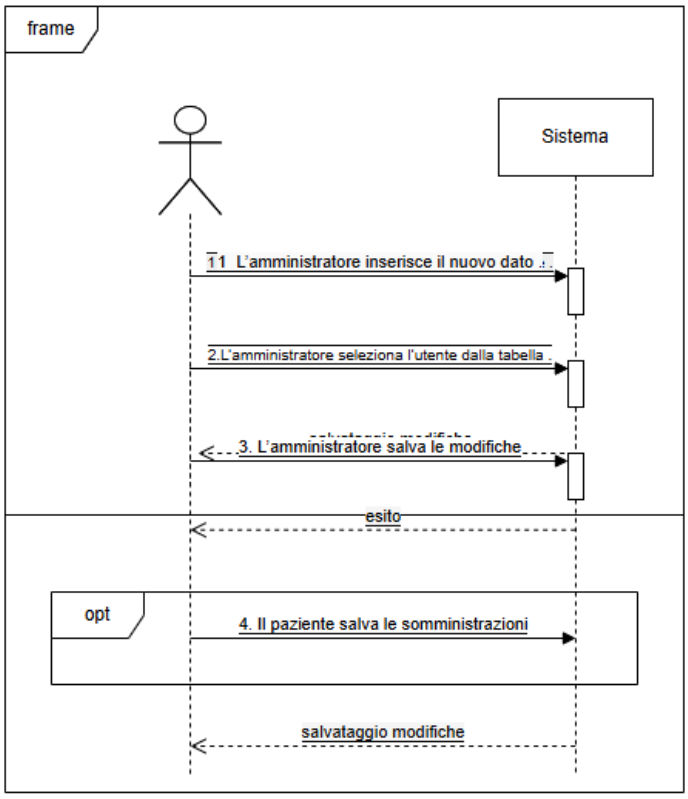


Modifica utente

L'amministratore può modificare un utente già esistente , inserisce il dato da modificare , quello nuovo , seleziona l'utente dalla tabella e salva .

Attori : amministratore
Precondizioni : l'amministratore dev'essersi autenticato .
Passi :
1. L'amministratore inserisce il nuovo dato .
2. L'amministratore seleziona l'utente dalla tabella .
3. L'amministratore salva le modifiche .

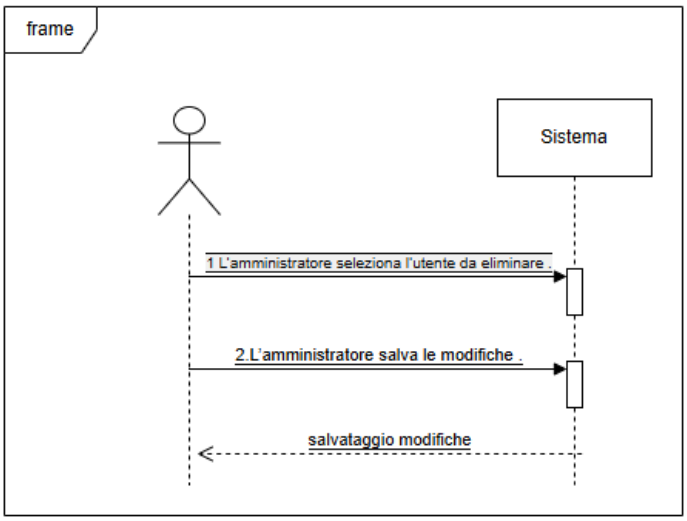
Postcondizioni : salvataggio nel database .



Elimina utente

L'amministratore può eliminare utente presente nella tabella .

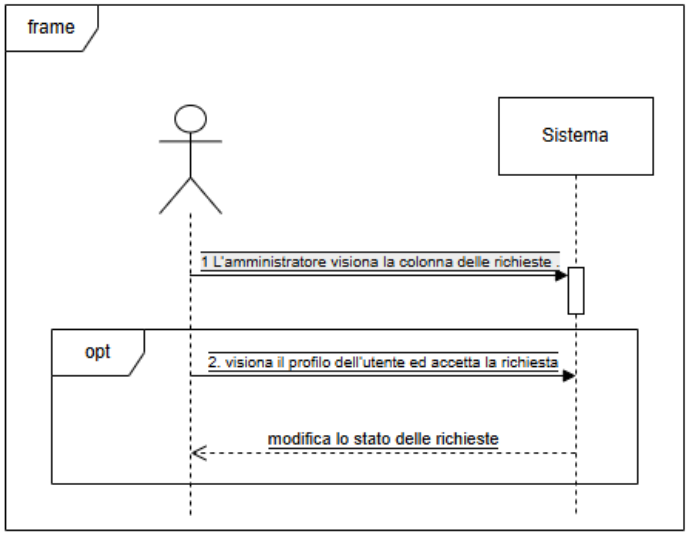
Attori : amministratore
Precondizioni : l'amministratore dev'essersi autenticato .
Passi :
1. L'amministratore seleziona l'utente dalla tabella da eliminare .
2. L'amministratore salva le modifiche .
Postcondizioni : salvataggio nel database .



Accetta richieste

L'amministratore può accettare le richieste per il cambio password .

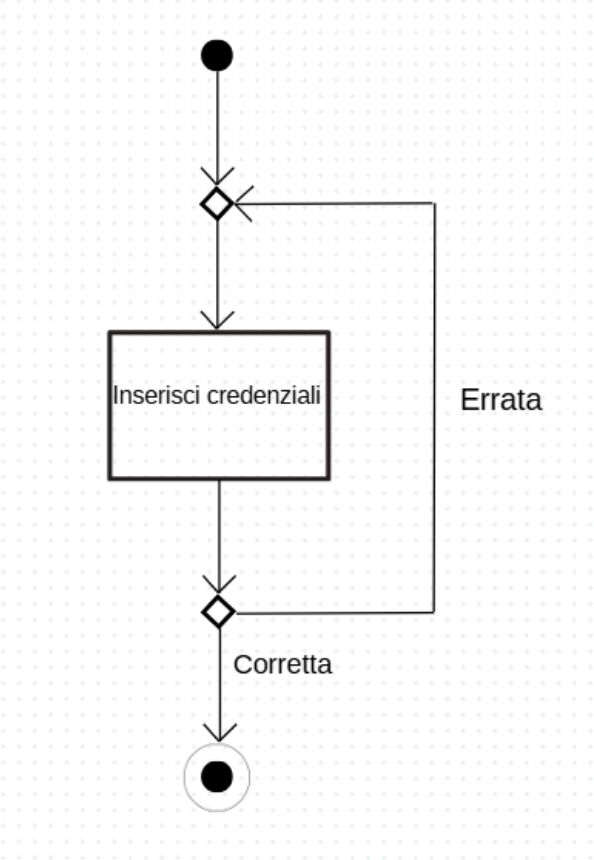
Attori : amministratore
Precondizioni : l'amministratore dev'essersi autenticato .
Passi :
1. L'amministratore visiona la colonna delle richieste .
2. Nel caso in cui ci sia una richiesta attiva visiona il profilo dell'utente ed accetta la richiesta .
Postcondizioni : modifica stato richiesta .



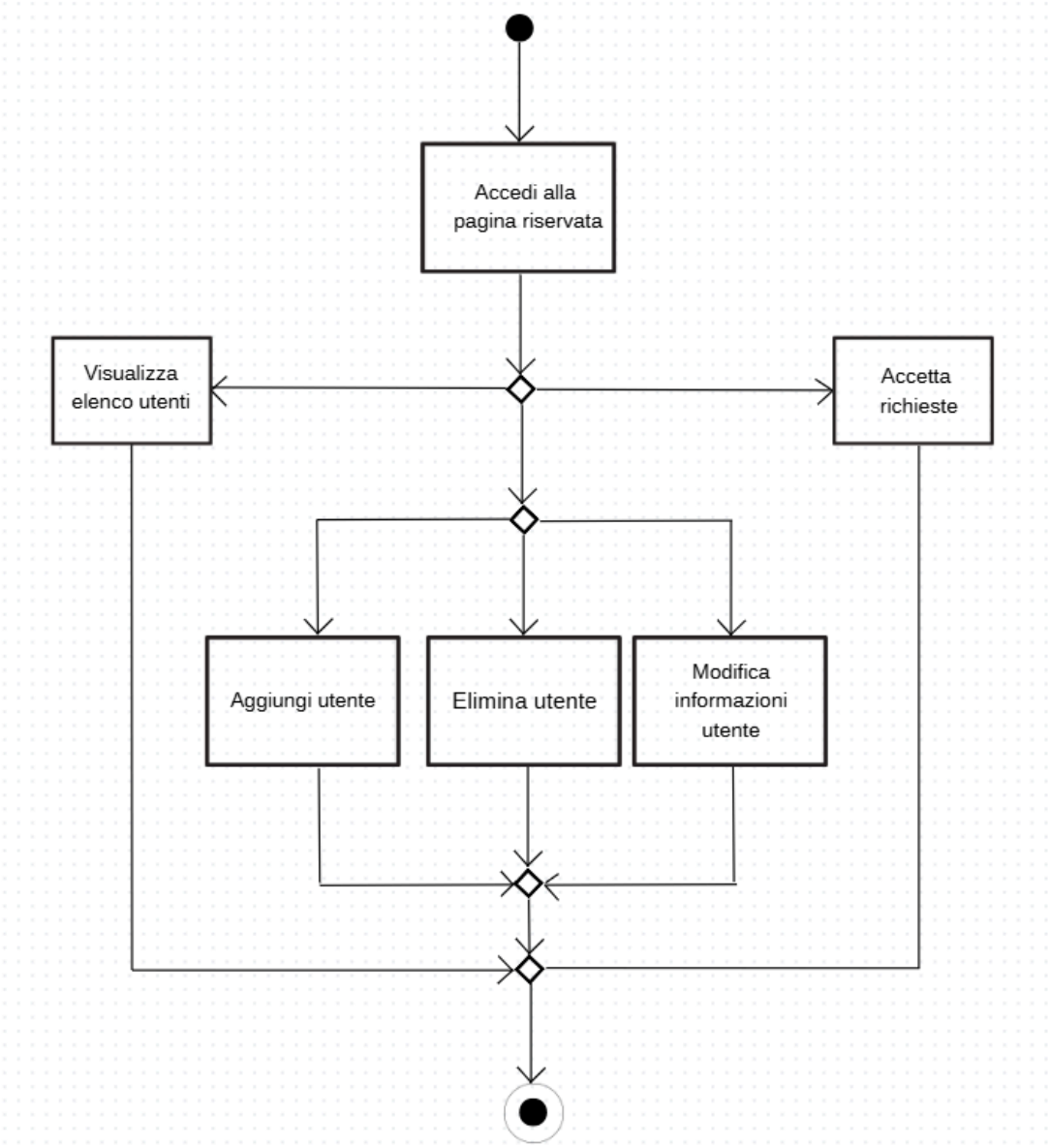
Diagrammi di attività

I seguenti diagrammi rappresentano il flusso di una singola serie di operazioni .

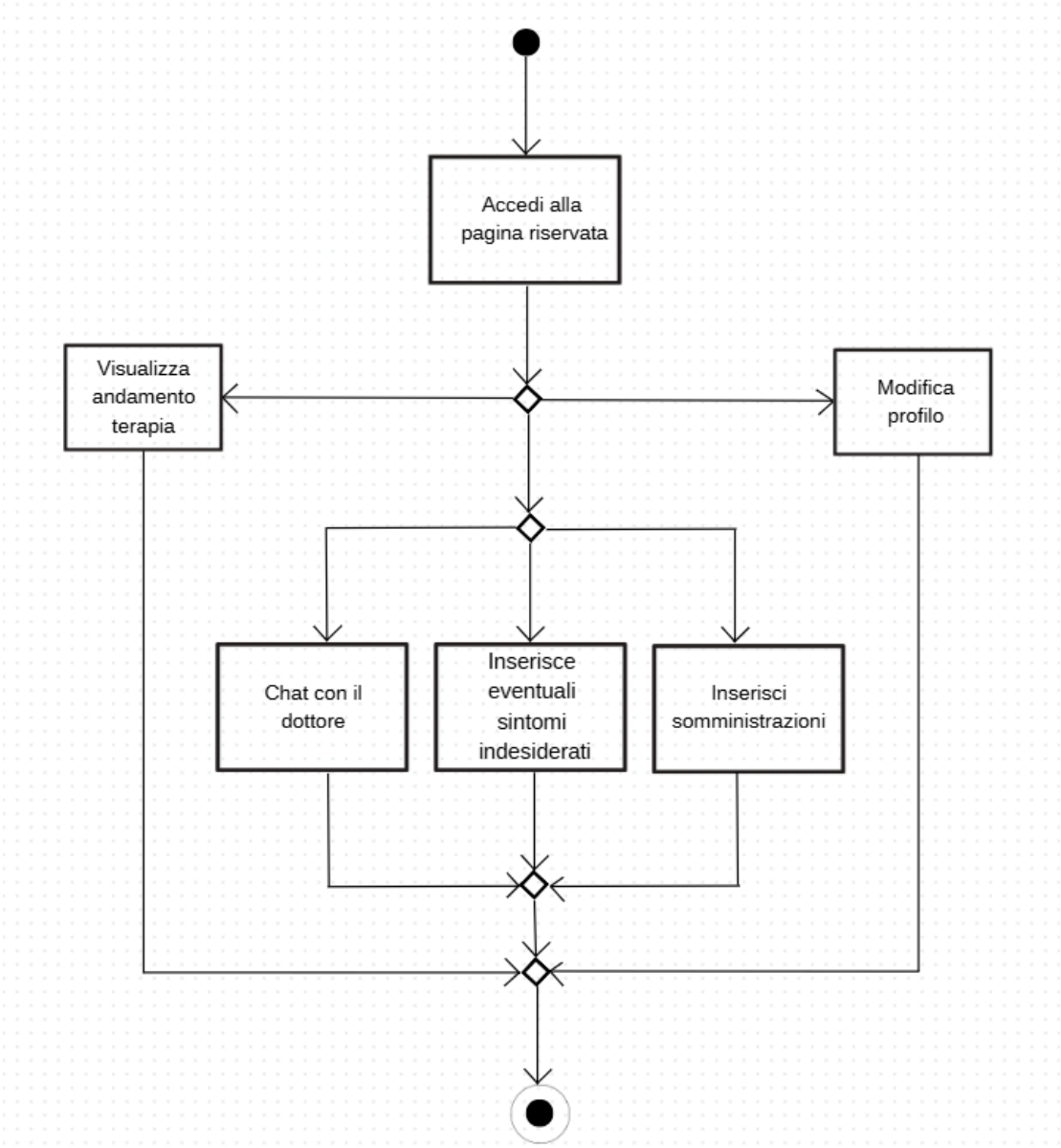
Autenticazione



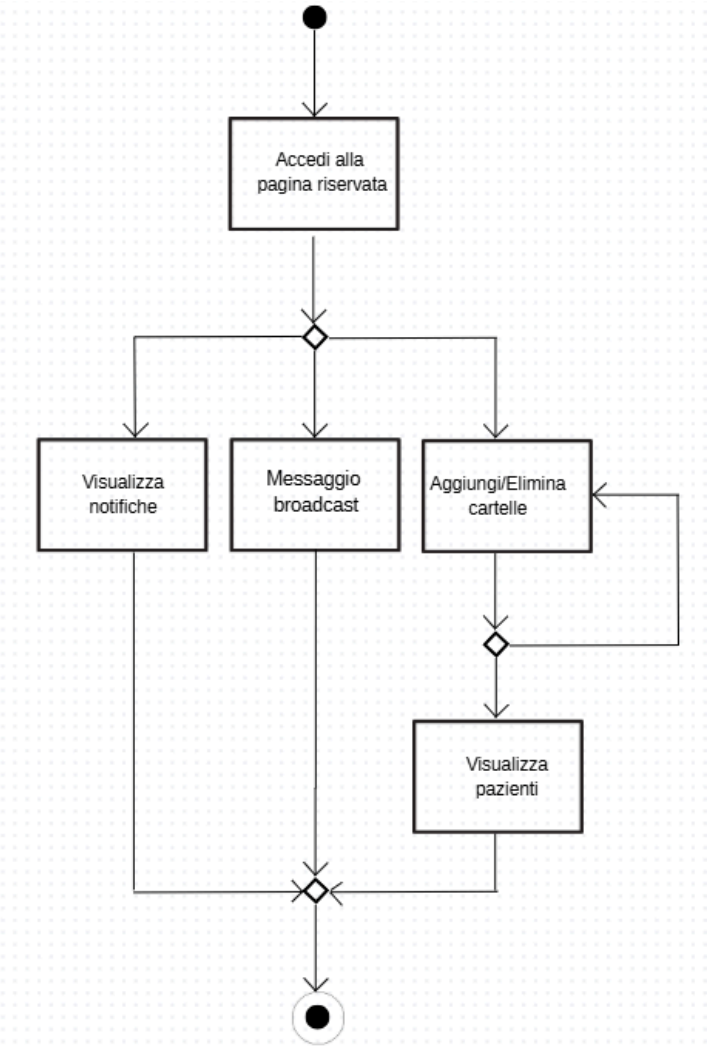
Admin


Accedi alla pagina riservata
Visualizza elenco utenti
Accetta richieste
Aggiungi utente
Elimina utente
Modifica informazioni utente

Paziente


Accedi alla pagina riservata
Visualizza andamento terapia
Modifica profilo
Chat con il dottore
Inserisce eventuali sintomi indesiderati
Inserisci somministrazioni

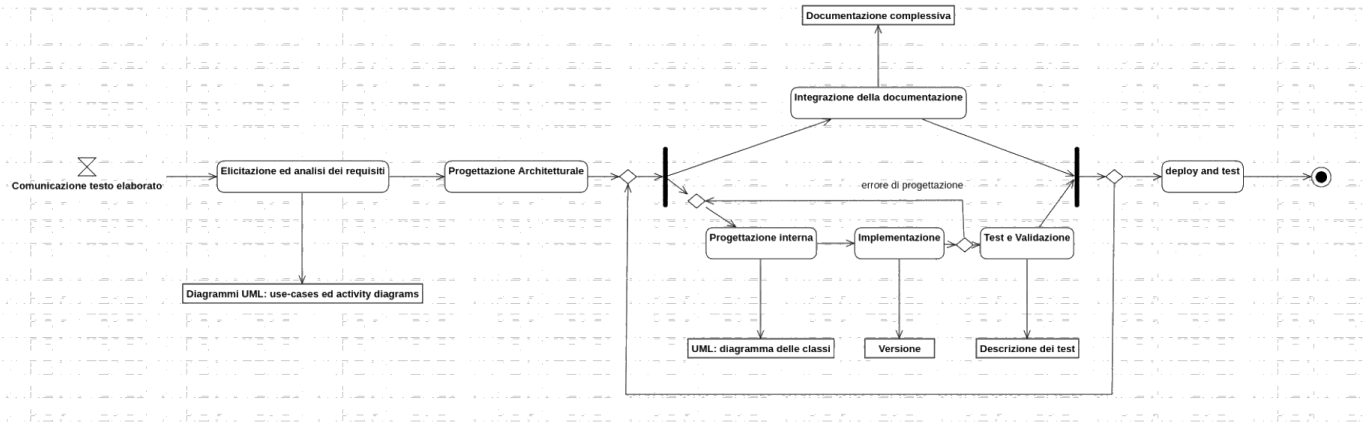
Medico


Accedi alla pagina riservata
Visualizza notifiche
Messaggio broadcast
Aggiungi/Elimina cartelle
Visualizza pazienti

Sviluppo: progetto dell’architettura ed implementazione del sistema

Note sul processo di sviluppo

Nel mio progetto è stata realizzata un’applicazione per la gestione di pazienti e medici per il diabete.
Nel processo di sviluppo si è cercato di mantenere sequenziali le fasi di progettazione , implementazione e validazione . Il motivo di questa scelta è stato quello di cercare di progettare e poi implementare .
Dopo ogni modifica sono stati svolti dei test .
Lo sviluppo del progetto non è sempre stato diretto e progressivo, ma ha avuto momenti di revisione e miglioramento del codice già scritto .
Durante lo sviluppo dell’architettura e dell’implementazione del sistema, è stata redatta una documentazione del codice volta a facilitare la comprensione, l’uso e la futura manutenzione del software .
La persistenza dei dati è gestita tramite il pattern DAO, che consente di astrarre l’accesso al database SQLite e mantenere separata la logica applicativa dalla logica di memorizzazione.



Organizzazione dei pacchetti

La suddivisione dei pacchetti nel progetto è la seguente :

- `controllers` : gestisce la logica di interazione con l’utente.
- `models` : definisce le entità e strutture dati.
- `DAO` : incapsula la logica di accesso al database.
- `utility` : contiene classi di supporto e metodi riutilizzabili.
- `view` : potrebbe includere logiche di visualizzazione o wrapper per le scene.
- `resources` : suddiviso in `fxml` , `css` , `img` per una gestione ordinata degli asset.

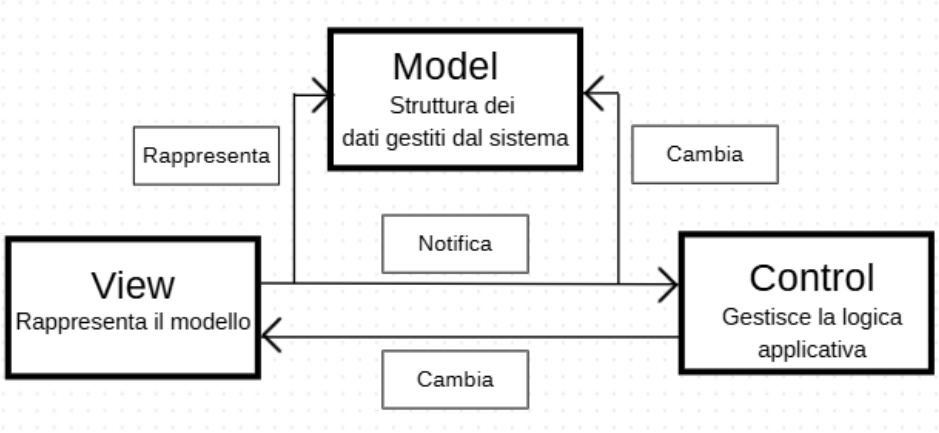
Progettazione e pattern architetturali usati

Il progetto è stato sviluppato seguendo un’architettura modulare e scalabile, con una chiara separazione tra i diversi livelli dell’applicazione. La struttura riflette l’adozione del pattern **MVC (Model-View-Controller)**, integrato con altri pattern di progettazione per migliorare la manutenibilità e la leggibilità del codice .
Ogni membro è stato inserito in un package separato .

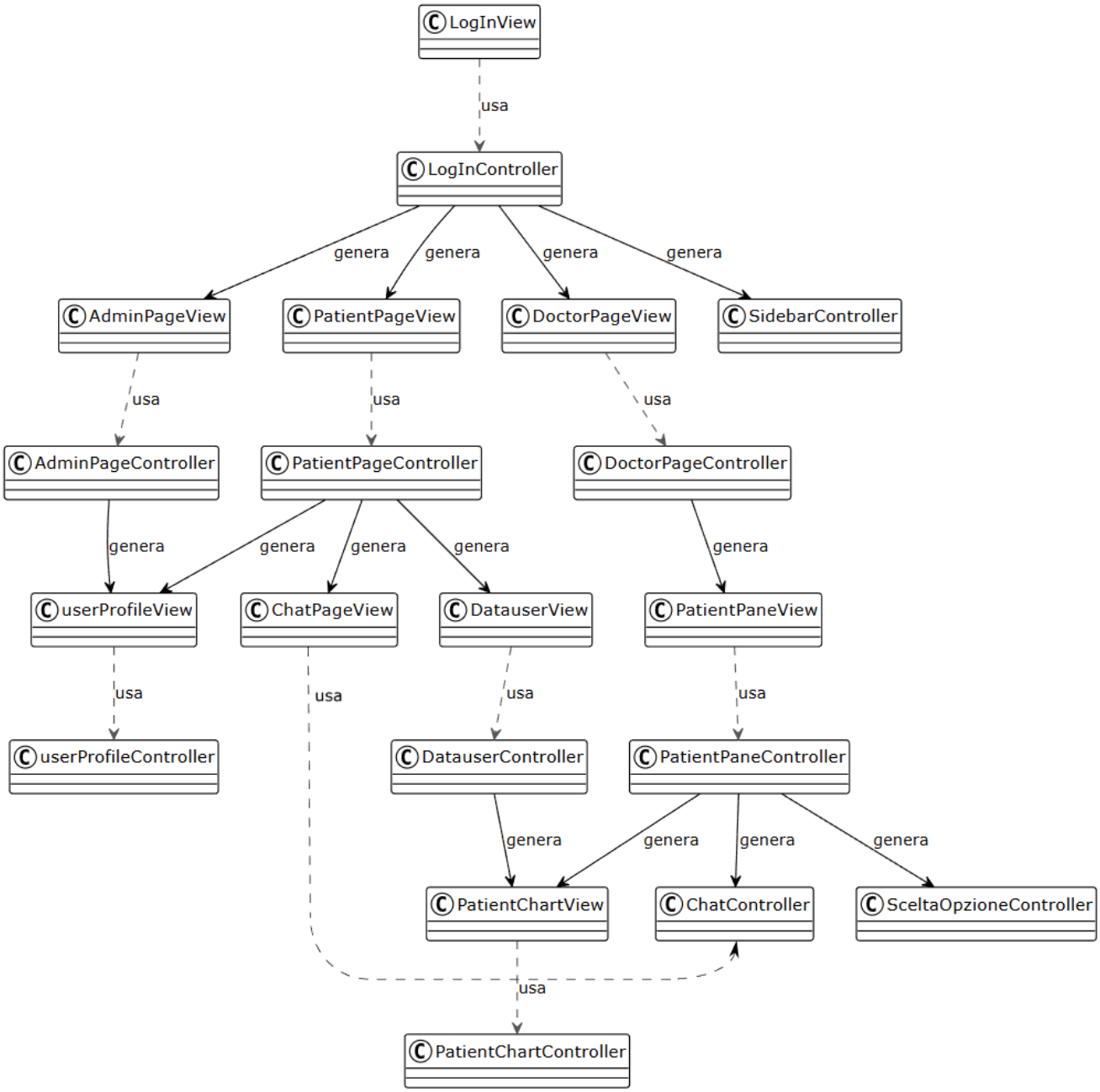
- **Model** (`models`): contiene le classi che rappresentano i dati e la logica di business .
- **View** (`resources/fxml`): definisce l’interfaccia utente tramite file FXML .
- **Controller** (`controllers`): gestisce gli eventi dell’interfaccia e collega la view al model .

Questa suddivisione consente una gestione chiara delle responsabilità e facilita eventuali modifiche o estensioni future.
Il progetto utilizza un database SQLite (`identifier.sqlite` , `miodatabase.db`) per la memorizzazione dei dati. L’accesso è gestito tramite il pattern DAO, che garantisce l’astrazione e la separazione tra logica applicativa e persistenza.

Architettura MVC



Di seguito viene mostrato il Diagramma delle classi :



Se si desidera avere un quadro completo di tutto il progetto, consigliamo la visione della seguente immagine :
https://github.com/battiatomatteo/Progetto-Java-Fx/blob/main/relazione/ProgettoJavaFX.png

Design pattern usati

Pattern	Problema risolto	Dove / come lo ho applicato	Vantaggi
MVC (Model-View-Controller)	Separare la logica dell'interfaccia (View), la gestione degli eventi e del flusso (Controller), e i dati / logica di business (Model)	In JavaFX, normalmente ogni finestra / scena ha un controller; il Model sono le classi che rappresentano pazienti, medici, etc., e il codice che interagisce col DB è separato. Potresti descrivere quali classi sono controllers, quali sono model.	Mantiene il codice più organizzato, facile da modificare la GUI senza toccare la logica dei dati, facilita il testing.
DAO (Data Access Object)	Incapsulare l'accesso al database, separando la logica SQL/DB dal resto dell'applicazione	Se hai una classe o un pacchetto che si occupa solo di interazioni col database (operazioni CRUD: create, read, update, delete) per entità `Paziente`, `Medico`.	Isola i dettagli del DB, cambiando DB (o query) non serve cambiare il resto del codice, migliora la manutenzione.
Observer / Listener	Notificare componenti interessate quando uno stato cambia (utile nelle GUI per aggiornare viste quando cambia il modello)	JavaFX già usa listener / binding per proprietà osservabili; se usi proprietà osservabili in model o controller per aggiornare automaticamente la GUI, è un esempio.	GUI più reattiva, meno codice boilerplate per l'aggiornamento manuale, migliore separazione.

Note :

Nel progetto, la logica di accesso al database è incapsulata in classi DAO dedicate. Ad esempio, ho una classe `PazienteDAO` che implementa i metodi `insert`, `update`, `delete`, `findById`, `findAll`. Questo permette al resto dell'applicazione (GUI) di non preoccuparsi delle query SQL .

Per l'interfaccia grafica è stata adottata una struttura che rispecchia il pattern MVC: ogni schermata JavaFX ha un controller che risponde agli eventi (clic, input utente), un modello (classi dati e logica) separato, e le viste definite nei file FXML o nel codice JavaFX .

Attività di test e validazione

Per garantire l'affidabilità e la qualità del software sviluppato, sono state condotte diverse attività di verifica e validazione :

1. Analisi del documento delle specifiche e confronto con i diagrammi realizzati.
2. Controllo della coerenza tra i diagrammi e il codice implementato.
3. Revisione del codice, con particolare attenzione alla corretta applicazione dei pattern e all'individuazione di possibili *code smell* o cattive pratiche.
4. Esecuzione di unit test automatizzati con **JUnit**, volti a verificare la correttezza delle classi considerate più critiche.
5. Sessioni di test interne da parte degli sviluppatori.
6. Attività di test da parte di utenti finali non coinvolti nello sviluppo, per valutare l'efficacia e l'usabilità del software.

Unit test

Ai fini di test automatizzati con JUnit è stata creata la seguente classe (*PatientPageDaoTest*) :

```
import DAO.DBConnection;
import models.*;
import org.junit.jupiter.api.*;
import org.mindrot.jbcrypt.BCrypt;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.SQLException;
import java.time.LocalDate;
import java.time.format.DateTimeFormatter;
import java.util.Map;

import static org.junit.jupiter.api.Assertions.*;

class PatientPageDaoTest {

    private static PatientPageDaoForTest patientDao;

    @BeforeAll
    static void setup() {
        patientDao = new PatientPageDaoForTest();
    }

    @BeforeEach
    void setupDatabase() throws SQLException {
        resetDatabase();

        try (Connection conn = DBConnection.getConnection()) {

            String addSamplePatient1 = "INSERT INTO utenti(username, tipo_utente, password, medico,
informazioni) VALUES (?, ?, ?, ?, ?)";
            try (PreparedStatement pstmt = conn.prepareStatement(addSamplePatient1)) {
                pstmt.setString(1, "TestPaziente1");
                pstmt.setString(2, "paziente");
                String password = "TestPaziente1.";
                pstmt.setString(3, BCrypt.hashpw(password, BCrypt.gensalt()) );
                pstmt.setString(4, "TestMedico1");
                pstmt.setString(5, "Paziente di Test 1");
                pstmt.executeUpdate();
            }
```

```
    }
}

// ====================== somministrazioneTabella ======================

@Test
void testSomministrazioneTabella() throws SQLException {
    try (Connection conn = DBConnection.getConnection()) {
        String insert = """
            INSERT INTO rilevazioni_giornaliere
            (data_rilevazione, orario, rilevazione_pre_pasto, rilevazione_post_pasto, username)
            VALUES (?, ?, ?, ?, ?)
            """;

        try (PreparedStatement ps = conn.prepareStatement(insert)) {
            ps.setString(1, LocalDate.now().toString());
            ps.setString(2, "08:00");
            ps.setFloat(3, 90);
            ps.setFloat(4, 120);
            ps.setString(5, "TestPaziente1");
            ps.executeUpdate();
        }
    }

    Map<String, Pasto> result = patientDao.somministrazioneTabella("TestPaziente1");

    assertNotNull(result);
    assertEquals(1, result.size());
    assertTrue(result.containsKey("08:00"));
    assertEquals("Colazione", result.get("08:00").getPasto());
}

// ====================== addSomministrazione ======================

@Test
void testAddSomministrazione() {
    Day day = new Day();

    day.addPasto(new Pasto("Colazione", "08:00", 100, 130));
    day.addPasto(new Pasto("Pranzo", "13:00", 110, 140));

    boolean result = patientDao.addSomministrazione(day, "TestPaziente1");

    assertTrue(result);
}

// ====================== checkSomministrazione ======================

@Test
void testCheckSomministrazione() throws SQLException {
    DateTimeFormatter formatter = DateTimeFormatter.ofPattern("yyyy-MM-dd");

    try (Connection conn = DBConnection.getConnection()) {
        String insert = """
            INSERT INTO rilevazioni_giornaliere
            (data_rilevazione, orario, username)
            VALUES (?, ?, ?)
            """;
        try (PreparedStatement ps = conn.prepareStatement(insert)) {
            ps.setString(1, LocalDate.now().format(formatter));
```

```
            ps.setString(2, "08:00");
            ps.setString(3, "TestPaziente1");
            ps.executeUpdate();
        }
    }

    boolean result = patientDao.checkSomministarazione(
            "08:00",
            LocalDate.now(),
            formatter,
            "TestPaziente1"
    );

    assertFalse(result);
}

// ====================== messageSomm ======================

@Test
void testMessageSomm() throws SQLException {
    patientDao.messageSomm("Test messaggio", "TestMedico1", "TestPaziente1");

    try (Connection conn = DBConnection.getConnection()) {
        var rs = conn.prepareStatement(
                "SELECT COUNT(*) FROM messages WHERE content = 'Test messaggio'"
        ).executeQuery();
        rs.next();

        assertEquals(1, rs.getInt(1));
    }
}

// ====================== recuperoNotifica ======================

@Test
void testRecuperoNotifica() throws SQLException {
    try (Connection conn = DBConnection.getConnection()) {
        String insert = """
            INSERT INTO messages(sender, receiver, content, visualizzato)
            VALUES (?, ?, ?, false)
            """;
        try (PreparedStatement ps = conn.prepareStatement(insert)) {
            ps.setString(1, "TestMedico1");
            ps.setString(2, "TestPaziente1");
            ps.setString(3, "Messaggio");
            ps.executeUpdate();
        }
    }

    boolean result = patientDao.recuperoNotifica("TestPaziente1");

    assertTrue(result);
}

// ====================== cambioVisualizzato ======================

@Test
void testCambioVisualizzato() throws SQLException {
    try (Connection conn = DBConnection.getConnection()) {
        conn.prepareStatement("""
```

```
            INSERT INTO messages(sender, receiver, content, visualizzato)
            VALUES ('TestMedico1','TestPaziente1','msg',false)
        """).executeUpdate();
    }

    patientDao.cambioVisualizzato("TestMedico1", "TestPaziente1");

    try (Connection conn = DBConnection.getConnection()) {
        var rs = conn.prepareStatement("""
            SELECT visualizzato FROM messages
        """).executeQuery();
        rs.next();

        assertTrue(rs.getBoolean(1));
    }
}

// ==================== cercoSintomi ====================

@Test
void testCercoSintomi() throws SQLException {
    DateTimeFormatter formatter = DateTimeFormatter.ofPattern("yyyy-MM-dd");

    try (Connection conn = DBConnection.getConnection()) {
        String sql = "INSERT INTO rilevazioni_giornaliere " +
                "(username, data_rilevazione, note_rilevazione) " +
                "VALUES (?, ?, ?)";

        try (PreparedStatement ps = conn.prepareStatement(sql)) {
            ps.setString(1, "TestPaziente1");
            ps.setString(2, LocalDate.now().toString());
            ps.setString(3, "note...");
            ps.executeUpdate();
        }

    }

    boolean result = patientDao.cercoSintomi(
            "Nuova nota",
            "TestPaziente1",
            LocalDate.now(),
            formatter
    );

    assertTrue(result);

    boolean result2 = patientDao.cercoSintomi(
            "Nuova nota",
            "TestPaziente1",
            LocalDate.now(),
            formatter
    );

    assertFalse(result2);
}

// ==================== CLEANUP ====================

//@AfterEach
void resetDatabase() throws SQLException {
```

```
        try (Connection conn = DBConnection.getConnection()) {
            conn.prepareStatement("DELETE FROM messages").executeUpdate();
            conn.prepareStatement("DELETE FROM rilevazioni_giornaliere").executeUpdate();
            conn.prepareStatement("DELETE FROM utenti").executeUpdate();
        }
    }
}
```

Cosa fa questo test :

Questo test si occupa di verificare che a qualsiasi utente che interagisce con la sua pagina non vengono mostrati dati errati , di conseguenza si occupa della raccolta dati (come ad esempio le somministrazioni) presenti nel database

Allegati all'interno della cartella **Test**, sono presenti altri tre esempi di **unitTest**

Test degli sviluppatori

Durante questa fase, gli sviluppatori hanno inserito nel sistema sia input corretti sia input errati, con l'obiettivo di verificare che il comportamento del software fosse quello atteso in tutte le situazioni. In particolare, sono stati eseguiti i seguenti test :

- **Verifica dell'autenticazione**: con credenziali errate l'accesso viene respinto; con credenziali corrette l'utente accede alla schermata associata al suo ruolo (admin, medico o paziente).
- **Admin con database vuoto**: è stato provato ad utilizzare i pulsanti di gestione utenti (aggiunta, eliminazione, aggiornamento) senza alcun dato registrato, per controllare che non si verificassero eccezioni o crash (ad eccezione dell'utente Admin che deve esistere per eseguire l'accesso al pannello amministratore) .
- **Medico con database vuoto**: analogamente, sono stati provati i comandi per la visualizzazione/gestione pazienti in assenza di dati, verificando che l'applicazione mostrasse messaggi corretti invece di errori.
- **Inserimento nuovo utente**: controllato che, dopo la creazione da parte dell'admin, l'utente sia effettivamente visibile nella tabella e che rispetti i vincoli previsti (es. il campo *medico* è nullo se si tratta di admin/medico).
- **Vincoli di visibilità**: verificato che il medico possa visualizzare solo i propri pazienti, mentre l'admin può vedere e gestire tutti gli utenti.
- **Validazione input**: sono stati testati input errati o malformati (campi vuoti, username non validi, password troppo deboli, stringhe con caratteri speciali non consentiti) per controllare che il sistema risponda con messaggi di errore adeguati.
- **Gestione update**: controllato che non sia possibile aggiornare un utente lasciando tutti i campi vuoti, e che i vincoli di formato siano rispettati anche in fase di modifica.
- **Gestione duplicati**: tentato l'inserimento di utenti con lo stesso username per verificare che il sistema mostri un errore e impedisca la creazione.

Test utente generico

Infine, il software è stato sottoposto a prova da parte di utenti esterni, non coinvolti nello sviluppo e con competenze informatiche limitate. L'obiettivo era osservare l'interazione naturale con l'applicazione, senza influenzare in alcun modo il comportamento dei tester.

Gli utenti hanno ricevuto soltanto una breve descrizione degli scopi del sistema (gestione utenti e relative funzionalità), senza ulteriori istruzioni sul funzionamento. Sono stati lasciati liberi di esplorare l'interfaccia e di utilizzare i comandi disponibili, intervenendo solo per rispondere a domande dirette.

Questo approccio ha permesso di :

- **Individuare errori o incoerenze** non facilmente visibili durante lo sviluppo.
- **Valutare l'usabilità** del sistema, osservando quanto l'interfaccia fosse chiara e intuitiva senza spiegazioni preliminari.
- **Raccogliere suggerimenti di miglioramento**: alcuni utenti hanno proposto nuove funzionalità o modifiche che potrebbero semplificare l'esperienza d'uso.