Progetto di ingegneria del software

Glicocare

UniVR - Dipartimento di Informatica

A.A. 2025/26

Marco Broccolato – VR501013

Mattia Nicolis – VR500356

Simone Pedretti – VR500040

Indice

1

Specifiche e analisi dei requisiti

Il sistema proposto consente l'accesso a personale medico e pazienti affetti da diabete.
Gli utenti sono classificati in due ruoli distinti: **Diabetologo** e **Paziente**.
L'accesso al sistema avviene tramite autenticazione mediante credenziali predefinite, fornite dagli amministratori di sistema, che determinano il caricamento della schermata iniziale associata al ruolo dell'utente autenticato.

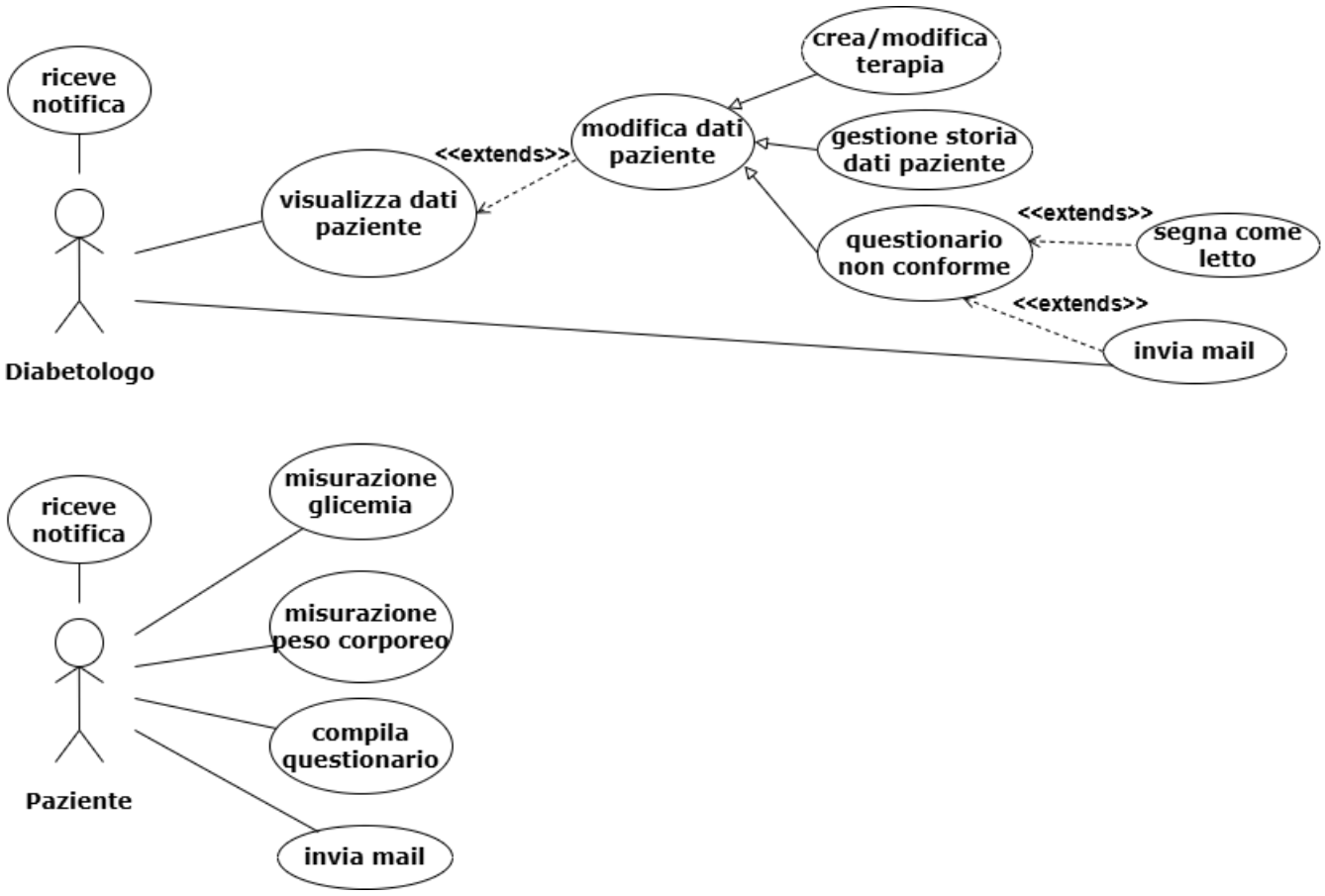


Figura 1.1: Casi d'uso - diabetologo e paziente

1.1 Use case

1.1.1 Diabetologo

A seguito di una corretta autenticazione, il diabetologo accede alla propria home page, dalla quale può visualizzare i propri dati anagrafici e la lista dei pazienti affetti da diabete.

UC_d – Visualizza/Modifica dati paziente

Attori: diabetologo

Precondizioni: diabetologo ha effettuato correttamente l'autenticazione al sistema

Flusso principale:
- ▷ diabetologo accede al sistema
- ▷ sistema mostra l'area riservata del diabetologo
- ▷ diabetologo seleziona un paziente
- ▷ sistema consente al diabetologo di:
  - ◇ creare, modificare o eliminare una terapia
  - ◇ visualizzare l'andamento della glicemia e del peso corporeo
  - ◇ aggiungere o rimuovere dati dallo storico clinico del paziente
  - ◇ visualizzare lo storico dei questionari compilati
- ▷ diabetologo visualizza i questionari non conformi
- ▷ diabetologo visualizza i pazienti con valori glicemici fuori dal range

Postcondizioni: dati visualizzati o modificati risultano aggiornati nel sistema

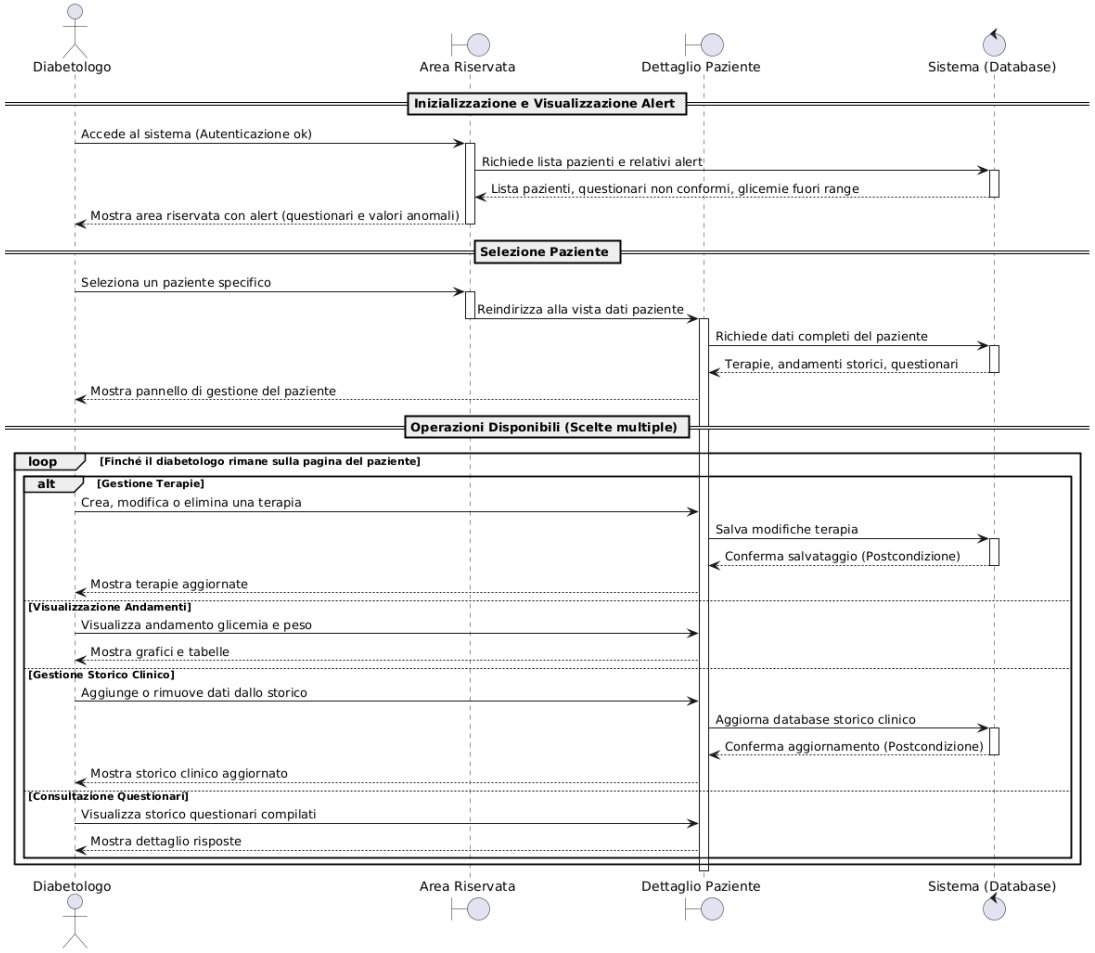


Figura 1.2: Diagramma di sequenza - Visualizza/Modifica dati paziente

UC_d - Ricevi notifica

Attori: diabetologo

Precondizioni: diabetologo ha effettuto correttamente l'auteniticazione al sistema

Flusso principale:

- ▷ diabetologo accede al sistema
- ▷ sistema mostra l'area riservata del diabetologo
- ▷ diabetologo clicca sul bottone "... Mail"
- ▷ sistema mostra l'area riservata alla mail
- ▷ sistema consente al diabetologo di:
  - ◇ inviare o rispondere a una mail
  - ◇ inviare una mail al paziente che ha compilato in maniera errata il questionario

Postcondizioni: notifica letta e/o comunicazione inviata

1.1.2 Paziente

A seguito di una corretta autenticazione, il paziente accede alla propria home page, dalla quale può visualizzare i propri dati anagrafici, i grafici relativi all'andamento della glicemia e del peso corporeo, le terapie assegnate e lo storico dei questionari compilati.

UC_p - Misurazione glicemia/peso corporeo

Attori: paziente

Precondizioni: paziente ha effettuto correttamente l'autenticazione al sistema

Flusso principale:

- ▷ paziente accede al sistema
- ▷ sistema mostra l'area riservata del paziente
- ▷ paziente inserisce il valore della glicemia, il momento della misurazione e l'ora / del proprio peso corporeo

Postcondizioni: visualizzazione del valore inserito nel grafico

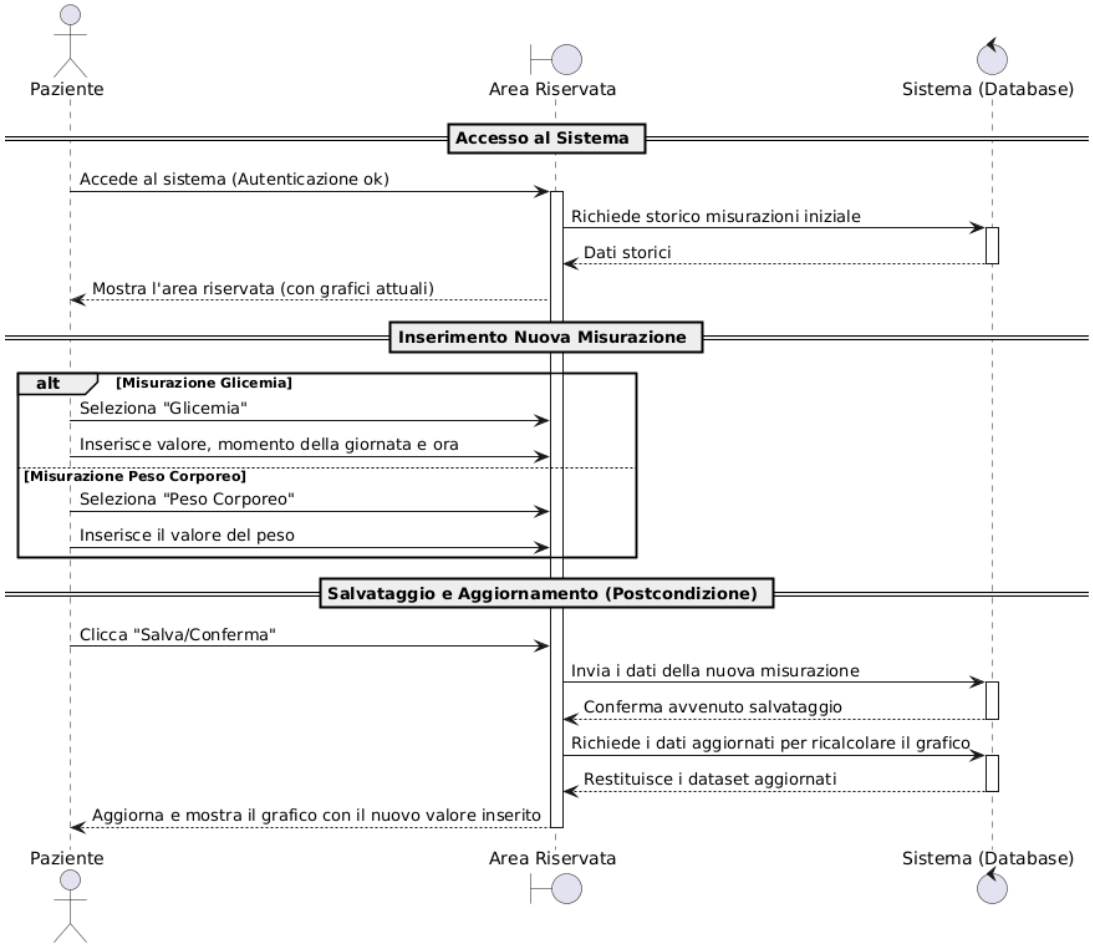


Figura 1.3: Diagramma di sequenza - Misurazioni del paziente

UC_p - Compilazone questionario

Attori: paziente
Precondizioni: paziente ha effettuto correttamente l'auteniticazione al sistema
Flusso principale:
- ▷ paziente accede al sistema
- ▷ sistema mostra l'area riservata del paziente
- ▷ paziente clicca il bottone "Compila questionario"
- ▷ sistema mostra l'area riservata alla compilazione del questionario
- ▷ paziente inserisce i dati della terapia nei campi appositi

Postcondizioni: questionario inserito nella lista dei compilati

UC_p - Riceve notifica

Attori: paziente
Precondizioni: paziente ha effettuto correttamente l'auteniticazione al sistema
Flusso principale:
- ▷ paziente accede al sistema
- ▷ sistema mostra l'area riservata del paziente
- ▷ paziente riceve una notifica se ha una o più terapie in corso (da iniziare)

Postcondizioni: notifica letta e/o comunicazione inviata

UC_p - Invia mail

Attori: paziente
Precondizioni: paziente ha effettuto correttamente l'auteniticazione al sistema
Flusso principale:
- ▷ paziente accede al sistema
- ▷ sistema mostra l'area riservata del paziente
- ▷ sistema mostra l'area riservata alla mail
- ▷ sistema consente al diabetologo di:
  - ◇ inviare o rispondere a una mail

Postcondizioni: comunicazione inviata

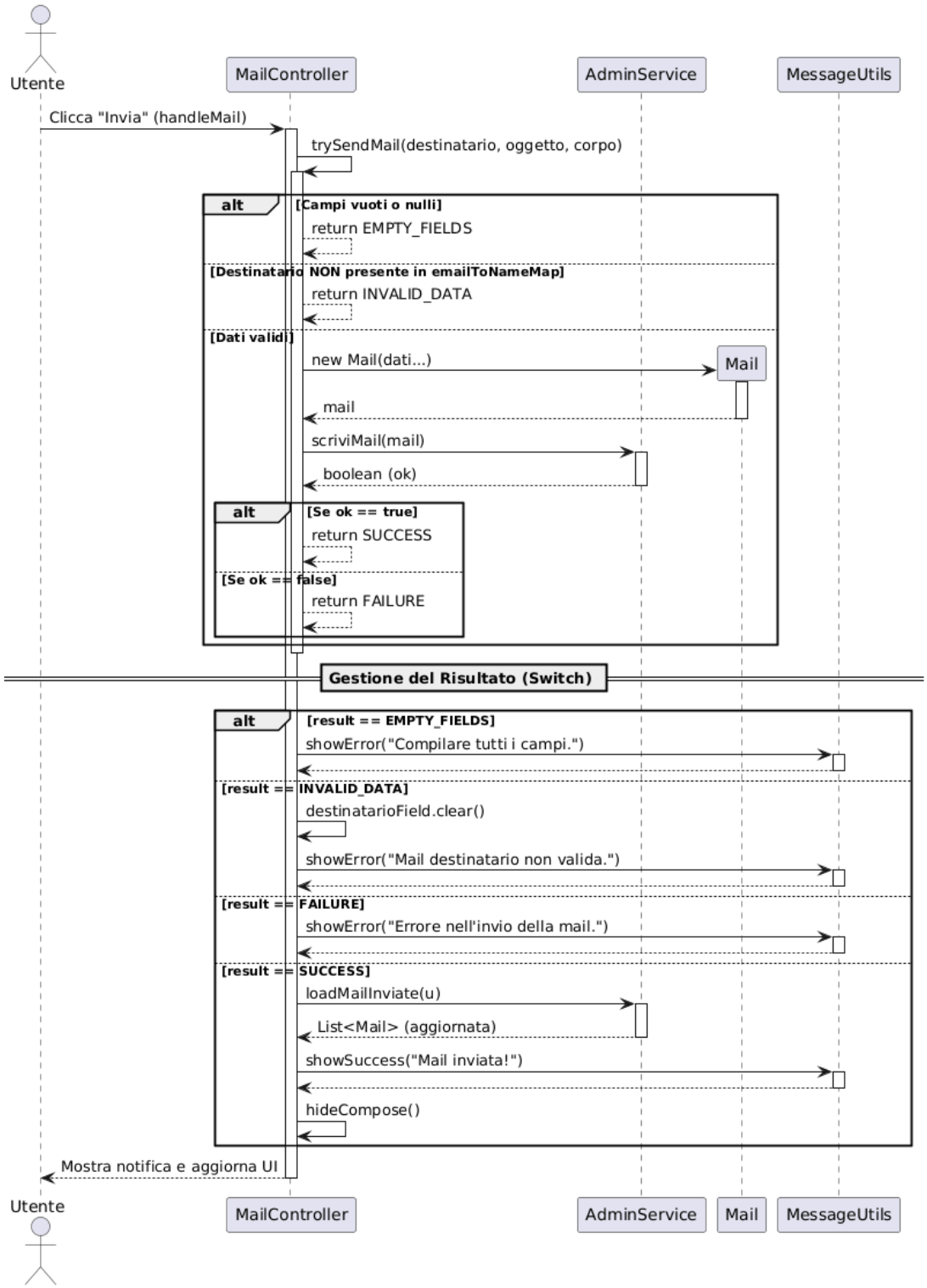


Figura 1.4: Diagramma di sequenza - Invio mail

1.2 Activity diagram

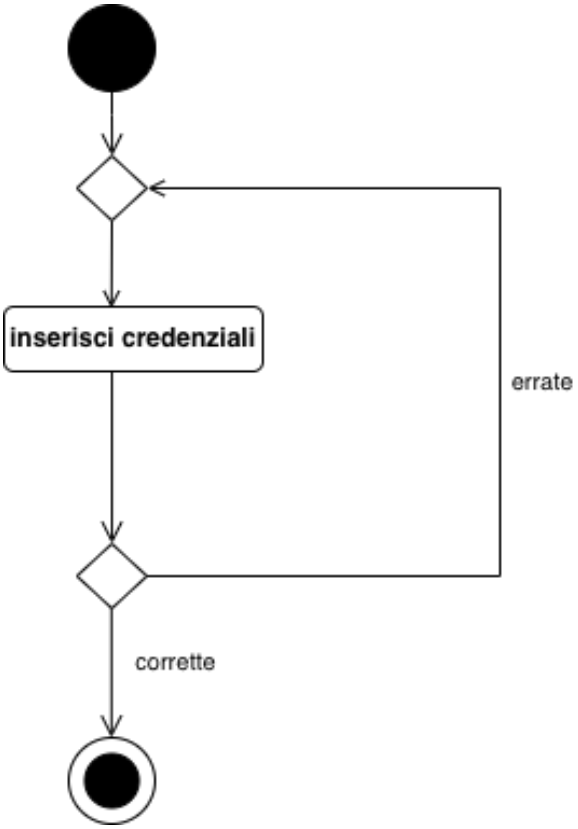


Figura 1.5: Autenticazione

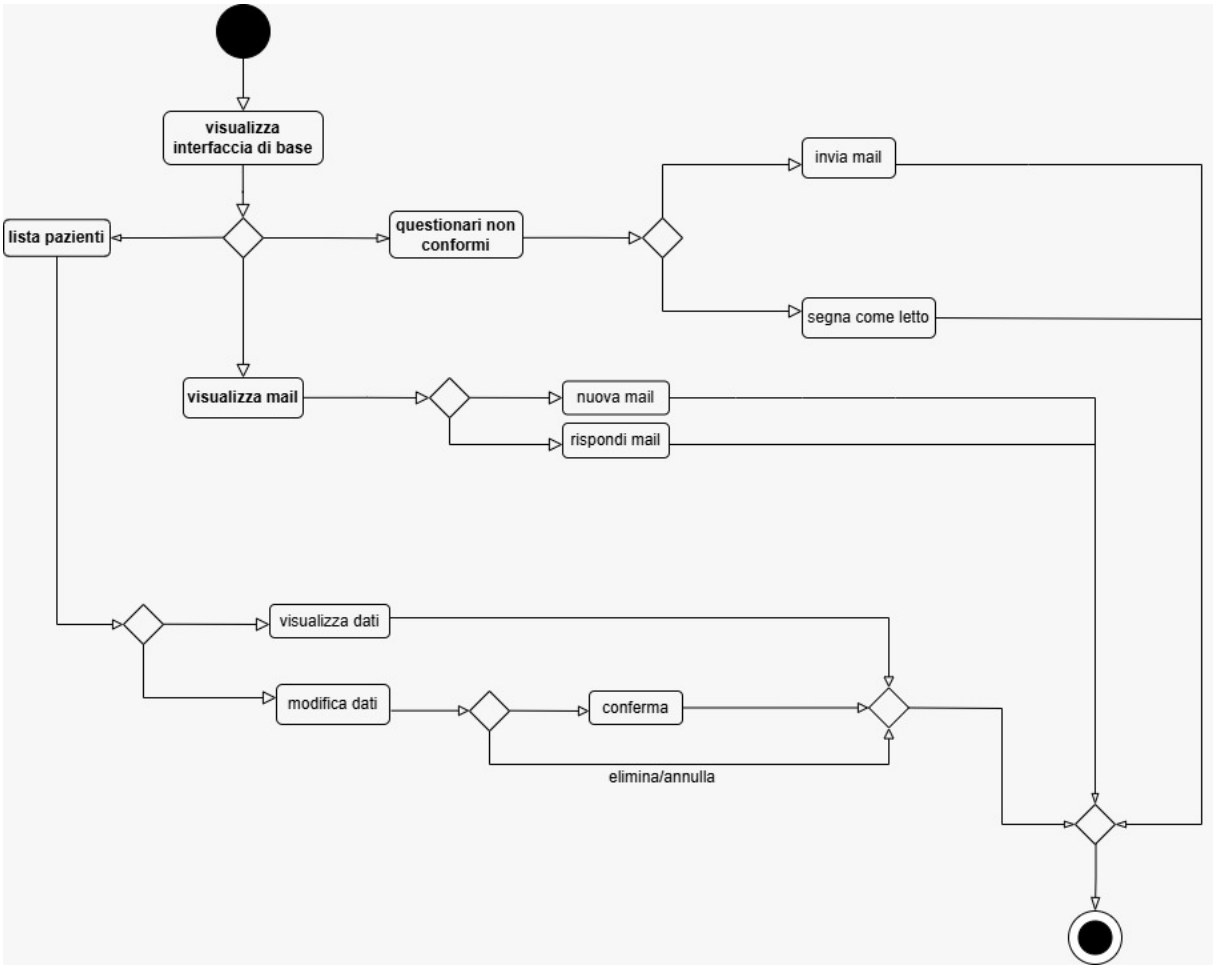


Figura 1.6: Diabetologo

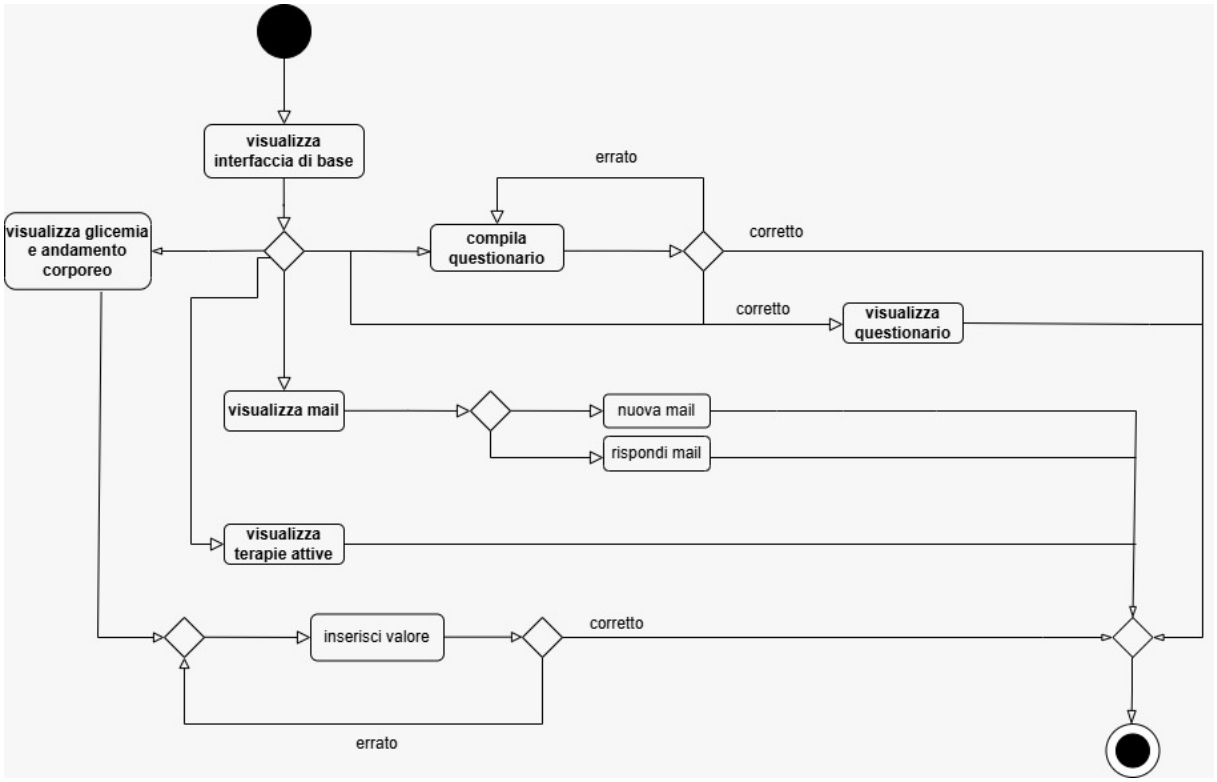


Figura 1.7: Paziente

1.3 Diagrammi UML

1.3.1 Classi

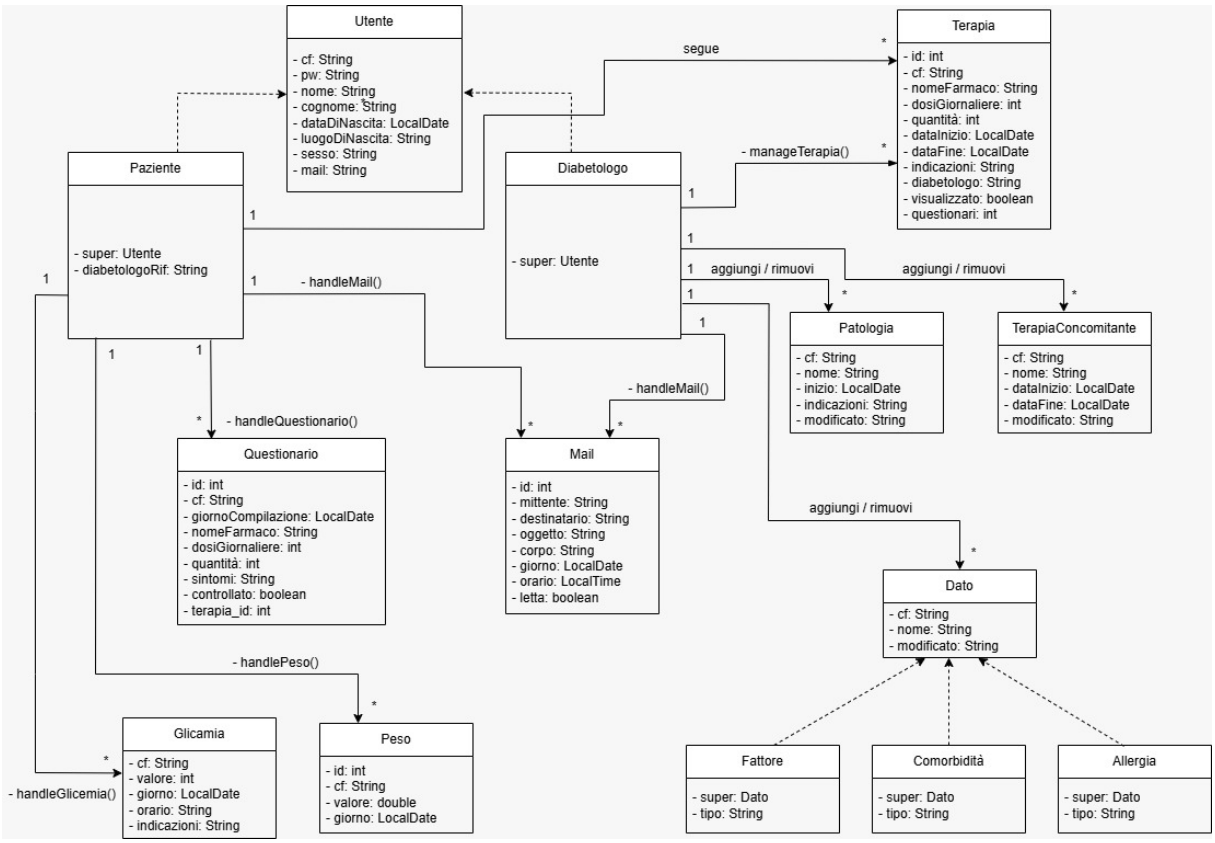
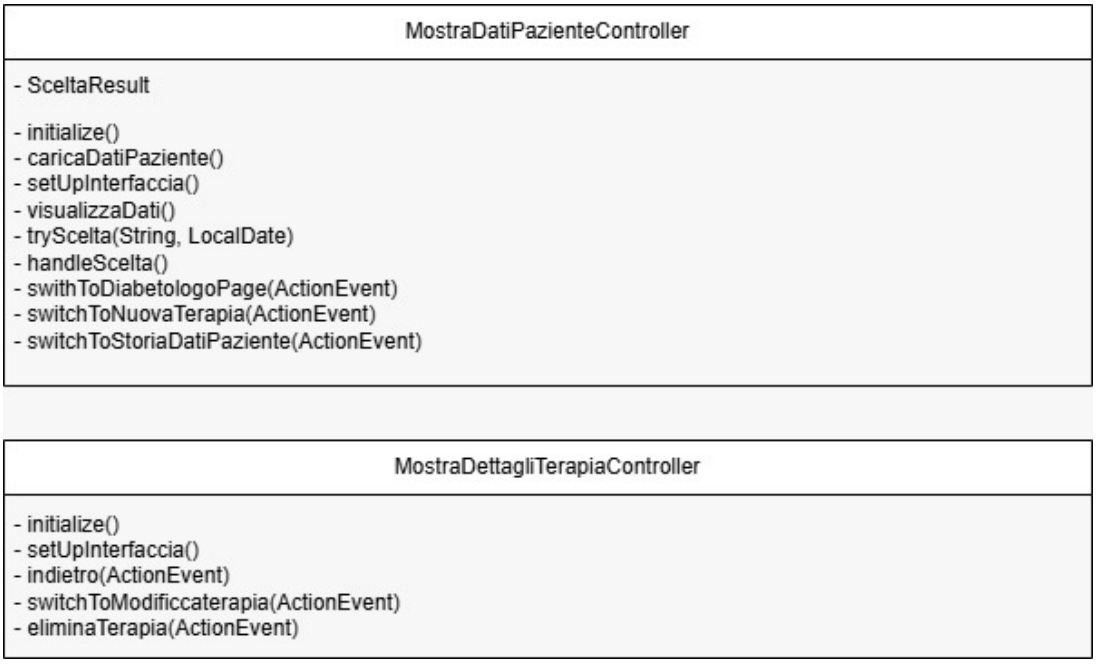


Figura 1.8: Diagramma UML delle classi

1.3.2 Controller



DiabetologoController
- caricaDatiDiabetologo() - setUpInterfaccia() - setupListaPazienti() - setupListaGlicemieSballate() - setupListaQiestionariNonConformi() - switchToLogin(ActionEvent) - switchToMailPage(ActionEvent)

LoginController
- LoginResult - initialize() - tryLogin(String, String) - handleLogin(ActionEvent)

MailController
- MailResult - inizialize() - caricaDati() - populateNameMapP(List<Paziente>) - populateNameMapD(List<Diabetologo>) - setupInterface() - trySendMail(String, String, String) - handleMail() - showMailRicevute(ActionEvent) - showMailInviate(ActionEvent) - setuopListView(List<Mail>, boolean) - updateFilter(String) + showCompose() - hideCompose() + rispondi(String, String) - indietro(ActionEvent)

TerapiaConcomitanteController
- initialize() - switchToMostraDatiPaziente(ActionEvent)

PazienteController
- GlicemiaResult - PesoResult - initialize() - caricaDatiPaziente() - setUpInterfaccia() - notificaTerapia() - visualizzaGraficoGlicemia(int) - visualizzaGraficoPeso() - setUpTerapieInCorso() - setUpQuestionari() - setUpCompilazioneQuest() - tryCreateGlicemia(String, String, String, String) - handleGlicemia (ActionEvent) - tryCreatePeso(String, boolean) - handlePeso(ActionEvent) - switchToLogin(ActionEvent) - switchToMailPage(ActionEvent) - switchToQuestionarioPage(ActionEvent)

Figura 1.9: Classi UML dei Controller

1.4 Diagrammi di sequenza del software

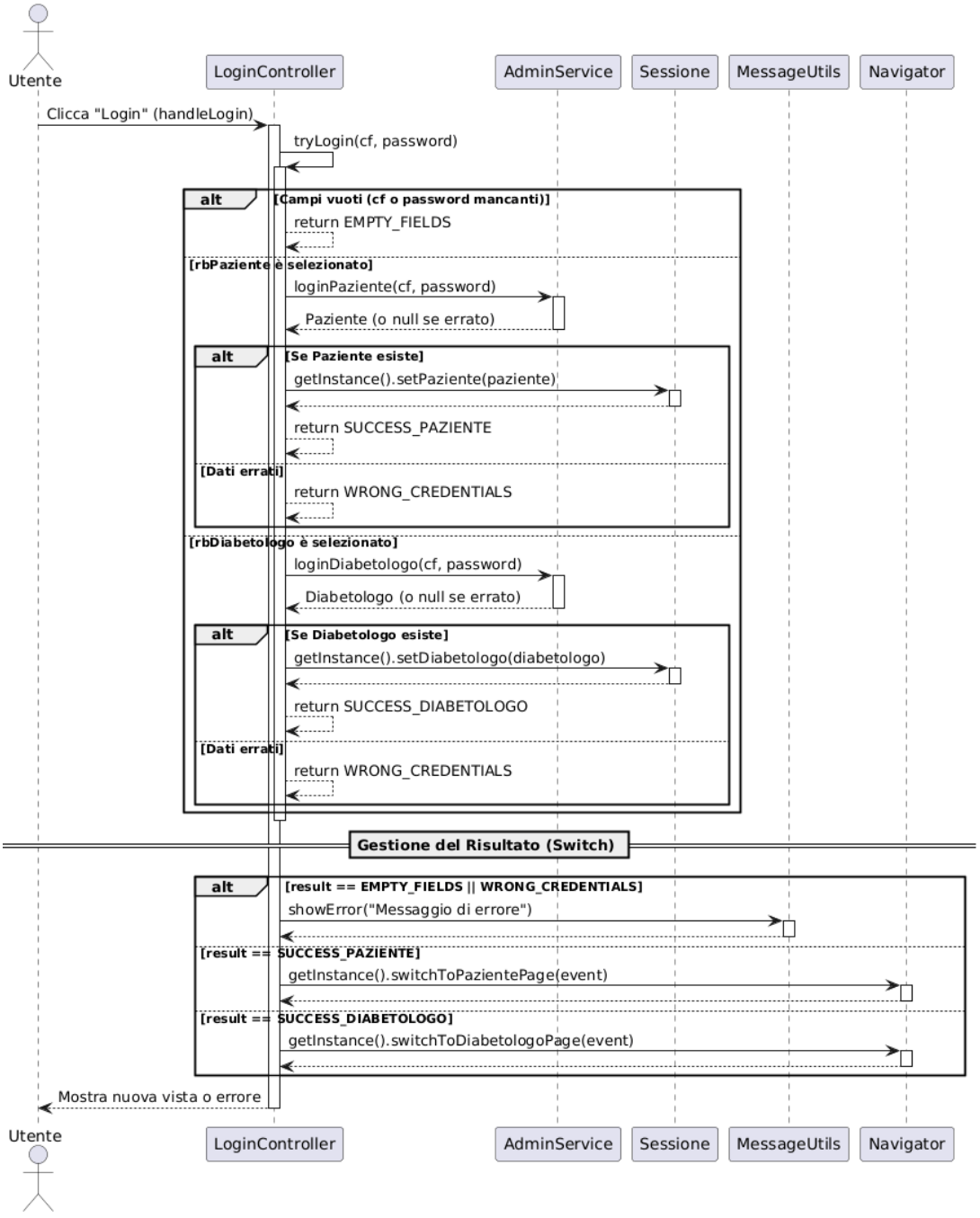


Figura 1.10: Fase di autenticazione

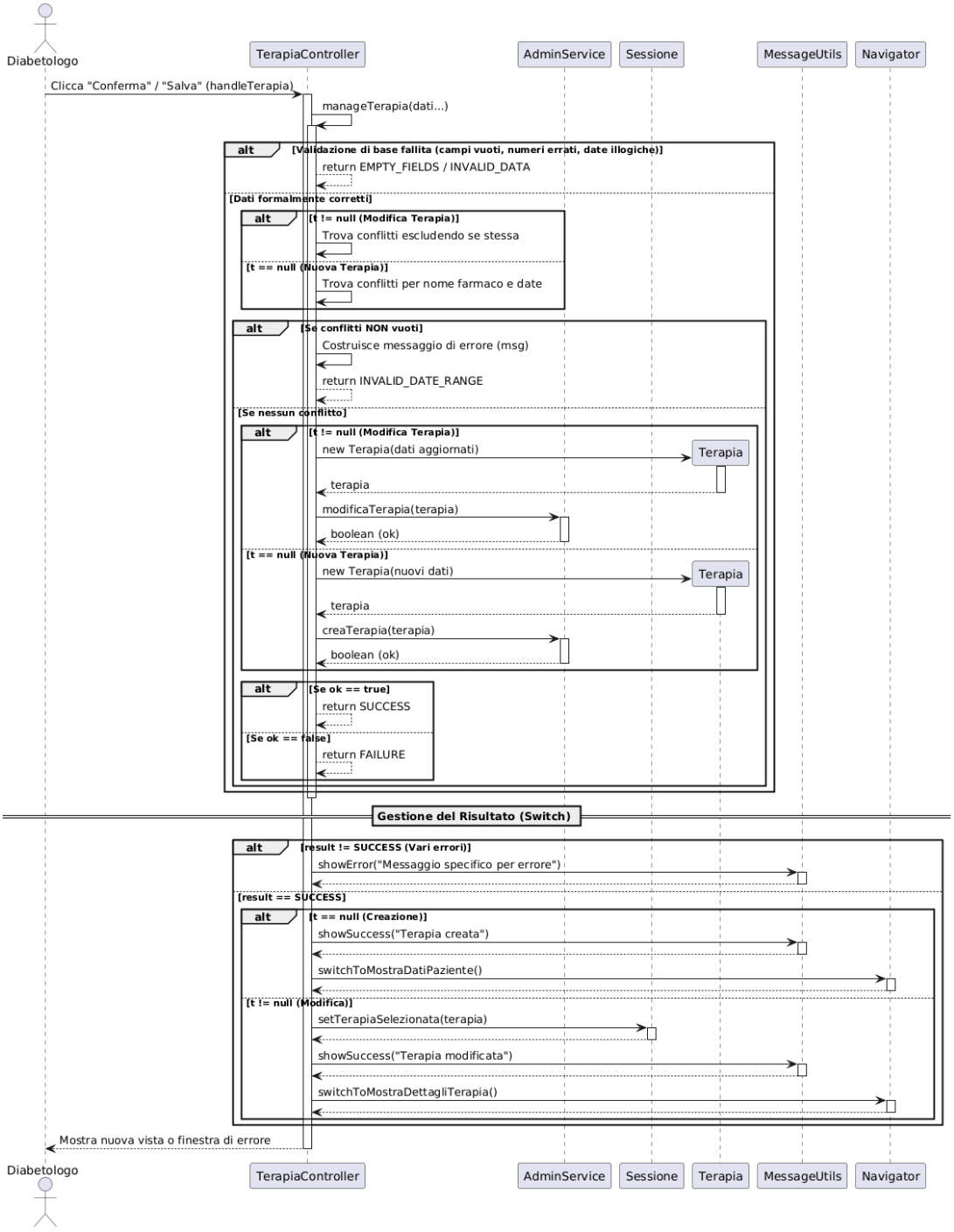


Figura 1.11: Gestione della terapia

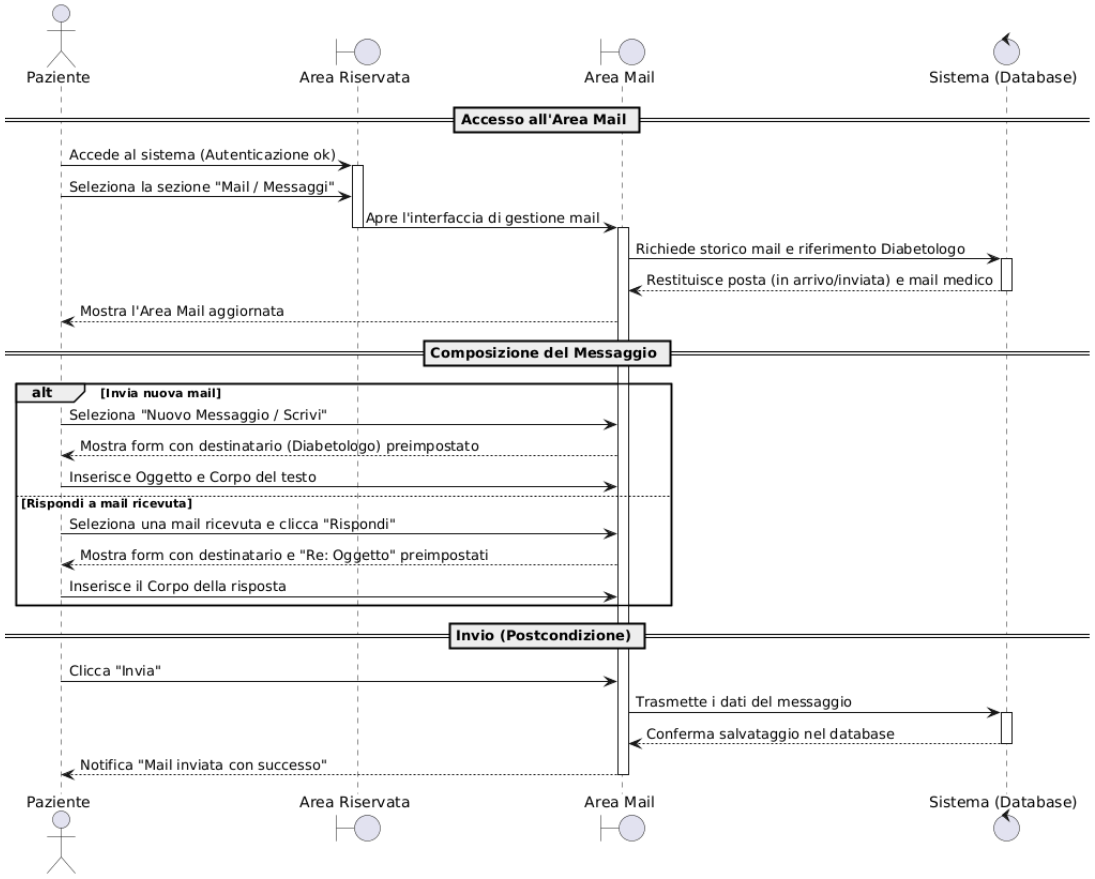


Figura 1.12: Gestione dell'invio di una mail

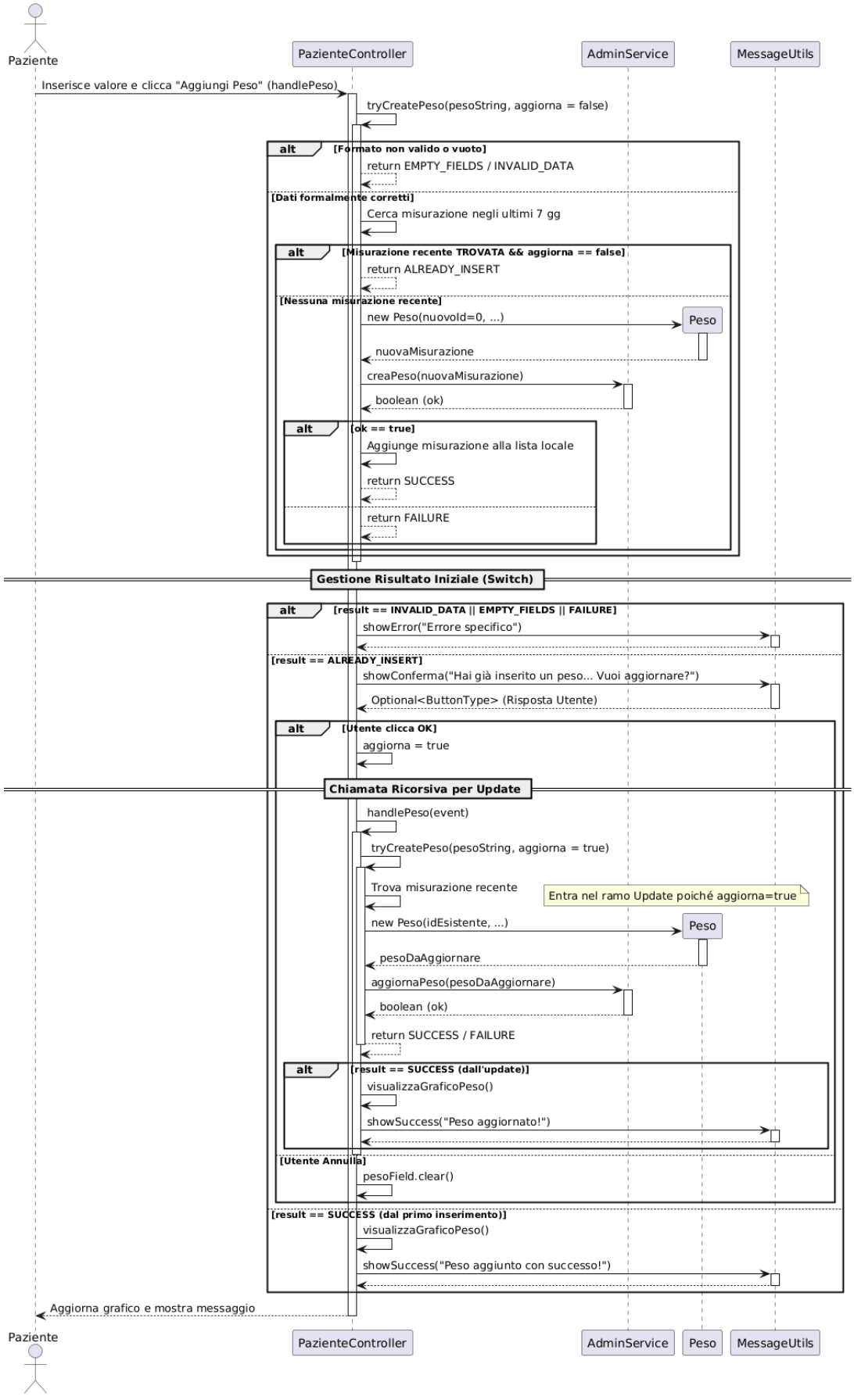


Figura 1.13: Pagina paziente

2

Processo di sviluppo e pattern utilizzati

2.1 Metodologia di sviluppo

In fase preliminare è stata condotta un'approfondita analisi dei requisiti, volta a identificare e formalizzare le funzionalità principali (*core*) del progetto.

Il processo di sviluppo del software ha poi seguito un approccio metodologico **agile e incrementale**. Nonostante la natura iterativa del processo, all'interno di ogni ciclo di sviluppo è stata mantenuta una rigorosa linearità operativa articolata nelle fasi di *progettazione*, *implementazione* e *validazione*.

Nello specifico, l'implementazione di ogni singolo caso d'uso (*use-case*) è stata seguita da una fase di verifica puntuale (testing). In caso di esito negativo dei test, si è proceduto con attività di *debugging* e *refactoring*, iterando il processo fino al raggiungimento della stabilità e della conformità ai requisiti funzionali prefissati.

A conclusione del ciclo di sviluppo, l'attività si è focalizzata sulla stesura della documentazione tecnica definitiva, arricchita dalla modellazione UML tramite diagrammi di attività e di sequenza, per rappresentare dettagliatamente il flusso logico del software.

2.2 Pattern Architetturali e Design Pattern

L'architettura del sistema è stata definita seguendo pattern consolidati nell'ingegneria del software, al fine di garantire modularità, manutenibilità e una chiara separazione delle responsabilità.

2.2.1 Model-View-Controller (MVC)

A livello macroscopico, il progetto adotta il pattern architetturale **MVC**. Questo ha permesso di disaccoppiare la logica di presentazione dalla logica di business e dai dati:

- **Model:** Rappresenta lo stato dell'applicazione.
- **View:** Si occupa della visualizzazione dei dati e dell'interazione con l'utente.

- **Controller:** Gestisce il flusso delle operazioni, ricevendo gli input dalla View e aggiornando il Model di conseguenza.

2.2.2 Data Access Object (DAO)

Per la gestione della persistenza è stato implementato il pattern **DAO**. Questo pattern astrae e incapsula tutti gli accessi alla base di dati (CRUD), nascondendo i dettagli complessi delle query SQL al resto dell'applicazione. Grazie ai DAO, la logica dei controllori non dipende direttamente dal database, facilitando eventuali migrazioni o modifiche alla struttura dati.

2.2.3 Singleton

Il pattern **Singleton** è stato utilizzato per garantire l'univocità di determinate istanze critiche all'interno dell'applicazione.

- Un esempio è la classe `Sessione.java`: il Singleton assicura che esista una sola istanza attiva relativa all'utente corrente, prevenendo conflitti di accesso ai dati di sessione.
- Un altro esempio è la classe `Navigator.java`: utilizzata per centralizzare la logica di navigazione, permettendo il cambio controllato delle viste (pagine) all'interno dell'interfaccia utente.

2.2.4 Service Layer

Per orchestrare le operazioni tra il Controller e il livello di persistenza (DAO), è stato introdotto un **Service Layer** (rappresentato dalla classe `AdminService`). Questa classe agisce come un punto di ingresso unificato per la logica del software: riceve le richieste dai controller, applica le regole di dominio necessarie e coordina le chiamate ai vari DAO. Sebbene tecnicamente definito come *Service Layer*, questo componente applica il principio del **Facade Pattern**, semplificando l'interfaccia verso i sottosistemi di dati e nascondendo la complessità delle transazioni sottostanti.

3

Test

In linea con la metodologia agile adottata, l'attività di verifica non è stata relegata alla sola fase finale, ma è stata integrata durante tutto il processo di sviluppo.

La strategia principale adottata è stata quella dello **Unit Testing**. Questo approccio consiste nell'isolare e testare singole porzioni di codice (solitamente metodi o classi) per accertarsi che funzionino correttamente in autonomia, indipendentemente dal resto del sistema.

Per l'implementazione automatizzata dei test è stato utilizzato il framework **JUnit**, lo standard industriale per l'ecosistema Java.

3.1 Test degli sviluppatori

3.1.1 Test della Logica

In questa fase sono stati eseguiti test mirati a validare la logica applicativa, sottoponendo il sistema a diverse tipologie di input (validi, errati e limiti) per verificare la conformità della risposta rispetto al comportamento atteso.

Tra i principali scenari di test effettuati si evidenziano:

- **Meccanismo di autenticazione:** Verifica della sicurezza in fase di login. È stato accertato che l'inserimento di credenziali errate o non presenti nel database comporti il blocco dell'accesso, reindirizzando l'utente alla schermata di errore appropriata.

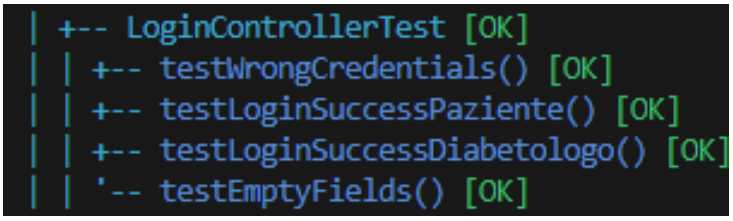


Figura 3.1: Test di autenticazione

```
@BeforeEach
void setup() {
    AdminService.setPazienteDAO(new MockPazienteDAO());
    AdminService.setDiabetologoDAO(new MockDiabetologoDAO());
    controller = new LoginController();
}

@AfterEach
void tearDown() {
    AdminService.setPazienteDAO(new PazienteDAO());
    AdminService.setDiabetologoDAO(new DiabetologoDAO());
}
```

Figura 3.2: Impostazione e "distruzione del DAO fittizio"

- **Dashboard del Diabetologo:** Verifica del corretto recupero dei dati dal database e del popolamento dinamico delle liste pazienti e delle metriche nella Home Page dedicata al medico.

```
| +-- DiabetologoPageTest [OK]
| | +-- Giorni di non compilazione < 3 [OK]
| | +-- Pre-Pasto: Deve essere NULL (Normale) se 100 [OK]
| | +-- Pre-Pasto: Deve essere ARANCIONE se 145 (Attenzione) [OK]
| | +-- Pre-Pasto: Deve essere ROSSO se > 150 (Iperglicemia Grave) [OK]
| | +-- Post-Pasto: Normale a 160 [OK]
| | +-- Post-Pasto: Soglie piu' alte (Rosso > 200) [OK]
| | +-- Pre-Pasto: Deve essere ROSSO se < 60 (Ipoglicemia Grave) [OK]
| | '-- Giorni di non compilazione 3 + [OK]
```

Figura 3.3: Test della pagina del diabetologo

```
// --- TEST CONTA GIORNI DI NON COMPILAZIONE
@Test
@DisplayName("Giorni di non compilazione < 3")
void testSituazioneRegolare() throws Exception {
    int giorni = 0;
    Paziente p = new Paziente(cf: "a", pw: null, nome: null, cognome: null, dataDiNascita: null, luogoDiNas… null, null, null, null);

    LocalDate dataInizio = LocalDate.now().minusDays(daysToSubtract: 2);
    LocalDate dataFine = LocalDate.now().plusDays(daysToAdd: 10);
    Terapia t = new Terapia(id: 0, cf: "a", nomeFarmaco: "testFarmaco", dosiGiornaliere: 1, quantità: 1, dataInizio, dataFine, … "", "", false, 0);
    boolean creata = AdminService.creaTerapia(t);

    LocalDate dataCompilazione = LocalDate.now().minusDays(daysToSubtract: 2);
    Questionario q = new Questionario(id: 0, cf: "a", dataCompilazione, nomeFarmaco: "testFarmaco", dosiGiornaliere: 1, quantità: 1, … null, false, 0);
    boolean creato = AdminService.creaQuestionario(q);
    if(creata && creato) {
        giorni = DiabetologoUtils.calcolaGiorniNonCompilati(p);
    }
    assertEquals(2, giorni);
}
```

Figura 3.4: Test conteggio giorni di non compilazione

- **Sistema di notifica:** Validazione del modulo di comunicazione, con particolare attenzione al corretto invio e alla ricezione delle e-mail di avviso generate dal sistema.

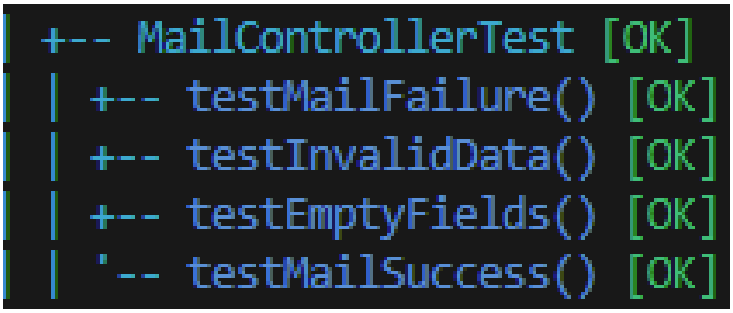


Figura 3.5: Test del sistema di comunicazione

```java
@Test
void testMailSuccess() {
    try {
        // Questo deve restituire SUCCESS
        Object result = invokeTrySendMail(utenteSuccesso.getMail(), oggetto: "Oggetto Ok", corpo: "Tutto ok");
        assertEquals("SUCCESS", result.toString());
    } catch (Exception e) {
        e.printStackTrace();
        fail(e.getMessage());
    }
}
```

Figura 3.6: Test del sistema di comunicazione: "SUCCESSO"

- **Gestione Dati Pazienti:** Test approfonditi sulle operazioni di inserimento, aggiornamento ed eliminazione dati. È stata verificata la robustezza del sistema a fronte di *input* non validi (es. formati errati) o dati mancanti, assicurando che le eccezioni vengano gestite senza causare crash del software.

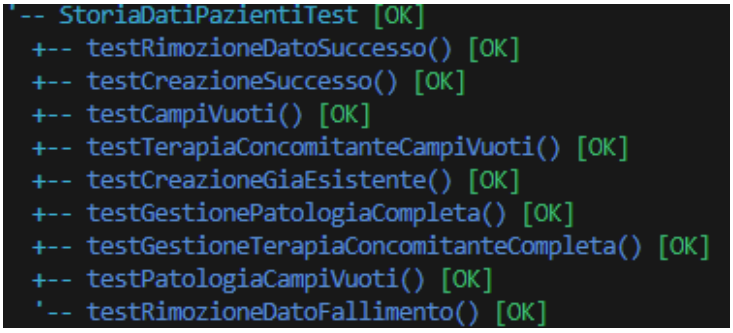


Figura 3.7: Test gestione dati paziente

```java
// --- TEST CASE: CREAZIONE ---

@Test
void testCreazioneSuccesso() throws Exception {
    assertEquals("SUCCESS", invokeTryCreate(tipo: "Allergia", nome: "acari").toString());
    assertEquals("SUCCESS", invokeTryCreate(tipo: "Comorbidità", nome: "Gastrite").toString());
}

    void test.StoriaDatiPazientiTest.testCreazioneGiaEsistente() throws Exception
    Source: Sw_Project_dc167784

@Test
void testCreazioneGiaEsistente() throws Exception {
    assertEquals("DATA_ALREADY_EXISTS", invokeTryCreate(tipo: "Allergia", nome: "polline").toString());
    assertEquals("DATA_ALREADY_EXISTS", invokeTryCreate(tipo: "Fattore Di Rischio", nome: "fumatore").toString());
}

@Test
void testCampiVuoti() throws Exception {
    assertEquals("EMPTY_FIELDS", invokeTryCreate(tipo: "Allergia", nome: "").toString());
    assertEquals("EMPTY_FIELDS", invokeTryCreate(tipo: "Allergia", nome: null).toString());
}
```

Figura 3.8: Test creazione dati paziente

3.1.2 Validazione dell'Interfaccia Grafica

Oltre ai test automatici sulla logica, è stata eseguita un'estesa fase di **testing manuale** sull'interfaccia utente. L'obiettivo di questa fase è stato verificare la robustezza del layer di presentazione (View) e la corretta integrazione con il Controller.

Nello specifico, è stata condotta una verifica esaustiva di tutti gli elementi interattivi (pulsanti, form) per accertarsi che:

- Ogni azione scatenata dall'utente (es. click su un pulsante di conferma) venisse gestita correttamente senza generare errori a runtime o crash dell'applicazione.
- La navigazione tra le diverse schermate avvenisse in modo fluido e coerente, senza "link rotti" o pagine irraggiungibili.
- Il sistema fornisse un feedback visivo adeguato ad ogni interazione (es. messaggi di conferma o alert di errore), garantendo una buona esperienza utente.

3.2 Test utente generico

Verso la fase finale dello sviluppo, il software è stato testato da un utente generico (un familiare esterno al team di sviluppo). L'esito di questa sessione è stato estremamente costruttivo, in quanto ha consentito di risolvere alcune imprecisioni nella gestione della logica di controllo e di affinare ulteriormente l'interazione tra utente e sistema.