

Documentazione progetto di

Ingegneria del Software

FRANCESCO MERIGHI - VR474954
MATTEO ZANDONAI - VR473622

Università degli Studi di Verona
a.a. 2022/2023

Interazioni utente-sistema	3
<u>Specifiche casi d'uso - alcune note generali</u>	<u>3</u>
<u>Casi d'uso: Pazienti</u>	<u>4</u>
<i>Visualizzazione terapia da assumere</i>	4
<i>Visualizzazione contatti medico associato</i>	5
<i>Inserimento rilevazioni</i>	6
<u>Casi d'uso: Medici</u>	<u>9</u>
<i>Visualizzazione lista di tutti i pazienti</i>	9
<i>Visualizzazione dettagli paziente</i>	10
<i>Visualizzazione tabella con rilevazioni</i>	11
<i>Inserimento terapia</i>	14
<u>Activity Diagram</u>	<u>16</u>
Architettura del progetto	19
<u>Processo di sviluppo</u>	<u>19</u>
<u>Progettazione</u>	<u>19</u>
<u>Tecnica di implementazione</u>	<u>20</u>
<u>Gestione dati</u>	<u>21</u>
<u>Pattern architetturali usati</u>	<u>23</u>
<i>MVC (Model View Controller)</i>	23
<i>DAO (Data Access Object)</i>	27
<u>Pattern progettuali usati</u>	<u>28</u>
Factory	28
Singleton	29
<u>Implementazione file di log</u>	<u>30</u>
<u>Grafica e design</u>	<u>31</u>
Test e validazione	33
<u>Verifica correttezza codice, pattern/diagrammi</u>	<u>33</u>
<u>Test sviluppatore</u>	<u>33</u>
<u>Test utente</u>	<u>34</u>

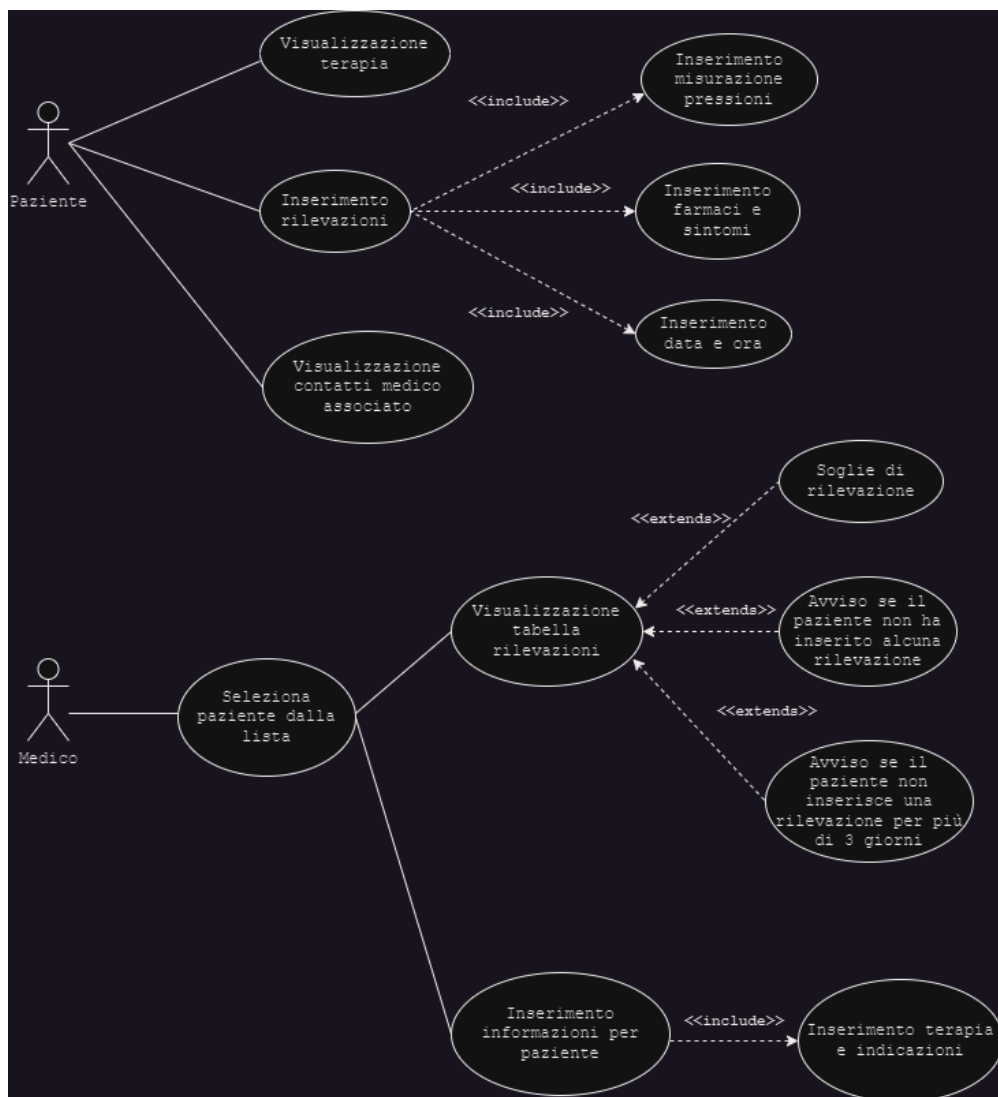
Interazioni utente-sistema

Specifiche casi d'uso - alcune note generali

Il sistema proposto supporta l'utilizzo da parte di **MEDICI** e **PAZIENTI**.

Entrambe le tipologie di utenti hanno a disposizione delle credenziali fornite dagli amministratori di sistema, salvate nelle apposite tabelle del DATABASE, con cui possono effettuare l'accesso al gestionale.

Se l'autenticazione va a buon fine, il medico e/o il paziente vengono reindirizzati alla rispettiva schermata successiva.



Casi d'uso: Pazienti

Dopo aver effettuato correttamente l'accesso, al paziente viene mostrata un'interfaccia che permette di visualizzare:

- la **terapia** personale assegnata da un medico.
- i **contatti** del medico associato.

e che permette di inserire:

- rilevazione delle **pressioni SBP/DBP** giornaliere.
- assunzione dei **farmaci** e relativi **sintomi** in base alla terapia prescritta.

Visualizzazione terapia da assumere

Nell'interfaccia sarà possibile visualizzare una sezione relativa alla **terapia**, prescritta da parte di un medico tra quelli presenti nel sistema (Ogni paziente possiede un medico di riferimento, ma non solamente quel suddetto medico può assegnare terapie e modificare le informazioni del paziente).

ATTORI → Paziente

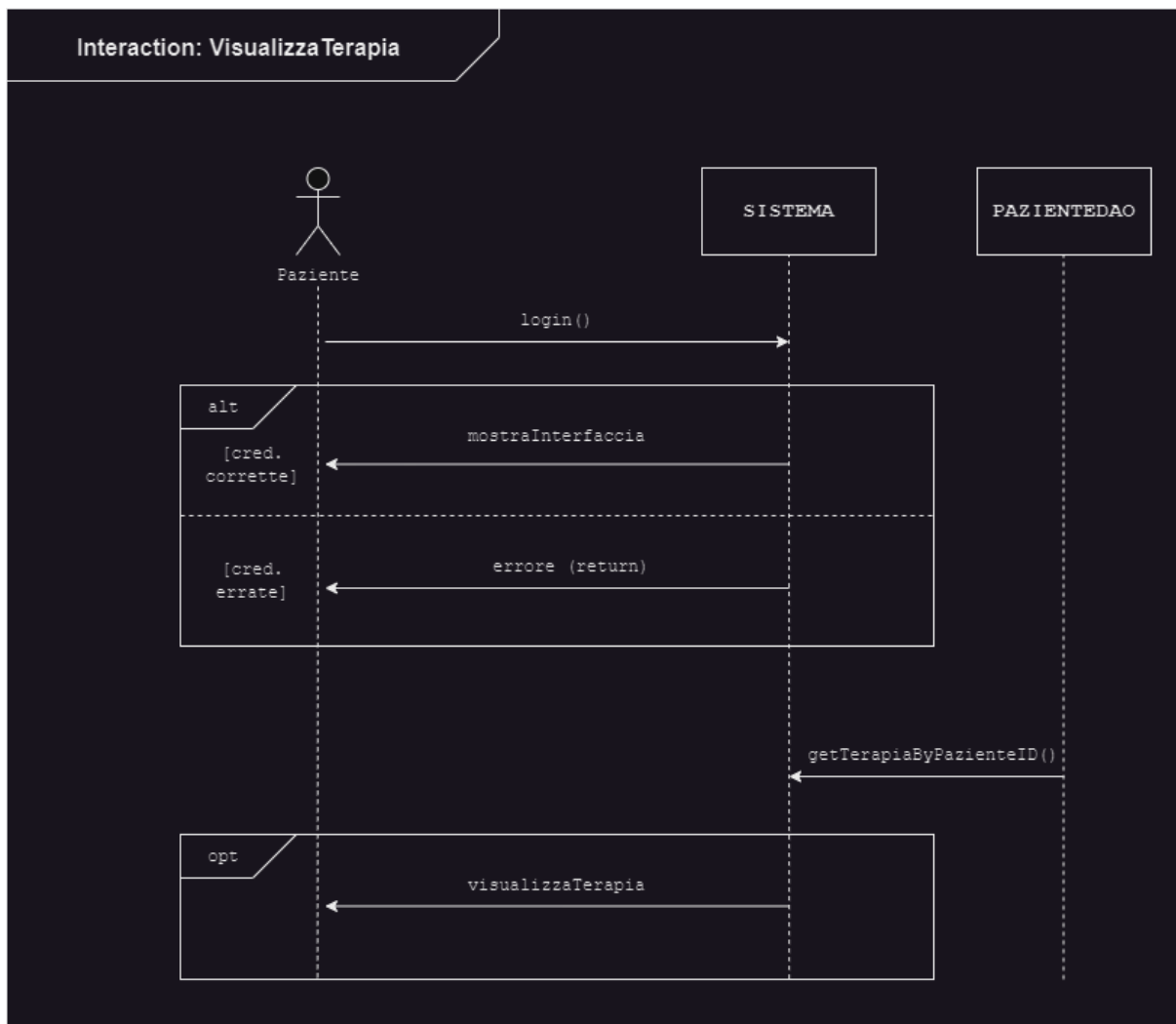
PRE-CONDIZIONI → Il paziente deve aver effettuato l'accesso e deve essergli stata prescritta una terapia

PASSI:

1. Il paziente accede al sistema
2. Il paziente visualizza l'interfaccia dei dettagli
3. Il paziente visualizza la terapia personale prescritta

POST-CONDIZIONI → Nessuna

Può capitare, che il paziente esegua l'accesso, ma che nessuna terapia gli sia stata assegnata, in questo caso viene notificato con un **avviso** (alert), che gli comunica di non aver assegnato nessuna terapia e, se necessario, di contattare il medico di riferimento.



Visualizzazione contatti medico associato

Nell'interfaccia sarà possibile visualizzare una sezione relativa ai **contatti** inerenti al medico associato.

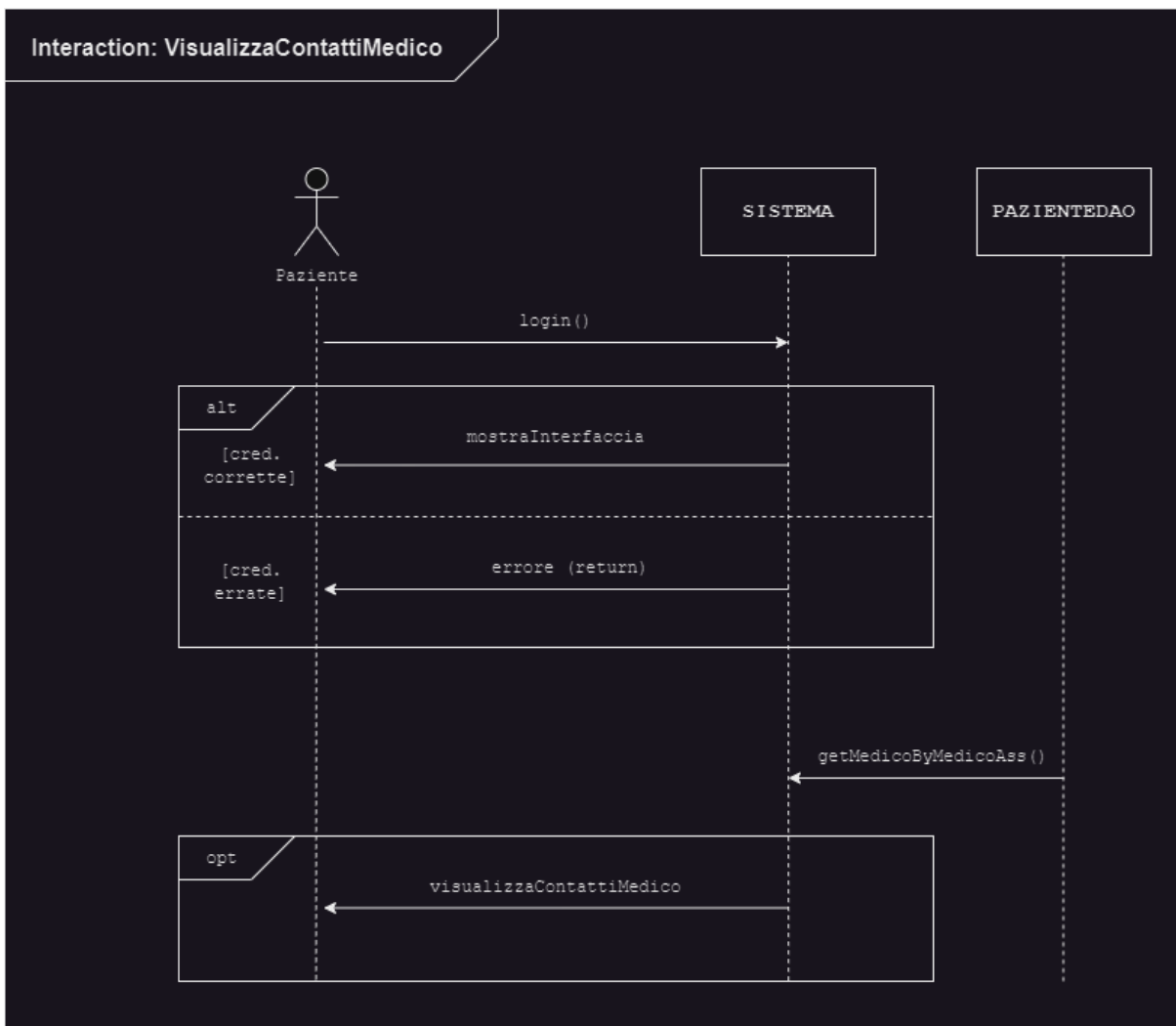
ATTORI → Paziente

PRE-CONDIZIONI → Il paziente deve aver effettuato l'accesso

PASSI:

1. Il paziente accede al sistema
2. Il paziente visualizza l'interfaccia dei dettagli
3. Il paziente visualizza i contatti del medico associato

POST-CONDIZIONI → Nessuna



Inserimento rilevazioni

Nell'interfaccia il paziente deve poter inserire **rilevazioni giornaliere** in merito alla terapia prescritta per esso.

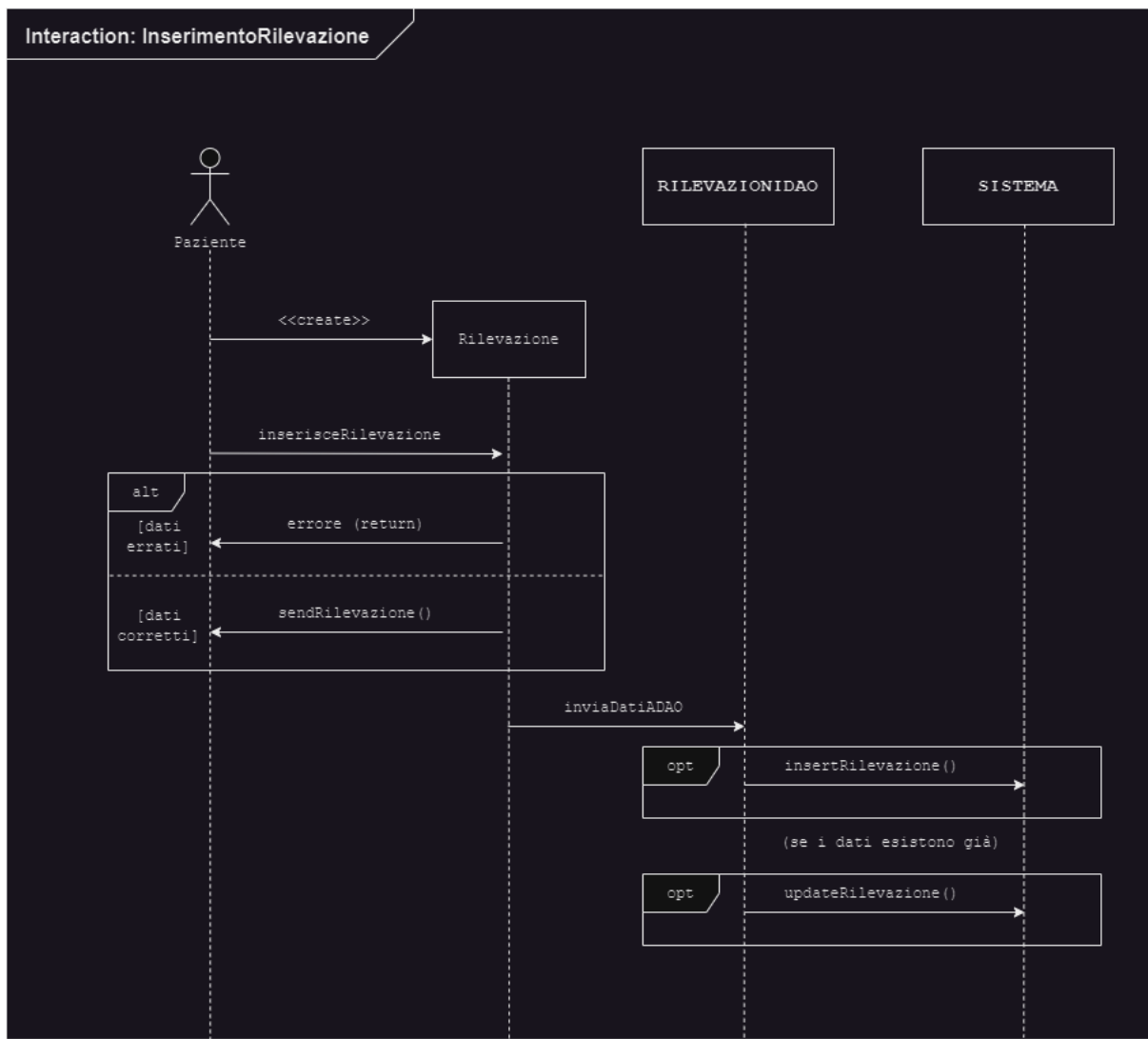
ATTORI → Paziente

PRE-CONDIZIONI → Il paziente deve aver effettuato l'accesso e deve essergli stata prescritta una terapia

PASSI:

1. Il paziente accede al sistema
2. Il paziente visualizza l'interfaccia dei dettagli
3. Il paziente inserisce la misurazione delle pressioni SBP/DBP
4. Il paziente inserisce assunzione farmaci e relativi sintomi

POST-CONDIZIONI → Senza eventuali errori, la rilevazione giornaliera viene inserita



Di seguito, un approfondimento per quanto riguarda l'inserimento della **misurazione pressioni SBP/DBP** e dell'**assunzione farmaci** e relativi **sintomi**.

MISURAZIONE PRESSIONI SBP/DBP

ATTORI → Paziente

PRE-CONDIZIONI → Il paziente deve aver effettuato l'accesso e deve essergli stata prescritta una terapia

PASSI:

1. Il paziente accede al sistema
2. Il paziente visualizza l'interfaccia dei dettagli
3. Il paziente inserisce il valore rilevato per SBP
4. Il paziente inserisce il valore rilevato per DBP

ASSUNZIONE FARMACI E RELATIVI SINTOMI

ATTORI → Paziente

PRE-CONDIZIONI → Il paziente deve aver effettuato l'accesso e deve essergli stata prescritta una terapia

PASSI:

1. Il paziente accede al sistema
2. Il paziente visualizza l'interfaccia dei dettagli
3. Il paziente inserisce eventuali sintomi rilevati
4. Il paziente inserisce la data e ora di rilevazione
5. Il paziente inserisce il farmaco prescritto, il n° di assunzioni e la quantità

I dati inseriti dal paziente devono essere consistenti:

- La **terapia** inserita (farmaco, n° assunzioni e quantità) deve coincidere con quella prescritta dal medico
- La **data** deve essere dello stesso giorno, oppure dei giorni precedenti (se il paziente si fosse dimenticato di inserire la/le rilevazione/i), ma NON può essere inserita una data successiva a quella attuale.
- L'**ora** deve essere compresa tra 00 e 23, altri valori vengono considerati errati

Casi d'uso: Medici

Dopo aver effettuato correttamente l'accesso, al medico viene mostrata un'interfaccia che permette di visualizzare una **lista** completa di TUTTI i pazienti registrati nel sistema.

Visualizzazione lista di tutti i pazienti

Il medico può accedere alle informazioni/dettagli di un paziente tra quelli della lista, dove può inoltre inserire/aggiornare la terapia, cliccando sul relativo bottone generato dinamicamente.

Dinamicamente significa che, all'aggiunta di un eventuale nuovo paziente al sistema, viene creato un nuovo bottone alla lista che si riferisce ad esso.

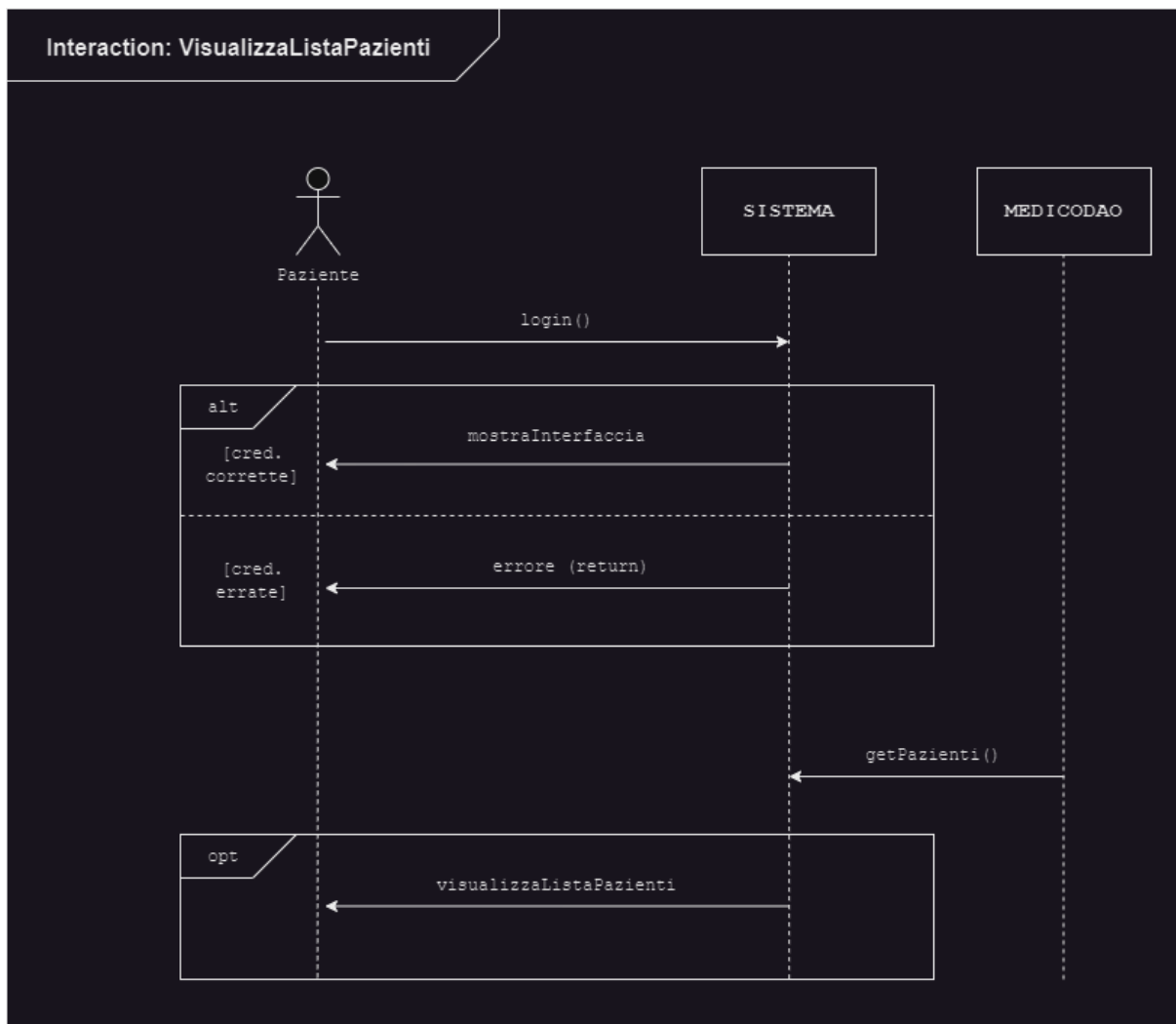
ATTORI → Medico

PRE-CONDIZIONI → Il medico deve aver effettuato l'accesso e deve essere presente almeno un paziente registrato nel sistema

PASSI:

1. Il medico accede al sistema
2. Il medico visualizza la lista dei pazienti registrati nel sistema

POST-CONDIZIONI → Il medico può accedere e visualizzare/aggiornare/inserire le informazioni di un paziente



Visualizzazione dettagli paziente

Dopo aver selezionato il paziente su cui il medico vuole visualizzare/aggiornare/inserire le relative informazioni e rilevazioni, al medico viene mostrata un'interfaccia che gli permette di:

- visualizzare **nome** e **cognome** del paziente di cui si stanno visualizzando le informazioni
- visualizzare le **informazioni** riguardanti le patologie del paziente e relativi **sintomi** (ultima rilevazione registrata)
- visualizzare una **tabella** contenente le ultime rilevazioni inserite dal paziente, con relativi avvisi e soglie (approfonditi in seguito)

- inserire/aggiornare la **terapia** prescritta al paziente e relative informazioni

ATTORI → Medico

PRE-CONDIZIONI → Il medico deve aver effettuato l'accesso e deve aver scelto uno dei pazienti dalla lista

PASSI:

1. Il medico accede al sistema
2. Il medico visualizza la lista dei pazienti registrati nel sistema e ne sceglie uno
3. Il medico visualizza l'interfaccia di visualizzazione e modifica delle relative informazioni

POST-CONDIZIONI → Nessuna

Visualizzazione tabella con rilevazioni

Nell'interfaccia relativa al paziente scelto, è possibile visualizzare una sezione contenente il **nome** e **cognome** del paziente e una **tabella** che contiene i valori:

DATA --- SBP --- DBP

Questi valori sono inerenti alla/e rilevazione/i giornaliera/e inserita dal paziente nella sua interfaccia.

ATTORI → Medico

PRE-CONDIZIONI → Il medico deve aver effettuato l'accesso e deve aver scelto uno dei pazienti dalla lista

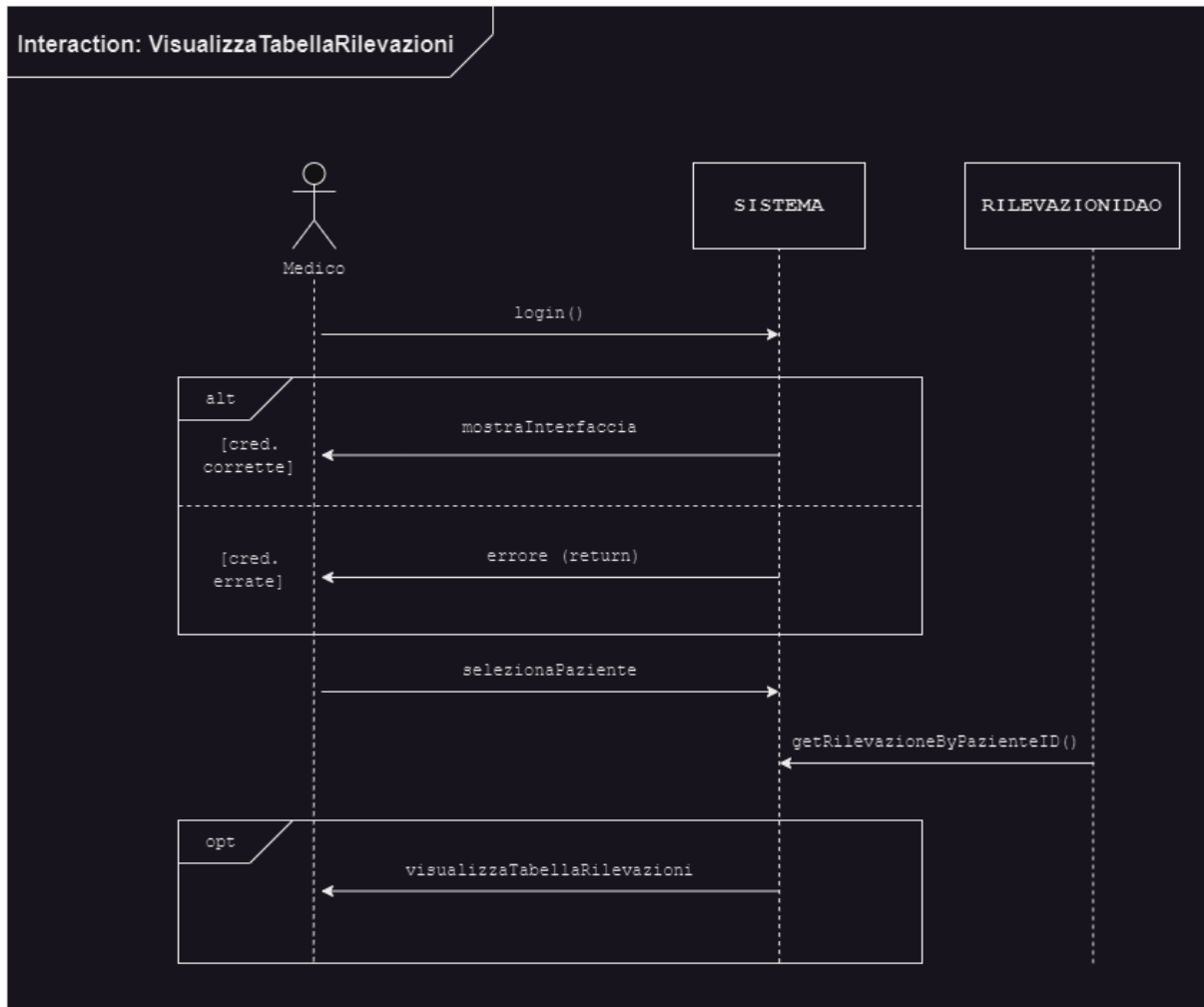
PASSI:

1. Il medico accede al sistema
2. Il paziente visualizza la lista dei pazienti registrati nel sistema e ne sceglie uno
3. Il medico visualizza la tabella contenente le rilevazioni (con soglie) compilata dal paziente in questione

POST-CONDIZIONI → Nessuna

All'entrata nel pannello dei dettagli, possono essere generati due tipi di **avvisi**:

- Il medico viene notificato con un alert apposito, se il paziente non ha inserito alcuna rilevazione al momento
- Il medico viene notificato con un alert apposito, se il paziente non ha inserito alcuna rilevazione da più di 3 giorni



Di seguito un approfondimento per quanto riguarda la visualizzazione delle **soglie** di rilevazione e degli **avvisi** generati dal sistema.

SOGLIE DI RILEVAZIONE

Le **soglie** di rilevazione vengono accentuate direttamente sul valore di SBP/DBP che appare nella tabella (se compilata).


Vengono applicate classi di stile CSS per rappresentare l'intensità dei valori di entrambe le pressioni SBP/DBP.

Se il valore di SBP inserito dal paziente è:

- **< 120** → il valore viene mostrato in **verde**
- **>= 120** e **< 140** → il valore viene mostrato in **giallo**
- **>= 140** e **< 180** → il valore viene mostrato in **arancione**
- **>= 180** → il valore viene mostrato in **rosso**

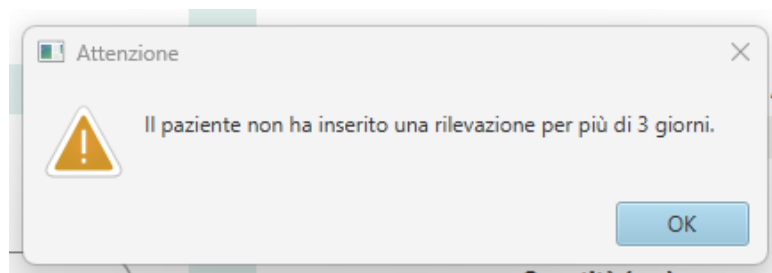
Se il valore di DBP inserito dal paziente è:

- **< 80** → il valore viene mostrato in **verde**
- **>= 80** e **< 90** → il valore viene mostrato in **giallo**
- **>= 90** e **< 110** → il valore viene mostrato in **arancione**
- **>= 110** → il valore viene mostrato in **rosso**

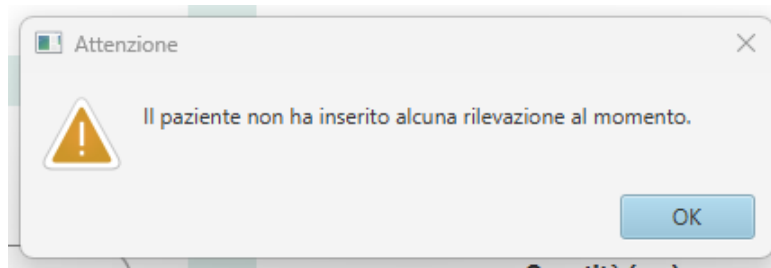
Dati Paziente		
 Mario Rossi		
Sintomi: Dolore al petto		
Info: Febbre alta e raffreddore		
Data	SBP	DBP
08/07/2023	190	190
07/07/2023	150	40
06/07/2023	180	180
09/07/2023	200	90

VISUALIZZAZIONE AVVISI

Il medico può essere notificato con due **avvisi** appena entra nell'interfaccia relativa al paziente selezionato.



L'avviso mostrato sopra viene generato quando il paziente selezionato non ha inserito una rilevazione per più di 3 giorni.



L'avviso mostrato sopra viene generato quando il paziente selezionato non ha inserito alcuna rilevazione al momento.

ATTORI → Medico

PRE-CONDIZIONI → Il medico deve aver effettuato l'accesso e deve aver scelto un paziente

PASSI:

1. Il medico accede al sistema
2. Il medico visualizza la lista dei pazienti registrati nel sistema e ne sceglie uno
3. Se necessario, viene mostrato uno dei due avvisi, in base alla situazione

Inserimento terapia

Nell'interfaccia relativa al paziente scelto, il medico deve poter inserire/aggiornare la **terapia** e le relative informazioni del paziente.

ATTORI → Medico

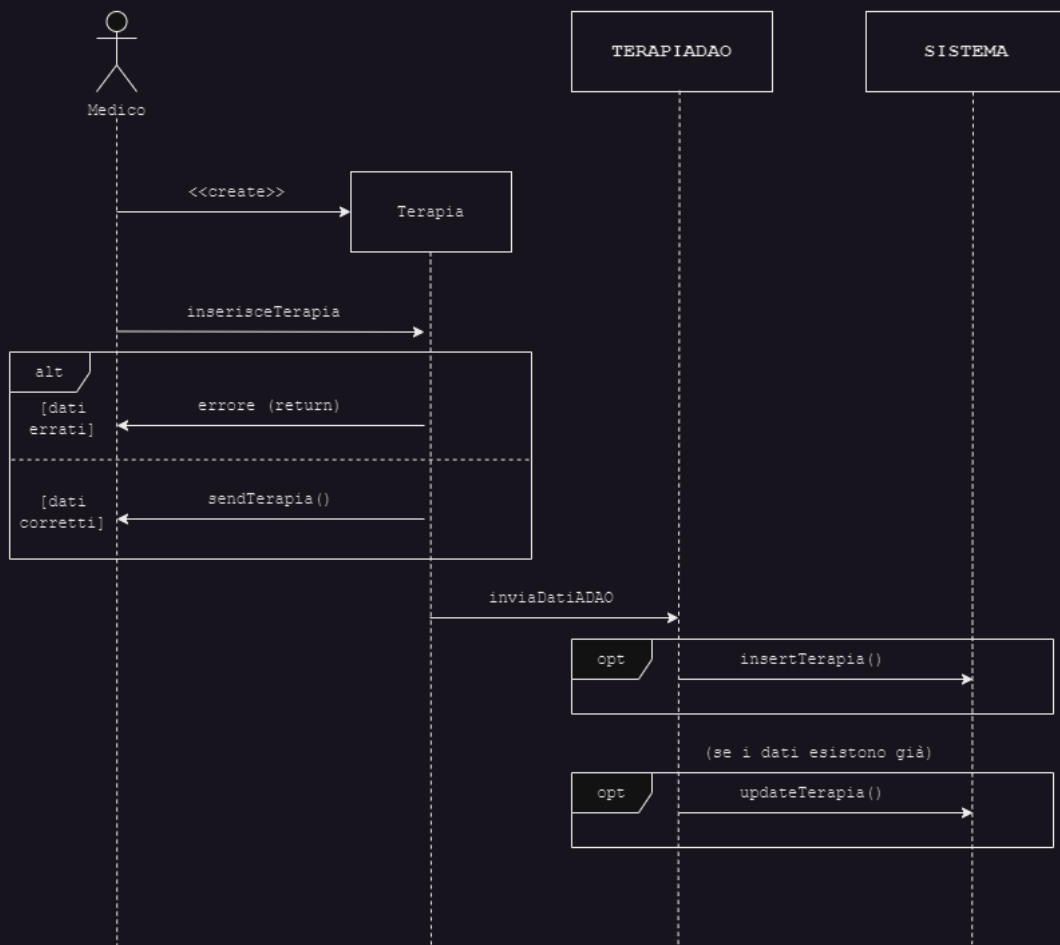
PRE-CONDIZIONI → Il medico deve aver effettuato l'accesso e deve e deve aver scelto un paziente

PASSI:

1. Il medico accede al sistema
2. Il medico visualizza la lista dei pazienti registrati nel sistema e ne sceglie uno
3. Il medico può inserire/aggiornare la terapia per il paziente

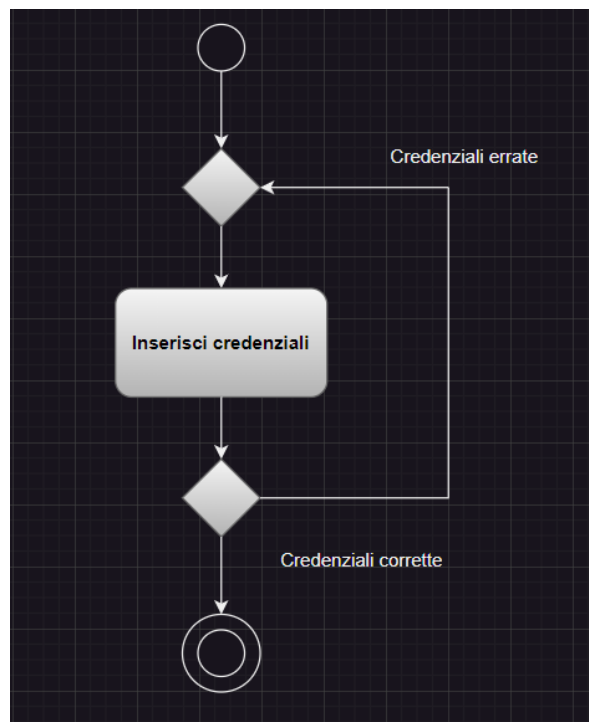
POST-CONDIZIONI → Senza eventuali errori, la terapia viene inserita/aggiornata

Interaction: Inserimento Terapia

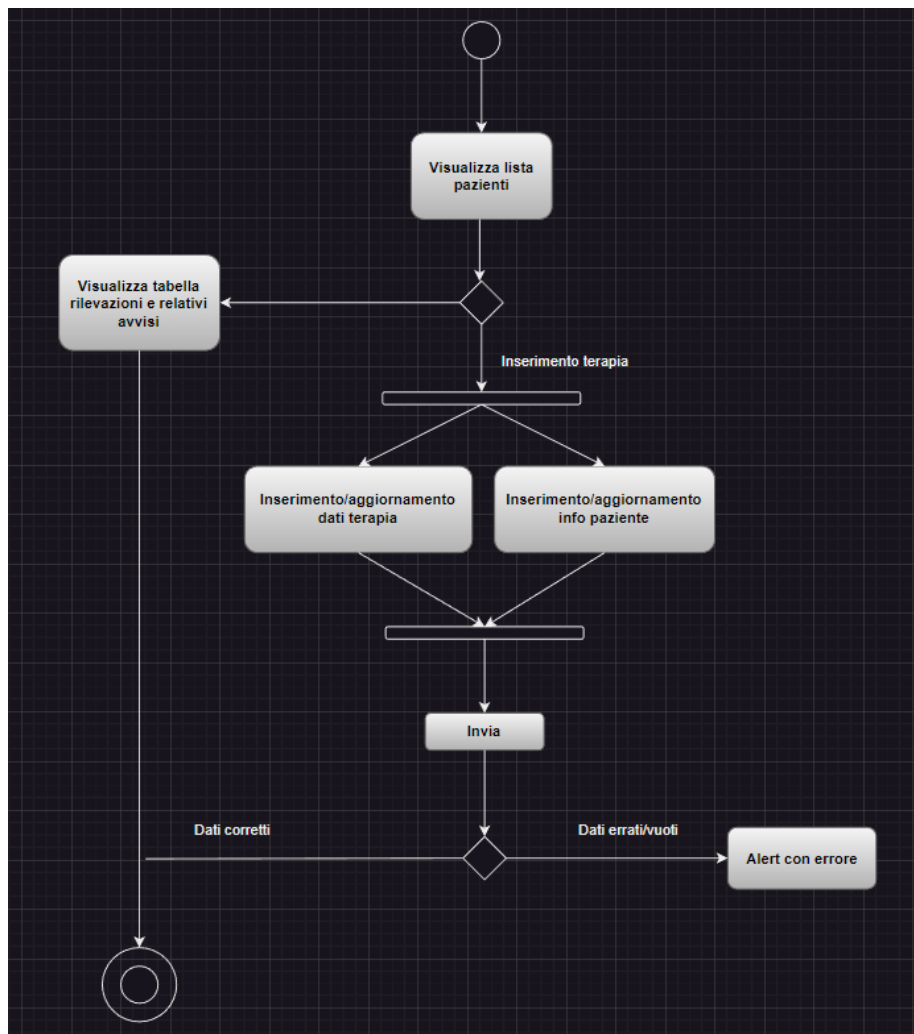


Activity Diagram

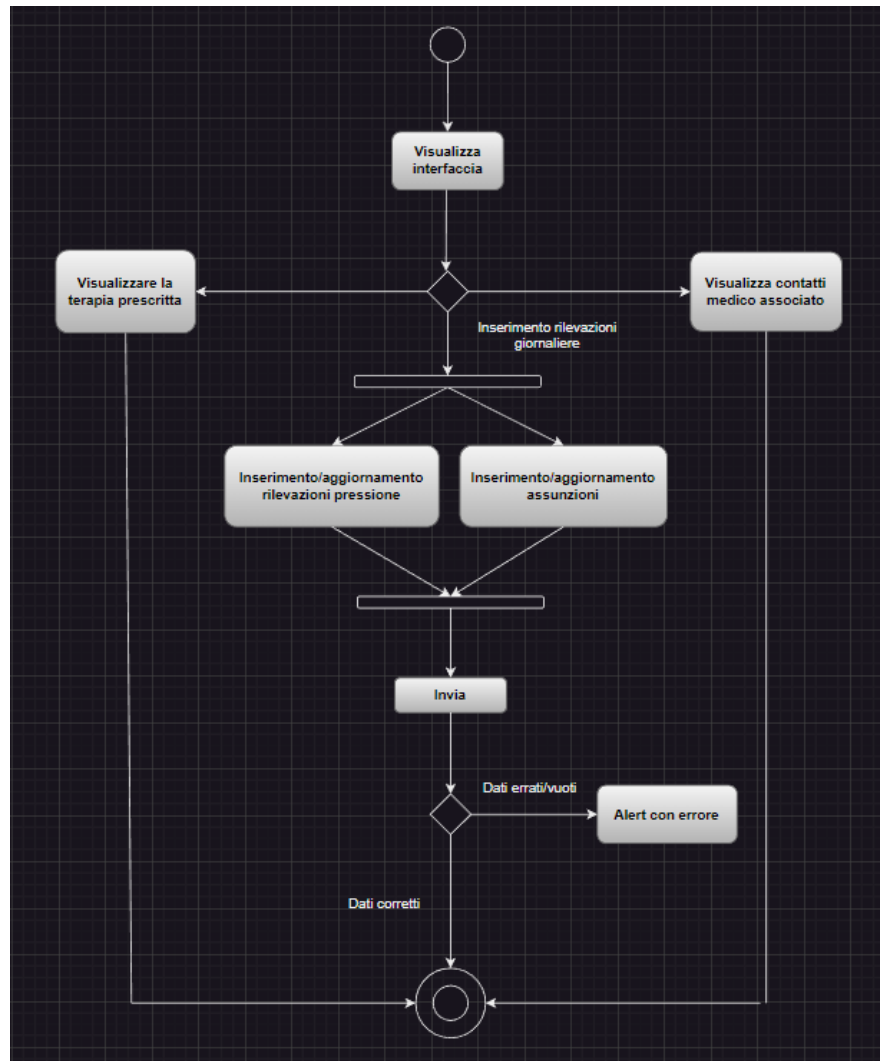
Qui di seguito, i **diagrammi di attività** del relativo utente rispetto al sistema:



AUTENTICAZIONE MEDICO/PAZIENTE



ATTIVITA' MEDICO



ATTIVITA' PAZIENTE

Architettura del progetto

Processo di sviluppo

Il processo di sviluppo è stato suddiviso in fasi:

- 1) Progettazione
- 2) Implementazione codice
- 3) Test e validazione

Progettazione

L'obiettivo principale è stato quello di ragionare sulle varie parti del progetto da realizzare prima di passare direttamente all'implementazione, in modo da non incappare in rielaborazioni inutili del codice.

Si è scelto di utilizzare **GitHub** come piattaforma per il versioning e il mantenimento del codice, per fare in modo che dopo ogni modifica significativa questo fosse salvato e mantenuto in modo sicuro.

In questa fase sono state effettuate varie scelte progettuali:

- Credenziali **username** & **password**:
 - Per quanto riguarda username è stato scelto un pattern comune: **nome.cognome**
 - Invece per quanto riguarda password è stato scelto il pattern comune: **pwdnome**
- Utilizzo **DB** piuttosto che file **JSON**:
 - Inizialmente si è scelto di utilizzare un file JSON per la gestione dei dati. Successivamente però abbiamo deciso di optare per l'utilizzo di un DATABASE SQLite.

- **Tabella storico** rilevazioni paziente:
 - Si è scelta la realizzazione di una tabella generale contenente lo storico delle rilevazioni inserite dal paziente nella sezione del medico.

Tecnica di implementazione

Lo sviluppo del codice è stato fatto man mano, seguendo i progressi del progetto, tutto seguito da commit e commenti per ricordare le modifiche effettuate.

Sono stati eseguiti circa 100 commit verificabili, dagli utenti: **franscescomerighi1202** e **jz4ndo**

[GestionaleMedicina GitHub](#)

Durante la scrittura del codice è stato usato un file di testo per segnare le cose da fare: **todo.txt**. Completata una parte di progetto veniva appuntata l'attività realizzata con successo.

Gestione dati

Per quanto riguarda la gestione dei dati, è stato scelto di utilizzare un **DATABASE** tramite **SQLite**, questo ci ha permesso di gestire i dati dei pazienti e medici in varie tabelle.

Il DATABASE che abbiamo utilizzato è chiamato: **GestionaleMedicinaDB**

Esso è composto dalle seguenti tabelle relazionali:

- **Medico**

- ID → valore univoco per il medico.
- Name → nome del medico.
- Surname → cognome del medico.
- Username → username del medico (nome.cognome)
- Password → password del medico (pwdnome)
- Email → email del medico (nome.cognome@gmail.com)

- **Paziente**

- ID → valore univoco per il paziente.
- Name → nome del paziente.
- Surname → cognome del paziente.
- Username → username del paziente. (nome.cognome)
- Password → password del paziente. (pwdnome)
- Symptoms → sintomi inseriti dal paziente.
- Medicine → nome del medicinale inserito dal paziente.
- Assumptions → numero di assunzioni inserito dal paziente
- Quantity → quantità di farmaco inserita dal paziente.
- Informations → informazioni aggiuntive.
- MedicoAss (*foreign key*) → ID del medico di riferimento.

- **Rivelazioni**

- ID → valore univoco relativo alla rilevazione.
- SBP → valore della pressione registrata dall'utente.
- DBP → valore della pressione registrata dall'utente.
- Day → valore del giorno di registrazione dei dati inserito dall'utente.
- Month → valore del giorno di registrazione dei dati inserito dall'utente.
- Year → valore dell'anno di registrazione dei dati inserito dall'utente.
- Hours → orario di registrazione dei dati inserito dall'utente.

- IDPaziente (*foreign key*) → valore univoco riferito all'ID del paziente.

- **Terapia**

- ID → valore univoco relativo alla terapia assegnata
- MedicineTherapy → farmaco assegnato
- AssumptionsTherapy → n° di assunzioni assegnate
- QuantityTherapy → quantità assegnata
- IndicationsTherapy → relative indicazioni assegnate
- IDPaziente (*foreign key*) → valore univoco riferito all'ID del paziente

Per gestire l'interazione tra il sistema e il DATABASE, è stata creata una classe chiamata **DBManager** che contiene le seguenti variabili e metodi:

- `private static final String URLDB` → contiene il path relativo al DATABASE, variabile costante
- `private static Connection connection` → oggetto di tipo `Connection` utilizzato nelle operazioni di chiusura e apertura della connessione con il DATABASE
- `public static Connection getConnection()` → metodo statico che gestisce l'apertura della connessione
- `public static void closeConnection()` → metodo statico che gestisce la chiusura della connessione

Poi, nei vari metodi delle classi DAO, ci è bastato dichiarare un oggetto di tipo `Connection`, che viene utilizzato nei `try` e `catch` per acquisire i dati dal database attraverso le query.

Questo *modus operandi* è stato attuato in tutti i metodi presenti nelle classi DAO che utilizzano le query per ottenere i dati da `GestionaleMedicinaDB`.

Per gestire eventuali eccezioni riguardanti l'apertura (o chiusura) della connessione e/o riguardanti la gestione dei dati nel DATABASE, viene utilizzata `SQLException`.

Pattern architetturali usati

Il sistema da noi progettato è basato su tecniche di modellazione ad oggetti gestite tramite i pattern architetturali **MVC** e **DAO**.

MVC (Model View Controller)

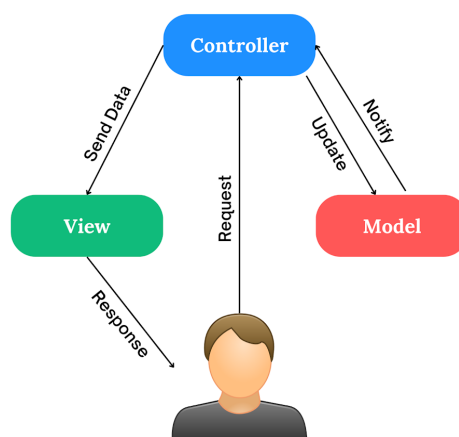
Questo pattern ci ha permesso di separare il progetto in parti distinte, ognuna con una responsabilità specifica.

MVC permette di favorire la modularità, il riutilizzo del codice e la manutenibilità dell'applicazione.

Abbiamo diviso le componenti del progetto secondo i vari livelli che compongono l'architettura presa in questione:

- **Model** → la parte responsabile dell'elaborazione dei dati, in modo da poter manipolare gli oggetti del sistema.
- **View** → la parte riguardante l'interfaccia dell'applicazione, responsabile della presentazione dei dati e dell'interazione con l'utente.
- **Controller** → parte del sistema che definisce la logica, comunica con la View e il Model, in modo che il sistema possa reagire. Richiama metodi dal Model per ottenere i dati, in seguito li fa visualizzare all'utente tramite View.

Non sono state usate ulteriori articolazioni: **Model**, **View**, **Control** sono tre blocchi monolitici, le varie interazioni tra componenti saranno presenti nei diagrammi successivi.



Per la gestione della **comunicazione** tra Modello, Vista e Controllore, sono state adottate le seguenti scelte:

- **Vista → Controllore**

Vengono utilizzati i seguenti meccanismi:

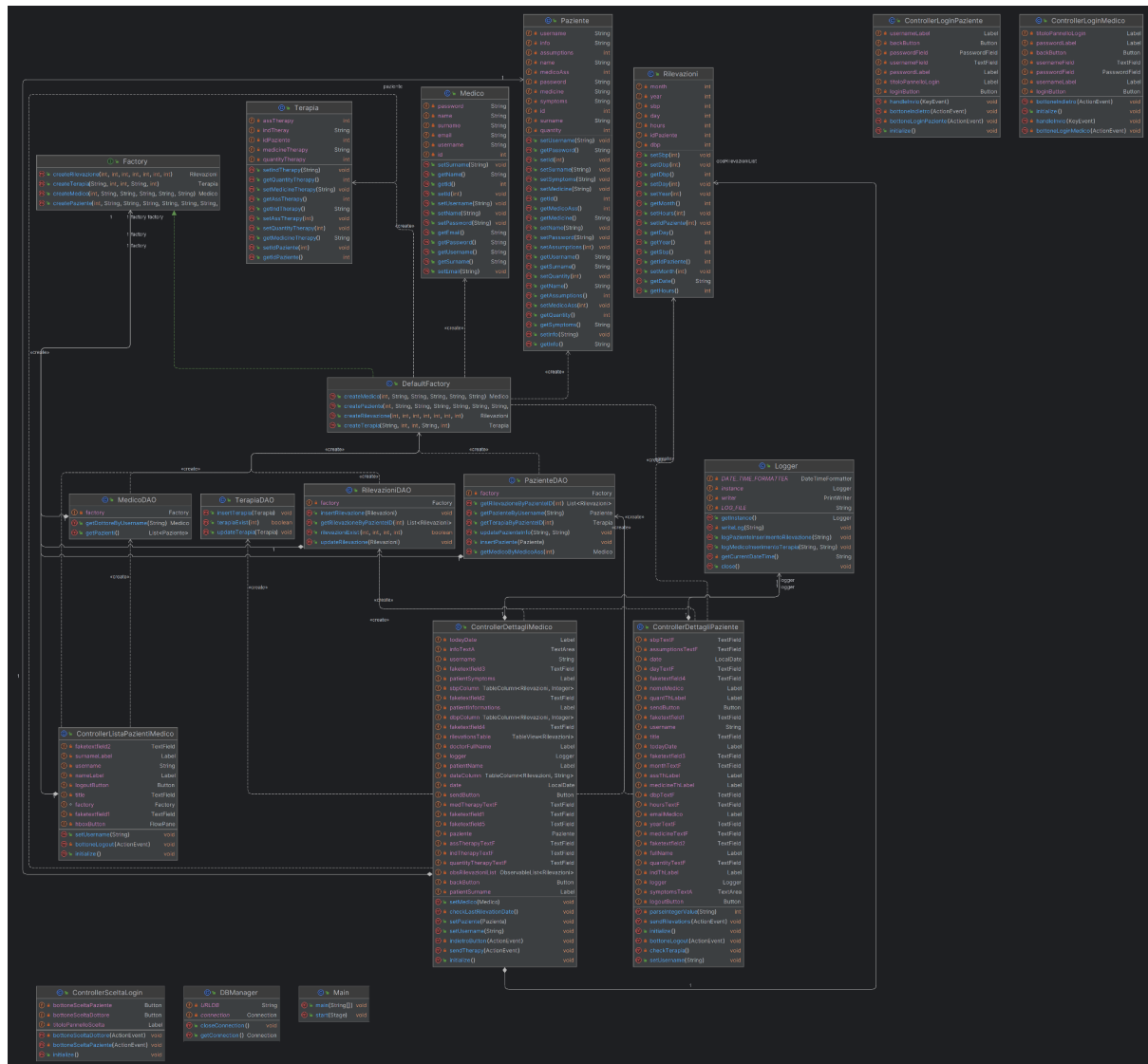
- ActionEvent → gestiscono le interazioni dell'utente con gli elementi dell'interfaccia grafica, il controller riceve i vari eventi e può eseguire le azioni appropriate.
- @FXML → annotazione utilizzata per stabilire una connessione tra un elemento dell'interfaccia utente definito in un file FXML e il codice associato all'elemento nel controller.
- FXMLLoader → consente di caricare le viste e stabilire la loro connessione con i rispettivi controller.

- **Vista → Modello**

- Vengono create le classi **DAO** (Data Access Object), per ogni entità, che, assieme alla classe DBManager, permettono di comunicare con il database e recuperare/salvare dati in esso.
- I **Controllori** ricevono gli input dall'utente e li elaborano (validazione dati), chiamano poi la relativa classe DAO per ottenere/aggiornare dati e successivamente, aggiornare la Vista in base ai risultati ottenuti.

- **Modello → Controllore**

- I **Modelli** contengono i dati relativi alle varie entità presenti nel sistema e la logica di business associata.
- I **Controllori** possono accedere ai dati ed effettuare operazioni su di essi.



DAO (Data Access Object)

Abbiamo utilizzato questo pattern per fornire un'interfaccia unificata per l'**accesso ai dati**, in modo che l'applicazione non debba preoccuparsi di come i dati vengono archiviati o recuperati.

Si compone di una serie di classi DAO (una per ogni classe "entità" presente nel Model):

- PazienteDAO → metodi di gestione dati per Paziente
- MedicoDAO → metodi di gestione dati per Medico
- RilevazioniDAO → metodi di gestione dati per Rilevazioni
- TerapiaDAO → metodi di gestione dati per Terapia

Queste classi includono metodi per **salvare, recuperare, aggiornare e eliminare** dati dal DATABASE.

All'interno di ognuno dei metodi contenuti nelle classi, si seguono i seguenti step:

1. **Apertura** della connessione
2. **Inserimento/Aggiornamento/Eliminazione** di relativi dati dalle tabelle contenute nel DATABASE, tramite delle *query*
3. **Chiusura** della connessione ed eventuale valore di ritorno

Pattern progettuali usati

Abbiamo utilizzato pattern progettuali per gestire alcune parti logiche del sistema.

Factory

Abbiamo deciso di implementare il pattern **Factory** nella parte di applicazione che riguarda le classi *DAO*, questo perchè offre i seguenti vantaggi nella creazione delle istanze:

- Consente di incapsulare la logica di creazione degli oggetti all'interno dell'interfaccia *Factory* e della classe *DefaultFactory*.
Questo significa che il codice che utilizza le istanze di *Medico*, *Paziente*, *Terapia* e *Rilevazioni* non deve preoccuparsi dei dettagli specifici di creazione di tali oggetti.
Ciò semplifica la gestione delle dipendenze e promuove una maggiore modularità nel codice.
- Il codice può affidarsi alla factory per ottenere un'istanza adeguata senza dover gestire i dettagli di implementazione.
- È possibile modificare o estendere il modo in cui vengono create le istanze senza dover modificare il codice che le utilizza.
- Raggruppando il codice di creazione delle istanze all'interno della factory, è possibile centralizzare e organizzare in modo più efficiente la logica di creazione degli oggetti.

Singleton

Abbiamo deciso di implementare il pattern **Singleton** nella parte di applicazione che riguarda la classe *Logger*, questo perchè offre i seguenti vantaggi:

- Garantisce l'esistenza di un'unica istanza del Logger all'interno dell'applicazione.
- Offre accesso globale al Logger, garantendo che tutti i messaggi di log vengano scritti nello stesso file di log senza sovrascrizioni o duplicazioni accidentali
- Migliora l'efficienza dell'applicazione evitando la creazione ripetuta di nuove istanze del Logger
- Consente una maggiore estensibilità e configurabilità per gestire i log futuri

Per capire come è stato implementato, fare riferimento al relativo [appendice](#) (cfr. *Implementazione file di log*).

Implementazione file di log

Abbiamo implementato un file di log, chiamato **log.txt** che tiene traccia degli inserimenti/aggiornamenti riguardanti la TERAPIA assegnata da un medico e le RILEVAZIONI inserite dal paziente.

La relativa voce nel file di testo per ogni inserimento viene visualizzata in questo modo:

```
[2023-07-13 12:36:51] Inserita TERAPIA per paziente 'Dino Tomasini' da parte del medico 'Davide Ricci'  
[2023-07-13 12:47:23] Inserita RILEVAZIONE da parte di 'Bruno Tedesco'
```

Per fare ciò, abbiamo creato una classe chiamata **Logger** che implementa la logica per creare un file se non esiste, scrivere sul file (in modalità *append*) e poi chiuderlo una volta terminato l'inserimento.

Alcuni metodi della classe sono:

- *public Logger* **getInstance()** → pattern Singleton che garantisce la creazione di una sola istanza della classe *Logger*.
- *public void* **logMedicoInserimentoTerapia(nomeMed, nomePaz)** → crea un "messaggio" contenente le stringhe da scrivere sul file per quanto riguarda la TERAPIA.
- *public void* **logPazienteInserimentoRilevazione(nomePaz)** → crea un "messaggio" contenente le stringhe da scrivere sul file per quanto riguarda la RILEVAZIONE.
- *private void* **writeLog(message)** → scrive il "messaggio" sul file tramite un'istanza della classe *PrintWriter*.
- *public void* **close()** → chiude il file.

Grafica e design

Per il design dell'applicazione, quindi per quanto riguarda il componente View, abbiamo utilizzato **FXML** (FXML Markup Language), un linguaggio basato su XML usato per definire l'interfaccia nelle applicazioni **JavaFX**.

Il design è semplice e minimale, abbiamo utilizzato una immagine principale di background, presente in tutte le schermate. I bottoni e gli input di testo sono rounded e sono state scelte delle immagini per rappresentare ogni sezione dell'interfaccia posta all'utente.

Tramite **FXML** abbiamo legato i componenti grafici ai vari Controller, grazie al tag **@FXML** possiamo manipolare dinamicamente i componenti grafici e gestire le azioni con l'utente.

L'utilizzo di JavaFX ci ha permesso di utilizzare il meccanismo di caricamento dei file FXML dove viene interpretato l'albero grafico dei nodi, contenuti in una scena e visualizzati poi nella finestra principale (**Stage**).

La parte/cartella riguardante la sezione grafica è composta da:

- **/View**
 - **MedicoDettagli.fxml** → **fxml** realizzato per visualizzare il pannello dei dettagli del medico. Contiene:
 - Sezione del profilo che ha effettuato l'accesso.
 - Sezione inserimento terapia per il paziente scelto.
 - Sezione storico dati del paziente scelto (tabella).
 - **PazienteDettagli.fxml** → **fxml** realizzato per visualizzare il pannello di inserimento dei dati del paziente. Contiene:
 - Sezione dati profilo del paziente che ha effettuato l'accesso.
 - Sezione inserimento assunzione farmaci.
 - Sezione contatto medico assegnato.
 - **MedicoListaPazienti.fxml** → **fxml** realizzato per visualizzare il pannello di seguito all'accesso effettuato dal medico. Contiene:
 - Sezione dati profilo del medico che ha effettuato l'accesso.

- **Button** dinamici contenenti i nomi dei pazienti selezionabili presenti nel sistema.
- **MedicoLogin.fxml** → **fxml** realizzato per visualizzare il pannello di login per i medici. Contiene:
 - **Input** di testo per il nome utente.
 - **Input** di testo per la password.
- **PazienteLogin.fxml** → **fxml** realizzato per visualizzare il pannello di login per i pazienti. Contiene:
 - **Input** di testo per il nome utente
 - **Input** di testo per la password.
- **SceltaLogin.fxml** → **fxml** realizzato per visualizzare il pannello di login in cui l'utente può scegliere se accedere come **PAZIENTE** o **MEDICO**. Contiene:
 - **Button** paziente + relativa immagine identificativa.
 - **Button** medico + relativa immagine identificativa.

Test e validazione

Per la verifica della solidità del software abbiamo svolto varie attività di test:

- 1) Verifica di correttezza del codice, pattern e diagrammi.
- 2) Test sviluppatore, informato sul progetto.
- 3) Test utente generico, non informato sul progetto.

Verifica correttezza codice, pattern/diagrammi

Nella prima fase si è rivista la correttezza del codice prodotto, e confrontato con le specifiche presenti nei diagrammi **UML** realizzati. É stata fatta una ispezione del codice, in modo da controllare e nel caso rimuovere, possibile codice inutilizzato o problematico.

Test sviluppatore

Nella fase di test dello sviluppatore è stato utilizzato il software seguendo la logica già conosciuta dallo sviluppatore, in modo da riscontrare la reazione del software e confrontarla con quella attesa. Le verifiche eseguite sono state:

- Verifica **funzionamento login**:
 - Verificato funzionamento della sezione di login per paziente/medico, tramite l'inserimento dei dati: username e password di tutti i pazienti/medici. Eventuali dati errati vengono respinti.
- Verifica visualizzazione corretta **dati** paziente/medico:
 - Verificati vari inserimenti di input errati, input vuoti, stringhe malformate, numeri negativi, ecc...
 - Verificati inserimenti/aggiornamenti corretti nel DATABASE, con eventuali voci duplicate
 - Verificata corretta visualizzazione dati nelle varie Label o altri elementi dell'interfaccia

- Verifica meccanismo di **avvisi** paziente/medico, in particolare:
 - Errore nella fase di login
 - Avvisi riguardanti le rilevazioni per il medico
 - Avvisi riguardanti la terapia per il paziente
 - Avvisi di dati correttamente inviati
 - Errori coerenza di dati e/o campi vuoti

Test utente

Il test utente generico è stato l'ultimo test sottoposto al software realizzato. Questa attività è stata svolta da persone inesperte, che non conoscono la logica e le procedure da seguire per utilizzare il software.

Alle persone che hanno eseguito il test **non è stati dati suggerimenti particolari**, è stato solo chiesto di utilizzare il software ed effettuare l'inserimento dei dati, prima sotto forma di paziente e in seguito sotto forma di medico.

Lo scopo principale è stato quello di individuare particolari errori di progettazione, ed ascoltare eventuali commenti da parte degli utilizzatori in modo da apportare modifiche al sistema.