Documentazione del progetto di Ingegneria del Software

Luglio 2023

SOMMARIO

1 REQUISITI ED INTERAZIONI UTENTE-SISTEMA

1.1 SPECIFICHE CASI D'USO

1.1.1 Note generali

Il sistema proposto supporta l'utilizzo da parte di medici e pazienti. Entrambe le categorie accedono al sistema inserendo il proprio codice fiscale e selezionando la propria categoria. Nel caso in cui il codice fiscale (inserito manualmente dal gestore del sistema) sia registrato nel database, l'utente sarà indirizzato alla propria Homepage.

1.1.2 Casi d'uso relativi ai Medici

1.1.2.1 Note generiche

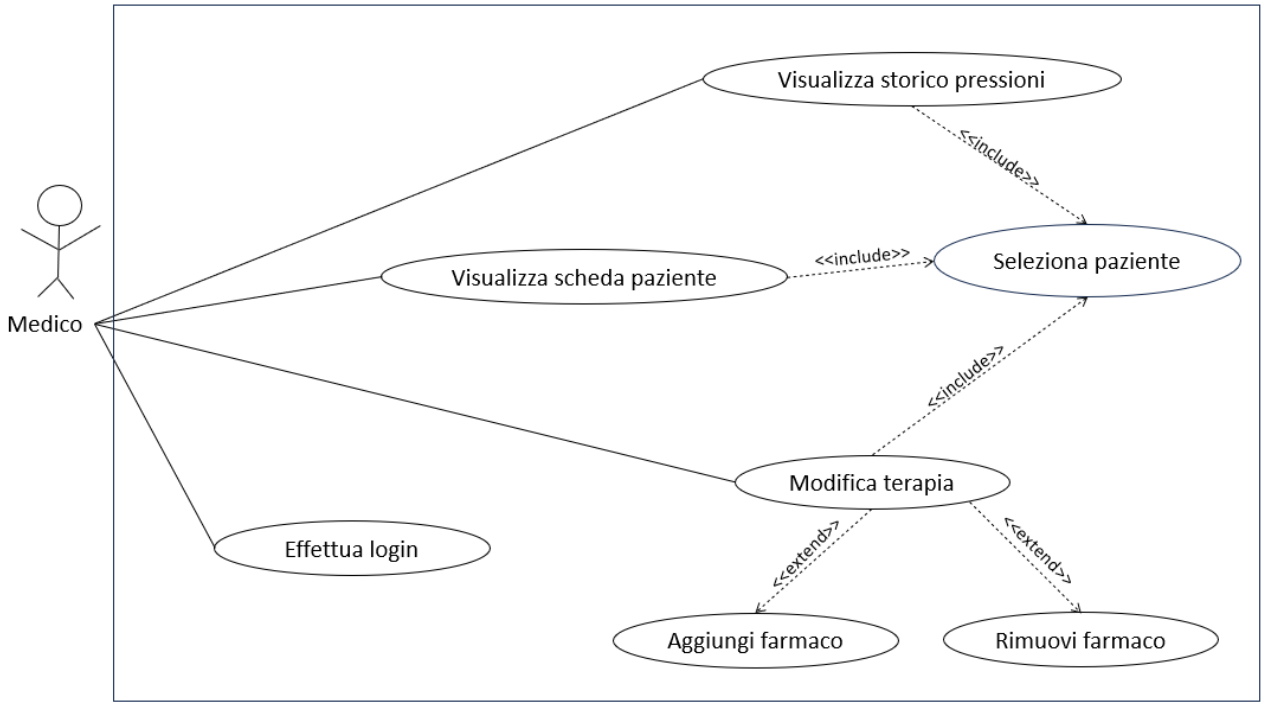

Figura 1 - Casi d'uso medico

Dopo opportuno accesso, il medico viene introdotto ad una interfaccia che permette di:

- Selezionare un paziente
- Visualizzarne i dati anagrafici
- Visualizzarne le statistiche sulle ultime misurazioni settimanali relative alle pressioni Sistoliche e Diastoliche
- Visualizzarne i sintomi dell'ultima settimana

Inoltre, sono presenti tre bottoni: "logout", "storico pressioni" e "modifica terapia", questi ultimi relativi al paziente selezionato.

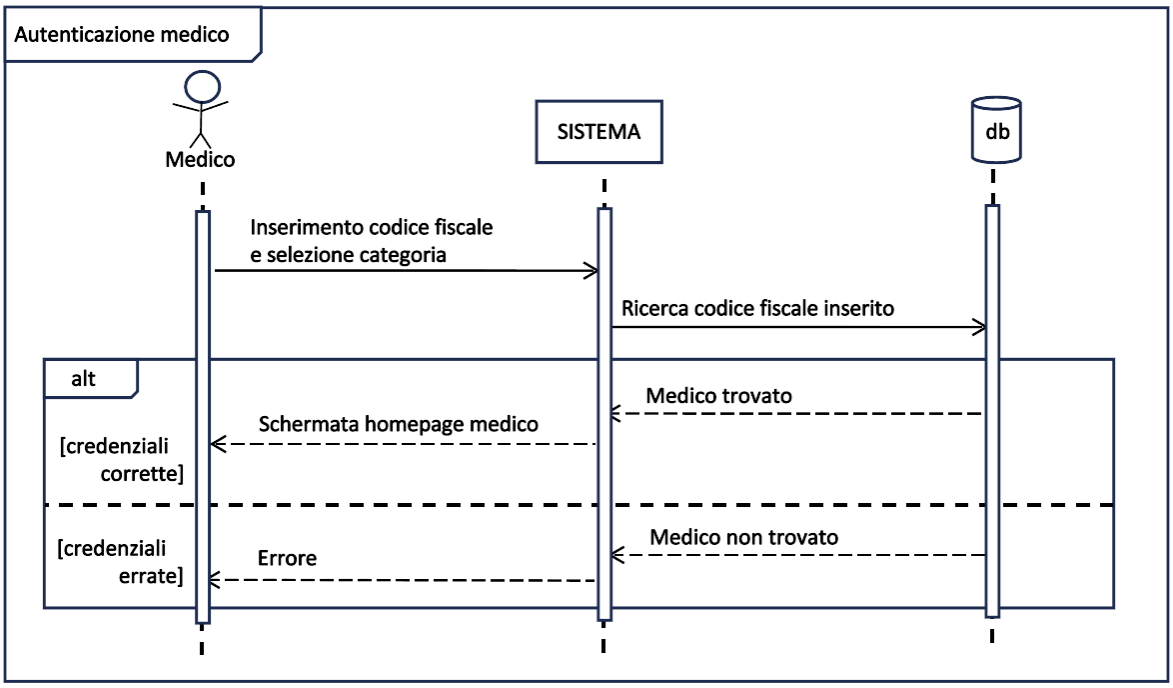

Figura 2 - Autenticazione medico

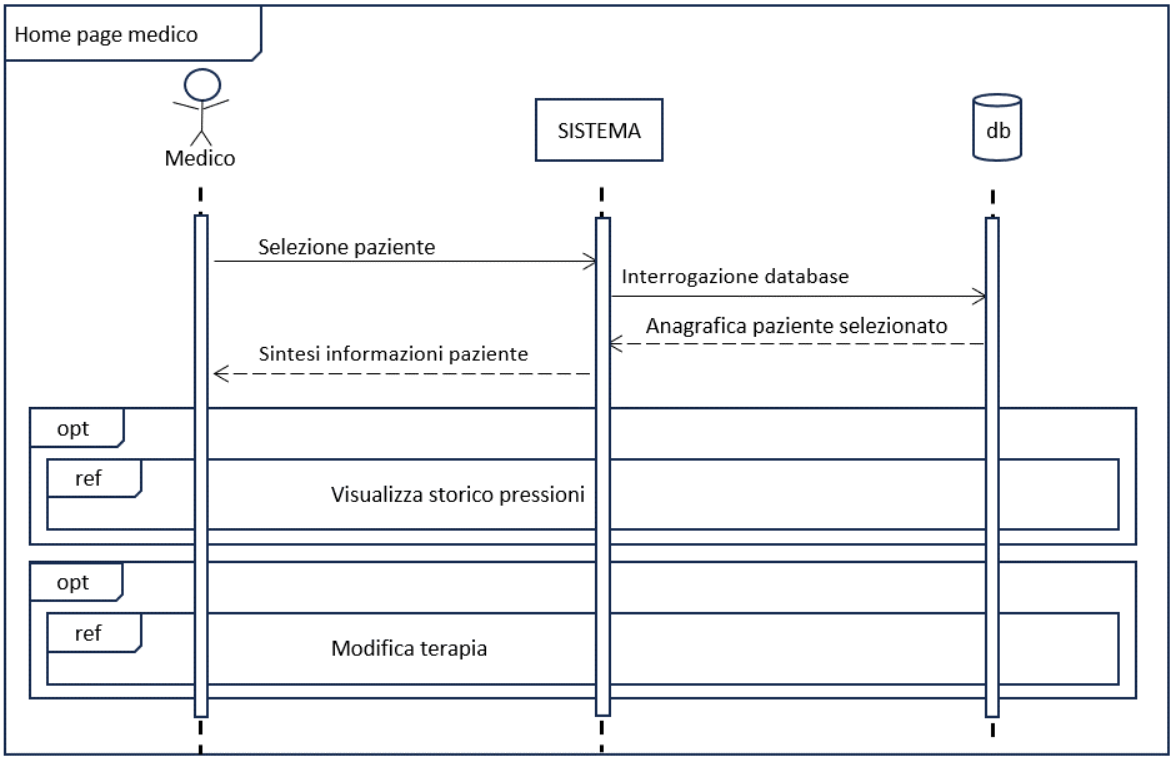

Figura 3 – Home-page medico

1.1.2.2 Visualizzazione dello storico delle pressioni

Il medico deve poter visualizzare lo storico delle pressioni inserite dal paziente. Per fare questo deve aver effettuato il login e aver selezionato il paziente interessato.

Attori: *Medico*
Precondizioni: Il medico dev'essersi autenticato.

Passi:

1. *Il medico accede al sistema*
2. *Il medico è introdotto alla home page*
3. *Il medico seleziona il paziente desiderato*
4. *Il medico accede all'interfaccia per lo storico pressioni*
5. *Il medico visualizza lo storico pressioni del paziente selezionato precedentemente*

Postcondizioni: *nessuna*

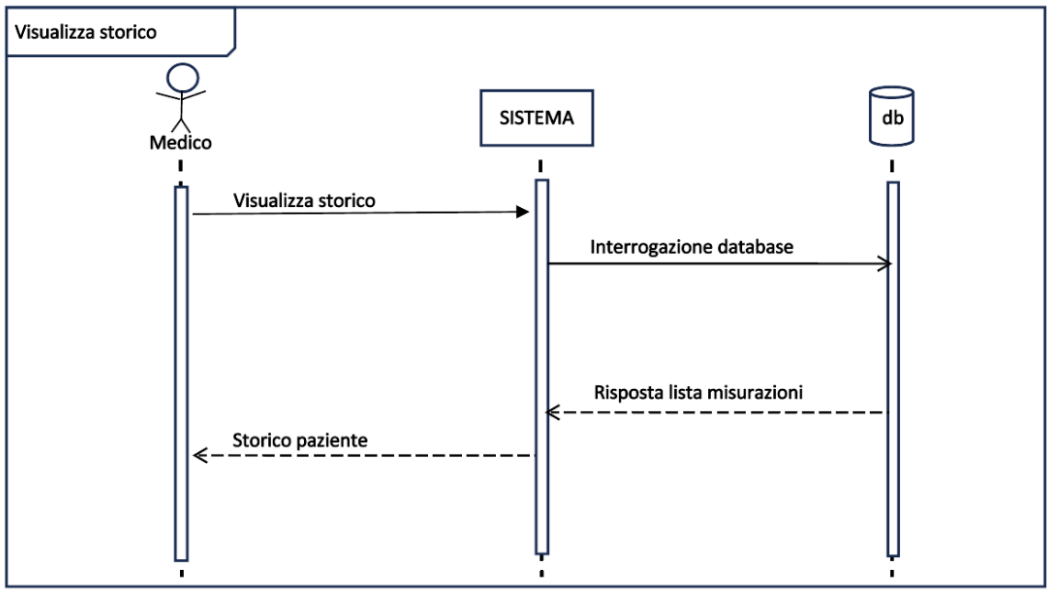

Figura 4 - Visualizzazione storico pressioni

1.1.2.3 Modifica della terapia

Ogni medico deve poter modificare la terapia di un paziente

Attori: *Medico*
Precondizioni: Il medico dev'essersi autenticato.

Passi:

1) *Il medico accede al sistema.*
2) *Il medico è introdotto alla home page.*
3) *Il medico accede all'interfaccia per la modifica della terapia.*
4) *A questo punto il Medico può:*
    a) *Aggiungere un farmaco specificando il nome, la quantità e la dose da assumere ogni giorno ed eventuali indicazioni.*
    b) *Rimuovere un farmaco (se presente).*
5) *Il medico conferma la modifica.*

Postcondizioni: *La nuova terapia deve essere stata memorizzata e il medico deve essere memorizzato come autore.*

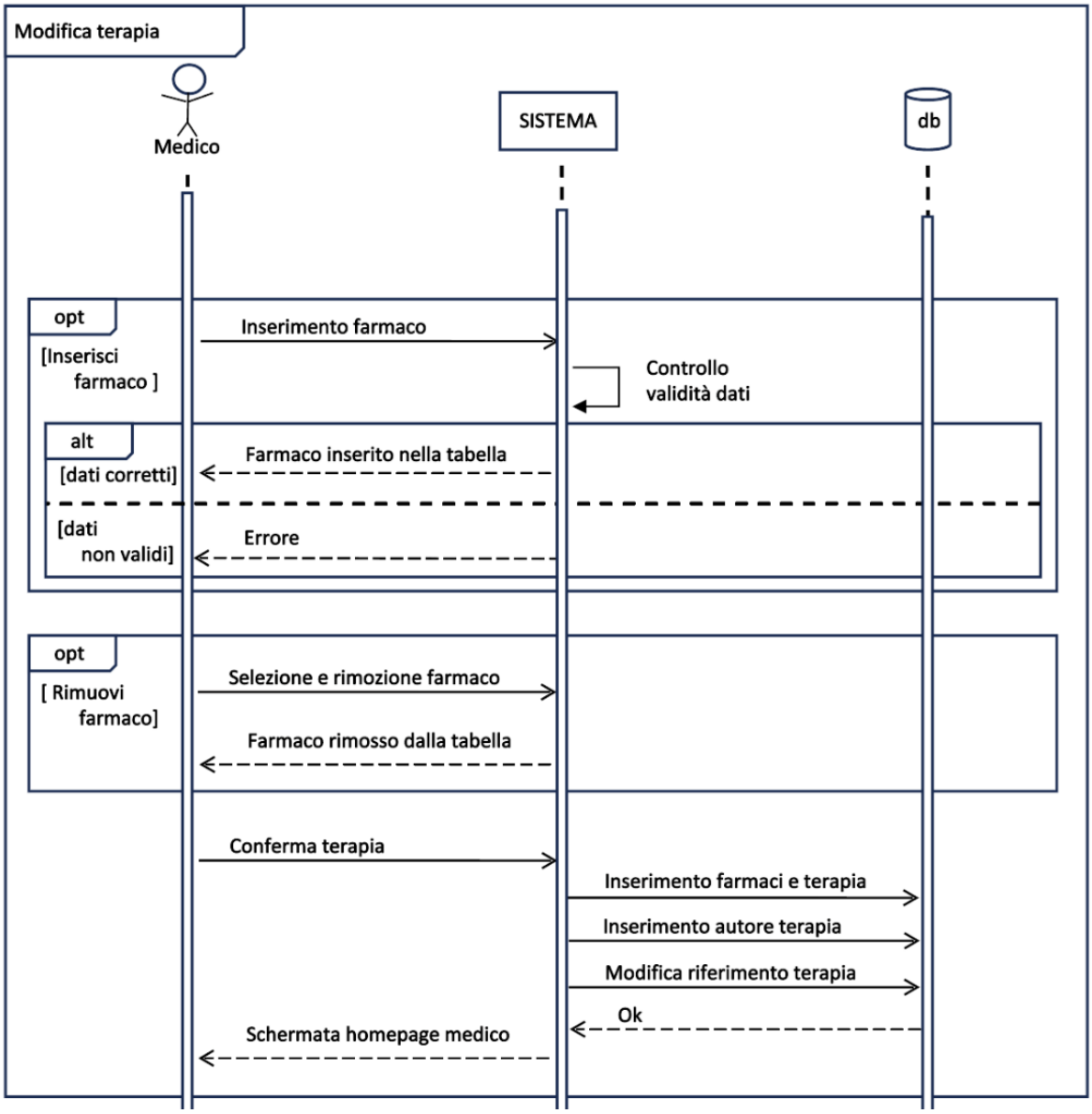

Figura 5 - Modifica terapia

1.1.3 Casi d'uso relativi ai Pazienti

1.1.3.1 Note generiche

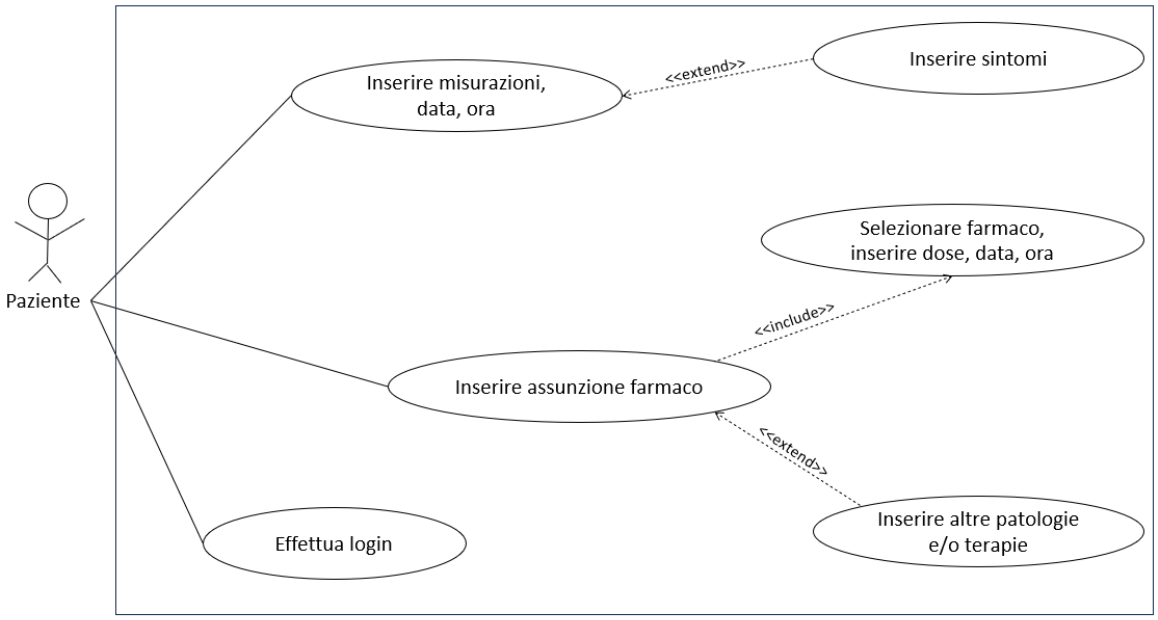

Figura 6 - Casi d'uso paziente

Dopo opportuno accesso tramite codice fiscale e selezione della opportuna categoria, il paziente viene introdotto ad una interfaccia che permette di selezionare uno dei seguenti servizi:

- Inserimento di una nuova misurazione
- Inserimento di un nuovo farmaco assunto
- Disconnessione

Sono presenti 3 bottoni: "Inserisci misurazione", "Inserisci farmaco", Logout".

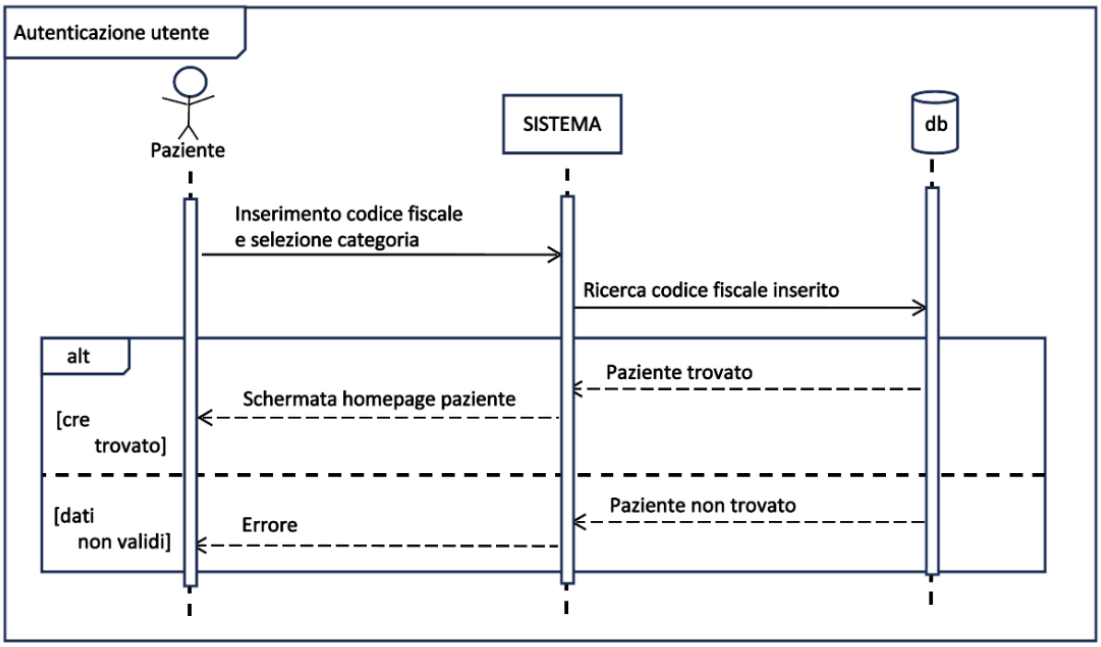

Figura 7 - Autenticazione paziente

1.1.3.2 Inserimento di una misurazione

Il paziente deve poter inserire i dati relativi alle misurazioni giornaliere.

Attori: *Paziente*
Precondizioni: Il paziente dev'essersi autenticato.

Passi:

1. *Il paziente accede al sistema.*
2. *Il paziente è introdotto alla home page.*
3. *Il paziente accede al servizio "Inserisci misurazione".*
4. *Il paziente inserisce la data e l'ora.*
5. *Il paziente inserisce i dati relativi alle misurazioni della pressione Sistolica e Diastolica.*
6. *Il paziente seleziona eventuali sintomi.*
7. *Il paziente può confermare l'inserimento o annullare la modifica tornando indietro.*

Post condizioni: *La misurazione ed i sintomi devono essere stati memorizzati*

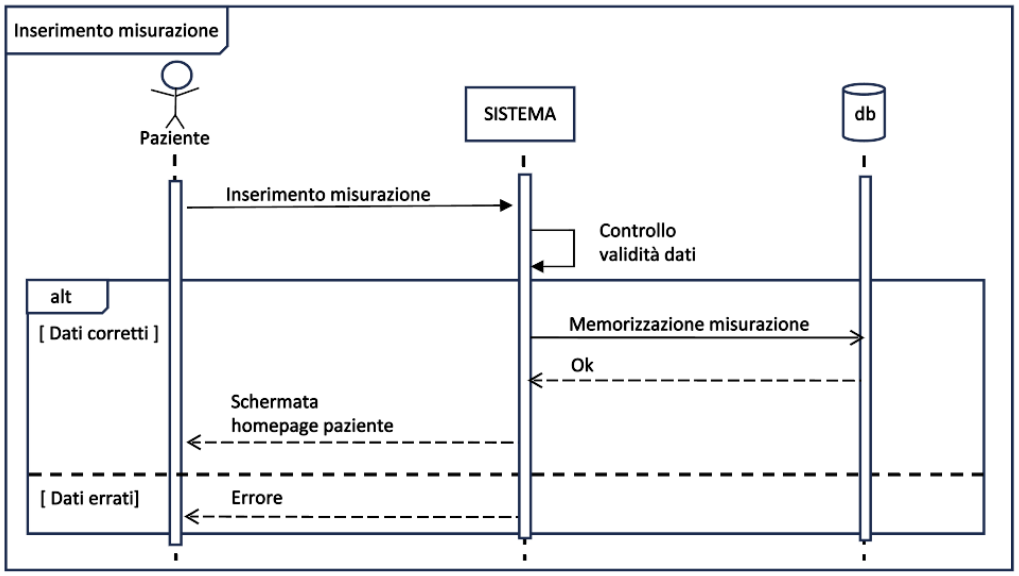

Figura 8 - Inserimento misurazione

1.1.3.3 Inserimento assunzione di un farmaco

Il paziente deve poter inserire i farmaci assunti.

Attori: *Paziente*
Precondizioni: Il paziente deve avere una terapia associata.

Passi:

1. *Il paziente accede al sistema.*
2. *Il paziente è introdotto alla home page.*
3. *Il paziente accede al servizio "Inserisci farmaco"*
4. *Il paziente seleziona il farmaco da inserire.*
5. *Il paziente seleziona la data.*
6. *Il paziente inserisce l'ora di assunzione.*
7. *Il paziente inserisce la dose.*
8. *Il paziente può indicare patologie/terapie concomitanti.*
9. *Il paziente conferma o annulla l'inserimento.*

Post condizioni: *Il farmaco assunto deve essere stato memorizzato*

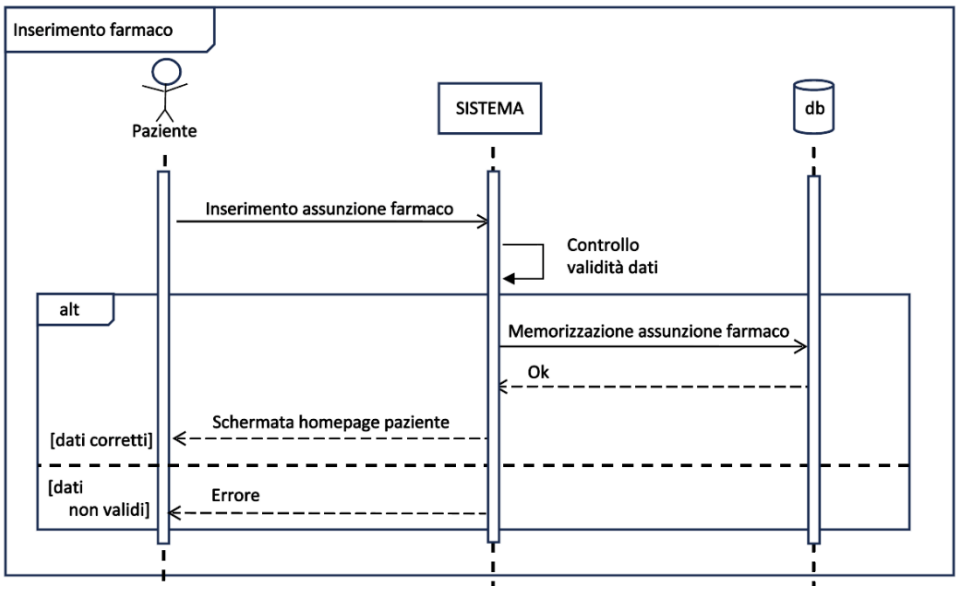

Figura 9 - Inserimento farmaco

1.2 ACTIVITY DIAGRAM

***Nota**: i seguenti diagrammi catturano una singola attività di un utente rispetto al sistema. Non è stata rappresentata nel diagramma la possibilità di ripetere più volte la stessa operazione, in sequenza, senza chiudere il software. Questo per semplici ragioni di chiarezza e leggerezza; si considerano quindi le singole attività d'interazione*

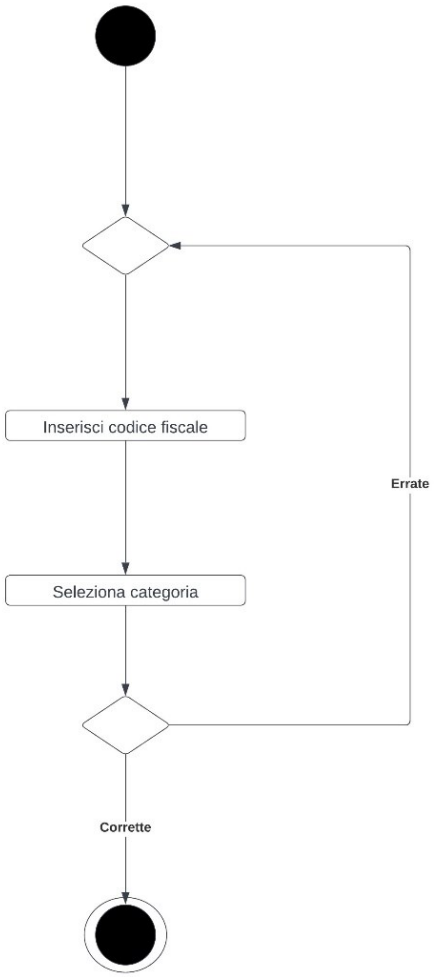

Figura 10 - Autenticazione

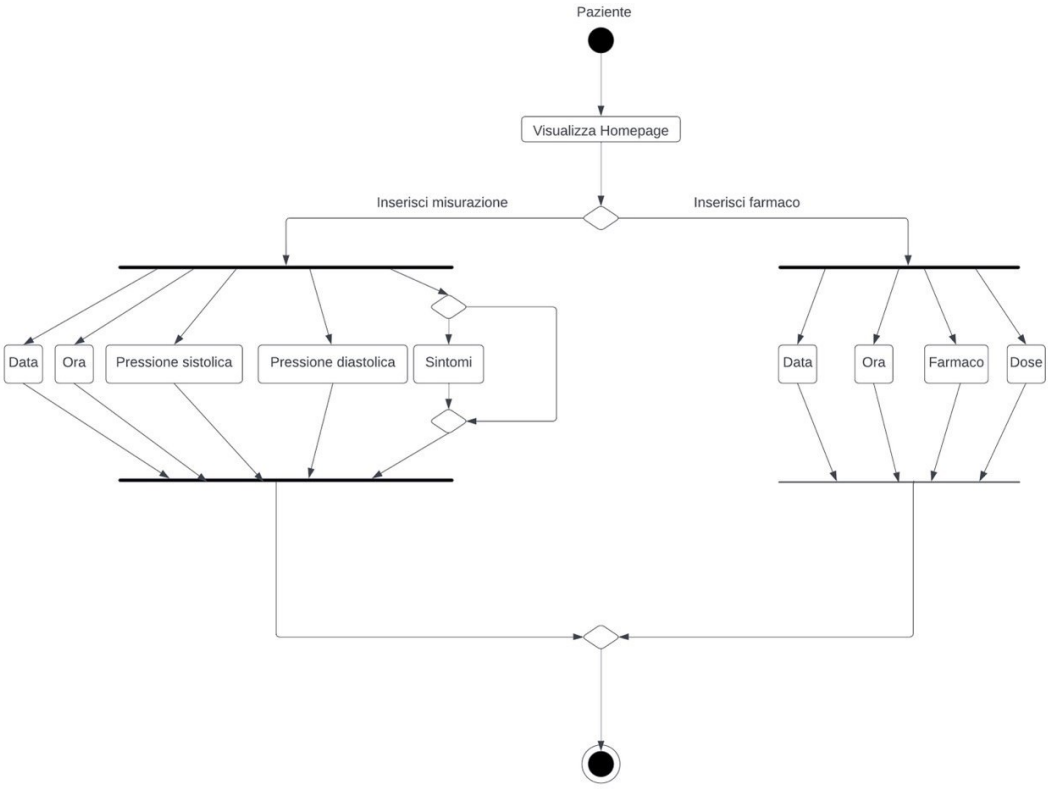

Figura 11 - Paziente

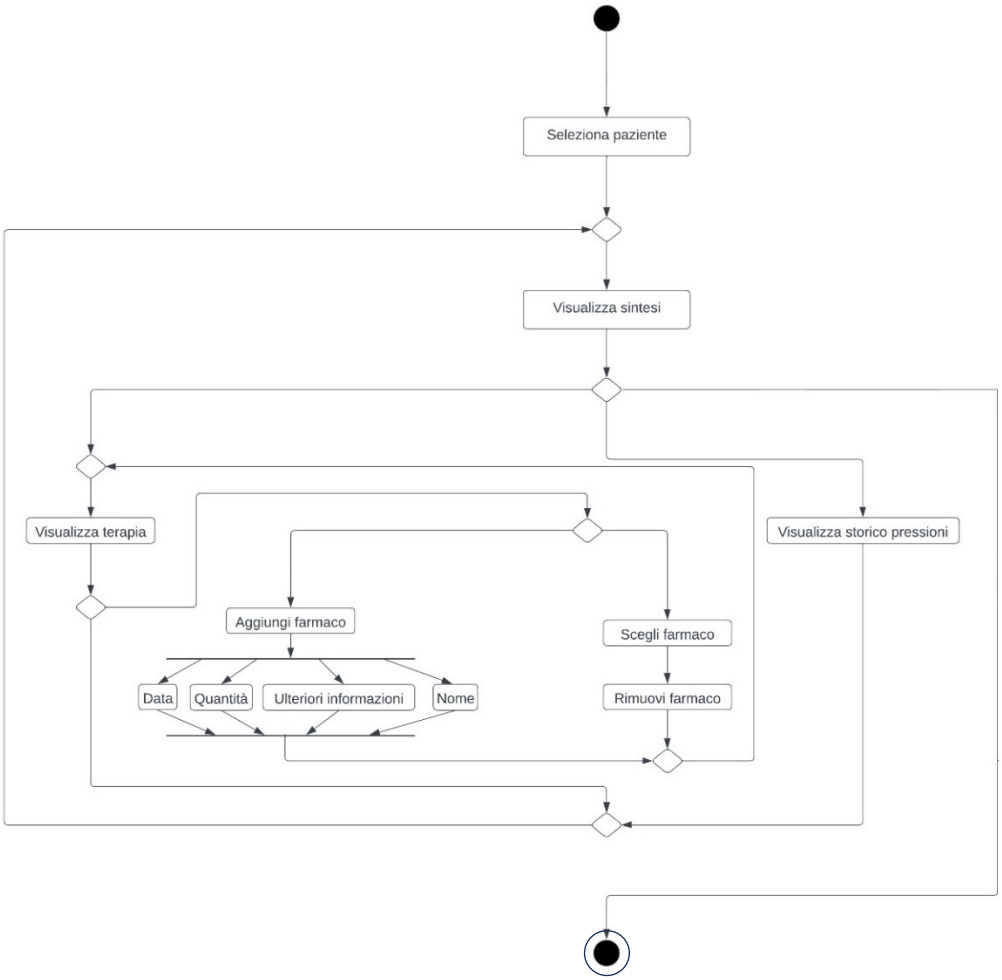

Figura 12 - Medico

1.3 CLASS DIAGRAM

1.3.1 Note Generali

Il Class Diagram mostra le classi di oggetti nel sistema e le associazioni tra queste classi.

1.3.2 Elenco campi entità Model

L'elenco dei campi rappresenta le classi con i loro rispettivi nomi ed i loro attributi. Gli attributi inseriti sono quelli richiesti dal progetto insieme ad altri attributi.

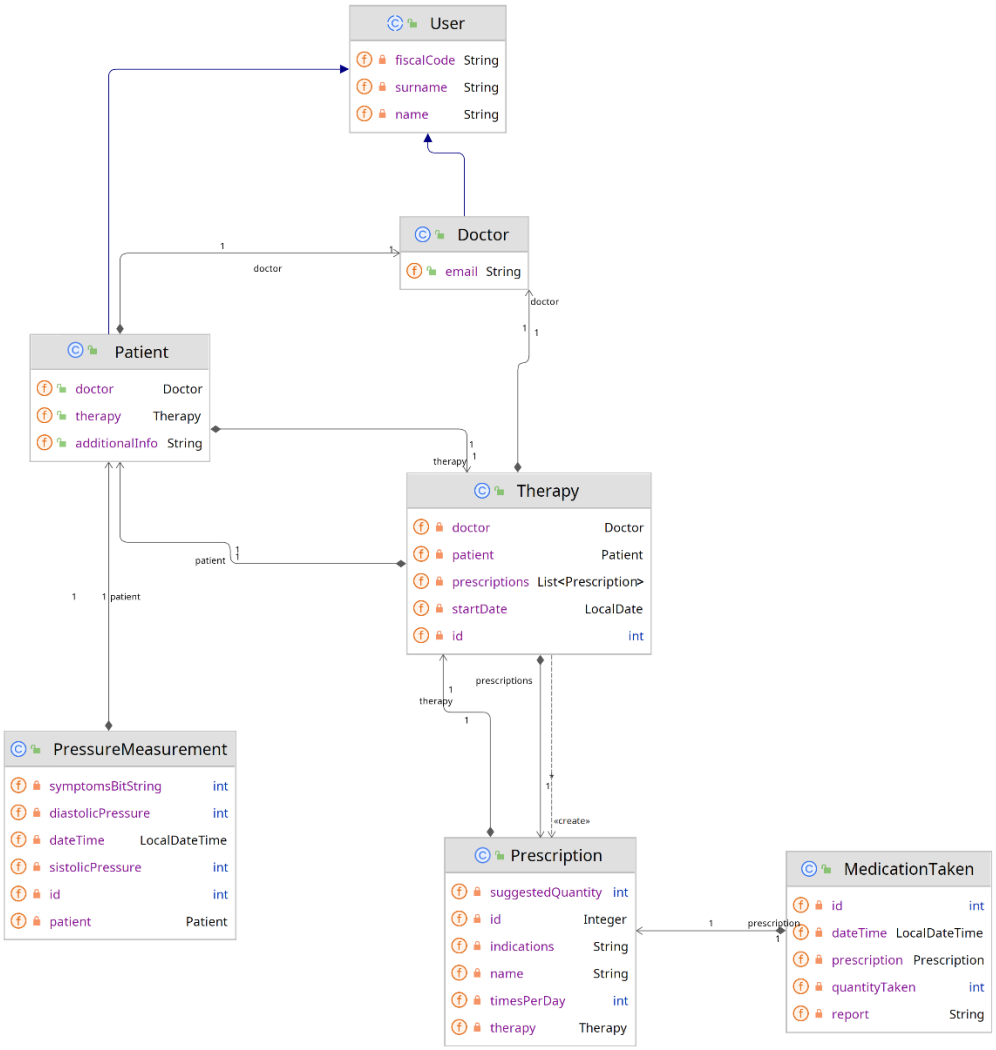

Figura 13 - Package model

1.3.3 Elenco metodi per ogni entità

Per avere una maggiore leggibilità del codice abbiamo adottato una classificazione delle entità tramite package. Possiamo qui di seguito osservare una visione dei package in cui è diviso il software.

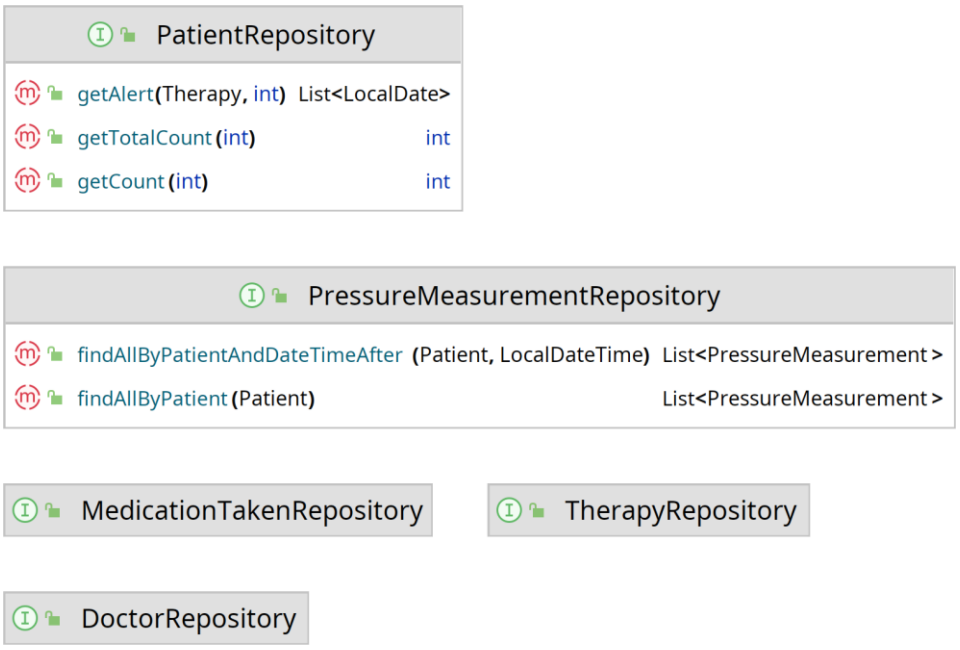

Figura 14 - Package repository

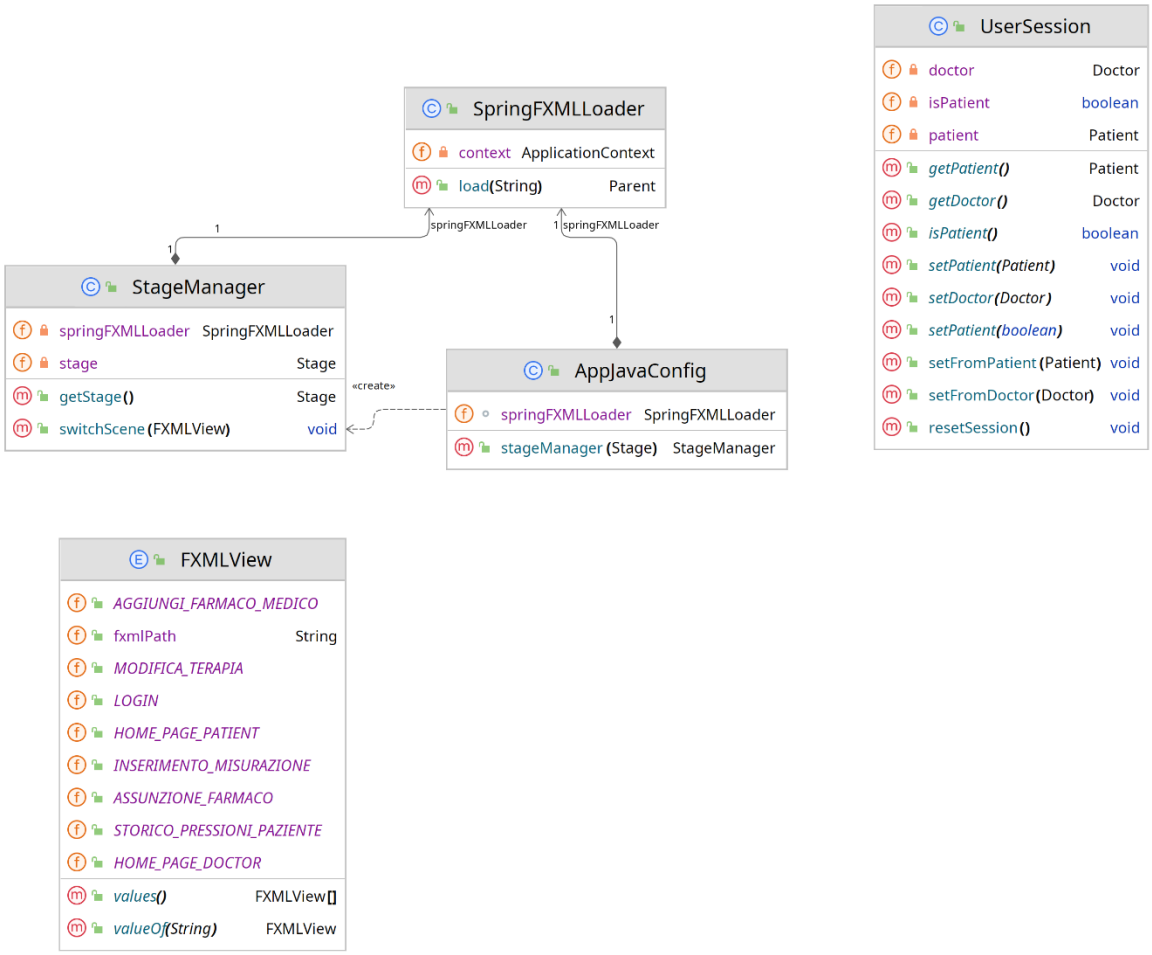

Figura 15 - Package utils

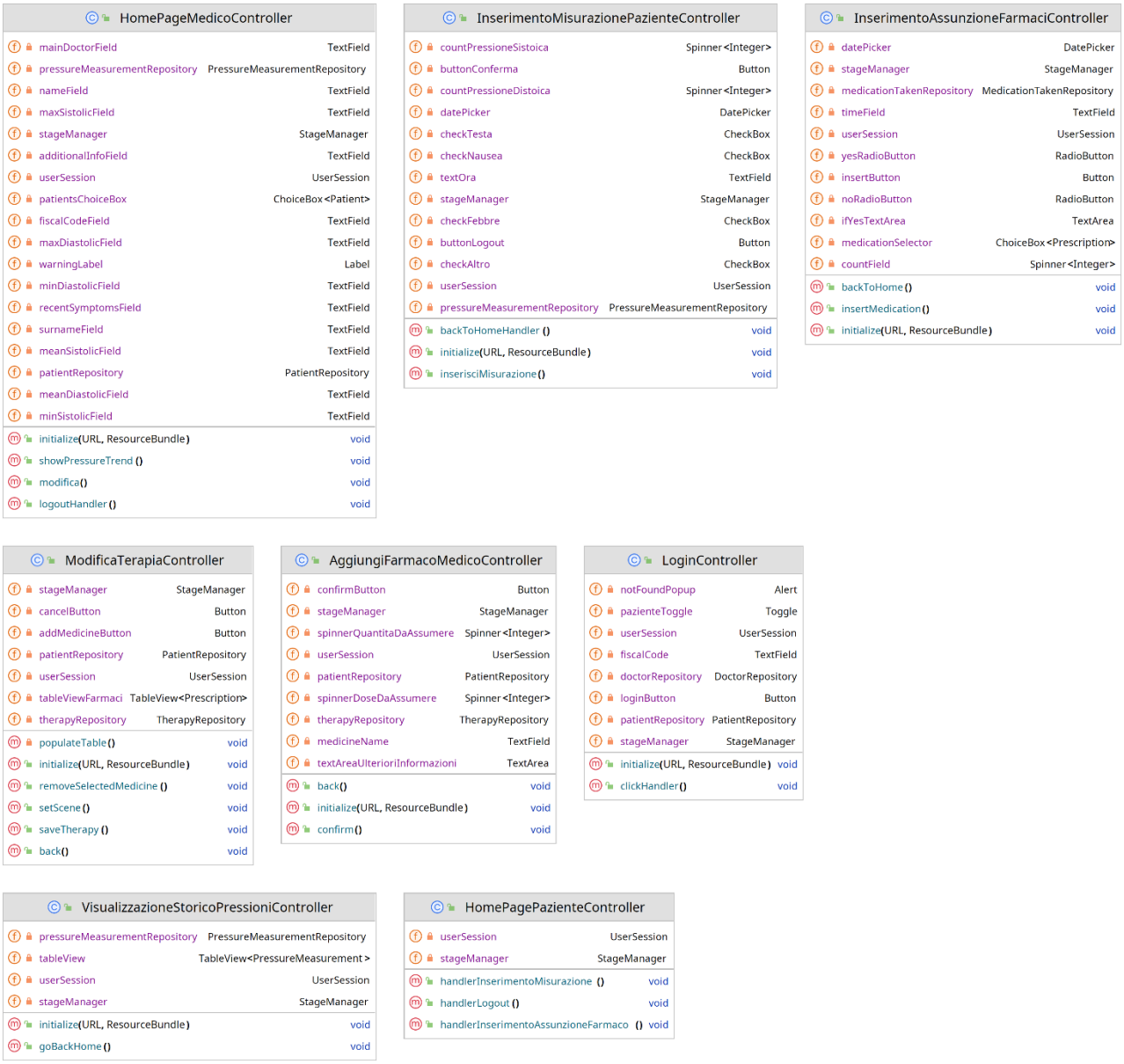

Figura 16 - Package controller

2 SVILUPPO:
PROGETTO DELL'ARCHITETTURA ED IMPLEMENTAZIONE DEL SISTEMA

2.1 NOTE SUL PROCESSO DI SVILUPPO

Il processo di sviluppo è stato essenzialmente incrementale di tipo plan-driven. Le attività sono state inizialmente programmate con una prima implementazione successivamente perfezionata. Prima di iniziare il processo di sviluppo c'è stata una fase iniziale di analisi dei requisiti, generando i relativi use-case e i diagrammi delle attività. Inoltre, è stata usata la piattaforma *GitHub* per il versioning distribuito del codice. Dopo ogni sostanziale implementazione/versione sono stati eseguiti gli appositi test. Non sempre le fasi di sviluppo sono state lineari, infatti sono state intervallate da numerose attività di refactoring.

2.2 IMPLEMENTAZIONE E PATTERN ARCHITETTURALI USATI

È stato realizzato in Java mediante l'utilizzo delle librerie JavaFX per la parte grafica e Spring Boot per l'accesso al database e gestione dei controller.

Nel nostro caso l'applicazione è un event processing system ovvero un sistema che risponde agli eventi dell'interfaccia utente. La caratteristica principale degli event processing systems è che la tempistica degli eventi è imprevedibile e il sistema deve essere in grado di far fronte a questi eventi quando si verificano. Il sistema rileva e interpreta gli eventi. Gli eventi dell'interfaccia utente rappresentano comandi impliciti per il sistema, che esegue alcune azioni per obbedire a quel comando. Nello specifico essendo un'applicazione grafica abbiamo usato uno stile architetturale event-driven ovvero un paradigma di architettura software che promuove la produzione, il rilevamento, il consumo e la reazione agli eventi. Oltre a ciò, abbiamo scelto il pattern architetturale MVC.

Oltre alla struttura presentata sopra Spring si basa sul principio di Inversion of Control: un principio dell'ingegneria del software che trasferisce il controllo degli oggetti o porzioni di un programma a un container o un framework.
In contrasto con la programmazione tradizionale, dove il nostro codice richiama le funzionalità di una libreria, IoC permette al framework di prendere il controllo del flusso di esecuzione ed effettuare chiamate al nostro codice. In Spring Boot il principio IoC è implementato attraverso il meccanismo detto Dependency Injection.

2.2.1 Dependency Injection (DI)

È un design pattern che possiamo usare per implementare IoC(Inversion of Control), dove il controllo invertito è quello del settaggio delle dipendenze di un'oggetto. Connettere gli oggetti con altri oggetti, o "iniettare" oggetti dentro altri oggetti, viene fatto da un'assembler invece che dagli oggetti stessi.

2.2.2 Spring Bean

Quando parliamo di DI in Spring non possiamo esimerci dal parlare anche dei Bean. Questi sono gli oggetti che formano la struttura dell'applicazione e che sono gestiti da Spring IoC container. Un bean è un'oggetto che è istanziato, assemblato e gestito da Spring IoC container. Nel nostro programma i Bean sono i Controller ed i Repository. Se non specificato altrimenti, i Bean inoltre implementano il pattern Singleton.

2.2.3 Singleton

Il pattern singleton è un meccanismo che assicura che esista solo un'istanza di un oggetto per applicazione.

Quando il costruttore viene richiamato per la prima volta l'oggetto viene istanziato, successivamente viene ritornato.

2.2.4 Data Access Object

Il pattern DAO separa la parte di accesso ai dati rispetto ai servizi di più alto livello. Questo pattern fa riferimento alla collezione di oggetti.
L'interfaccia Data Access Object definisce le operazioni standard da effettuare sugli oggetti del modello definiti nel package model. La classe concreta Data Access Object generata dal framework Spring Boot è invece implementa la logica di accesso al database.
Il POJO (Plain Old Java Object) è un oggetto che rappresenta un pezzo dell'informazione che si sta considerando attraverso un insieme di metodi per accedere e modificare gli attributi dell'oggetto (metodi get e set).

2.2.5 Model-View-Control

È un pattern architetturale che separa la rappresentazione e l'interazione dai dati del sistema. Il sistema è strutturato in tre componenti logiche che interagiscono tra loro.

- *Model*: gestisce i dati di sistema e le operazioni associate su quei dati
- *View*: definisce e gestisce il modo in cui i dati sono presentati all'utente
- *Control*: gestisce l'interazione dell'utente (ad esempio, pressioni di tasti, clic del mouse, ecc.) e passa queste interazioni alle altre componenti

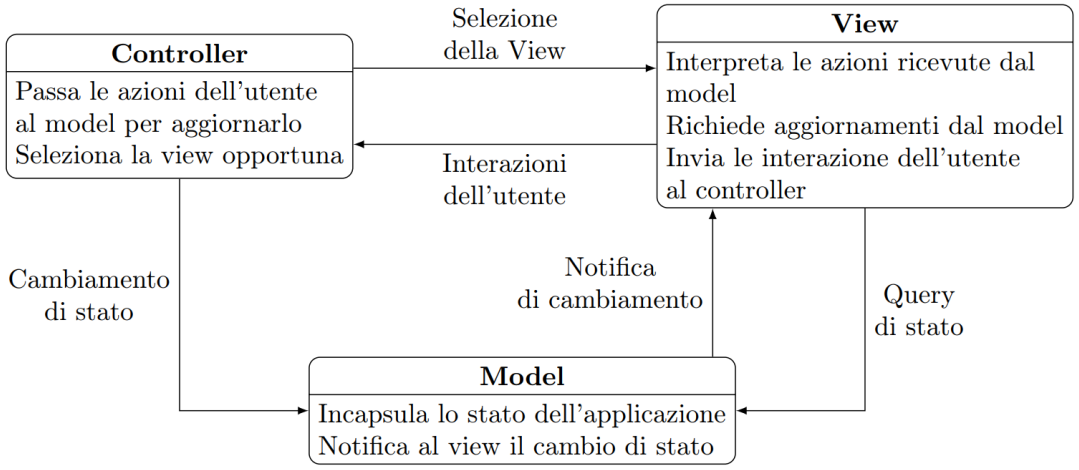

Figura 17 - MVC pattern

Nel nostro progetto il controller corrisponde alle classi Controller, la View ai file FXML e il model alle entità. Il Dependecy Injection è usato per iniettare nel controller i componenti grafici e gli oggetti Singleton chiamati StageManager per gestire le interfacce mostate e UserSession per memorizzare e traferire i dati dell'utente tra un controller e l'altro.

2.3 PROGETTO DELL'ARCHITETTURA ED IMPLEMENTAZIONE DEL DATABASE

Per lo sviluppo del sistema è stato il database relazione H2 su file e la libreria Spring Data JPA per interfacciare il sistema al Database. La logica del database è stata di tipo incrementale seguita da numerosi refactoring per implementare nuove funzionalità. Questo ha portato al non seguire uno schema logico. Di sotto possiamo trovare un piccolo schema dell'attuale database.

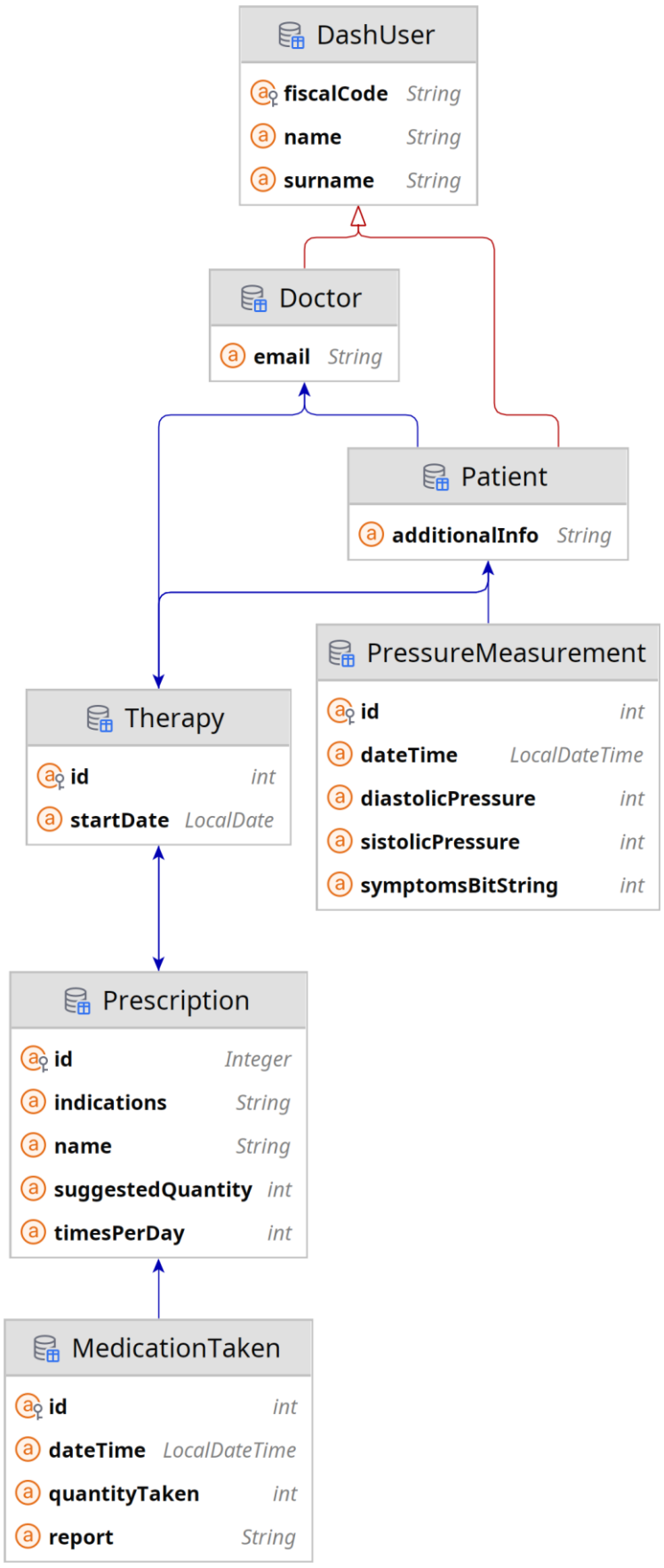

Figura 18 - Entity Relationship Diagram

3 ATTIVITÀ DI TEST E VALIDAZIONE

3.1 GENERALITÀ

Il test ha lo scopo di dimostrare che un software fa quello che è destinato a fare e scoprirne i difetti prima che venga messo in uso.

3.2 END-TO-END TESTING

Questi test sono stati eseguiti solo sui Controller. In questo tipo di test viene avviato l'intero applicativo senza scorporarlo dalle dipendenze e quindi viene testato il sistema nel suo insieme.

Per effettuare questi test abbiamo utilizzato le seguenti librerie:

- **JUnit5**: è al centro dell'infrastruttura di testing e serve principalmente come base per definire le funzioni di test, funzioni di supporto e le classi di test mediante le apposite annotazioni.
- **AssertJ**: è una libreria per semplificare la scrittura e migliorare la leggibilità delle asserzioni nei test. Un'asserzione è un'espressione che incapsula una logica verificabile specificata su un obiettivo sottoposto a test.
- **TestFX**: consente di interagire con l'interfaccia simulando passo dopo passo le azioni di un utente. Inoltre, permette di interrogare la GUI per verificare che l'esistenza e lo stato dei controlli ed altri elementi grafici che compongono l'interfaccia.

Inoltre, anche durante i test è stato utilizzato il pattern Dependency Injection per iniettare le dipendenze e gli oggetti fittizi nelle classi di test.

Vediamo un esempio di classe di test in cui viene utilizzata la libreria TestFX. L'obiettivo di questo test è verificare se un paziente registrato può effettuare correttamente il login.

```java
import ...;

@ExtendWith(ApplicationExtension.class)
@SpringBootTest(classes = IpertesiApplication.class)
class LoginControllerIntegrationTest {
    @Autowired
    PatientRepository patientRepository;
    @Autowired
    ConfigurableApplicationContext springContext;

    @Start
    void start(Stage stage) {
        StageManager stageManager = springContext.getBean(StageManager.class, stage);
        stageManager.switchScene(FXMLView.LOGIN);
        stage.show();
    }

    @Test
    void testPatientIsRegistered(FxRobot robot) {
        String fiscalCode = "RSSMRA80L05F593A";
        // inserisco un paziente nel database
        Patient patient = new Patient();
        patient.setFiscalCode(fiscalCode);
        patientRepository.save(patient);

        // digito il suo codice fiscale
        robot.clickOn("#fiscalCode").write(fiscalCode);
        robot.clickOn("#patientRadioButton");
        // premo su login
        robot.clickOn("#loginButton");

        // verifico se è stato effettuato il cambio schermata
        FxAssert.verifyThat("#patientPane", Node::isVisible);
    }
}
```

Figura 19 - Esempio di test

La classe è annotata con @ExtendWith(ApplicationExtension.class) che permette di utilizzare l'estensione di TestFX per eseguire i test su applicazioni JavaFX. Inoltre, è annotata con @SpringBootTest(classes = IpertesiApplication.class) che avvia l'applicazione Spring durante l'esecuzione del test.

La classe contiene due metodi: start e testPatientIsRegistered.

Il metodo start viene annotato con @Start e riceve un parametro di tipo Stage. Questo metodo viene eseguito prima dell'avvio dei test e ha il compito di inizializzare lo stage della JavaFX application. All'interno del metodo, viene ottenuto un bean chiamato StageManager dal contesto di Spring e viene invocato il metodo switchScene su di esso per passare alla scena di login.

Il metodo testPatientIsRegistered è annotato con @Test e riceve un parametro di tipo FxRobot. Questo metodo rappresenta il test vero e proprio. Inizia creando una stringa fiscalCode che rappresenta il codice fiscale di un paziente registrato. Viene quindi creato un oggetto Patient e viene impostato il codice fiscale. Successivamente, l'oggetto Patient viene salvato nel repository utilizzando patientRepository.save(patient).

A questo punto, il robot TestFX interagisce con l'interfaccia utente simulando il comportamento dell'utente. Vengono simulate le azioni di cliccare sul campo di testo "fiscalCode" e di inserire il codice fiscale. Viene inoltre simulato il clic sul pulsante "patientRadioButton" per selezionare il tipo di utente come paziente. Infine, viene simulato il clic sul pulsante "loginButton" per effettuare il login.

Dopo aver simulato il login, viene effettuata una verifica utilizzando FxAssert.verifyThat per verificare se l'elemento con l'id "patientPane" è visibile. Se l'elemento è visibile, significa che il cambio di schermata è avvenuto correttamente. In caso contrario viene generato un errore da JUnit per segnalare l'anomalia allo sviluppatore.

3.3 RELEASE TESTING

Per la fase di Release testing abbiamo fatto usare il software ad alcuni colleghi, i quali, oltre a verificare la corretta funzionalità del software ci hanno fatto alcuni consigli per migliorare la qualità del codice. Alcuni dei test svolti sono:

- Verifica del corretto funzionamento dell'autenticazione: i codici fiscali errati non vengono accettati, gli utenti non registrati vengono respinti ed a fronte dei dati corretti all'utente viene mostrata la schermata iniziale alla categoria di appartenenza.
- Paziente senza una terapia: si è provato ad inserire un'assunzione di un farmaco in assenza di una terapia, per vedere se questo creasse eccezioni o errori.
- Medico a database vuoto: si è provato a pigiare ogni pulsante pigiabile in assenza assoluta di dati, per vedere se questo creasse eccezioni o errori.
- Inserimento di una misurazione da parte di un paziente. Verifica che le date inserite siano congrue. Verifica che tale misurazione sia poi visibile dai tutti i medici e che sia categorizzata correttamente.
- Verificato che ogni medico possa modificare una terapia e che l'autore della modifica sia registrato nel database.
- Verifica di vari inserimenti di input errati, input vuoti, stringhe malformate, numeri negativi, date future.

3.4 USER TESTING

Come ultimo scaglione il software è stato sottoposto ad un breve test da parte di alcuni individui con limitata dimestichezza informatica. In questa fase non si è cercato in nessun modo di guidare o strutturare l'esperienza, per non influenzare in alcun modo il risultato; piuttosto si è lasciato che il soggetto navigasse liberamente il sistema. È stata fornita loro solo una breve descrizione dello scopo del software. L'unico scopo del test era quello di individuare errori e migliorare la User Experience.

Grazie a quest'ultimo test sono state inoltre individuate anche alcune funzionalità aggiuntive per il software tra cui:

- Visualizzazione dello storico delle misure di pressioni sanguigna relative ad un paziente con classificazione della tipologia di ipertensione anche a colpo d'occhio anche attraverso i colori.
- Visualizzazione di alcune informazioni chiave relative all'utente corrente nel titolo dell'applicazione.
- Visualizzazione del medico di riferimento nella schermata relativa ai dati anagrafici di un paziente.