Documentazione progetto Pazienti Ipertesi

Alex Gaiga, VR471343
Michele Cipriani, VR471337
Tommaso Vilotto, VR471487

Luglio 2023

Indice

Interazione con il sistema

Casi d’uso

Il sistema sviluppato permette l’interazione di due attori principali, i medici e i pazienti ipertesi. Entrambi gli attori hanno in loro possesso delle credenziali di accesso al sistema con cui possono autenticarsi.
Qualora venga riconosciuta una tupla d’accesso valida verra’ caricata una schermata relativa all’attore (paziente o medico) per il quale sono riconosciute le credenziali.

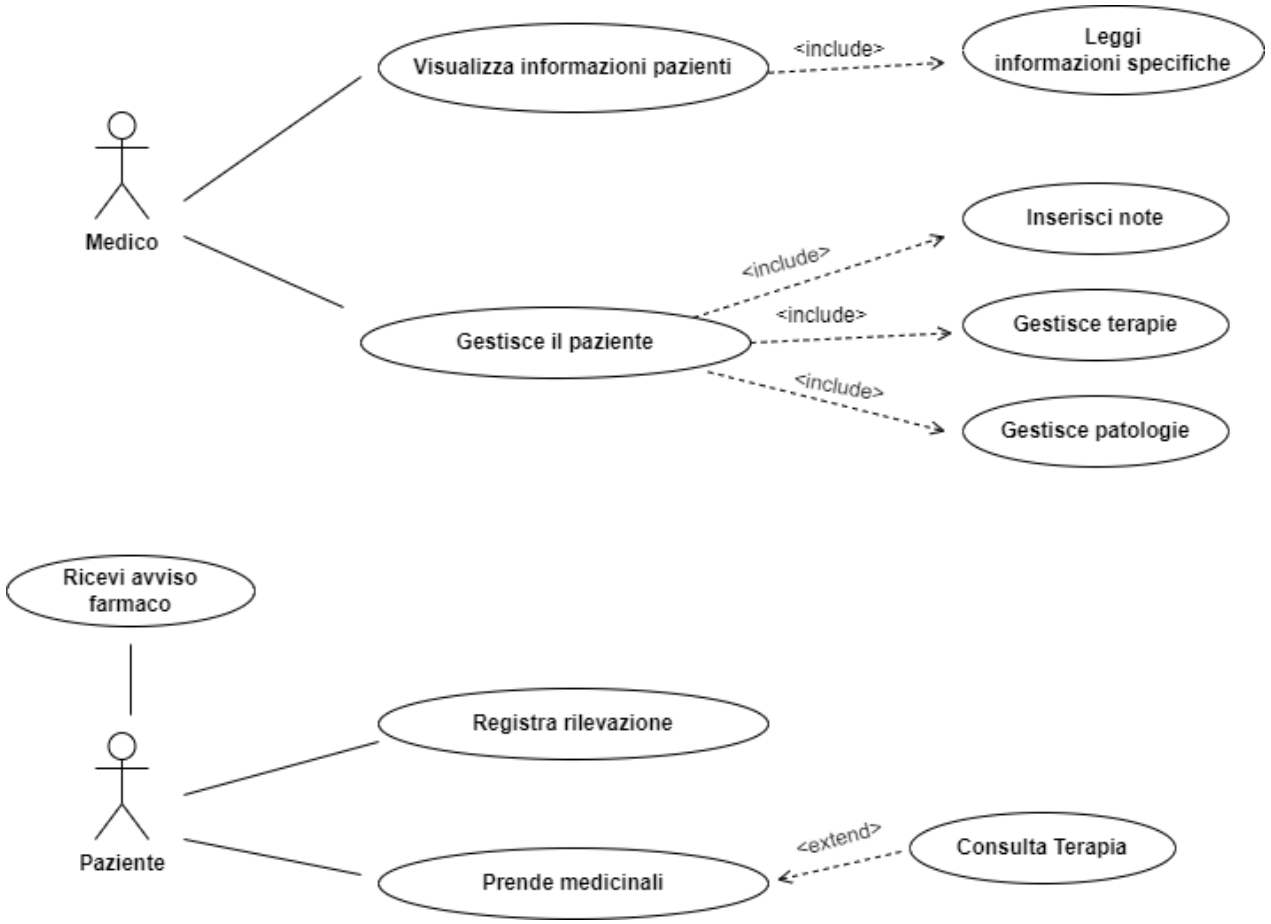

Figure 1: casi d’uso

Casi d’uso Medici

Dopo opportuna autenticazione, viene proposta ai medici un’interfaccia nella quale possono visualizzare in breve o nello specifico le generalità dei pazienti, aggiornare le patologie correlate e gestirne eventuali terapie.

Visualizzazione info generiche pazienti

I medici devono poter visualizzare tutte le informazioni dei pazienti che seguono e consultarne il relativo grado di ipertensione.

Caso d’uso: **VisualizzaInformazioni**
Attori: Medico
Precondizioni: Il medico deve essere autenticato
Passi: 1. Il medico accede al sistema 2. Il medico visualizza l’interfaccia di base 3. Il medico accede alla seconda scheda
Postcondizioni: Il medico consulta le informazioni generiche e il grado di ipertensione di ogni paziente da lui seguito

Visualizzazione info specifiche paziente

I medici devono poter visualizzare tutte le informazioni per esteso di ogni paziente che desiderano consultare, con particolare attenzione al grado di ipertensione e allo stato della terapia, ovvero se il paziente sta osservando correttamente le prescrizioni.

***Caso d'uso:* VisualizzaInformazioniSpecifiche**
Attori: Medico
Precondizioni: Il medico deve essere autenticato
Passi: 1. Il medico accede al sistema 2. Il medico visualizza l'interfaccia di base 3. Il medico accede alla seconda scheda 4. Il medico cerca il paziente tramite il relativo codice fiscale. 5. Il medico viene introdotto in un'altra schermata
Postcondizioni: Il medico consulta le informazioni specifiche del paziente interessato

Inserimento note paziente

I medici devono poter inserire informazioni generiche per ogni paziente da loro seguito.

***Caso d'uso:* InserisciAnnotazioneGenericaPaziente**
Attori: Medico
Precondizioni: Il medico deve aver selezionato il paziente
Passi: 1. Il medico inserisce l'annotazione generica 2. Il medico seleziona il tasto "Add Info"
Postcondizioni: Annotazione inserita

Gestione patologie paziente

I medici devono poter inserire patologie pregresse\in corso per ogni paziente da loro seguito.
In questa fase, vi sono due alternative:

- registrare una nuova patologia
- visualizzare e modificare una patologia già registrata

Caso d'uso: **GestionePatologiePaziente**
Attori: Medico
Precondizioni: Il medico deve aver selezionato il paziente
Passi: 1. Se il medico vuole aggiungere una patologia 1.1 Il medico seleziona una nuova patologia dall'apposito selettore 1.2 Il medico compila i campi rimanenti 1.3 Il medico seleziona il tasto "Add pathology" 2. Se il medico vuole modificare una patologia già esistente 2.1 Il medico seleziona una patologia già esistente 2.2 Il medico modifica/compila i campi rimanenti 2.3 Il medico seleziona il tasto "Modify pathology"
Postcondizioni: Patologia aggiunta/aggiornata

Visualizzazioni dati grafici pressione e sintomi paziente

I medici devono poter visualizzare l'andamento della pressione (SBP/DBP) di ogni paziente, selezionando il periodo e visualizzando relativi sintomi registrati.

***Caso d'uso:* VisualizzaGraficoPressione**
Attori: Medico
Precondizioni: Il medico deve aver selezionato il paziente
Passi: 1. Il medico accede alla seconda scheda 2. Il medico seleziona il periodo temporale interessato 3. Il medico seleziona il tasto "Show Data"
Postcondizioni: Il medico visualizza il grafico mostrato a video

Gestione terapie paziente

I medici devono poter inserire terapie da far seguire ad ogni paziente da loro seguito.
In questa fase, vi sono due alternative:

- registrare una nuova terapia
- visualizzare e modificare una terapia già registrata

Caso d'uso: **GestioneTerapiePaziente**
Attori: Medico
Precondizioni: Il medico deve aver selezionato il paziente
Passi: 1. Se il medico vuole aggiungere una nuova terapia 1.1 Il medico compila una nuova terapia 1.2 Il medico seleziona il tasto "+" 2. Se il medico vuole modificare una terapia già esistente 2.1 Il medico seleziona una terapia già esistente 2.2 Il medico seleziona il tasto *Simbolo penna* 2.3 Il medico modifica i campi desiderati 2.4 Il medico Il medico seleziona il tasto "+" per confermare o "x" per annullare
Postcondizioni: Terapia aggiunta/aggiornata

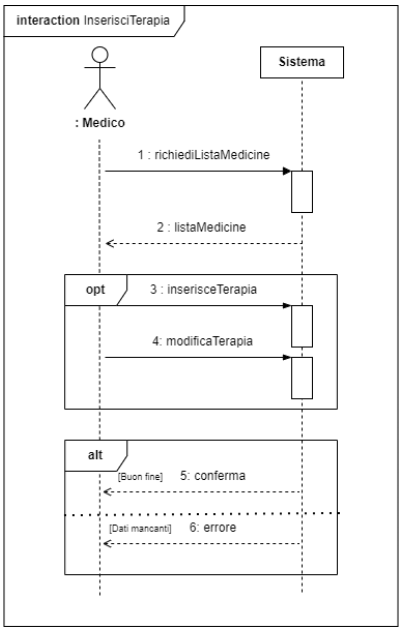

Figure 2: Registrazione terapia

Casi d’uso Pazienti Ipertesi

Dopo opportuna autenticazione, viene proposta ai pazienti un’interfaccia nella quale, dopo essere stati avvisati delle medicine giornaliere che devono ancora prendere, possono registrare le rilevazioni di pressione (SBP/DBP), segnalare le assunzioni dei medicinali prescritti e contattare il medico che li segue.

Avviso medicinali giornalieri da assumere

Il paziente deve essere avvisato, ogni qual volta che esegue l’autenticazione, dei medicinali giornalieri che deve ancora assumere.

***Caso d’uso:* AvvisoMedicinaliGiornalieri**
Attori: Paziente
Precondizioni: Il paziente deve essere autenticato
Passi: 1. Il paziente visualizza la schermata di avviso contente un promemoria con i medicinali che deve ancora assumere
Postcondizioni: Il paziente per poter continuare la navigazione nel sistema deve chiudere la schermata di avviso tramite relativo tasto “ok”

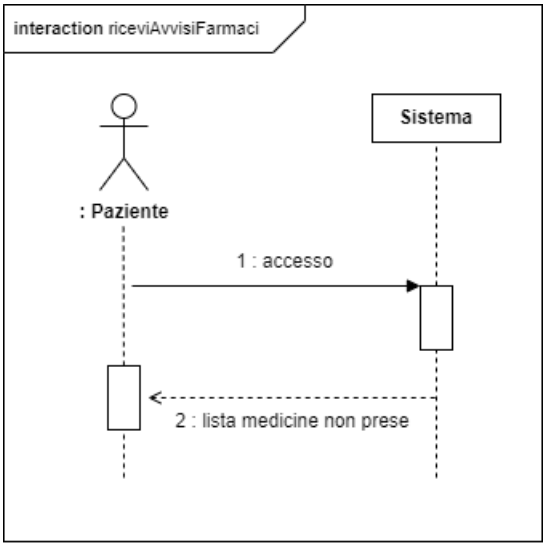

Figure 3: Avviso farmaci

Registrazione rilevazione pressione

Il paziente deve poter inserire le rilevazioni giornaliere di pressione arteriosa.

Caso d'uso: **RegistraRilevazionePressione**
Attori: Paziente
Precondizioni: Il paziente deve essere autenticato
Passi: 1. Il paziente accede alla seconda scheda 2. Il paziente inserisce le rilevazioni di SBP e DBP 3. Il paziente puo' selezionare eventuali sintomi legati alla rilevazione (CTRL per selezione multipla) 4. Il paziente seleziona l'icona del "dischetto" per salvare
Postcondizioni: Rilevazione aggiunta

Registrazione assunzione medicinali

Il paziente deve poter inserire i medicinali giornalieri prescritti dal medico.

Caso d'uso: **RegistraAssunzioneMedicinali**
Attori: Paziente
Precondizioni: Il paziente deve essere autenticato
Passi: 1. Il paziente accede alla terza scheda 2. Il paziente seleziona la medicina da assumere 3. Il paziente legge nel box informativo le relative indicazioni 4. Il paziente seleziona il tasto "Take it"
Postcondizioni: Assunzione medicina aggiunta

Email al medico

Il paziente deve poter contattare il medico tramite l'indirizzo *mail* che quest'ultimo ha fornito al sistema.

Caso d'uso: **EmailAlMedico**
Attori: Paziente
Precondizioni: Il paziente deve essere autenticato
Passi: 1. Il paziente seleziona l'icona della casella di posta 2. Il paziente invia la mail
Postcondizioni: Mail inviata al medico associato

Diagrammi di attività

Nota: i diagrammi di attività proposti descrivono una singola attività di interazione di un utente rispetto al sistema.

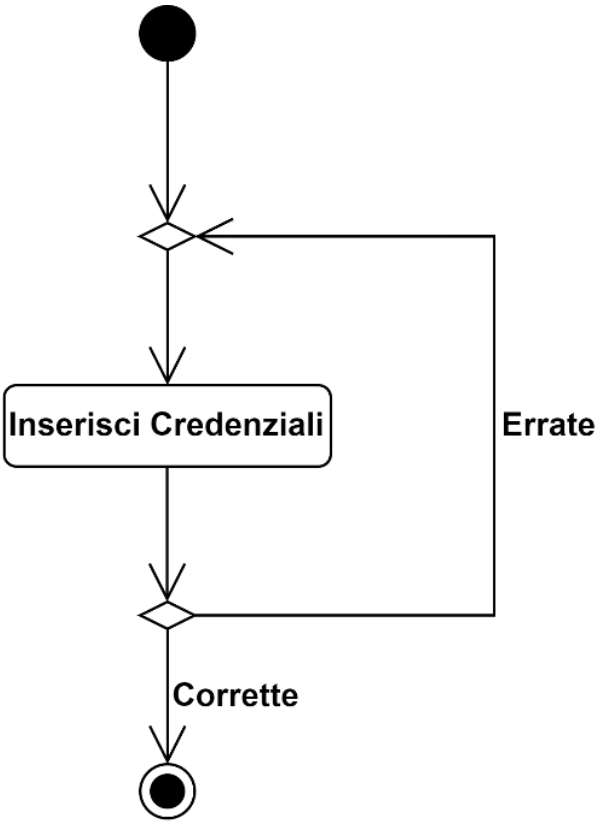

Figure 5: Autenticazione

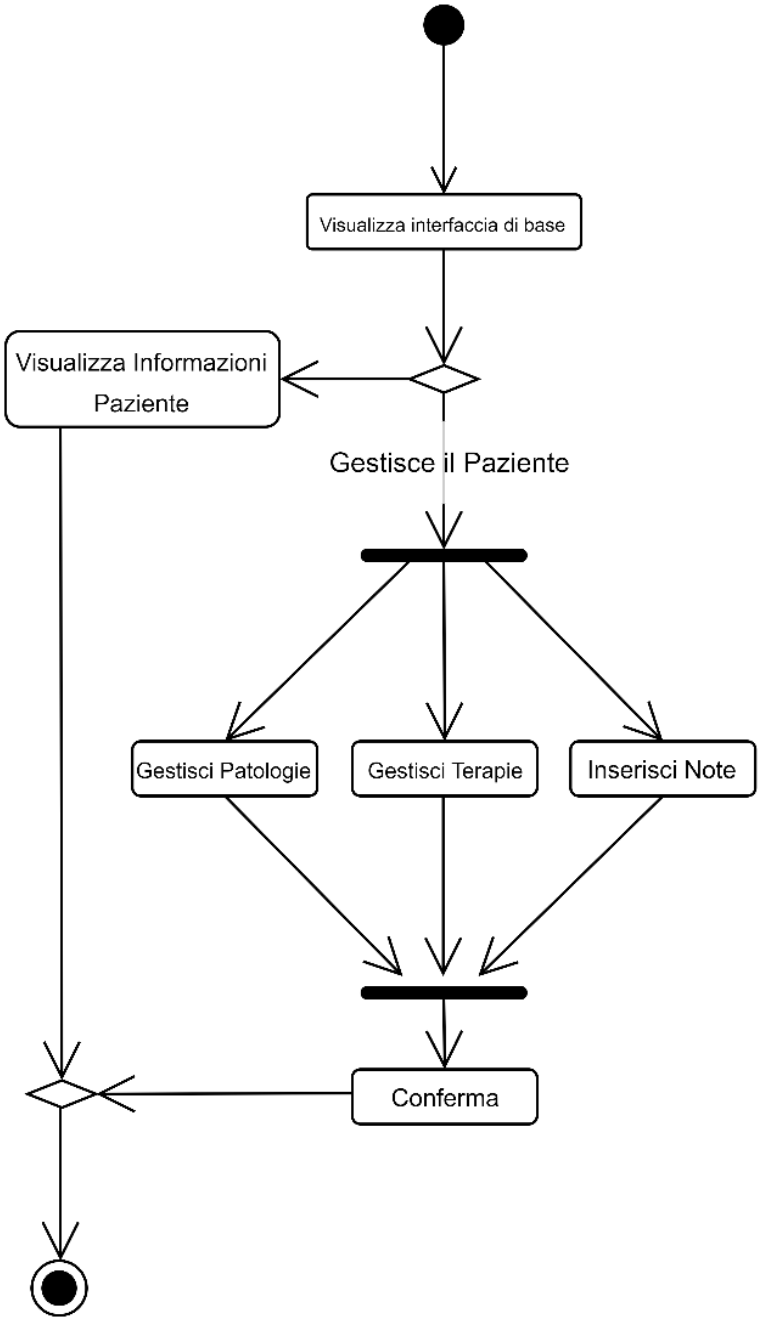

Figure 6: Attività medico

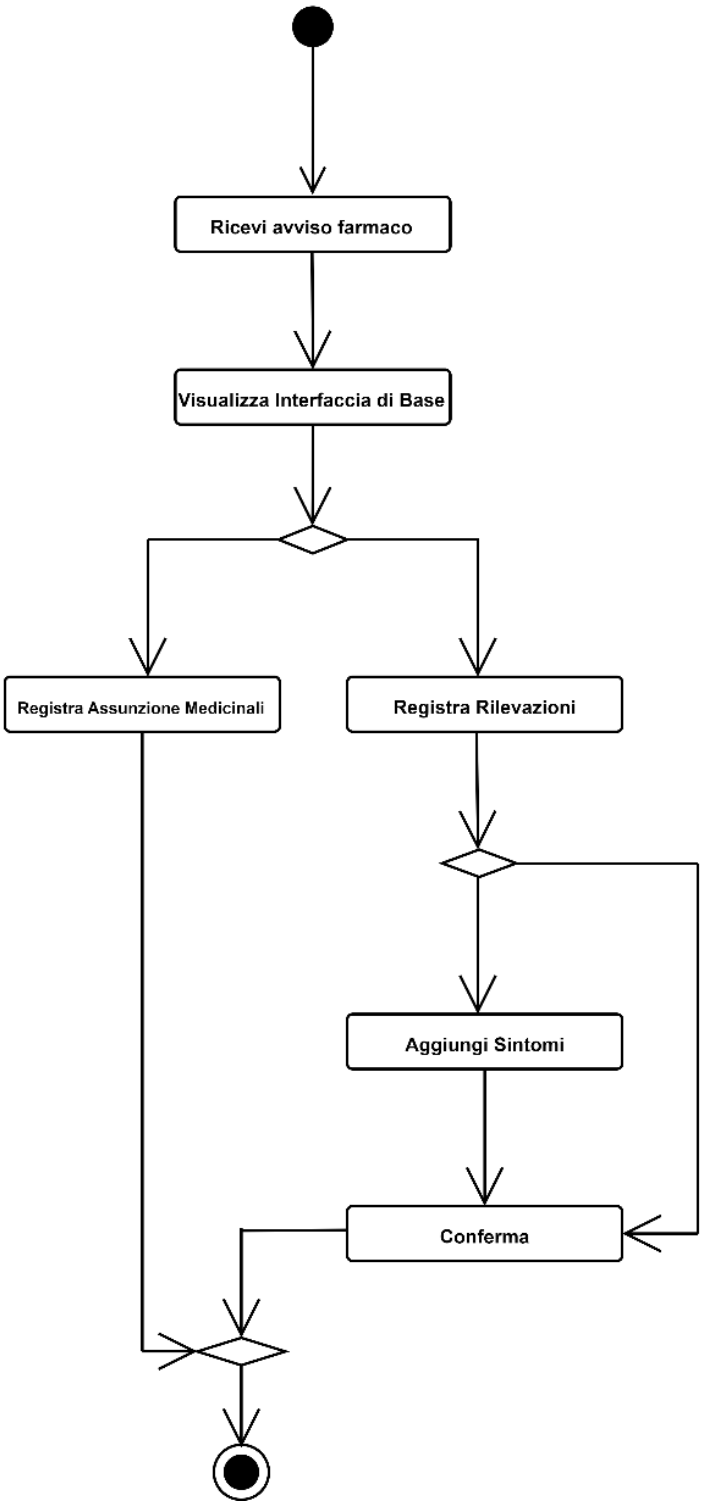

Figure 7: Attività paziente

Sviluppo ed implementazione del sistema

Processo di sviluppo e organizzazione generale

Il processo di sviluppo adottato e' stato di tipo *Incremental Planning Approach* in quanto il nostro obiettivo era utilizzare un approccio incrementale che incorporasse elementi del modello *plan-driven*.
Si e' optato per un modello *plan-driven* in quanto si e' deciso di pianificare a priori tutte le attività di sviluppo nel dettaglio per poi condurre una sequenza di fasi lineari verificandone i progressi secondo il piano stilato.
Si e' deciso di adottare il metodo incrementale in quanto la suddivisione dello sviluppo in piccoli incrementi funzionali ha reso possibile la verifica e il test per ogni fase del progetto senza doverne attendere l'intera conclusione.
E' stata condotta una fase di test per ogni ciclo, andando a correggere tempestivamente eventuali inesattezze rilevate.
La documentazione relativa ai casi d'uso è stata prodotta in fase iniziale mentre i *class diagram*, così come i *sequence diagram*, dopo l'effettiva implementazione del codice relativo.
In sintesi, l'adozione dell'approccio "Incremental Planning" ci ha permesso di combinare la pianificazione dettagliata del modello "plan-driven" con i vantaggi della metodologia incrementale, migliorando l'efficienza e la qualità del nostro processo di sviluppo.

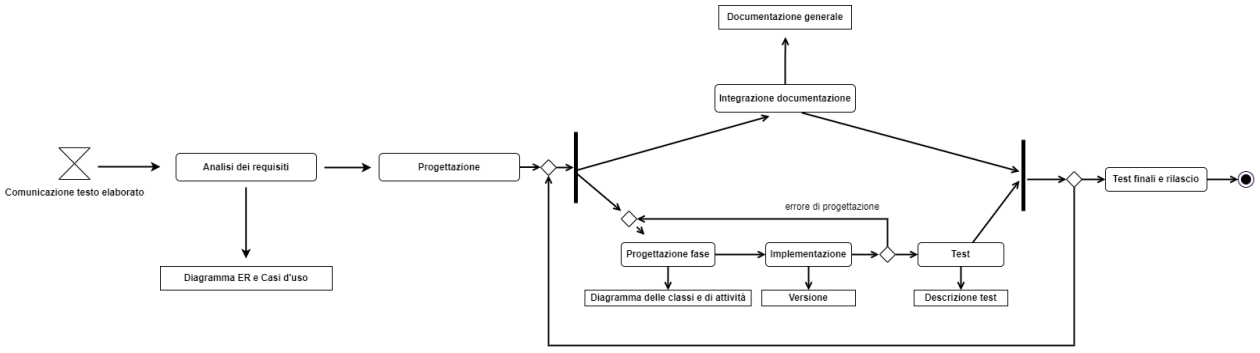

Figure 8: Processo di sviluppo

Diagramma delle classi del modello

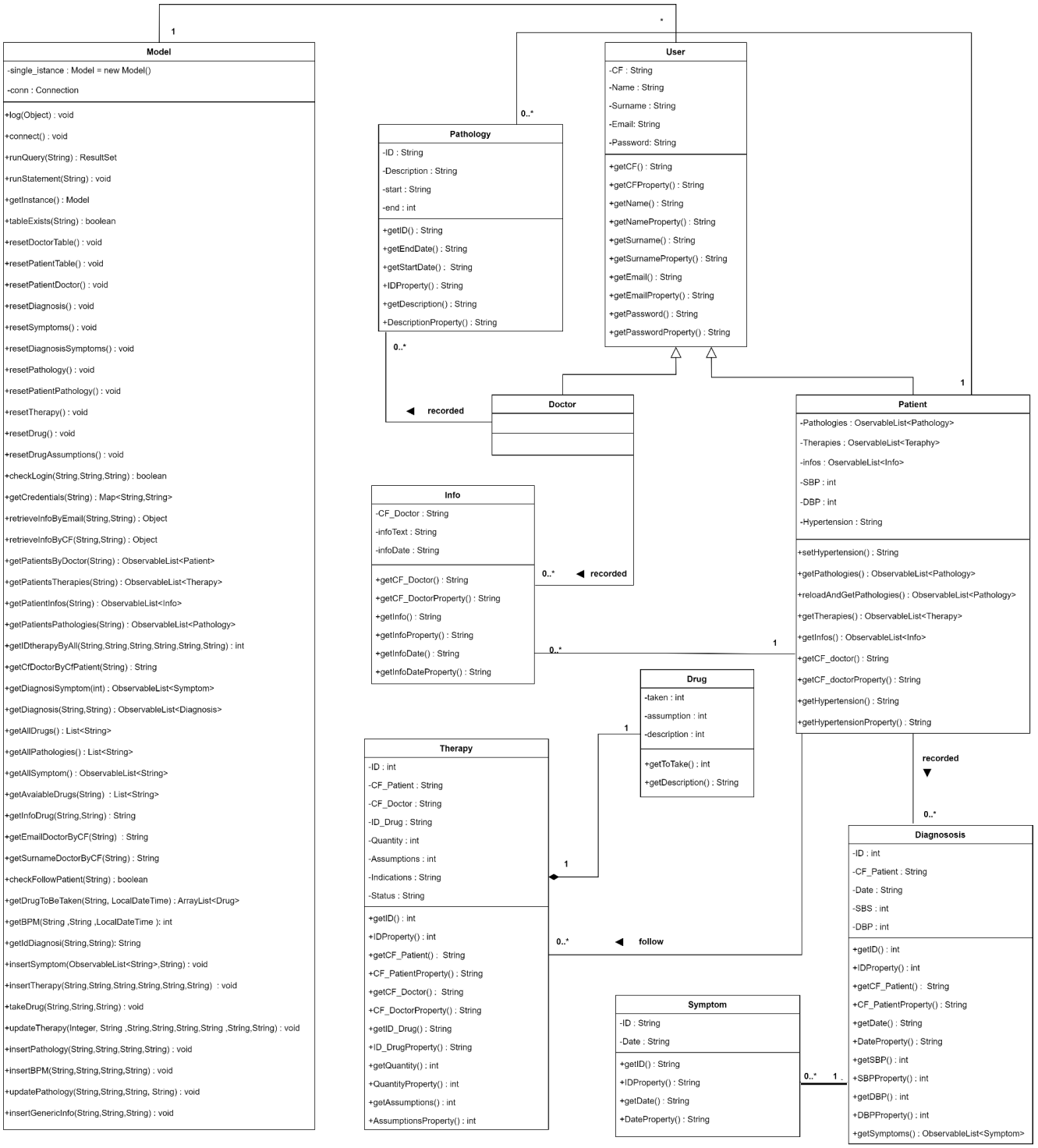

Figure 10: Diagramma delle classi del modello

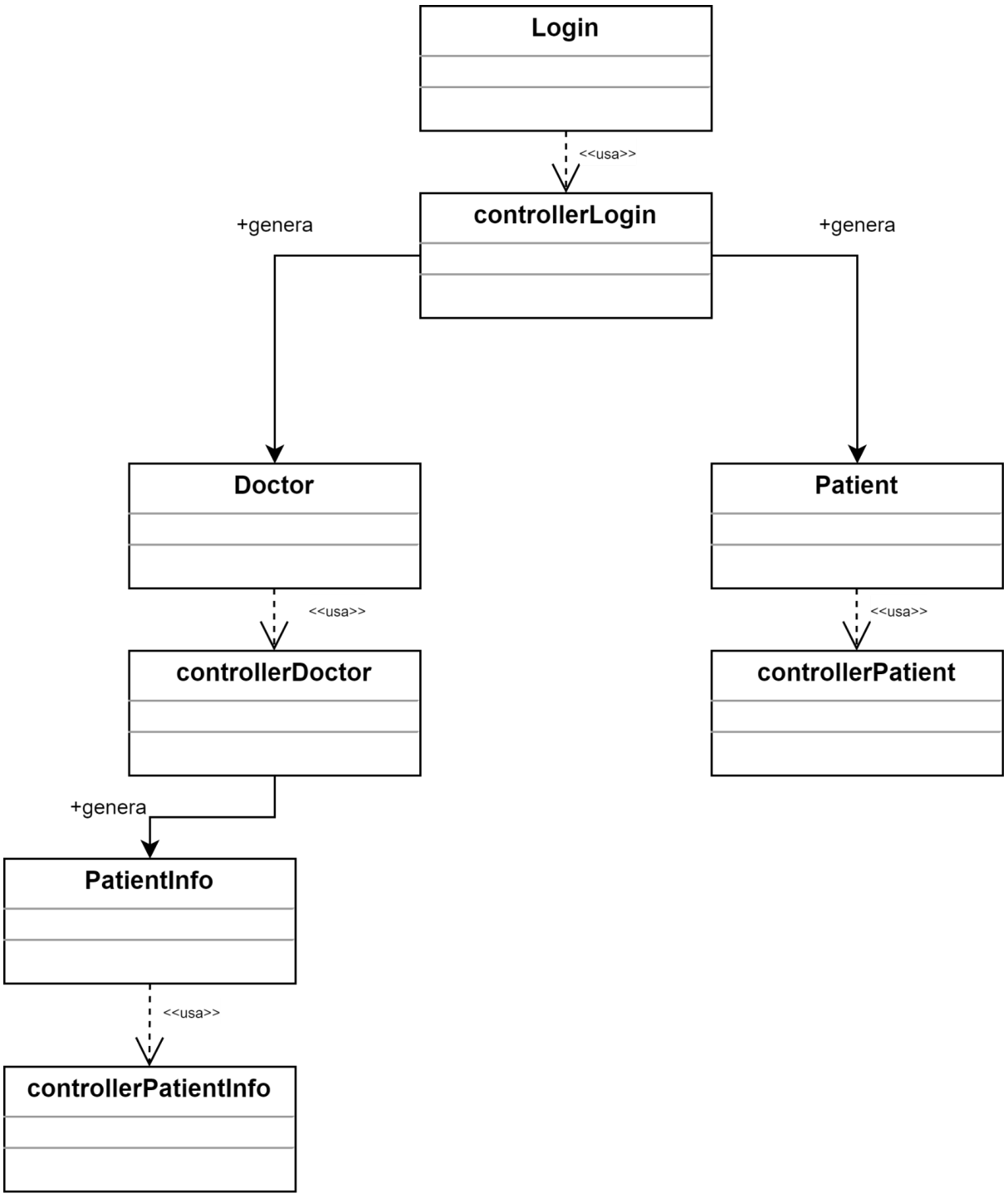

Figure 11: Diagramma delle classi della vista-controllore.

Breve commento sulle interfacce

La comunicazione Model-Controller e Model-View avviene grazie alle chiamate alla classe *Model* che permette di reperire le informazioni necessarie.
A seguire alcune delle chiamate più importanti alla classe *Model:*

- *connect():* per collegarsi alla base di dati
- *runQuery():* performa la query nel database
- *getInstance():* ottieni istanza singleton fondamentale per il funzionamento del sistema
- *getCredentials():* ottieni lista credenziali
- *checkLogin():* per autenticare l'utente
- *insertGenericInfo():* per aggiungere info generiche sul paziente
- *insertBPM():* per registrare rilevazioni SBP/DBP
- *insertSymptom():* per registrare sintomi correlati ad una rilevazione SBP/DBP
- *insertPathology():* per aggiungere patologie al paziente
- *getDrugToBeTaken():* per ricevere lista dei farmaci giornalieri ancora da assumere
- *checkFollowPatient():* ottieni informazioni se il paziente sta seguendo la terapia assumendo i medicinali
- *getInfoDrug():* ottieni informazioni sui farmaci della terapie

Per la comunicazione View-Controller sono state adottate le soluzioni nativamente suggerite dalla libreria JavaFX.

Diagrammi di sequenza del software implementato

Ecco i diagrammi di sequenza di alcune delle interazioni tra classi fondamentali per il funzionamento del sistemi.

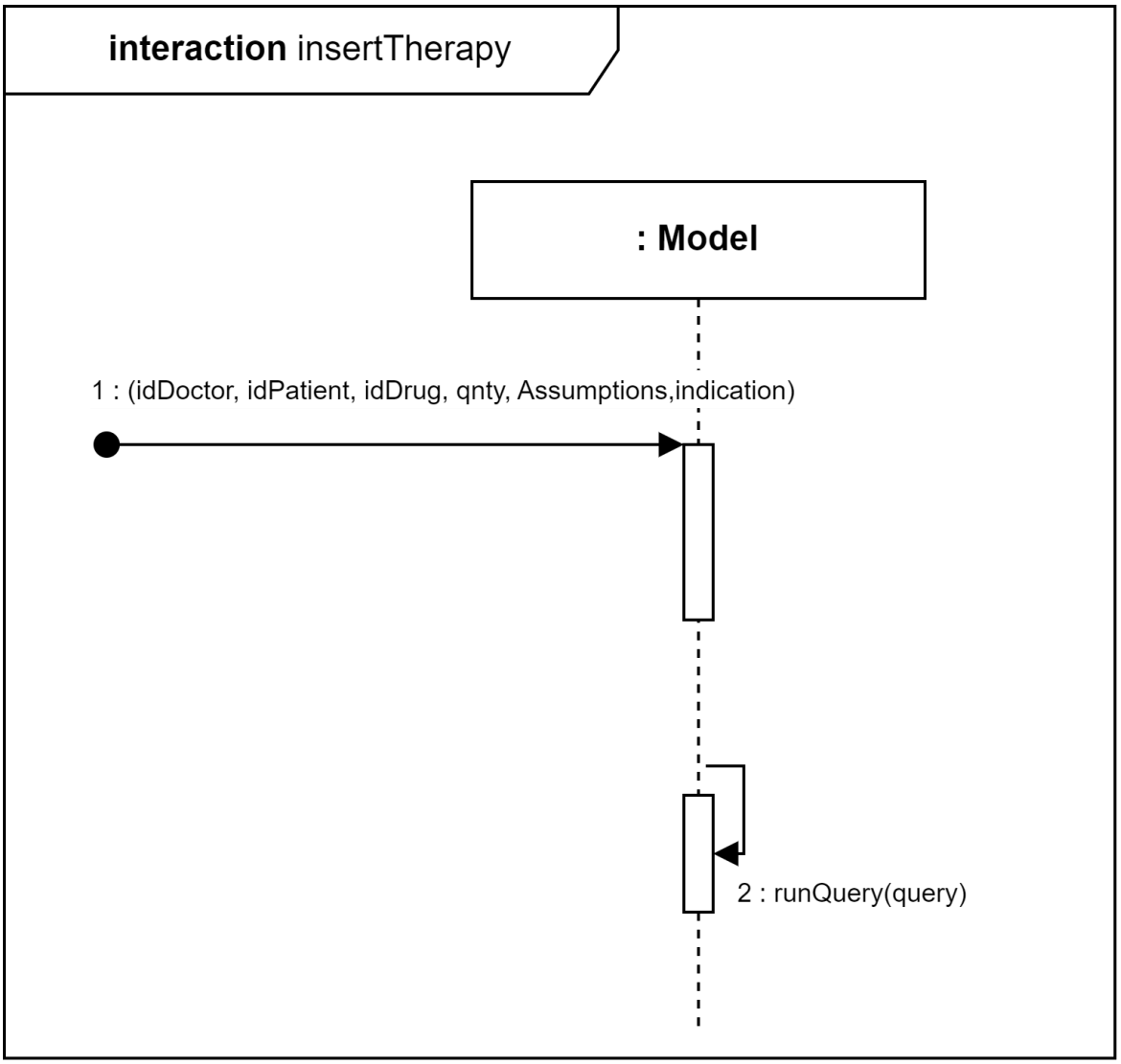

Figure 12: inserimento terapia

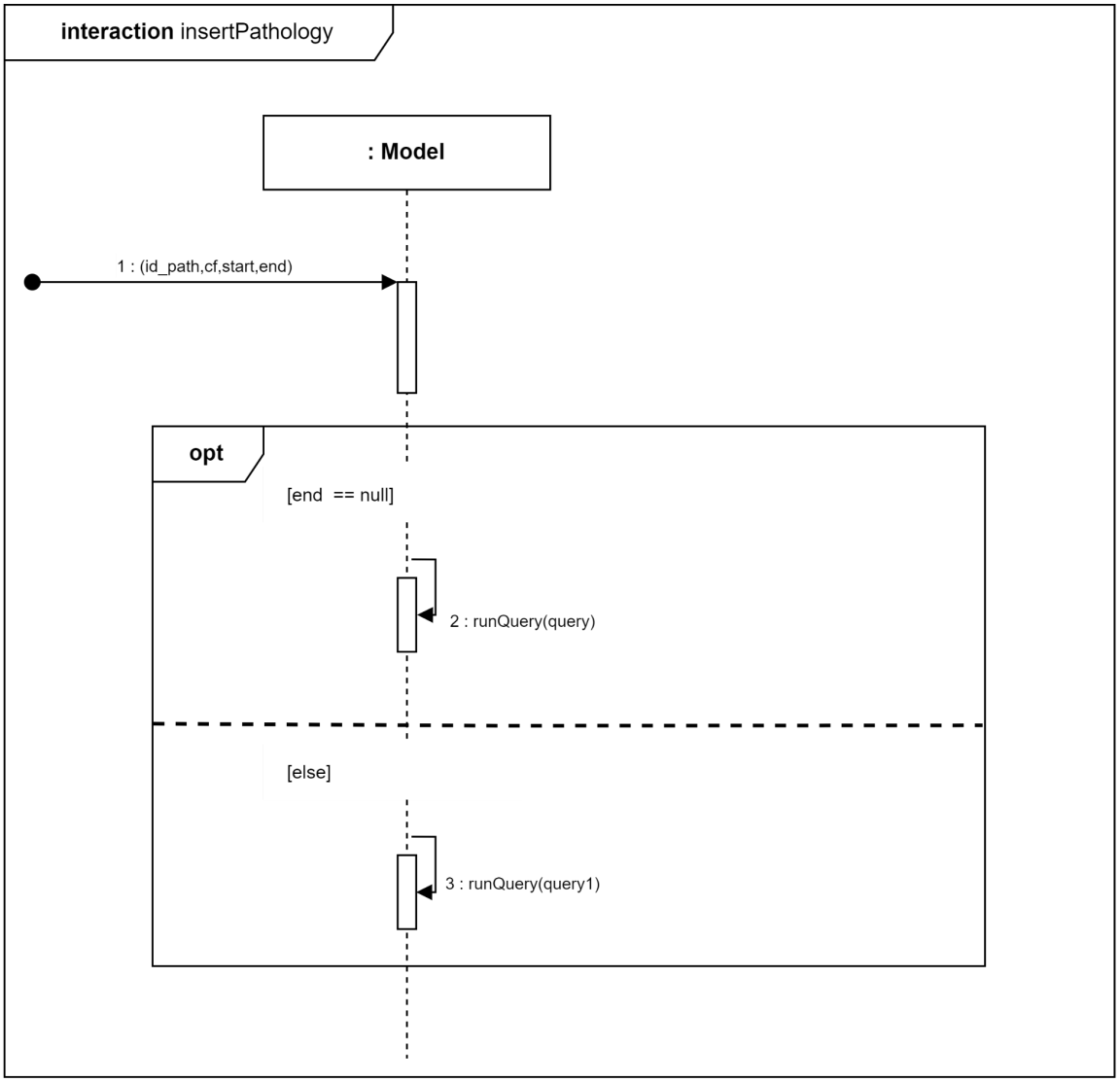

Figure 13: inserimento patologia

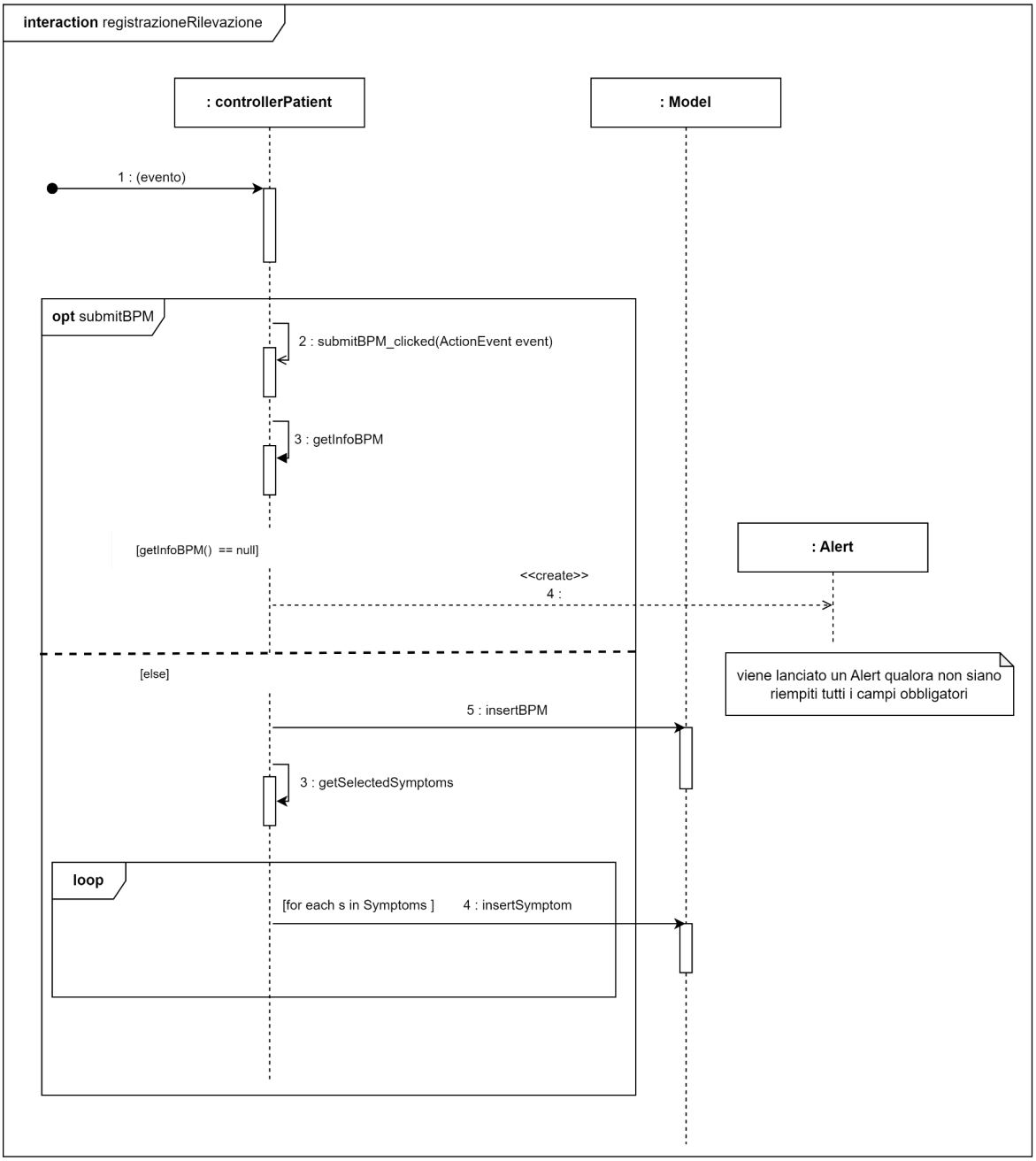

Figure 14: registrazione rilevazione pressione paziente

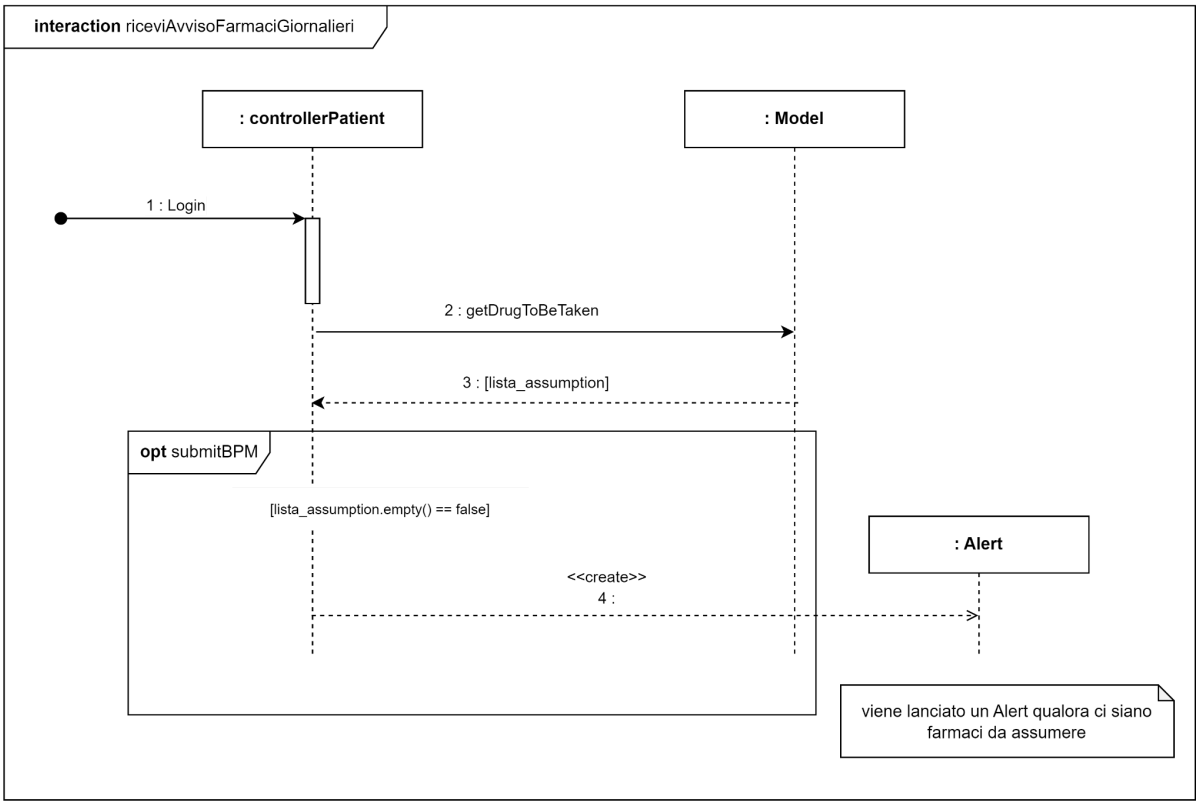

Figure 15: avviso farmaci giornalieri da assumere

Ulteriori dettagli riguardo al codice, l'implementazione ed il suo funzionamento sono consultabili presso il JavaDoc presente nella directory del progetto.

Attività di test e validazione

In questa fase si sono effettuate diverse operazioni mirate a verificare la solidità del software ultimato, ossia:

1. Rianalisi del documento contenente le specifiche e confronto diretto con i diagrammi prodotti.
2. Verifica di congruenza tra diagrammi e l'effettivo codice elaborato.
3. Verifica del codice con conseguente ricerca di eventuali ridondanze.
4. Verifica di consistenza dei dati prodotti.
5. Unit test automatizzato mediante l'utilizzo di JUnit per verificare il comportamento e la correttezza delle classi interessate.
6. Test da parte degli sviluppatori.
7. Test da parte di utente esterno.

Ispezione codice e documentazione

Dopo un'ulteriore analisi del testo contenente le specifiche, si e' verificato che il codice fosse conforme ai diagrammi UML prodotti, verificando con particolare attenzione i casi d'uso, i diagrammi d'attivita' e delle classi.
Si è controllato il codice per individuare parti ridondanti che lo renderebbero ripetitivo ed inutile, e se necessario, sono state rimosse.

Unit test

Questa fase ha svolto un ruolo centrale nelle operazioni di test, in quanto grazie all'utilizzo di JUnit si sono testate alcune delle funzionalità del codice che per noi ricoprivano le aree con più importanza e criticità.
Di seguito sono riportati alcuni esempi dei test svolti.

```
@Test
public void testConnect() throws SQLException {
        String url = "jdbc:sqlite:table.db";
        Connection conn = DriverManager.getConnection(url);
        // verifico connessione non sia null
        assertNotNull("Connection is null", conn);
        // verifico che la connessione non sia chiusa
        assertTrue("Connection is closed", !conn.isClosed());
}

@Test
public void testRunQuery() throws SQLException {
        var m = Model.getInstance();
        String query = "SELECT * FROM doctor";
        ResultSet rs = m.runQuery(query);
        // verifico risultato query non sia null
        assertNotNull("ResultSet is null", rs);
        // verifico risultato contenga almeno una riga
        assertTrue("ResultSet is empty", rs.next());
}

@Test
public void testGetInstance() throws SQLException {

        // Due istanze model guardo non siano null, e che siano la stessa
        // Considerando che sono singleton
        var m1 = Model.getInstance();
        assertNotNull("Model instance is null", m1);
        var m2 = Model.getInstance();
        assertNotNull("Model instance is null", m2);
        assertSame("Model instances are not the same object", m1, m2);}
```

```
	@Test
	public void testTableExists() throws SQLException {
		var m = Model.getInstance();

		// Testo se tabella esiste nel DB
		assertTrue("Table does not exist", m.tableExists("doctor"));

		// Testo se tabella non esiste nel DB
		assertFalse("Table exists", m.tableExists("tommaso"));
	}

	@Test
	public void testCheckLogin() {

		// Creazione dell'istanza del modello
		Model model = null;
		try {
			model = Model.getInstance();
		} catch (SQLException e) {
			e.printStackTrace();
			fail("Errore durante l'inizializzazione del modello");
		}

		// credenziali corrette
		String email = "mail@mail.com";
		String psw = "d4a1ecebe00a30d870e88bf18828709d";

		// Verifica del login con credenziali corrette
		boolean loginResult = model.checkLogin(email, "wirecard", psw);
		assertTrue(loginResult);

		// Verifica del login con password errata
		loginResult = model.checkLogin(email, "wrongpassword", psw);
		assertFalse(loginResult);
	}

}
```

Test degli sviluppatori

In questa fase gli sviluppatori hanno simulato diversi casi d'uso, svolgendo sia dei classici test, che test mirati a minare completamente le funzionalità del sistema.
Alcuni test svolti sono stati:

- Autenticazioni al sistema di vario genere: autenticazione con credenziali corrette, con la sola email errata, con la sola password errata o con entrambe le credenziali errate. In ogni caso, viene segnalato opportunamente il campo errato e/o mancante mediante un apposito alert pop-up.
- Verificato che il medico veda solo i suoi pazienti ma abbia accesso a tutte le informazioni (anche pregresse) relative al paziente nel sistema.
- Per i seguenti 3 test viene segnalato opportunamente il campo errato e/o mancante mediante un apposito alert pop-up.
    - Inserimento annotazione paziente: si e' provato ad inserire annotazioni vuote e a cliccare ripetutamente il tasto “Add Info”.
    - Inserimento terapie paziente: si e' provato ad inserire terapie vuote e a cliccare ripetutamente il tasto “+”.
    - Inserimento patologie paziente: si e' provato ad inserire patologie con dati inconsistenti (ad esempio senza la data ‘Start’) e cliccare ripetutamente il tasto “Add pathology”.
    - Inserimento rilevazioni giornaliere: si e' provato ad inserire rilevazioni giornaliere vuote e a cliccare ripetutamente il tasto “SUBMIT”.
- Verificato il sistema di farmaci giornalieri ancora da assumere.
- Verificato che il sistema inserisca solo i farmaci da assumere delle terapie attive.

Test utente generico

Il software è stato sottoposto ad un ultimo test da parte di alcune persone con poca conoscenza informatica e che non erano coinvolte nel suo sviluppo. Durante questa fase, non si è cercato di guidare o strutturare l'esperienza degli utenti in alcun modo al fine di non influenzare il risultato del test. Invece, si è permesso agli utenti di navigare liberamente nel sistema senza fornire alcuna spiegazione sull'utilizzo del software, ad eccezione di una breve descrizione generale dei suoi obiettivi. Si è risposto solo alle domande che sono state poste durante il test. Il test ha individuato alcuni errori di interfaccia grafica che non erano stati notati durante lo sviluppo del software.

Pattern architetturali usati

Il sistema e' stato progettato decidendo di adottare il pattern MVC, nativamente implementato dalle librerie *JavaFX*, utilizzate per l'interfaccia grafica del progetto.
Le motivazioni che hanno spinto questa scelta derivano dal fatto di voler avere una suddivisione logica ben definita delle classi all'interno del progetto sfruttando quindi il principio base del MVC ossia il disaccoppiamento.
Ad ogni membro del MVC e' stato assegnato un *package* differente.
Si e' suddiviso il tutto quindi come segue:

- **Model:** gestisce i dati del sistema e fornisce i metodi di accesso al DB.
- **View:** visualizza i dati contenuti nel Model e si occupa dell'interazione tra utente (sia esso Medico o Paziente) e sistema.
- **Controller:** riceve i comandi dell'utente (mediante la *view*) e li attua modificando lo stato degli altri due componenti (model e *view*).

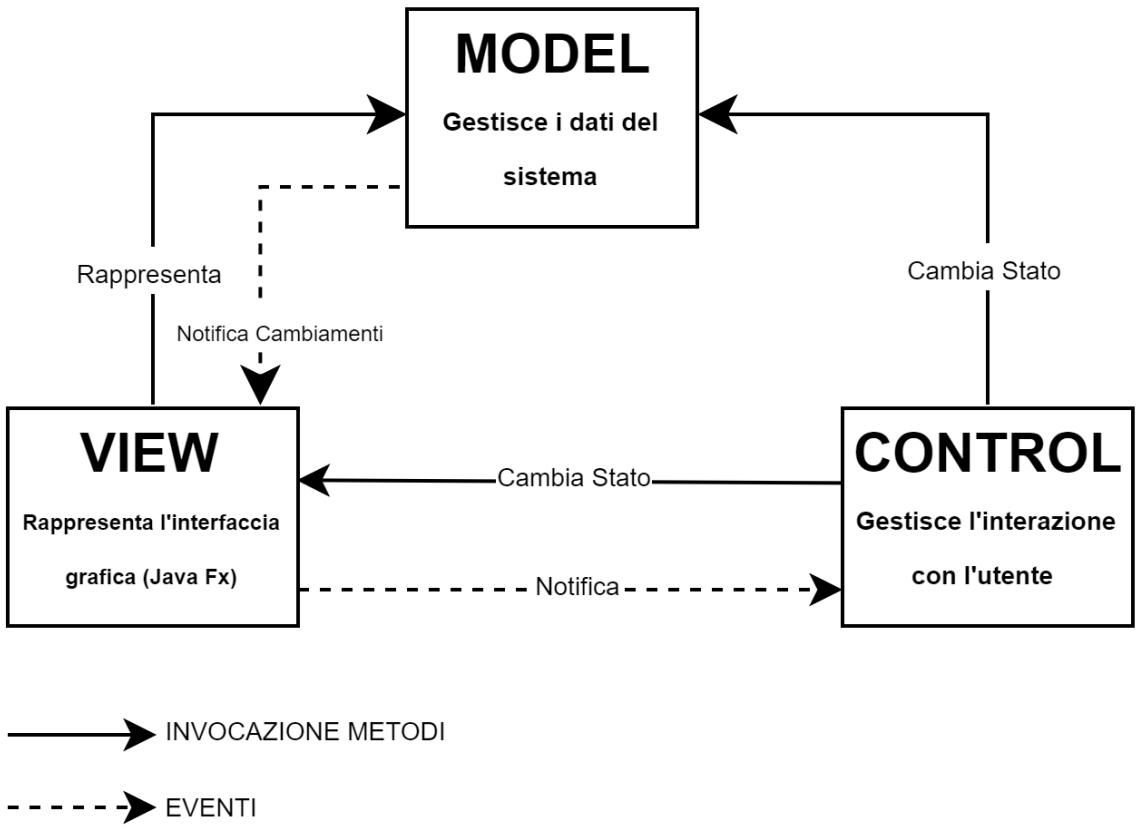

Figure 9: Architettura MVC

Implementazione e design pattern usati

Nel seguente paragrafo si discute l'implementazione e l'applicazione dei vari pattern utilizzati.

Pattern Singleton
E' stato scelto di utilizzare un pattern singleton per facilitare il passaggio di informazioni tra View-Controller differenti, ad esempio il passaggio di informazioni tra il login e l'interfaccia di base relativa che si carica in seguito ad una autenticazione.
Questo pattern viene utilizzato anche per la classe *Model*, cuore del sistema, che gestisce i dati del sistema e fornisce i metodi di accesso al DB.

Pattern Facade
Si è optato di usare il pattern facade, implementando diversi metodi che nascondano all'utente finale la complessità del sistema.

Pattern Decorator
Abbiamo sfruttato il pattern decorator per aggiungere funzionalità ad oggetti già esistenti, ad esempio l'oggetto Patient e Doctor estendono l'oggetto base User.

Pattern Iterator
Il pattern Iterator viene usato nel progetto in diverse porzioni di codice per scorrere sequenzialmente collezioni di oggetti.

Pattern Observer
Il pattern Observer e' utilizzato per gestire le azioni catturate dalle view notificando ai relativi controller gli eventi generati.