Università degli studi di Verona

Progetto di Ingegneria del software

Vax Reax

22 settembre, 2022

Autore	*Matricola*
Karol Pasieczny	VR447002
Giovanni Bee	VR446879

Contents

1 Traccia elaborato

Si vuole progettare un sistema software per gestire le segnalazioni di reazioni avverse (ad esempio, asma, dermatiti, insufficienza renale, miocardiopatia, ...) da vaccini anti-Covid. Ogni segnalazione è caratterizzata da un codice univoco, dall'indicazione del paziente a cui fa riferimento, dall'indicazione della reazione avversa, dalla data della reazione avversa, dalla data di segnalazione, e dalle vaccinazioni ricevute nei due mesi precedenti il momento della reazione avversa. Per ogni paziente sono memorizzati: un codice univoco, l'anno di nascita, la provincia di residenza e la professione. Per ogni paziente è possibile memorizzare gli eventuali fattori di rischio presenti (paziente fumatore, iperteso, sovrappeso, paziente fragile per precedenti patologie cardiovascolari/oncologiche), anche più d'uno. Ogni fattore di rischio è caratterizzato da un nome univoco, una descrizione e il livello di rischio associato. Per ogni paziente è, inoltre, memorizzata l'intera storia delle sue vaccinazioni precedenti, anti-Covid-19 e antinfluenzali. Ogni vaccinazione è caratterizzata da: paziente a cui si riferisce, segnalazioni a cui è legata, vaccino somministrato (AstraZeneca, Pfizer, Moderna, Sputnik, Sinovac, antinfluenzale, ...), tipo della somministrazione (I, II, III o IV dose, dose unica), sede presso la quale è avvenuta la vaccinazione e data di vaccinazione. Per ogni reazione avversa sono memorizzati un nome univoco, un livello di gravità (da 1 a 5) e una descrizione generale, espressa in linguaggio naturale. Una reazione avversa può essere legata a molte segnalazioni. Per ogni paziente sono memorizzati il numero di reazioni avverse segnalate ed il numero di vaccinazioni ricevute.

Il sistema deve supportare i medici che effettuano la segnalazione. Dopo opportuna autenticazione, il medico viene introdotto ad una interfaccia che permette l'inserimento dei dati delle reazioni avverse e dei pazienti. Il codice univoco dei pazienti è gestito dal sistema, che tiene traccia dei pazienti indicati da ogni medico. Ogni medico vede solo i codici identificativi dei pazienti, dei quali ha già segnalato qualche reazione avversa, e le relative informazioni.

Ad ogni fine settimana o quando il numero di segnalazioni raggiunge la soglia di 50, il sistema manda un avviso ad uno dei farmacologi responsabili della gestione delle segnalazioni di reazioni avverse. Il farmacologo, dopo autenticazione, accede alle segnalazioni (tutte, con l'indicazione del medico che le ha fatte) e può effettuare alcune analisi di base (quante segnalazioni per vaccino, quante segnalazioni gravi in settimana, quante segnalazioni per provincia e quante segnalazioni per sede di vaccinazione). Il sistema, inoltre, avvisa il farmacologo quando un vaccino ha accumulato in un mese oltre 5 segnalazioni di gravità superiore a 3. In base alle segnalazioni e agli avvisi del sistema, il farmacologo può proporre di attivare una fase di controllo del vaccino. Tale proposta viene registrata dal sistema, che tiene traccia di tutte le proposte relative ai vaccini segnalati.

2 Specifica dei requisiti

2.1 Casi d'uso e relative schede di specifica

Vax Reax è un programma che supporta il personale medico a gestire le segnalazioni avverse al vaccino. Le due categorie in cui si suddividono gli utenti, medico e farmacologo, hanno operazioni di interazione distinte, gestite nell'applicazione. Un'unica schermata di autenticazione permette all'utente di effettuare il login. Dopodiché, l'utente viene indirizzato alla pagina associata alla categoria corrispondente (Figure 1)

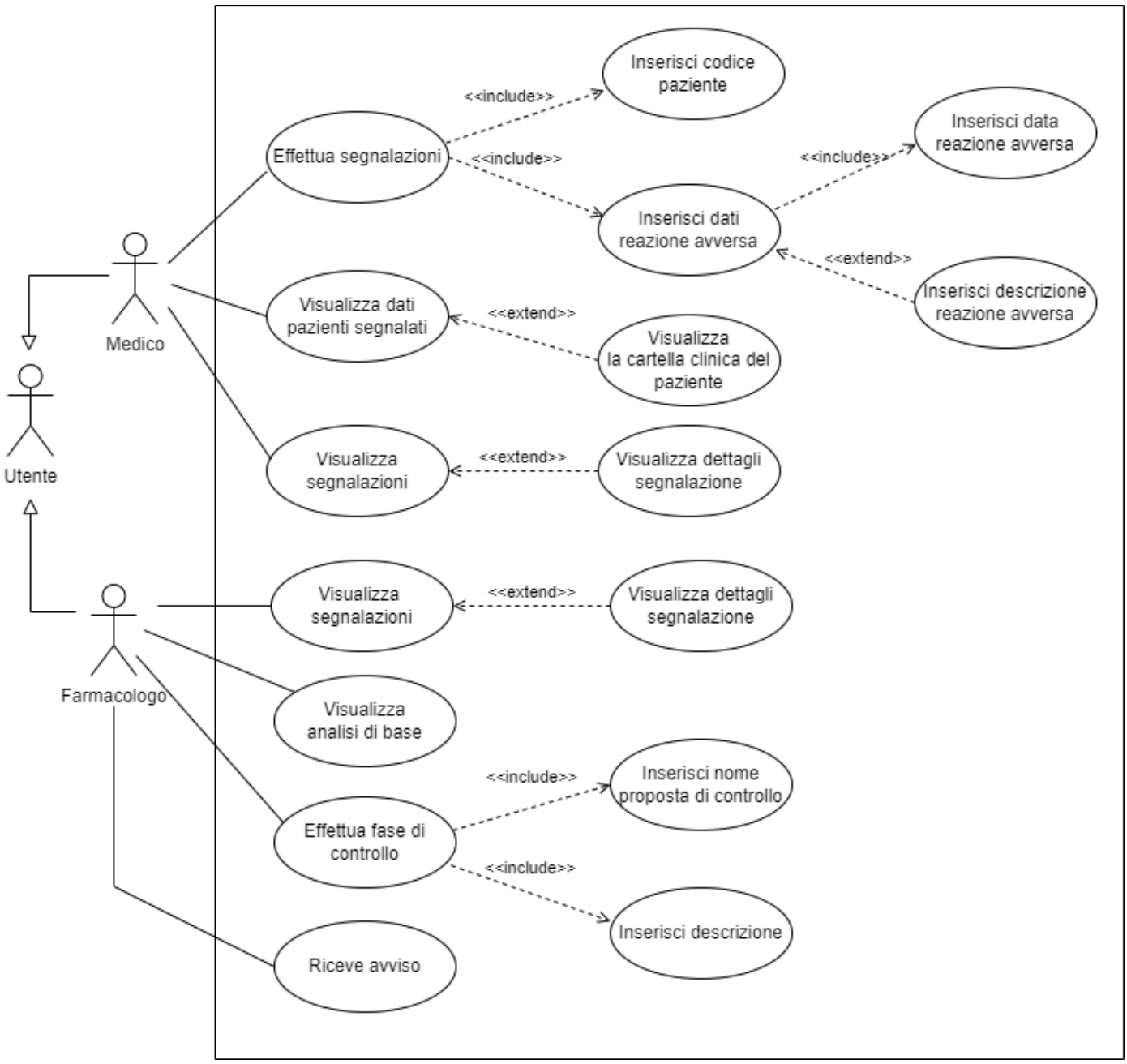

Figure 1: Use Case generico

2.2 Casi d'uso: medico

Il medico, dopo opportuna autenticazione, viene introdotto ad un interfaccia che permette tre operazioni principali: effettuare segnalazioni di reazioni avverse al vaccino, visualizzare le segnalazioni e visualizzare i dati relativi ai pazienti segnalati

2.2.1 Effettua segnalazioni

Il medico deve scegliere il codice del paziente e la reazione avversa per registrare la segnalazione (Figure 2).

***Attori**: Medico*

***Pre-condizioni**: Autenticazione come medico*

***Passi**:*
1. Accedere all'interfaccia di segnalazione (New Reports)
2. Inserire il codice del paziente
3. Inserire i dati della reazione avversa
4. Confermare la segnalazione

***Post-condizioni**: E' stata memorizzata la segnalazione*

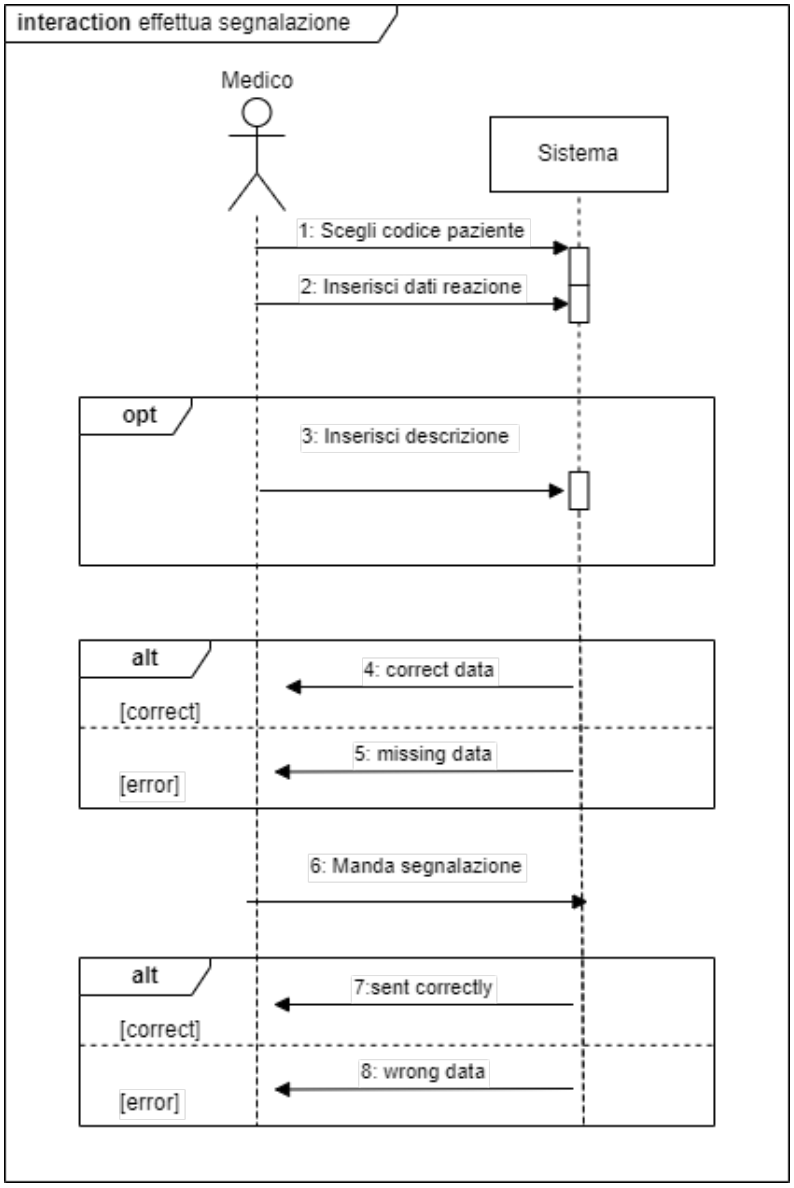

Figure 2: Interazione di segnalazione

2.2.1.1 Inserimento dati paziente

I pazienti, già registrati nel sistema, sono visualizzati solo se il medico in questione ha già effettuato una segnalazione a loro carico: altrimenti, dovrà inserirne l'identificativo durante la segnalazione.

2.2.1.2 Inserimento reazioni avverse

Bisogna scegliere una reazione avversa (es. tosse) tra le varie possibilità, indicando la data della reazione avversa, coerente con quelle delle vaccinazioni e con la data corrente, ovvero quella della segnalazione.

2.2.2 Visualizza segnalazioni

Il medico può visualizzare le segnalazioni passate

***Attori**: Medico*

***Pre-condizioni**: Autenticazione come medico*

***Passi**:*
1. Accedere all'interfaccia di visualizzazione di segnalazioni (Reports)
2. Visualizzare le segnalazioni

***Post-condizioni**: Nessuna*

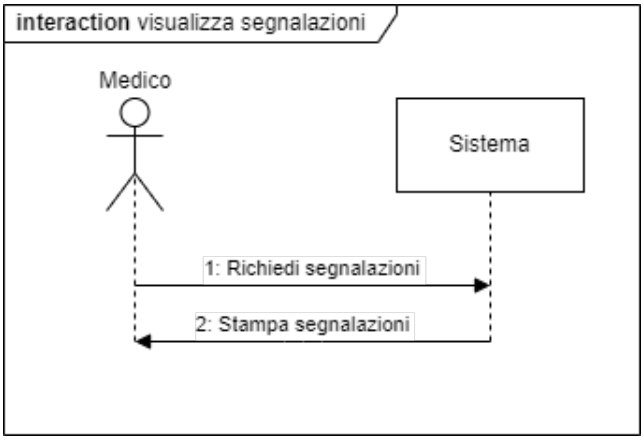

Figure 3: Interazione di visualizzazione delle segnalazioni

2.2.3 Visualizza pazienti

Il medico può visualizzare le informazioni relative ai pazienti

***Attori**: Medico*

***Pre-condizioni**: Autenticazione come medico*

***Passi**:*
1. Accedere all'interfaccia di visualizzazione dei pazienti (Patients'info)
2. Visualizzare i pazienti

***Post-condizioni**: Nessuna*

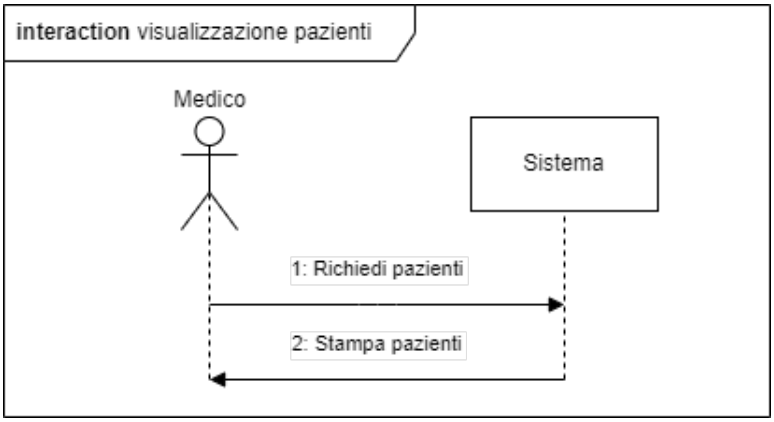

Figure 4: Interazione di visualizzazione dei pazienti

2.3 Casi d'uso: farmacologo

Il farmacologo deve essere in grado di accedere alle segnalazioni, analizzando eventualmente i dettagli delle singole segnalazioni. Il farmacologo può inoltre effettuare delle analisi di base. Ogni 50 segnalazioni viene inviato un avviso al farmacologo. Quest'ultimo, infine, è in grado di effettuare una fase di controllo del vaccino. Anche il farmacologo deve autenticarsi per poter compiere azioni.

2.3.1 Accede alle segnalazioni ed effettua le analisi

Il farmacologo accede alle segnalazioni e, eventualmente, effettua delle analisi.

***Attori**: Farmacologo*

***Pre-condizioni**: Autenticazione come farmacologo*

***Passi**:*
1. Accedere all'interfaccia di segnalazione (report)
2. Visualizzare le segnalazioni
3. Effettuare delle analisi
4. Visualizzare i risultati

***Post-condizioni**: Nessuna*

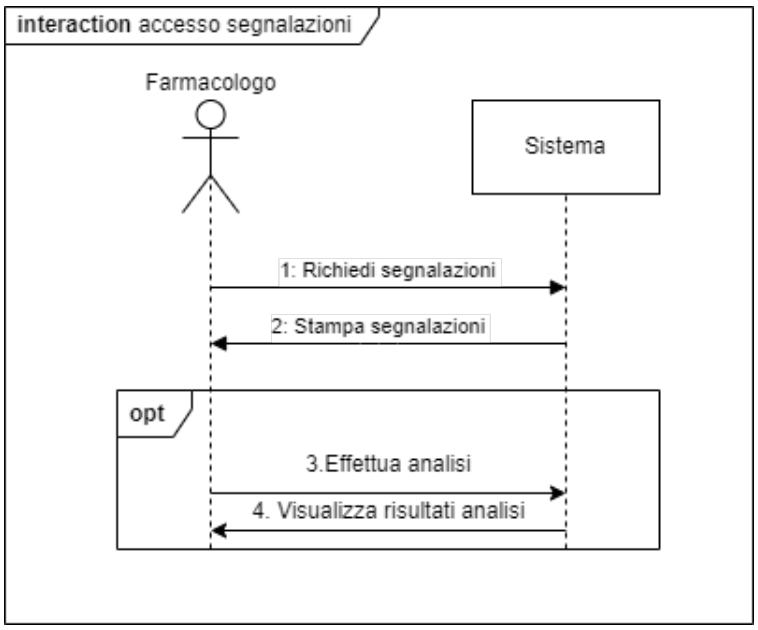

Figure 5: Interazione di accesso alle segnalazioni

2.3.2 Riceve avvisi

Periodicamente (*ogni 50 segnalazioni di reazioni avverse*), viene mandato un avviso al farmacologo.

***Attori**: Farmacologo*

***Pre-condizioni**: Autenticazione come farmacologo*

***Passi**: 1. Ricevere l'avviso, ogni 50 segnalazioni*

***Post-condizioni**: Nessuna*

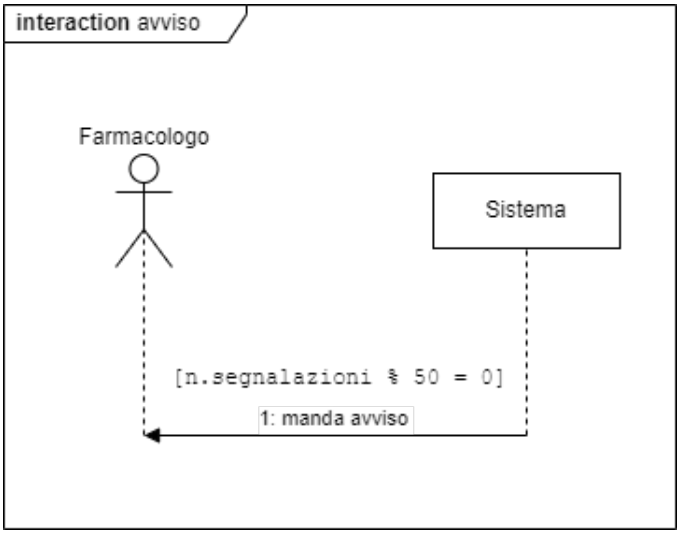

Figure 6: Interazione dell'operazione di avviso

2.3.3 Effettua proposte

Infine, il farmacologo può effettuare delle proposte per agire in merito al problema di reazioni avverse (ad esempio, sospensione o riesaminazione del vaccino)

***Attori**: Farmacologo*

***Pre-condizioni**: Autenticazione come farmacologo*

***Passi**:*
1. Effettuare la proposta
2. Visualizzare l'esito della proposta

***Post-condizioni**: La proposta è stata registrata*

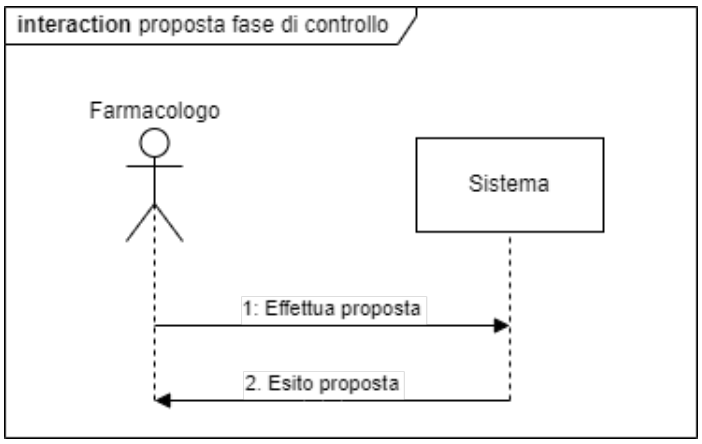

Figure 7: Interazione dell'operazione di proposta

3 Diagrammi UML

3.1 Diagrammi di attività

3.1.1 Login

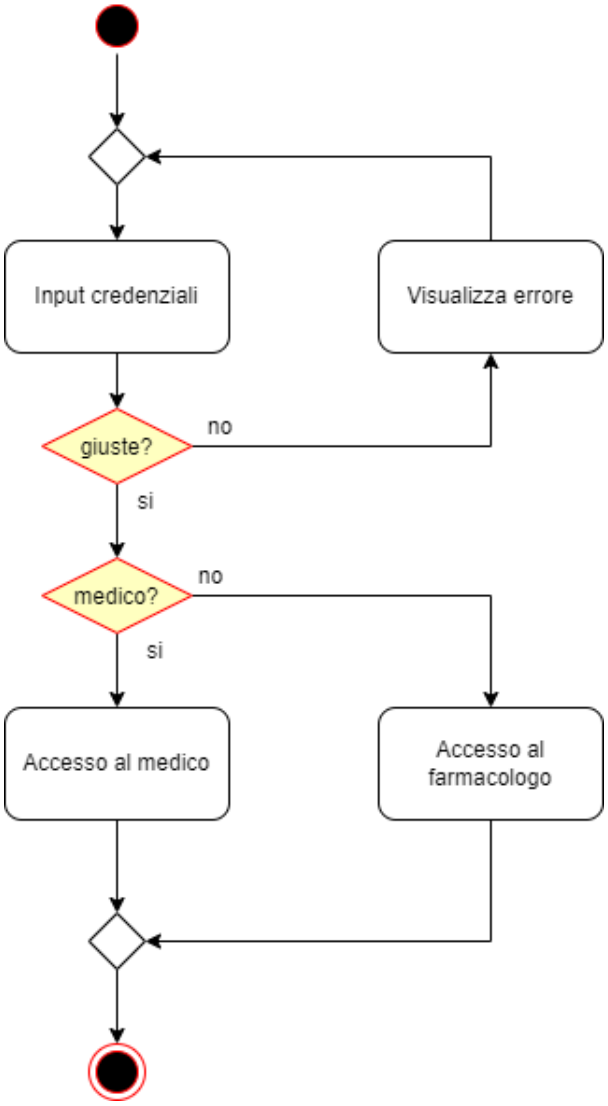

Figure 8: Diagramma di attività del login

3.1.2 Medico

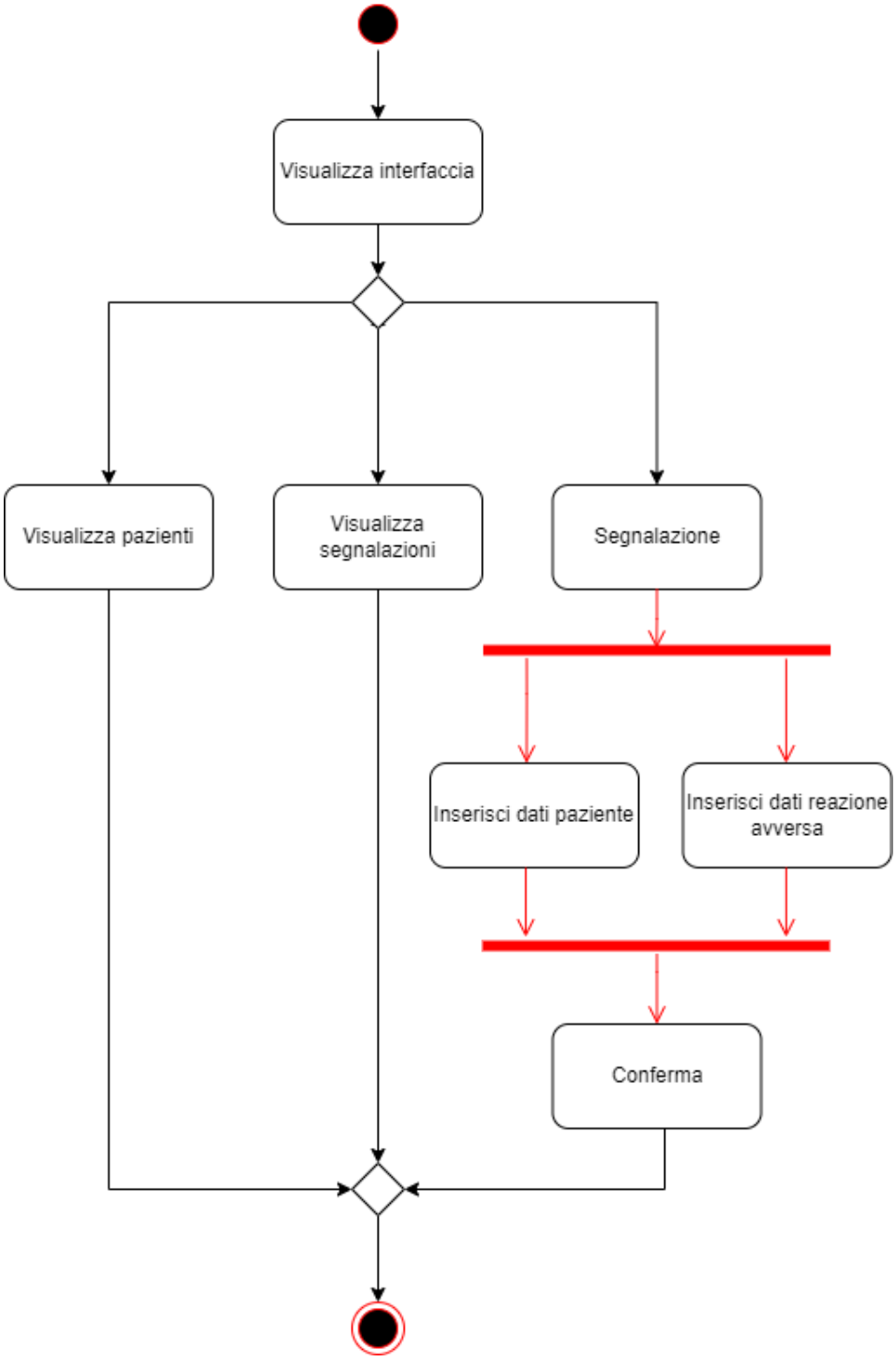

Figure 9: Diagramma di attività del medico

3.1.3 Farmacologo

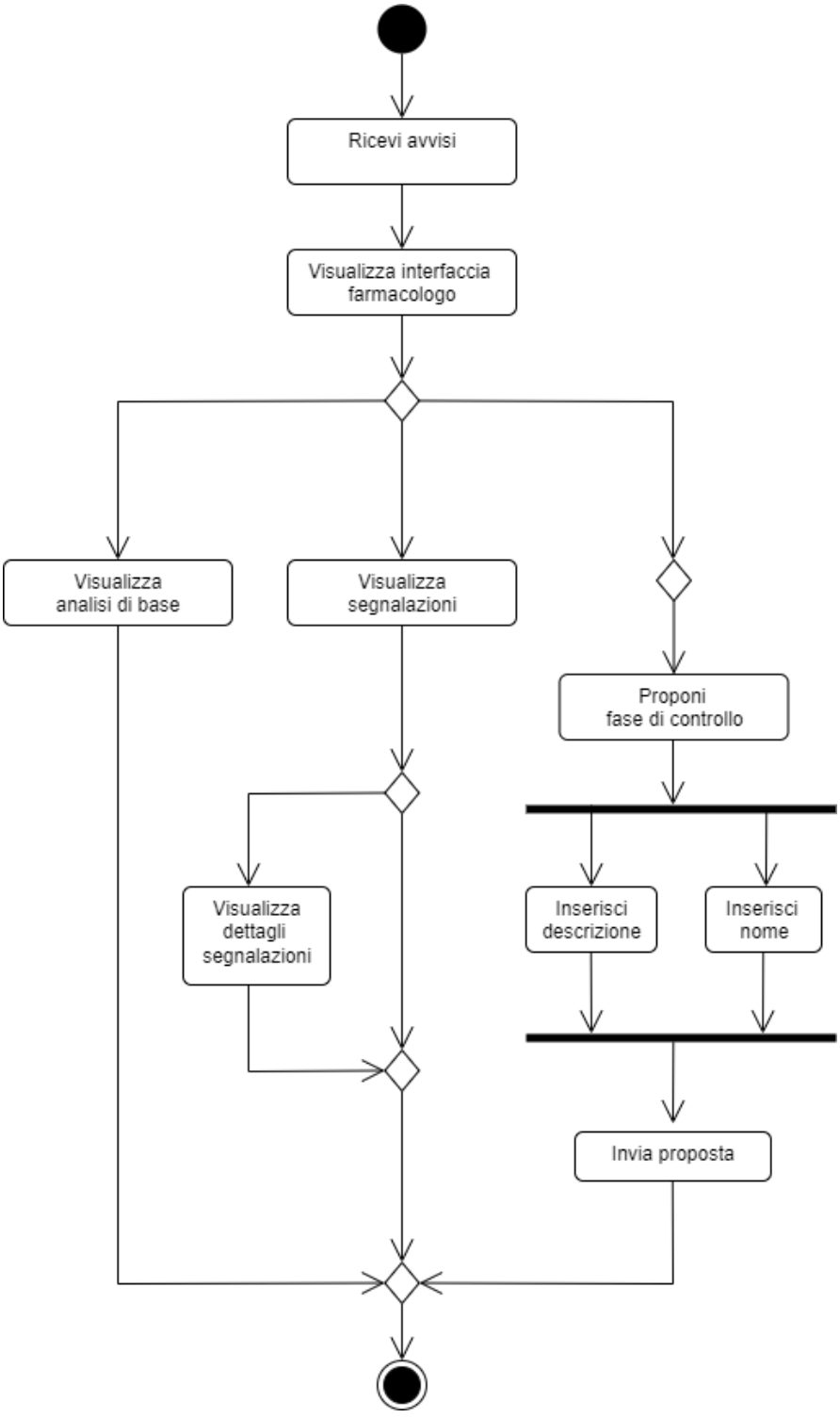

Figure 10: Diagramma di attività del farmacologo

3.2 Diagrammi delle classi

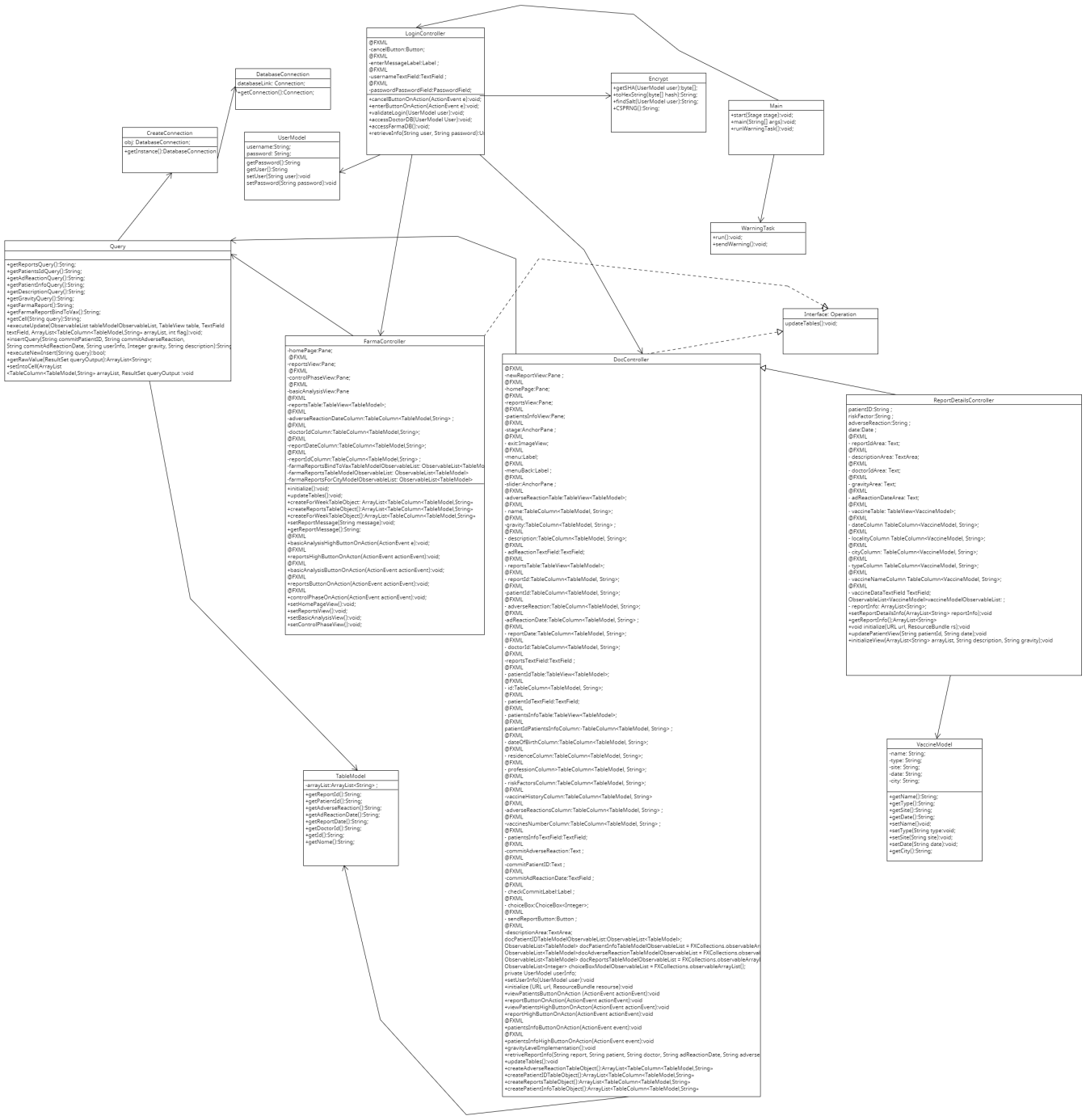

Figure 11: Diagramma delle classi

4 Analisi e sviluppo

4.1 Cenni sullo sviluppo

Pensando a "Vax Reax" come ad un software interamente nuovo e soggetto ad eventuali future releases e versioni, la scelta per lo sviluppo e la progettazione del software è ricaduta sull'utilizzo di un metodo ***agile***. Questa scelta è stata inoltre ponderata dal fatto che il team di sviluppo è composto da due persone. L'interazione, avvenuta per buona parte a distanza, si è quindi basata sulla scelta di opportuni strumenti per la condivisione in rete di materiale per lo sviluppo del software e costanti scambi di opinioni e dubbi circa le scelte e le tecniche di sviluppo. Dopo una preliminare e rapida fase di sviluppo concettuale ed astratto, con l'ausilio di diagrammi e schemi ad alto livello si è deciso di procedere in maniera ***incrementale*** per quanto riguarda lo sviluppo del programma. Questo ha richiesto l'utilizzo di una pianificazione iterativa e a breve termine, che potesse rispondere ad ogni cambiamento più che seguire un piano prestabilito e che permettesse quindi di avanzare nella programmazione per piccoli blocchi/moduli. In quest'ottica si è dimostrata valida, seppur utilizzata con moderazione, la tecnica del ***riuso*** di generici blocchi di codice per catturare gli elementi in comune tra differenti contesti, ed individuando i possibili plug-points per inserire le parti che li specializzano. Ad esempio ,per costruire l'interfaccia grafica del medico è stata usata una parte di quella utilizzata per il farmacologo, sia per una questione di coerenza con la forma e con l'estetica dell'interfaccia, sia per mantenere lo stesso comportamento del programma a seconda delle funzionalità richieste dai due tipi di utenza. L'attività di ***refactoring*** nella fase implementativa dei singoli blocchi è servita per mantenere il codice il più semplice possibile e confrontabile con le varie parti del programma.

4.2 Pattern architetturale

Si è deciso di adottare il **pattern architetturale MVC** sfruttando le librerie con funzionalità grafiche di **JavaFX**. Per quanto riguarda la *vista* si è deciso di adottare il linguaggio **FXML** ed il corrispondente tool **Scene builder** integrato a JavaFX come visual editor. Questa scelta è dovuta sostanzialmente alla possibilità offerta dal tool stesso di separare la componente grafica da quella del codice che gestisce il funzionamento e la logica del programma, mantenendo comunque un canale di comunicazione tra le due componenti attraverso l'injection delle dipendenze (@FXML) tra le due parti del software.

Sin dall'inizio si è voluto lavorare e sperimentare molto sul ***front-end*** cercando di sfruttare il più possibile le potenzialità di JavaFX.

Tramite oggetti di tipo *TableView* sono stati rappresentati la maggior parte dei dati. Questi dati sono stati inseriti all'interno di oggetti di tipo *ObservableList*, sfruttando il **pattern observer** già implicitamente implementato nella libreria di javaFX, *FXCollection*. Per rendere i dati più maneggevoli e facilmente accessibili da parte dell'utente finale è stata introdotta l'opzione di eseguire ricerche sui dati all'interno delle tabelle del database attraverso dei fitri dinamici. Per ogni tabella, dunque, è stata creata una classe *modello* che ne immagazzinasse i dati.

Per la connessione al database si è deciso di utilizzare il **pattern singleton** in modo tale da creare una sola istanza per tutto il ciclo di vita del software, restituendo un riferimento all'istanza già esistente attraverso il metodo statico *getInstance()*.

In linea con il concetto di *information hiding* si è deciso di ispirarsi al **pattern facade** così da nascondere ai due client principali (Medico e Farmacologo) la complessità del sistema.

4.3 Progettazione concettuale

4.3.1 Formalizzazione requisiti (ERD)

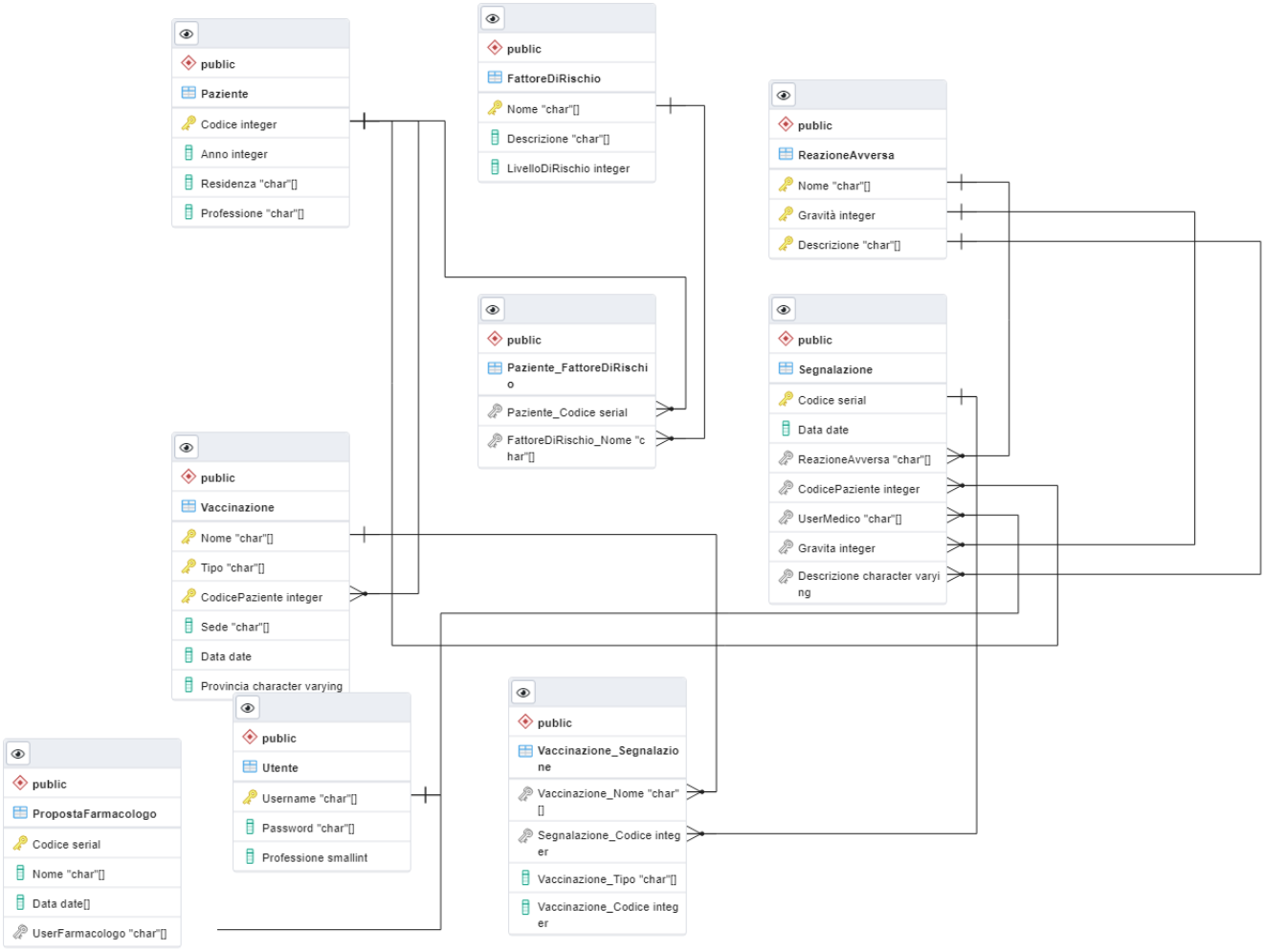

Figure 12: Diagramma ER

4.3.2 Regole non codificabili nel diagramma

La data delle vaccinazioni deve essere precedente a quelle della reazione avversa. Inoltre, la data della reazione avversa deve essere precedente a quella della segnalazione La gravità della reazione avversa va da 1 a 5, come per il livello di rischio della reazione avversa.

4.3.3 Verifica della qualità del database

Il database è stato realizzato come da traccia, per poi essere normalizzato

4.3.3.1 Normalizzazione

1-2NF: Le colonne sono già atomiche alla realizzazione dell'ERD quindi è già in prima forma normale. Inoltre, i campi non chiave dipendono dalla chiave primaria composta nella sua completezza, quindi lo schema è già in seconda forma normale.

3NF: La gravità e la descrizione della reazione avversa non dipendono dalla reazione avversa ma dalla singola segnalazione, per passare in terza forma normale è necessario rimuovere la tabella, accorpando i campi in segnalazione. Anche il campo relativo alla provincia della sede crea ridondanza, quindi è stata creata una nuova tabella Sede.

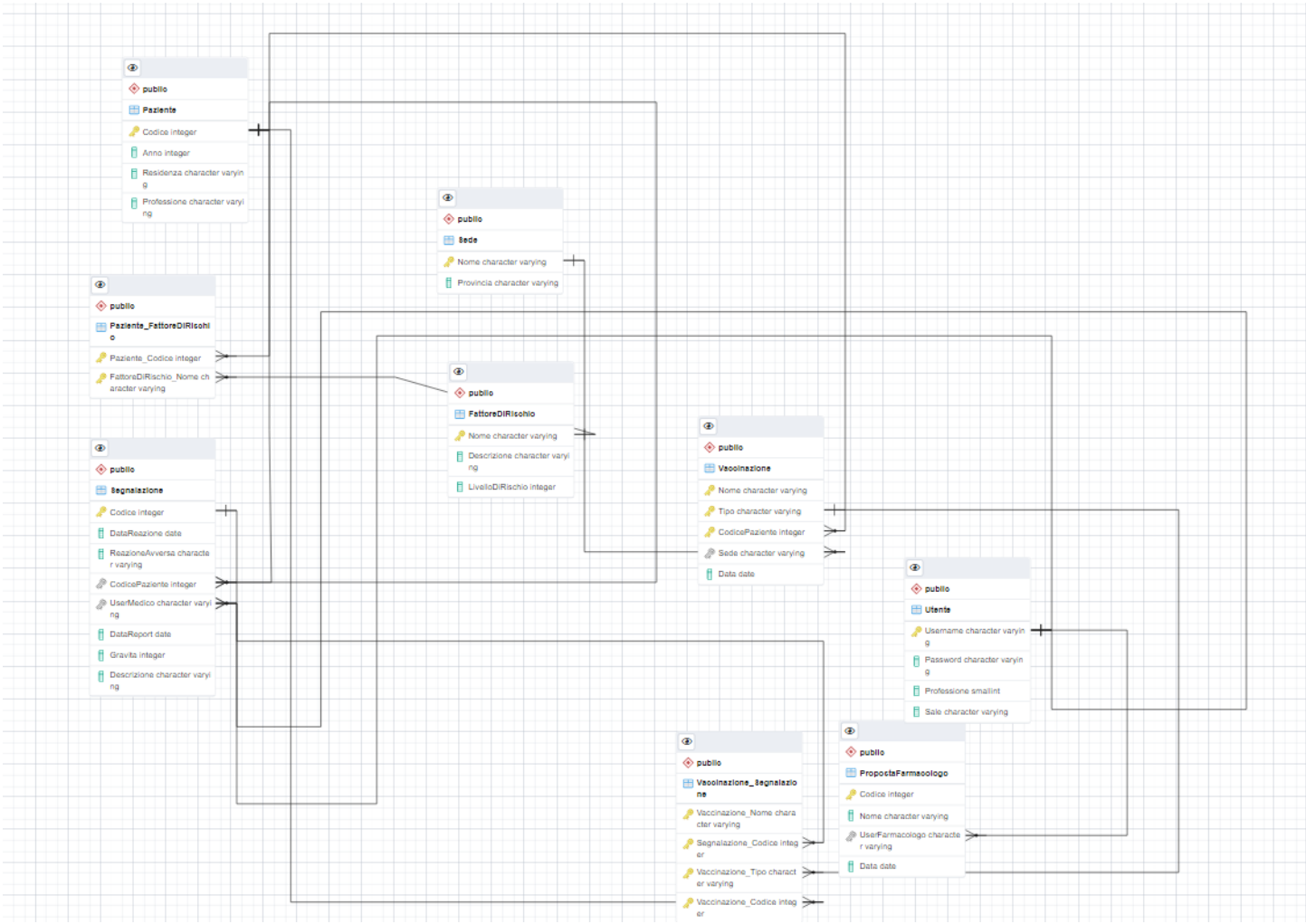

Figure 13: Diagramma Normalizzato

4.4 Progettazione logica

4.4.1 Schema relazionale

Paziente(<u>Codice</u>, Anno, Residenza, Professione)

PK: Codice

Paziente_FattoreDiRischio(Paziente_Codice, FattoreDiRischio_Nome)

PK: Paziente_Codice, FattoreDiRischio_Nome

FK: Paziente_Codice $\xrightarrow{references}$ Paziente(Codice)

FK: FattoreDiRischio_Nome $\xrightarrow{references}$ FattoreDiRischio(Nome)

FattoreDiRischio(Nome, Descrizione, LivelloDiRischio)

PK: Nome

Segnalazione(Codice, DataReazione, ReazioneAvversa, CodicePaziente, UserMedico, DataReport, Gravità, Descrizione)

PK: Codice

FK: CodicePaziente $\xrightarrow{references}$ Paziente(Codice)

FK: UserMedico $\xrightarrow{references}$ Utente(Username)

Vaccinazione(Nome, Tipo, CodicePaziente, Sede)

PK: Nome, Tipo, CodicePaziente

FK: CodicePaziente $\xrightarrow{references}$ Paziente(Codice)

FK: Sede $\xrightarrow{references}$ Sede(Nome)

Vaccinazione_Segnalazione(Vaccinazione_Nome, Vaccinazione_Tipo,Vaccinazione_Codice, Segnalazione_Codice)

PK: Vaccinazione_Nome, Vaccinazione_Tipo, Vaccinazione_Codice

FK: Vaccinazione_Nome $\xrightarrow{references}$ Vaccinazione(Nome)

FK: Vaccinazione_Tipo $\xrightarrow{references}$ Vaccinazione(Tipo)

FK: Vaccinazione_Codice $\xrightarrow{references}$ Vaccinazione(CodicePaziente)

FK: Segnalazione_Codice $\xrightarrow{references}$ Segnalazione(Codice)

Sede(Nome, Provincia)

PK: Nome

Utente(Username, Password, Professione, Sale)

PK: Username

PropostaFarmacologo(Codice, Nome, UserFarmacologo, Data)

PK: Codice

FK: UserFarmacologo $\xrightarrow{references}$ Utente(Username)

4.5 Cenni sulla sicurezza

4.5.1 Credenziali

Per garantire la sicurezza delle password è stata utilizzata una funzione di hash integrata in Java (*security.MessageDigest*) scegliendo come algoritmo **SHA-256**. Per generare i numeri casuali crittograficamente sicuri necessari, ossia i vari **salt** e il **pepper**, ci si è affidati a *java.security.SecureRandom*, una classe che integra un CSPRNG.

4.5.2 Queries

Le queries sono gestite con i *PreparedStatement*, una sotto-interfaccia di Statement. La differenza sostanziale è all'oggetto PreparedStatement viene data un'istruzione SQL nel momento in cui viene creato. Il vantaggio principale è che la dichiarazione è pre-compilata, evitando problemi di sicurezza come la *SQL injection*

5 Test e validazione

L'obiettivo della fase di test è stato dimostrare che il programma svolgesse i compiti per il quale è stato realizzato ed individuare eventualmente gli errori più probabili prima di renderlo pronto all'uso.

Si è pensato di procedere con dei test per tutte le **funzionalità del sistema** incorporate in questa prima release del prodotto, in modo tale da cercare di individuare eventuali input o sequenze di input per i quali il comportamento del software fosse errato.

Nella fase di **test di sviluppo** il *tester* del software ha suddiviso il lavoro in tre fasi sequenziali:

- *Test delle unità*
- *Test dei componenti*
- *Test del sistema*

5.1 JUnit test

Il test delle unità è stato automatizzato mediante il framework di automazione **JUnit**. In questa fase sono stati testati i metodi preposti a svolgere le funzionalità più importanti del programma come ad esempio il metodo *CreateReportsTableObject*:

```
package com.vaxreact;

import javafx.scene.control.TableColumn;
import org.junit.jupiter.api.Test;

import java.util.ArrayList;

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

class DocControllerTest {

    @Test
    void testCreateReportsTableObject() {

        String string = "PazienteZero";

        TableColumn<TableModel, String> column = new TableColumn<>();
        column = null;

        ArrayList<TableColumn<TableModel, String>> arrayListTest = new ArrayList<>();
```

```
        DocController docController = new DocController();

        int MAX_VALUE = docController.createReportsTableObject().size();

        for(int i = 0; i < MAX_VALUE ; i++)
        arrayListTest.add(column);

        assertEquals(arrayListTest,docController.createReportsTableObject());

        assertNotNull(docController.createReportsTableObject());

    }
```

oppure il metodo *getRowValue()*:

```
class QueryTest {

...

        @Test
    void testGetRowValue() throws SQLException {

        String query = "SELECT \"Nome\" FROM \"ReazioneAvversa\"";
        Query query1 = new Query();

        Connection connectDB = CreateConnection.getInstance().getConnection();
        Statement statement = connectDB.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE, R
        ResultSet queryOutput = statement.executeQuery(query);

        assert queryOutput != null;
        if (!queryOutput.next()) System.exit(0);
        queryOutput.next();

        ArrayList<String> s1 = new ArrayList<>();

        s1.add(queryOutput.getString(1));

        assertEquals(s1, query1.getRowValue(queryOutput));

        assert queryOutput != null;
        if (!queryOutput.next()) System.exit(0);
        queryOutput.next();

        ArrayList<String> s2 = new ArrayList<>();
        s2.add(queryOutput.getString(1));

        assertEquals(s2, query1.getRowValue(queryOutput));

        assert queryOutput != null;
        if (!queryOutput.next()) System.exit(0);

        queryOutput.next();
        ArrayList<String> s3 = new ArrayList<>();
        s3.add(queryOutput.getString(1));
```

```
        assertEquals(s3, query1.getRowValue(queryOutput));

...

}
    }
```

Nonstante sia stato privilegiato il fatto di testare i singoli metodi talvolta è stato necessario effettuare sequenze di test per il fenomeno dell'**incapsulamento**. Per esempio, per poter testare il metodo *checkReportLimit*, che verifica se sono state superate le 50 segnalazioni, è necessario aver eseguito il metodo *updatePatientView*.

5.2 Test dei componenti

Una volta analizzato il comportamento dei singoli metodi e delle singole classi si è cercato di testare il codice da un punto di vista più generale, studiando l'interazione tra più classi differenti.

In questa fase si è seguito il seguente iter:

1. Analisi del codice da testare ed elencazione di tutte le chiamate dei componenti esterni. Il valore dei parametri passati ai componenti esterni, laddove fossero presenti, sono stati scelti ai limiti estremi dei loro range per cercare di rilevare le inconsistenze tra le varie interfacce.
2. Effettuare stress test cercando di generare uno scambio di interazioni maggiore rispetto ad una situazione reale tra i vari componenti del sistema. Ad esempio l'inserimento di molteplici segnalazioni in rapida sucessione da parte del medico determina il superamento dell' *upper bound* per far scattare l'avviso di notifica del sistema verso l'interfaccia del farmacologo.

5.3 Test di sistema

Al termine dell'implementazione del software è stata effettuata una verifica di tutte le singole componenti integrate assieme. Dal momento che il team di sviluppo ha collaborato sempre a stretto contatto, il test di sistema si è sovrapposto al test delle componenti ed in questo modo è stato notevolmente semplificato il lavoro. Con l'ausilio dei diagrammi di sequenza sono stati effettuati vari test finali in modo tale da vedere quali fossero gli input ed output previsti:

- Validazione in fase di login
- Validazione del sistema di notifica
- Validazione inserimento delle segnalazioni