Università degli Studi dell'Insubria
Laurea triennale in Informatica
Progetto Laboratorio A

Centri Vaccinali

Manuale Tecnico

Gruppo:

Cristian Arcadi Mat: 745389
David Poletti Mat: 746597
Eros Marsichina Mat: 745299
Tommaso Morosi Mat: 741227

Professoressa:
Locoro Angela

29 agosto 2022

Indice

1 Introduzione

Centri vaccinali è un progetto sviluppato per il corso Laboratorio B di laurea in Informatica.

Lo scopo era quello di realizzare un'applicazione per la gestione eorganizzazionedi Centri Vaccinali, mediante l'uso del paradigma client-server, dei threads e del database, in particolare usando il DBMS PostgreSQL.

Il programma risulta thread-safe, inoltre, ogni operazione bloccante, è gestita da un thread diverso, cosi da non risultare in un blocco della UI.

Il progetto è sviluppato in Java16, l'interfaccia grafica è costruita con JavaFX 16 ed è stato sviluppato e testato su Windows 10/11.

1.1 Requisiti

Per poter eseguire questo programma è necessario scaricare il pacchetto Java (JRE) dal sito Oracle,inoltre, per non aver problemi di compatibiiltà, si consiglia di avere l'ultima versione installata di Windows. Per macOS bisogna installare la libreria JavaFX,con annesso SDK, scaricabile dal sito.

1.2 Installazione

Questo programma non ha bisogno di installazione, semplicemente se rispettati i **requisiti**, basta avviare i file jar (Server.jar e CentriVaccinali.jar) . Per avviare i file jar basta cliccare due volte su di esso, oppure, aprire il **terminale** ed eseguire il seguente comando:

```
java -jar "path_del_jar" [IP]
```

[Note:] Questa operazione va eseguita con entrambi i jar. [IP] è un parametro che il jar CentriVaccinali accetta, ma non obbligatorio. Se non si inserisce niente, si suppone che entrambi i jar si stiano eseguendo in locale. Se invece, i jar sono su due computer diversi, bisognerà procedere ad inserire al posto di [IP], l'indirizzo ip del server.

1.3 Librerie esterne utilizzate

Per lo sviluppo di questo progetto, come unica libreria esterna, abbiamo utilizzato la JavaFX 16. Questa libreria permette la integrazione nel progetto Java della parte grafica in modo facile ed efficace.

1.4 Struttura generale del sistema di classi

Le classi sono organizzate in tre package e una cartella resources

- centrivaccinali [PACKAGE]
  - PortaleOperatori.java
  - RegistraNuovoCentro.java
  - RegistraNuovoVaccinato.java
  - SelectionUI.java
  - SingoloCentroVaccinale.java
- cittadini [PACKAGE]
  - EventiAvversi.java
  - LoginUtente.java
  - MainCittadini.java
  - RegistraEventiAvversi.java
  - RegistrazioneUtente.java
- client-server [PACKAGE]
  - Server.java
  - ServerHandler.java
- resources [RISORSE]

- centrivaccinali
  - crowd.png
  - introduction.png
  - operatorPortalIcon.png
  - hub.png
  - medici.png
  - ospedale.png
  - person.png
  - siringa.png
  - stickman.png
- cittadini
  - citizenPortalIcon.png
  - errorWithDb.png
  - login.png
  - noCenters.png
  - noSearchResult.png
  - openCenterInfo.png
  - fiorellino.png
  - ospedale.png
  - search.png
  - stickmanCittadino.png
  - termometro.png
  - vaccnialiCittadino.png
- fxml
  - LoginUtente.fxml
  - LoadingPopup.fxml
  - MainCittadini.fxml
  - PortaleOperatori.fxml
  - RegistraEventiAvversi.fxml
  - RegistraNuovoCentroVaccinale.fxml
  - RegistraNuovoVaccinato.fxml
  - RegistraUtente.fxml
  - SelectionUI.fxml
  - VisualizzazioneCentroVaccinalePage1.fxml
  - VisualizzazioneCentroVaccinalePage2.fxml

2 Progettazione del Database

Il database "CentriVaccinali" è composto da 4 tabelle:

- centrivaccinali
- eventiavversi

- utente
- vaccinati

2.1 Tabelle

2.1.1 centrivaccinali

La tabella centrivaccinali, contiene tutte le informazioni che descrivono un centro vaccinale nel sistema

Nome Campo	Tipo	Lunghezza	Chiave 1*	Vincolo
ID	integer	11	PK	NOT NULL, AUTO INCREMENT
qualificatore	varchar	256		NOT NULL
via	varchar	256		NOT NULL
civico	varchar	2		NOT NULL
comune	varchar	256		NOT NULL
provincia	char	2		NOT NULL
cap	varchar	256		NOT NULL
tipologia	varchar	256		NOT NULL
nome	varchar	256		NOT NULL

2.1.2 eventiavversi

La tabella eventiavversi, descrive gli eventiavversi e li collega rispettivamente all'utente che li ha inseriti e al centro vaccinale dove è stato vaccinato

Nome Campo	Tipo	Lunghezza	Chiave 1*	Vincolo
ID	integer	11	PK	NOT NULL, AUTO INCREMENT
male_testa	integer			NOT NULL
febbre	integer			NOT NULL
dolori_muscolari	integer			NOT NULL
linfoadenopatia	integer			NOT NULL
tachicardia	integer			NOT NULL
crisi_ipertensiva	integer			NOT NULL
altri_sintomi	varchar	256		
id_centro	integer		FK (tabella centrivaccinali: ID)	NOT NULL
cf_utente	integer		FK (tabella utente: CF)	NOT NULL

2.1.3 utente

La tabella utente, contiene tutte le informazioni che descrivono un utente nel sistema

Nome Campo	Tipo	Lunghezza	Chiave 1*	Vincolo
nome	varchar	255		NOT NULL
cognome	varchar	255		NOT NULL
cf	char	16	PK	NOT NULL
data_nascita	date			NOT NULL
email	varchar	255		NOT NULL
password	varchar	255		NOT NULL

2.1.4 vaccinati

la tabella vaccinati, contiene tutte le informazioni relative a un paziente vaccinato (un utente può non essere vaccinato) e le collega al codice fiscale dell'utente e al centro vaccinale dove è stato vaccinato

Nome Campo	Tipo	Lunghezza	Chiave 1*	Vincolo
ID	integer		PK	NOT NULL, AUTO INCREMENT
nome	varchar	255		NOT NULL
cognome	varchar	255		NOT NULL
vaccino	varchar	255		NOT NULL
centrovaccinale	integer		FK(tabella centrivaccinali : ID)	
data_vaccinazione	date			NOT NULL
cf_utente	char	16		NOT NULL

2.2 SQL

In questa sezione verranno riportate tutte le query per la creazione della struttura del database (DDL)

2.2.1 centrivaccinali

```
    CREATE TABLE public.centrivaccinali (
    id integer NOT NULL,
    qualificatore character varying(256) NOT NULL,
    via character varying(256) NOT NULL,
    civico character varying(256) NOT NULL,
    comune character varying(256) NOT NULL,
    provincia character(2) NOT NULL,
    cap character varying(256) NOT NULL,
    tipologia character varying(256) NOT NULL,
    nome character varying(256)
);

CREATE SEQUENCE public.centrivaccinali_id_seq
    AS integer
    START WITH 1
    INCREMENT BY 1
    NO MINVALUE
    NO MAXVALUE
    CACHE 1;

ALTER TABLE ONLY public.centrivaccinali
    ADD CONSTRAINT centrivaccinali_nome_key UNIQUE (nome);

ALTER TABLE ONLY public.centrivaccinali
    ADD CONSTRAINT centrivaccinali_pkey PRIMARY KEY (id);
```

2.2.2 eventiavversi

```
    CREATE TABLE public.eventiavversi (
    id integer NOT NULL,
    male_testa integer NOT NULL,
    febbre integer NOT NULL,
    dolori_muscolari integer NOT NULL,
    linfoadenopatia integer NOT NULL,
    tachicardia integer NOT NULL,
    crisi_ipertensiva integer NOT NULL,
    altri_sintomi character varying(255),
    id_centro integer NOT NULL,
    cf_utente character(16) NOT NULL
);

CREATE SEQUENCE public.eventiavversi_id_seq
    AS integer
    START WITH 1
    INCREMENT BY 1
    NO MINVALUE
    NO MAXVALUE
    CACHE 1;

ALTER TABLE ONLY public.eventiavversi
    ADD CONSTRAINT eventiavversi_pkey PRIMARY KEY (id);

ALTER TABLE ONLY public.eventiavversi
    ADD CONSTRAINT cf_utente_fk FOREIGN KEY (cf_utente) REFERENCES public.utente(cf);

ALTER TABLE ONLY public.eventiavversi
```

```
    ADD CONSTRAINT eventiavversi_id_centro_fkey FOREIGN KEY (id_centro) REFERENCES
    public.centrivaccinali(id);
```

2.2.3 utente

```
    CREATE TABLE public.utente (
    nome character varying(255) NOT NULL,
    cognome character varying(255) NOT NULL,
    cf character(16) NOT NULL,
    data_nascita date NOT NULL,
    email character varying(255) NOT NULL,
    password character varying(255) NOT NULL
);

ALTER TABLE ONLY public.utente
    ADD CONSTRAINT cf_pk PRIMARY KEY (cf);
```

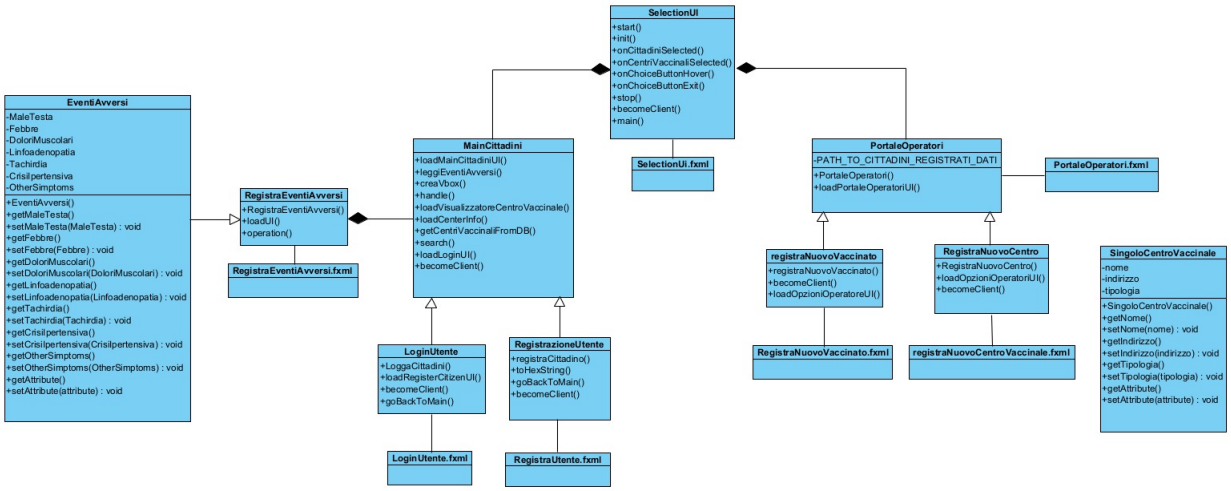
2.2.4 vaccinati

```
    CREATE TABLE public.vaccinati (
    id integer NOT NULL,
    nome character varying(256) NOT NULL,
    cognome character varying(256) NOT NULL,
    vaccino character varying(256) NOT NULL,
    centrovaccinale integer,
    data_vaccinazione date NOT NULL,
    cf_utente character(16) NOT NULL
);

CREATE SEQUENCE public.vaccinati_id_seq
    AS integer
    START WITH 1
    INCREMENT BY 1
    NO MINVALUE
    NO MAXVALUE
    CACHE 1;

ALTER TABLE ONLY public.vaccinati
    ADD CONSTRAINT vaccinati_pkey PRIMARY KEY (id);

ALTER TABLE ONLY public.vaccinati
    ADD CONSTRAINT centrovaccinale_fk FOREIGN KEY (centrovaccinale)
    REFERENCES public.centrivaccinali(id);
```

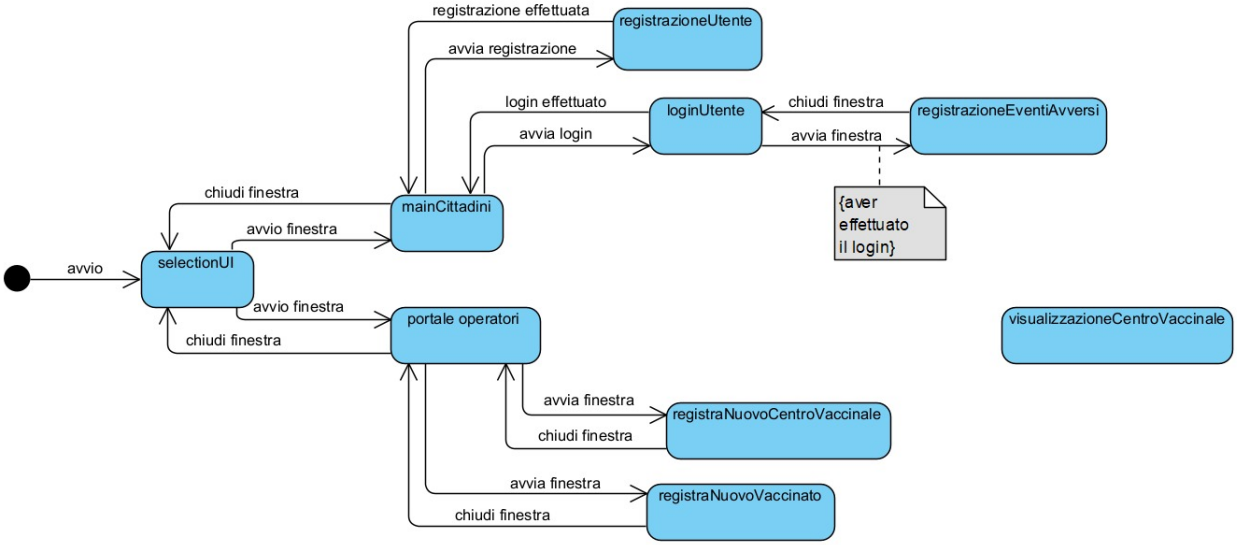
3 UML

In questa sezione verranno presentati gli schemi UML richiesti:

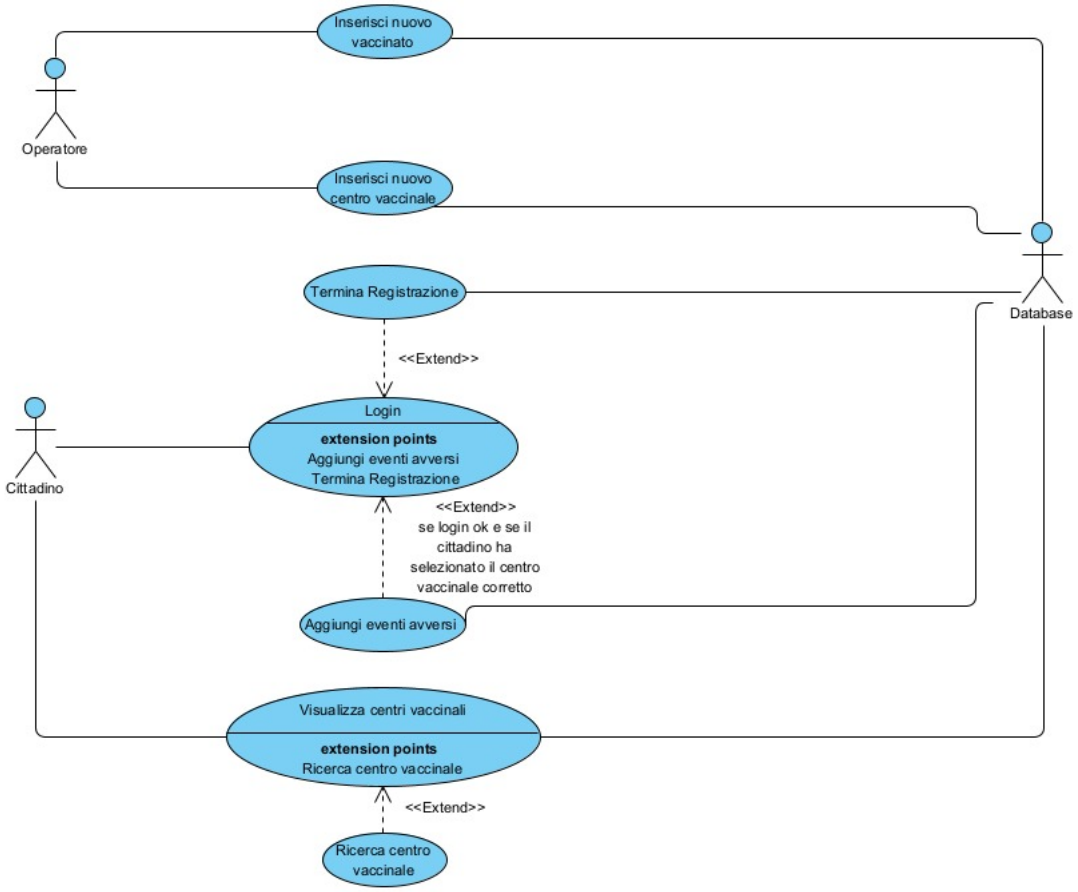
Class diagram: Rappresentazione del class diagram dell'intero sistema:



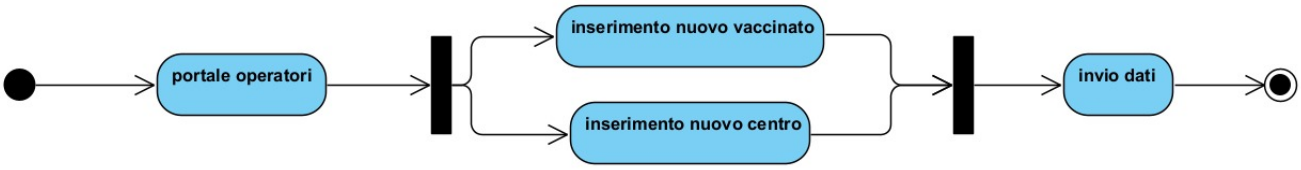
State diagram: Rappresentazione dello state diagram dell'intero sistema:



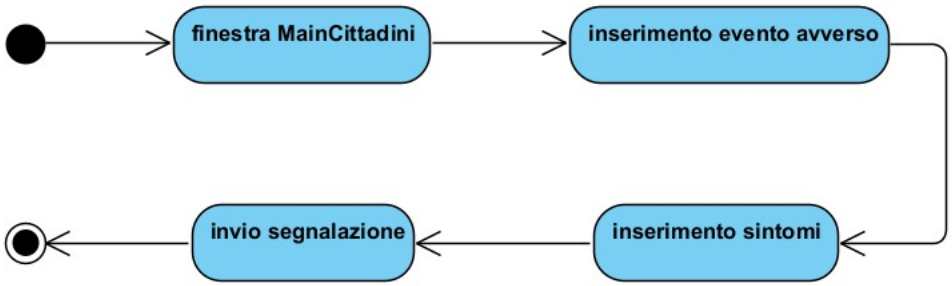
Use case diagram: Rappresentazione dello use case diagram dell'intero sistema:



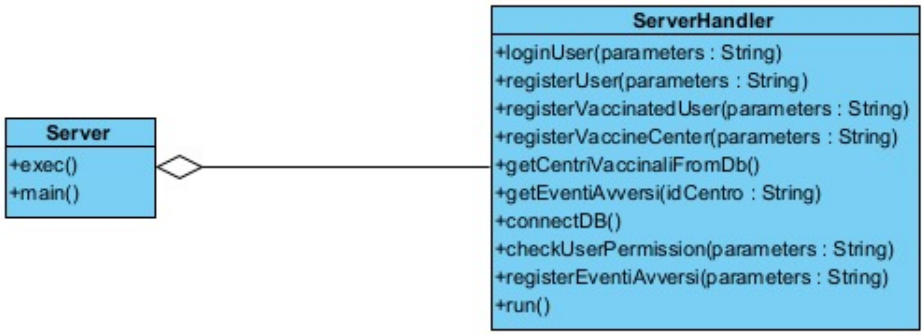
State diagram Operatori: Rappresentazione dello state diagram relativo agli operatori:



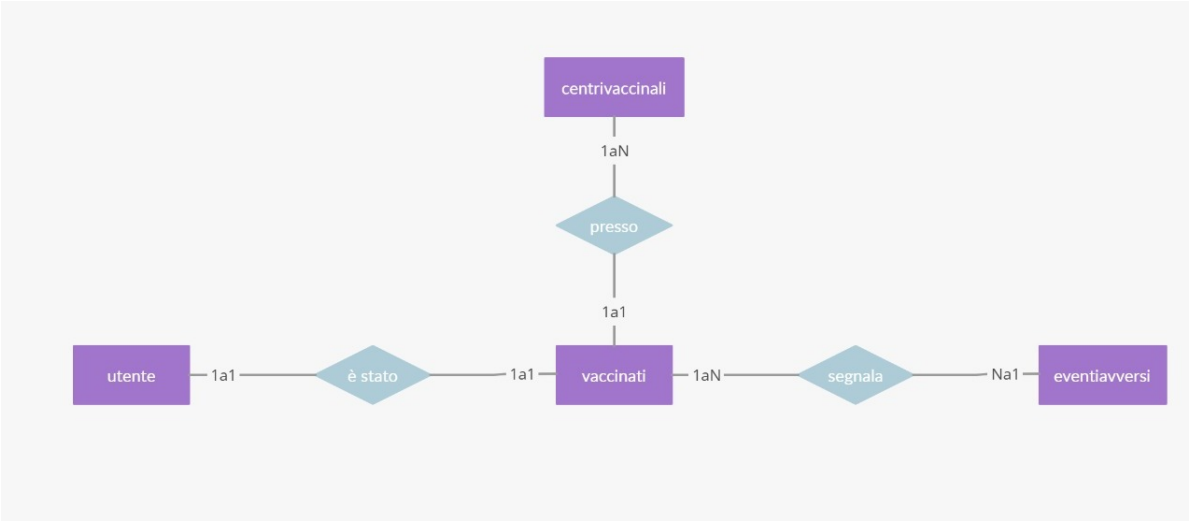
Activity diagram : Rappresentazione dell'activity diagram relativo alla pagina principale:



Server class diagram : Rappresentazione dell server class diagram:



Schema ER : Rappresentazione dello schema ER del database:



4 Troubleshooting

4.1 Errore connessione al server in avvio

Se dovesse presentarsi questo errore, significa che non si è avviato il server. Per utilizzare il programma bisogna: chiudere il programma, avviare il server, avviare il programma

4.2 Errore connesiona al database

Se dovesse presentarsi un errore di connessione al database, verificare che le porte al quale ci si connette siano libere.

Ora verranno presentate le classi in dettaglio, con una descrizione marginale della gestione dell'interfaccia grafica

5 Package: centrivaccinali

5.1 SelectionUI.java

SelectionUI.java è la prima classe che viene eseguita , si occupa di connettersi al server, e di caricare la UI riguardante la scelta di utilizzare il programma come cittadino , o come operatore vaccinale. I metodi piu importanti:

5.1.1 Start

l metodo Start si occupa di caricare e inizializzare il file SelectionUI.fxml, che contiene tutte le informazioni per creare il primo men'u di avvio del programma

5.1.2 becomeClient

Il metodo becomeClient si occupa di, qualora sia possibile, effetuare la prima connessione al server, per poi salvare il socket connesso e gli stream di input/output, per le altre classi

5.1.3 onCentriVaccinaliSelected

Il metodo onCentriVaccinaliSelected si occupa di reindirizzare l'utente alla sezione degli operatori qualora si abbia cliccato quel bottone

5.1.4 onCittadiniSelected

Il metodo onCentriVaccinaliSelected si occupa di reindirizzare l'utente alla sezione dei cittadini qualora si abbia cliccato quel bottone

5.2 PortaleOperatori.java

PortalePoeratori.java è la classe che si occupa di gestire se l'operatore, vuole inserire un nuovo paziente vaccinato, oppure inserire un nuovo centro di vaccinazione. I metodi piu importanti:

5.2.1 onNuovoCentroSelected

il metodo onNuovoCentroSelected si occupa di reindirizzare l'utente all'inserimento di un nuovo centro vaccinale

5.2.2 onNewVaccinate

il metodo onNewVaccinate si occupa di caricare reindirizzare l'utente all'inserimento di un nuovo paziente vaccinato

5.3 RegistraNuovoCentro.java

RegistraNuovoCentro.java si occupa di caricare la UI per la registrazione del centro vaccinale e di gestire l'inserimento a sistema di un centro vaccinale. I metodi piu importanti:

5.3.1 RegistraCentroVaccinale

Il metodo RegistraCentroVaccinale si occupa di:

- prendere i dati dalla UI e impacchettarli per il server:

  [Dati:] nome,qualificatore,via,vicivo,comune,provincia,cap,tipologia

```
1         Scene currentScene = ((Button) event.getSource()).getScene();
2         String nome = ((TextField) currentScene.lookup("#txt_nomeCentro")).getText()
              ;
3         String qualif = ((ChoiceBox<String>) currentScene.lookup("#cbx_qualificatore
              ")).getValue();
4         String via = ((TextField) currentScene.lookup("#txt_via")).getText();
5         String civico = ((TextField) currentScene.lookup("#txt_numeroCivico")).
              getText();
6         String com = ((TextField) currentScene.lookup("#txt_comune")).getText();
7         String prov = ((TextField) currentScene.lookup("#txt_provincia")).getText();
8         String cap = ((TextField) currentScene.lookup("#txt_cap")).getText();
9         String tipolog = ((ChoiceBox<String>) currentScene.lookup("#cbx_tipologia"))
              .getValue();
10        String parameters = nome + ";" + qualif + ";" + via + ";" + civico + ";" +
              com + ";" + prov + ";" + cap + ";" + tipolog;
11    }
```

  [Note:] **parameters** è l'insieme dei parametri, uniti in una stringa, pronti per essere spediti al server

- controllare la conformità dei dati:

  [Controllo:]Si controlla che la provincia sia minore di 2 caratteri, e che nessun campo sia rimasto vuoto

```
12        if(prov.length()>2){
13            Alert alert = new Alert(Alert.AlertType.ERROR);
14            alert.setTitle("Errore");
15            alert.setHeaderText(null);
16            alert.setContentText("Controllare i dati inseriti");
17            alert.showAndWait();
18        }
19        if (nome.equals("") || qualif == null || via.equals("") || civico.equals("")
              || com.equals("") || prov.equals("") || cap.equals("") || tipolog ==
              null) {
20            Alert alert = new Alert(Alert.AlertType.ERROR);
21            alert.setTitle("Errore");
22            alert.setHeaderText(null);
23            alert.setContentText("Controllare i dati inseriti");
24            alert.showAndWait();
25    }
```

- spedire i dati al server:

  [Spedizione:] Si spediscono i dati al server

```
26            becomeClient(parameters);
```

- gestire l'output:

  [Controllo:] si gestisce l'output del server, qualora ci sia un errore o meno

```
27                  String result = in.readLine();
28                  if (result.equals("true")) {
29                      Alert alert = new Alert(Alert.AlertType.INFORMATION);
30                      alert.setTitle("Successo");
31                      alert.setHeaderText(null);
32                      alert.setContentText("Centro vaccinale registrato");
33                      alert.showAndWait();
34                  } else if (result.equals("false")) {
35                      Alert alert = new Alert(Alert.AlertType.ERROR);
36                      alert.setTitle("Errore");
37                      alert.setHeaderText(null);
38                      alert.setContentText("Problemi con il server");
39                      alert.showAndWait();
40                  }
41                  ((Stage) currentScene.getWindow()).close();
```

  [Note:] result è l'esito dell'operazione che riceviamo dal server

5.3.2 becomeClient

il metodo becomeClient si occupa di prendere il socket e gli stream di input e output, resi disponibili precedentemente dalla **SelectionUI** e comunicare con il server, inviando la lista di parametri per registrare il centro

```
42          public void becomeClient(String parameters){
43              System.out.println("[CLIENT] - Sono gia connesso, prendo gli stream ");
44              Socket s = SelectionUI.socket_container;
45              out = SelectionUI.out_container;
46              in = SelectionUI.in_container;
47              out.println(parameters);
48              out.println(REGISTER_VACCINECENTRE_OPERATION_CODE);
49          }
```

5.4 RegistraNuovoVaccinato.java

RegistraNuovoVaccinato.java si occupa di caricare la UI per la registrazione del nuovo vaccinato e di gestirne l'inserimento di un vaccinato a sistema. I metodi piu importanti:

5.4.1 registraVaccinato

Il metodo registraVaccinato si occupa di:

- Recupero dati di input e impacchettarli per il server

  [Dati:] nome,cognome,codice fiscale,tipo vaccino,centro vaccinale,id vaccino,data vaccino,data vaccinazione

- Controllo sui dati

  [Controlli:]Si controlla che l'utente abbia inserito tutti i dati e che siano corretti

- Spedizione dati al server

Note: Il codice di queste operazioni è simile a quello della classe precedente, quindi non verrà riportato

Complessità
La complessità stimata di questo metodo è O(1) poichè non si usano ne cicli ne strutture dati, il tutto è lineare

5.5 SingoloCentroVaccinale.java

SingoloCentroVaccinale è la classe che permette la creazione dell'oggetto centro vaccinale.La classe è serializzata poichè verrà usata per riempire una struttura dati dal server e inviarne i dati al client, e contiene i seguenti metodi:

5.5.1 SingoloCentroVaccinale

Questo metodo è il costruttore della classe. I parametri che lo definiscono sono **String nome, String indirizzo, String tipologia**.

5.5.2 Metodi getter & setter e toString

La classe presenta dei metodi getter & setter per andare, al di fuori della classe, a prendere certi attributi o assegnargli un certo valore. E' presente un metodo toString per andare a descrivere il centro vaccinale

6 Package: cittadini

6.1 RegistrazioneUtente.java

La classe RegistrazioneUtente si occupa di prendere i dati di registrazione dell'utente, hashare la password per una maggiore sicurezza e inviare tutto al server. I metodi piu importanti:

6.1.1 registraCittadino

Il metodo registraCittadino si occupa di:

- Recupero dati di input e impacchettarli per il server

  [Dati:] nome,cognome,user,codice fiscale,password,password-confermata,data di nascita

- Controllo sui dati

  [Controlli:]Si controlla che l'utente abbia inserito tutti i dati e che siano corretti

- Hashare la password

```
50                  MessageDigest messageDigest = MessageDigest.getInstance("SHA-256");
51                  byte[] hash = messageDigest.digest(pwd.getBytes(StandardCharsets.
                        UTF_8));
52                  pwd = toHexString(hash);
```

  [Note:] in questo frammento di codice è descritto l'hashing della password. Si usa come "MessageDigest" lo standard SHA-256 poichè ritenuto sicuro e affidabile. L'hashing vero e proprio è effettuato dalla funzione **toHexString(hash)**

- Spedizione dati al server

Note: Il codice di queste operazioni è simile a quello della classe precedente, quindi non verrà riportato

Complessità
La complessit'a generale 'e data dalle complessit'a delle funzioni di Hash. la funzione di hash ha complessit'a, semplificata, O(n1 + n2) dove n1 'e la stringa di input e n2 'e la stringa di output

6.1.2 toHexString

Il metodo toHexString si occupa di ricostruire la password dopo aver ottenuto i byte dalla funzione di hash

- Converte un array di byte in una stringa. Viene utilizzato dopo aver effettuato l'hasing di una stringa, per ricomporre quest'ultima

```
53  private String toHexString(byte[] array) {
54          StringBuilder sb = new StringBuilder(array.length * 2);
55
56          for (byte b : array) {
57              int value = 0xFF & b;
58              String toAppend = Integer.toHexString(value);
59
60              sb.append(toAppend);
61          }
62          sb.setLength(sb.length() - 1);
63          return sb.toString();
64      }
```

Complessità
La complessità di questo metodo , è data dall'uso di risorse dato dallo scorrimento dell'array. Quindi O(n)

6.2 LoginUtente.java

La classe LoginUtente.java si occupa di prendere username e password tramite input dell'utente e confrontare i dati tramite il server per verificare che siano corretti per la fase di login. I metodi piu importanti:

6.2.1 loggaCittadini

- Recupero dati di input

  [Dati:] Nome, password

- Hashing password

  [Hashing:] si effettua l'hashing della password tramite il metodo **toHexString**

- Invio parametri al server e gestione del suo output

  [Server:] se si ha un riscontro positivo dal server, si effetua il login, sennò apparirà un popup di errore

6.3 RegistraEventiAvversi.java

La classe RegistraEventiAvversi.java si occupa di prendere i dati inerenti agli eventi avversi comunicati dall'utente, e di spedirli al server per memorizzarli. I metodi piu importanti:

6.3.1 registraEventiAvversi

- Recupero dati di input

  [Dati:] mal di testa,febbre,dolori muscolari o articolari,linfoadenopatia,tachicardia,crisi ipertensiva

- Spedizione dati al sever

6.4 RegistrazioneUtente.java

La classe RegistrazioneUtente.java si occupa di registrare l'utente a sistema. I metodi più importanti:

6.4.1 registraCittadino

- recupero dati di input

  [Dati:] nome,cognome,user,codice_fiscale,pasword,conferma_passwprd,datanascita

- controlli sulla password

  [Controlli:] si controlla che le due password siano uguali. (password e conferma della password)

- hashing password

  [Hashing:] si utilizza il metodo **toHexString** per hashare la password

- invio dati al server

Note: il codice è del tutto simile a quello visto in metodi precedenti, quindi non viene descritto

7 Package: Client-Server

7.1 Server.java

La classe Server.java si occupa di istanziare il server e restare in attesa per eventuali connessioni, è un server multi threading, quindi non c'è un limite al numero di connessioni che può ricevere. Ogni qualvolta il server riceve una connessione, istanzia un ServerHandler. I metodi piu importanti:

7.1.1 exec

- Attesa di connessioni
- Creazione **ServerHandler**

```
65          public void exec() {
66              try {
67                  while (true) {
68                      s = server_socket.accept();
69                      new ServerHandler(s);
70                  }
71              } catch (IOException e) {
72                  e.printStackTrace();
73              }
74          }
```

7.2 ServerHandler.java

La classe ServerHandler.java si occupa di gestire tutte le operazioni che il client richiede al server e di comunicarne l'esito.

7.2.1 Struttura e funzionamento del serverHandler

Il serverHandler viene creato dal Server, dopo aver accettato una connessione. La sua struttura consiste in uno switch che riceve dal client due dati:

- parametri

  i parametri sono tutti i dati necessari, che il client invia al server, per eseguire una richiesta, ad esempio , i dati di un paziente da inserire a sistema. I parametri ricevuti, sono tutti formattati secondo lo standard **CSV**: "nome;cognome"

- operation code

  l'operation code, è un numero costante che consiste nella operazione da svolgere da parte del server, questo serve al server per capire, grazie allo switch, quale funzione deve eseguire

  [Codice:]

```
75          public void run() {
76              super.run();
77              System.out.println("[THREAD] - Server thread startato");
78              try{
79                  in = new BufferedReader(new InputStreamReader(s.getInputStream()));
80                  out = new PrintWriter(new BufferedWriter(new OutputStreamWriter(s.
                        getOutputStream())), true);
81                  while (true){
82                      System.out.println("[THREAD] Ascolto");
83                      parameters = in.readLine();
84                      op = in.readLine();
85                      op_converted = Integer.parseInt(op);
86                      }
87                      switch (op_converted) {
88                          case LOGIN_USER_OP_CODE -> {
89                              System.out.println("[THREAD] Login chiamata");
90                              login(parameters);
91                          }
```

  [Note:] parameters e op vengono ricevuto dal client, come si può vedere dal metodo **becomeClient**. Questo è solamente un pezzo di codice per descriverne il funzionamento, non è il server completo

7.2.2 connectDB

Il metodo connectDB si occupa di connettere il server al database

- Tentativo di connessione

```
92          private Connection connectDB() throws SQLException {
93              Connection conn = DriverManager.getConnection("URL DATABASE", "NOME
                    DATABASE", "PASSWORD DATABASE");
```

- Gestione del risultato

```
94              if (conn != null) {
95                  System.out.println("[DB - THREAD] - Sono connesso al db");
96              } else {
97                  System.err.println("[DB - THREAD] - Non sono connesso al db");
98              }
99              return conn;
100         }
```

7.2.3 login

Metodo del server per effetuare il login

- spacchettamento dati ricevuti dal client

```
101         private void login(String parameters) {
102             String[] parameters_splitted = parameters.split(";");
103             String email = parameters_splitted[0];
104             String cf=null;
105             String pwd = parameters_splitted[1];
```

- connessione al database

```
106             Connection con = connectDB();
```

- preparazione ed esecuzione query sql

```
107             PreparedStatement stm = con.prepareStatement("SELECT cf FROM public.
                    utente where email=? and password =?");
108             stm.setString(1, email);
109             stm.setString(2, pwd);
110             ResultSet result = stm.executeQuery();
111             if(result.next()){
112                 cf = result.getString("cf").toUpperCase();
113             }
114             out.println(cf);
```

- comunicazione dell'esito dell'operazione al client

```
115                 out.println(cf)
```

Note: in questo caso viene mandato al client il suo CF come esito dell'operazione, negli altri casi verrà mandato "true" per esito positivo, o "false" per esito negativo

7.2.4 registerUser

Metodo del server per gestire la richiesta di registrazione utente

- spacchettamento dati ricevuti dal client
- connessione al database
- preparazione ed esecuzione query sql

```
116             String sql ="insert into public.utente(nome,cognome,cf,data_nascita,
                    email,password) values (?,?,?,?,?,?)" ;
117             PreparedStatement stm = con.prepareStatement(sql);
118             stm.setString(1,name);
119             stm.setString(2,surname);
120             stm.setString(3,userCF);
121             stm.setDate(4,date1);
122             stm.setString(5,user);
123             stm.setString(6,pwd);
124             int result = stm.executeUpdate();
```

- comunicazione dell'esito dell'operazione al client

Note: il funzionamento di questo metodo è del tutto analogo agli altri, usa solo dati diversi, verrà descrita solamente la query

7.2.5 registerVaccinatedUser

Metodo del server per gestire la richiesta di registrazione di un utente vaccinato

- spacchettamento dati ricevuti dal client
- connessione al database
- preparazione ed esecuzione query sql

```
125             String sql ="insert into vaccinati (nome,cognome,cf_utente,vaccino,
                    centrovaccinale,data_vaccinazione) VALUES (?,?,?,?,?,?)";
126             PreparedStatement stm = con.prepareStatement(sql);
127             stm.setString(1,nome);
128             stm.setString(2,cognome);
129             stm.setString(3,codice_fiscale);
130             stm.setString(4,tipoVaccino);
131             stm.setString(5,centroVaccinale);
132             stm.setDate(6,dataVaccinazioneSQL);
133             int result = stm.executeUpdate();
```

- comunicazione dell'esito dell'operazione al client

Note: il funzionamento di questo metodo è del tutto analogo agli altri, usa solo dati diversi, verrà descrita solamente la query

7.2.6 registerVaccineCenter

Metodo del server per gestire la richiesta di registrazione di un centro vaccinale

- spacchettamento dati ricevuti dal client
- connessione al database
- preparazione ed esecuzione query sql

```
134             String sql = "INSERT INTO public.centrivaccinali (qualificatore,via,
                    civico,comune,provincia,cap,tipologia,nome)\n" +
135                     "VALUES(?,?,?,?,?,?,?,?)";
136             PreparedStatement stm = con.prepareStatement(sql);
137             stm.setString(1, qualificatore);
138             stm.setString(2, via);
139             stm.setString(3, civico);
140             stm.setString(4, comune);
141             stm.setString(5, provincia);
142             stm.setString(6, cap);
143             stm.setString(7, tipologia);
144             stm.setString(8, nome);
145             int result = stm.executeUpdate();
```

- comunicazione dell'esito dell'operazione al client

Note: il funzionamento di questo metodo è del tutto analogo agli altri, usa solo dati diversi, verrà descrita solamente la query

7.2.7 getCentriVaccinaliFromDb

Metodo del server che preleva dal database tutti i centri vaccinali e li spedisce al client

- creazione struttura dati Vector e preparazione variabili di appoggio

```
146             private void getCentriVaccinaliFromDb(){
147                 Vector<SingoloCentroVaccinale> vector = new Vector<>();
148                 String nome_db;
149                 String qualificatore_db;
150                 String via_db;
151                 String civico_db;
152                 String comune_db;
153                 String provincia_db;
```

```
154                     String cap_db;
155                     String tipologia_db;
```

- connessione al database
- preparazione ed esecuzione query sql

```
156             String sql = "SELECT * from centrivaccinali";
157             PreparedStatement stm = con.prepareStatement(sql);
158             ResultSet res = stm.executeQuery();
```

- prelievo del risultato e riempimento struttura dati

```
159              String sql = "SELECT * from centrivaccinali";
160              PreparedStatement stm = con.prepareStatement(sql);
161             ResultSet res = stm.executeQuery();
162                 while(res.next()){
163                     nome_db = res.getString("nome");
164                     via_db = res.getString("via");
165                     qualificatore_db = res.getString("qualificatore");
166                     civico_db = res.getString("civico");
167                     comune_db = res.getString("comune");
168                     provincia_db = res.getString("provincia");
169                     cap_db = res.getString("cap");
170                     String indirizzo = via_db+" "+nome_db+", "+civico_db+", "+
                            comune_db+" ("+provincia_db+") "+cap_db;
171                     tipologia_db = res.getString("tipologia");
172                     vector.add(new SingoloCentroVaccinale(nome_db, indirizzo,
                            tipologia_db));
173               }
```

- invio struttura dati , tramite serializzazione, al client

```
174             os = new ObjectOutputStream(s.getOutputStream());
175             os.writeObject(vector);
176 }
```

Note: **os** è una variabile di tipo ObjectOutputStream, usata per inviare l'intero vector al client, contentente oggetti di tipo **SingoloCentroVaccinale**

7.2.8 getEventiAvversi

Metodo del server per prelevare tutti gli eventi avversi dal database

- creazione struttura dati Vector e preparazione variabili di appoggio
- connessione al database
- preparazione ed esecuzione query sql

```
177             String sql="SELECT * FROM eventiavversi ea JOIN centrivaccinali cv ON ea
                    .id_centro=cv.id WHERE cv.id="+idCentro;
178             PreparedStatement prepSt=con.prepareStatement(sql);
179             ResultSet result=prepSt.executeQuery();
```

- prelievo del risultato e riempimento struttura dati
- invio struttura dati , tramite serializzazione, al client

Note: Il codice è molto simile a uno de metodi già descritti, quindi non verrà descritto

7.2.9 registerEventiAvversi

Metodo del server per registrare gli eventi avversi inseriti da un utente

- spacchettamento dati ricevuti dal client
- connessione al database
- preparazione ed esecuzione query sql

```
180         String sql="INSERT INTO eventiavversi VALUES (DEFAULT,?,?,?,?,?,?,?,?,?)";
181         try {
182             PreparedStatement prepSt = con.prepareStatement(sql);
183             prepSt.setInt(1,maleTesta);
184             prepSt.setInt(2,febbre);
185             prepSt.setInt(3,doloriMuscolari);
186             prepSt.setInt(4,linfoadenopatia);
187             prepSt.setInt(5,tachicardia);
188             prepSt.setInt(6,crisiIpertensiva);
189             prepSt.setString(7,otherSymptoms);
190             prepSt.setInt(8,idCentro);
191             prepSt.setString(9,cfUtente);
192             prepSt.executeUpdate();
193             out.println(true);
194         } catch (SQLException e) {
195             e.printStackTrace();
196             out.println(false);
197         }
```

- comunicazione dell'esito dell'operazione al client

Note: il funzionamento di questo metodo è del tutto analogo agli altri, usa solo dati diversi, verrà descrita solamente la query

7.2.10 checkUserPermission

- spacchettamento dati ricevuti dal client
- connessione al database
- preparazione ed esecuzione query sql

```
198             String sql="SELECT COUNT(*) AS rowCount FROM vaccinati v JOIN utente u
                    ON v.cf_utente=u.cf WHERE u.cf=? AND v.centrovaccinale=?";
199             sql="SELECT COUNT(*) AS rowCount FROM eventiavversi ea WHERE ea.
                    id_centro=? AND ea.cf_utente=?";
```

- comunicazione dell'esito dell'operazione al client

Note: il funzionamento di questo metodo è del tutto analogo agli altri, usa solo dati diversi, verrà descrita solamente la query

8 Gestione interfaccia

Per la gestione della interfaccia, sono presenti dei file .fxml dove all'interno, in **markup language**, ne è definita la struttura

Note: il **markup language** è un insieme di regole che descrivono i meccanismi di rappresentazione (strutturali, semantici, presentazionali) o layout di un testo; facendo uso di convenzioni rese standard, tali regole sono utilizzabili su più supporti. Il markup language ha una struttura fortemente gerarchica

8.0.1 SelectionUI.fxml

Questo file si occupa della descrizione della pagina principale del programma, presenta due **Button** che, grazie all'evento **onAction** scatenato dall'utente cliccandoci, richiamano due funzioni diverse:

- onCittadiniSelected
- onCentriVaccinaliSelected

8.0.2 RegistraNuovoVaccinato.fxml

RegistraNuovoVaccinato descrive la pagina per inserire un paziente all'interno del database al momento della vaccinazione. Presenta 3 **textfields**(nome paziente, cognome paziente, codice fiscale), due **choicebox**(vaccino somministrato, centro vaccinale) e un **datepicker**(data vaccinazione). Alla pressione del pulsante **conferma** viene richiamata la funzione **registraVaccinato**

8.0.3 PortaleOperatori.fxml

PortaleOperatori descrive la pagina per scegliere l'inserimento di un nuovo vaccinato o di un nuovo centro vaccinale. Presenta due **Button** che, grazie all'evento **onAction** scatenato dall'utente cliccandoci, richiamano due funzioni differenti:

- onNewVaccinate
- onNuovoCentroSelected

8.0.4 RegistraNuovoCentroVaccinale.fxml

RegistraNuovoCentroVaccinale descrive la pagina per inserire un nuovo centro vaccinale all'interno del database. Presenta 6 **textfields**(nome centro, Via, numero civico,Comune, Provincia, cap), 2 **choicebox**(Qualificatore, Tipologia), alla pressione del pulsante **registra** viene richiamata la funzione **registraCentroVaccinale**

8.0.5 mainCittadini.fxml

mainCittadini è la pagina per ricercare tramite input dell'utente un centro vaccinale all'interno del database. Presenta 1 **Textfield** in cui si può inserire la ricerca per nome, comune e tipologia, 1 **scrollpane** in cui vengono mostrati i centri vaccinali che soddisfano la ricerca

8.0.6 loginCittadino.fxml

loginCittadino descrive la pagina per eseguire il login del cittadino, presenta 2 **textfields** (Email, Password) e 2 **Button**, alla pressione del pulsante **login** viene richiama la funzione **loggaCittadini**, il pulsante registrati indirizza l'utente nella pagina della registrazione **RegistraUtente.fxml**

8.0.7 visualizzazioneCentroVaccinale.fxml

visualizzazioneCentroVaccinale descrive la pagina per la visualizzazione degli eventi avversi inerenti al centro vaccinale scelto. Inoltre presenta un **Button** (Inserisci Evento Avverso) per,se loggati, inserire gli eventi avversi individuati dal singolo cittadino e una **textfield** per scrivere un commento.

8.0.8 registraEventiAvversi.fxml

registraEventiAvversi descrive la pagina per segnalare eventuali eventi avversi da vaccinazione covid19, presenta 1 **textfields** in cui si possono inserire altri sintomi, 6 **choicebox** con un intervallo da 1 a 5 per segnalare la severità del sintomo rappresentato, 2 **Button**:

- Annulla (freccia indietro (¡-) , in alto a sinistra)
  - Permette di tornare alla pagina precedente
- Conferma
  - Richiama la funzione **registraEventiAvversi**

8.0.9 RegistraUtente.fxml

RegistraUtente descrive la pagina di registrazione di un nuovo utente, presenta 6 **textfields** per l'inserimento dei seguenti dati (nome, cognome, codice fiscale, password, conferma passwoed, email), 1 **datapicker** per inserire la data di nascita, e 2 **button**

- Annulla (freccia indietro (¡-) , in alto a sinistra)
  - Permette di tornare alla pagina precedente
- Conferma
  - Richiama la funzione **registraCittadino**