



Progetto Centri Vaccinali
Laboratorio Interdisciplinare B

Centro Operatori

Manuale Tecnico

Preparato da

Daniel Satriano Mat. 745232

Claudio Menegotto Mat. 745394

Sommario

INTRODUZIONE	3
Librerie esterne utilizzate	3
Impostare parametri personalizzati per la connessione al server RMI	4
Struttura generale delle classi.....	5
CORE CLASSES.....	6
Classi enumerative	8
Interfacce	10
UML DIAGRAMS	11
Class Diagram.....	11
State Diagram	12
Use-case Diagram.....	12

INTRODUZIONE

1. **Centro Operatori** è un progetto sviluppato nell'ambito del progetto di Laboratorio A per il corso di laurea in Informatica dell'Università degli Studi dell'Insubria.
Il progetto è sviluppato in Java 12, usa un'interfaccia grafica costruita con OpenJFx 12 ed è stato sviluppato e testato sul sistema operativo Windows 10 e Windows 11.

Librerie esterne utilizzate

1.1 L'applicazione fa uso di 3 librerie:

- **OpenJFx 12:** La libreria OpenJFx 12 contiene tutti gli elementi per lo sviluppo dell'interfaccia grafica. L'utilizzo di questa libreria si è reso necessario poiché JavaFx non è più incluso nel Java Development Kit di Oracle dalla release di Java 9.
- **Jfoenix 9.0.10:** La libreria Jfoenix è una open source di java che permette di implementare (graficamente) "Google Material Design". Usato assieme a scene builder.
- **Gson 2.8.6:** La libreria Gson è una libreria sviluppata da Google che permette la serializzazione e la deserializzazione di oggetti a JSON o il contrario.

Impostare parametri personalizzati per la connessione al server RMI

Se si vuole cambiare l'indirizzo IP e/o la PORTA per la connessione al server è possibile farlo modificando il file startPortaleOperatori.bat, situato nella cartella "../Eseguibili".

È possibile aprire il file facendo tasto destro del mouse su di esso e selezionando la voce "modifica" o "edit".

Il file al suo interno presenterà una stringa di questo tipo [Figura 1], dove "localhost" rappresenta l'indirizzo IP e "1099" la porta del registro RMI.

E' possibile cambiare l'indirizzo di connessione al server semplicemente eliminando localhost e inserendo l'indirizzo desiderato come d'esempio in [Figura 2].

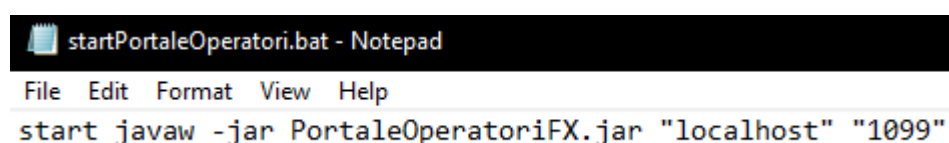


FIGURA 1

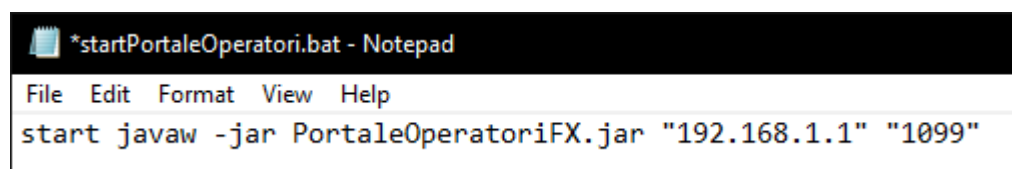


FIGURA 2

Nota bene che le virgolette sono **sempre** presenti e che è possibile fare lo stesso procedimento anche per la porta.

Struttura generale delle classi

1.2 Il progetto è strutturato fondamentalmente in 2 rami: le core classes + classi enumerative e le classi adibite alla gestione dell'interfaccia grafica [Figura 1]

- **Classi "core":**
 - CentroVaccinale
 - EventoAvverso
 - Indirizzo
 - Storico
 - UtenteVaccinato
 - ApiRequest
 - DatabaseHelper
- **Classi enumerative:**
 - Evento
 - Qualificatore
 - Severita
 - Tipologia
 - Vaccini
- **Controllers:**
 - MainWindow
- **Rispettivi FXML:**
 - MainWindow
- **Interfacce:**
 - intOperators

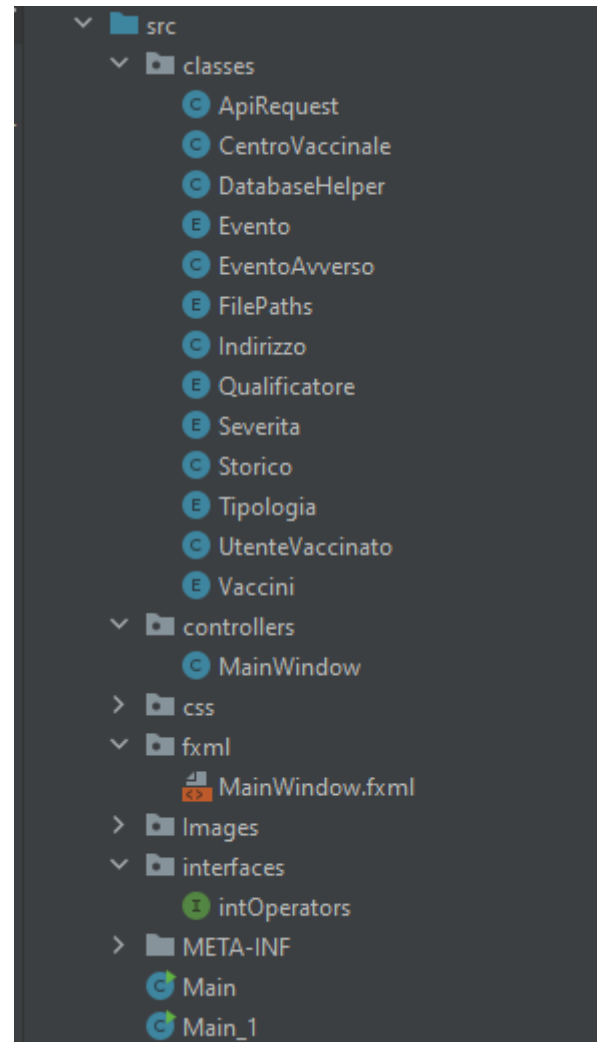


FIGURA 3

CORE CLASSES

1.1 Verranno presentate ora le core classes nel dettaglio.

(N.B -> la maggior parte delle classi presenta un override del toString e un metodo costruttore che non andremo a citare)

CentroVaccinale:

1.2 La classe **Centro Vaccinale** identifica i centri vaccinali e le loro funzioni.

Più in particolare presenta al suo interno quattro tag per l'identificazione:

- 1 `LinkedList<Short> IDVaccinazioni` : è una lista di ID delle vaccinazioni effettuate nel centro, in modo da poterne tenere traccia.
- 2 `String Nome` : indica il nome del centro vaccinale.
- 3 [Indirizzo](#) `Indirizzo` : va ad indicare l'indirizzo stradale del centro.
- 4 [Tipologia](#) `Tipologia` : indica la tipologia del centro.

La classe inoltre presenta inoltre un metodo:

- `getNome()` : che restituisce il nome del centro vaccinale.

EventoAvverso:

1.3 La classe **Evento Avverso** identifica e classifica gli eventi avversi che sono capitati agli utenti a cui è stato somministrato il vaccino.

Presenta al suo interno quattro tag, i quali verranno adesso elencati:

- 1 [Evento](#) `evento` : va ad identificare il tipo di evento che è capitato all'utente.
- 2 [Severita](#) `severità` : indica, come descritto dal nome, la severità dell'evento subito dall'utente.
- 3 `Short IDVaccinazione` : serve per tenere traccia della vaccinazione somministrata all'utente che sta riportando l'evento avverso.
- 4 `String noteOpzionali`: come da nome, sono note opzionali che l'utente può aggiungere prima di pubblicare l'evento avverso.

Indirizzo:

1.4 La classe **indirizzo** serve come classe complementare alla classe [CentroVaccinale](#), in quanto contiene al suo interno tutte le informazioni riguardanti l'indirizzo stradale del centro. Al suo interno possiamo trovare sei tag che la caratterizzano:

- 1 [Qualificatore](#) `qualificatore` : va ad indicare il qualificatore dell'indirizzo stradale.
- 2 `String nome` : nome della strada a cui fa riferimento l'indirizzo.
- 3 `Int numeroCivico` : numero civico della strada ove situato il centro.
- 4 `String comune` : comune dove è situato il centro.
- 5 `String provincia` : provincia dove è situato il centro.
- 6 `Int cap` : codice di avviamento postale del comune.

Storico:

1.5 La classe **Storico** serve a implementare la funzione dello storico dei vaccini nell'applicazione, presenta al suo interno due tag:

- 1 StringProperty **informazioniSomministrazioni** : contiene al suo interno informazioni riguardanti il vaccino usato e il nome e cognome della persona a cui è stato somministrato.
- 2 StringProperty **dataSomministrazione** : indica la data di somministrazione del vaccino.

UtenteVaccinato:

1.6 La classe **Utente Vaccinato** serve a salvare le informazioni degli utenti vaccinati di tutti i centri registrati nell'applicativo. Al suo interno presenta 8 tag:

- 1 String **nomeCentroVaccinale** : nome del centro.
- 2 String **nome** : nome dell'utente vaccinato.
- 3 String **cognome** : cognome dell'utente vaccinato.
- 4 String **codiceFiscale**: codice fiscale dell'utente vaccinato.
- 5 String **dataSomministrazione** : data somministrazione del vaccino.
- 6 [Vaccini](#) **vaccino** : tipo di vaccino effettuato.
- 7 Short **idVaccinazione** : id della vaccinazione.
- 8 [EventoAvverso](#) **evento** : **

La classe presenta tre metodi:

- **getIdVaccinazione()** : restituisce l'idVaccinazione dell'utente.
- **getInformation()** : restituisce tutte le informazioni dell'oggetto sotto forma di stringa in UPPER case.
- **getDataSomministrazione()** : restituisce la data di somministrazione-

ApiRequest:

1.7 La classe **Api request** è una classe utilitaria alla UI, in quanto ha il compito di andare a prelevare le informazioni che vengono mostrate nella home.

Per far questo adopera la libreria java.net, in particolare le sottoclassi della classe http.

E anche la libreria gson di google per la manipolazione del json.

Presenta al suo interno tre metodi:

- **makeRequest(String url)** : prende in input una pagina web e ne restituisce le informazioni sotto forma di JSONArray andando a richiamare il metodo [convertJSONintoJSONObject](#). (N.B assicurarsi che la pagina web data in input printi a schermo un array di json per il corretto funzionamento del metodo.)
- **convertJSONintoJSONObject(String json)** : metodo che presa in input una stringa contenente della sintassi json la trasforma in un oggetto JSONObject.
- **infoGrabber(JSONObject tmp, String nodo)**: restituisce al codice chiamante una determinata sezione del json object definita dal tag *nodo*.

DatabaseHelper:

1.8 La classe **DatabaseHelper** viene utilizzata per la comunicazione con il server RMI, implementa i metodi dell'interfaccia [intOperators](#) (che verranno discussi nella sezione riguardante l'interfaccia).

La classe presenta inoltre due costruttori dedicati alla connessione al database:

- **DatabaseHelper()** : Il quale utilizza parametri default impostati nella classe per instaurare la connessione con il server RMI, rispettivamente **1099** per la porta e **"localhost"** per l'indirizzo ip.
- **DatabaseHelper(int port, String address)** : Il quale permette di scegliere una porta e un indirizzo diversi da quelli default. (Attualmente questo metodo non viene utilizzato dal programma).

Classi enumerative

Vengono presentate ora le classi enumerative usate nel progetto.

- **Evento:**

Classe enumerativa utilizzata nella gestione degli eventi avversi, in particolare possiamo trovare al suo interno i seguenti tag:

- 3 Mal di testa
- 4 febbre,
- 5 dolori_muscolari_e_articolari
- 6 infodonopatia
- 7 tachicardia
- 8 crisi_ipertensiva
- 9 altro

- **Qualificatore:**

Classe enumerativa utilizzata per indicare i diversi tipi di qualificatori di un indirizzo. Possiamo trovare all'interno della classe i seguenti tag:

- 1 Via
- 2 Viale
- 3 Piazza
- 4 Corso.

- **Severita:**

Enum utilizzata per la gestione della severità. Al suo interno troviamo i seguenti tag:

- 1 Molto_bassa_1
- 2 bassa_2
- 3 fastidiosa_3
- 4 sopportabile_4
- 5 insopportabile_5

- **Tipologia:**

Enum utilizzato per la definizione della tipologia di un centro vaccinale. Al suo interno troviamo i seguenti tag:

- 1 Aziendale
- 2 Ospedaliero
- 3 Hub.

- **Vaccini:**

Enum utilizzato per la definizione dei diversi vaccini disponibili in commercio. Al suo interno abbiamo i seguenti tag:

- 1 Pfizer
- 2 AstraZeneca
- 3 Moderna
- 4 JeJ

Interfacce

Viene presentata ora l'interfaccia utilizzata da DatabaseHelper:

- **intOperators:**

Questa interfaccia è responsabile per l'implementazione dei metodi utilizzati dalla classe DatabaseHelper. Questa interfaccia estende [Remote](#) e i metodi che contiene lanciano una RemoteException se qualcosa va storto.

Possiede quattro metodi:

- **registraCentroVaccinale**(CentroVaccinale nuovoCentro) throws RemoteException : Questo metodo viene usato dal client per richiedere al server RMI la registrazione di un nuovo centro vaccinale.
- **registraVaccinato**(UtenteVaccinato utenteVaccinato) throws RemoteException : Questo metodo permette all'utente di registrare un nuovo vaccinato su richiesta.
- **List<CentroVaccinale> pullCentriVaccinali()** throws RemoteException : Viene utilizzato dalla MainWindow allo startup del programma per recuperare la lista di tutti i centri registrati.
- **List<UtenteVaccinato> pullUtentiVaccinati()** throws RemoteException : Viene utilizzato dalla MainWindow allo startup del programma per recuperare la lista di tutti i vaccinati. Questo metodo viene utilizzato dallo storico.

UML DIAGRAMS

Di seguito vengono mostrati i diagrammi UML del progetto al quale questo manuale si riferisce. Nello specifico verranno mostrati: **UML state diagram**, **UML class diagram** e **UML use case diagram**

Class Diagram

A seguito [Figura 2] viene mostrato il class diagram del progetto in una vista compresa di package in modo da rendere il tutto più leggibile. Nel class diagram non è compresa la classe controller, il package css che comprende gli stili di alcuni item grafici e tutte le relative classi XML di jfoenix.

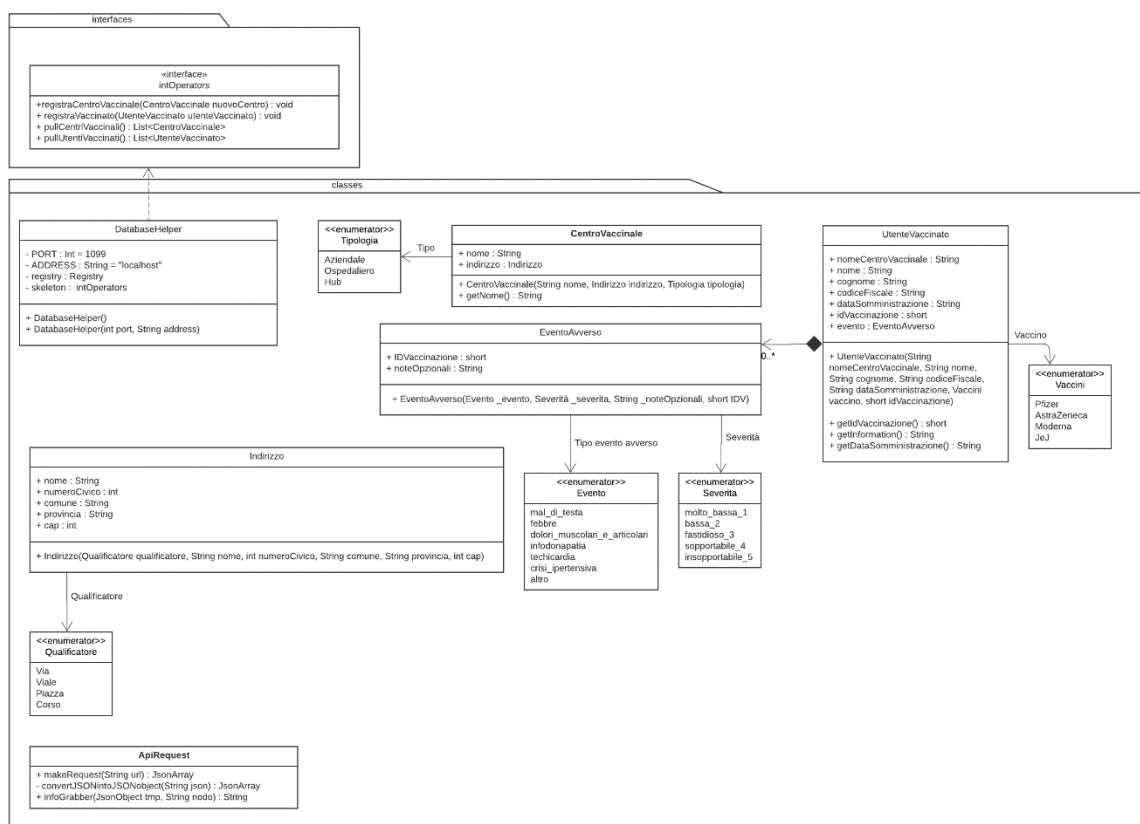


FIGURA 4

State Diagram

A seguito [Figura 3] viene mostrato il digramma di stati che rappresenta le possibili azioni dallo start dell'applicativo alla chiusura.

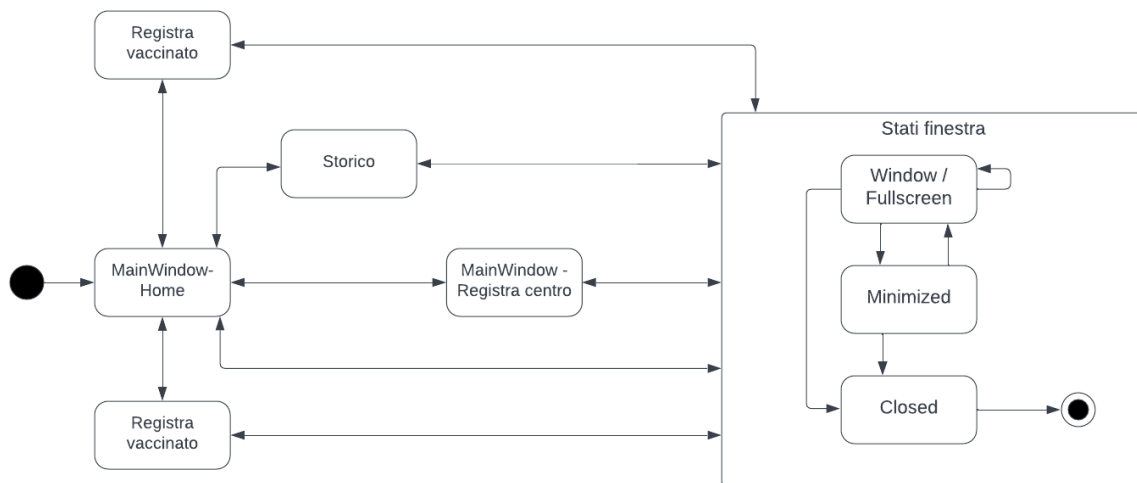


FIGURA 5

Use-case Diagram

A seguito [Figura 4] viene mostrato uno scenario di use-case per l'operatore vaccinale.

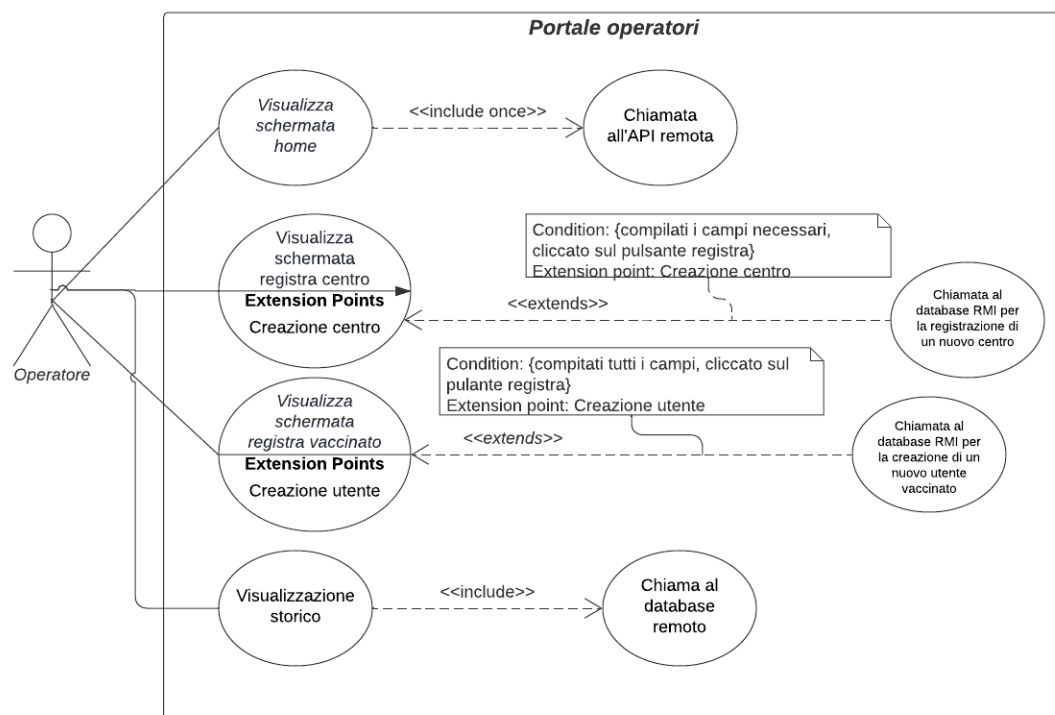


FIGURA 6