**Progetto Centri Vaccinali**

Laboratorio Interdisciplinare B

# Portale Cittadini

## Manuale Tecnico


Preparato da

**Cristian De Nicola** **Mat. 744954**

**Francesco Cavallini** **Mat. 746933**

# Sommario

# INTRODUZIONE

1. **Portale cittadini** è un progetto iniziato nell'ambito del progetto di Laboratorio A e continuato a sviluppare per il progetto di Laboratorio B, 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.

## 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].

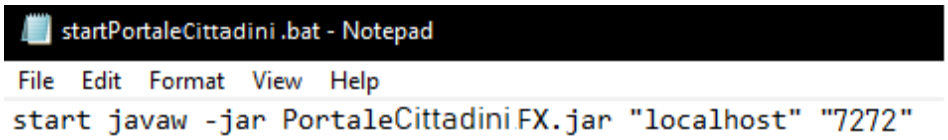


**FIGURA 1**

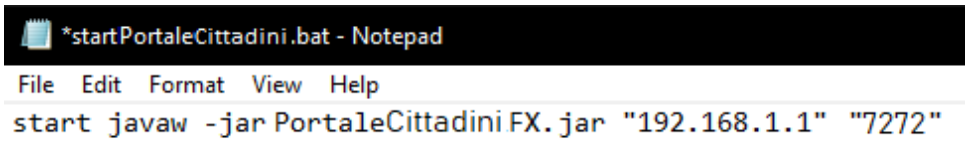


**FIGURA 2**

*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
  - DatabaseHelper
  - EventoAvverso
  - EventoAvversoTMP
  - Indirizzo
  - LoginBox
  - UtenteVaccinato
  - UtenteCredenziali
- Classi Deprecate:
  - ~~FilePaths~~
  - ~~JsonReadWrite~~
- Classi enumerative:
  - Evento
  - FilePaths
  - Qualificatore
  - Severita
  - Tipologia
  - Vaccini
- Controllers:
  - CentroVaccinaleRG
  - EventoAvverso
  - Home
  - Login
  - Registrazione
- Rispettivi FXML:
  - CentroVaccinaleRG
  - EventoAvversoForm
  - Home
  - Login
  - Registrazione
- Interfaccia:
  - CittadiniMetodi

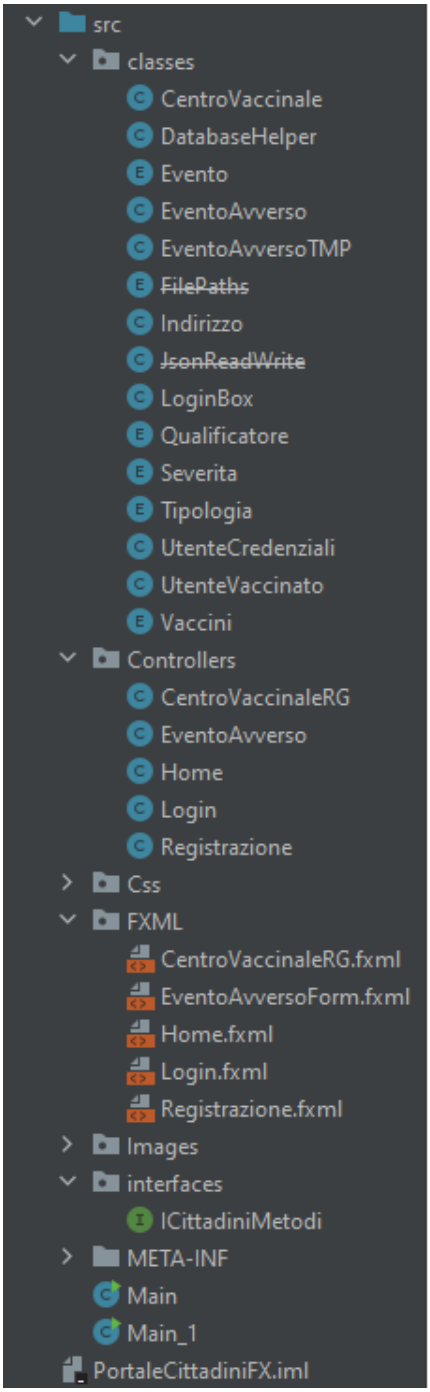


FIGURA 1

# CORE CLASSES

1.1 Verranno presentate ora le core classes nel dettaglio.

(**N.B** -> la maggior parte della 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.

**DatabaseHelper:**

1.3 La classe **DatabaseHelper** viene utilizzata per la comunicazione con il server RMI. Implementa i metodi dell'interfaccia CittadiniMetodi. La classe presenta al suo interno due tipi di costruttore:

- DatabaseHelper(): utilizza parametri default (messi come variabili globali) impostati nella classe; verranno utilizzati per creare la connessione con il server RMI, rispettivamente **8080** per la porta e "**localhost**" per l'indirizzo.
- DatabaseHelper(int port, String address): costruttore che permette di scegliere una porta e un indirizzo diversi da quelli dati di default.

**EventoAvverso:**

1.4 La classe **Evento Avverso** identifica e classifica gli eventi avversi che sono capitati agli utenti a cui è stato sottoministrato 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.

**EventoAvversoTMP:**

1.5 La classe **Evento Avverso TMP** è una classe clone di evento avverso che non poteva essere figlia in quando era necessario che estendesse la classe RecursiveTreeObject<EventoAvversoTMP>. Lo scopo di questa classe è quella di immagazzinare i dati allo stesso modo di Evento Avverso ma in più essere anche "Adapter" per la listView contenente gli eventi avversi presente nel controller "CentroVaccinaleRG"
Presenta al suo interno gli stessi quattro tag contenuti in EventoAvverso, più altri 4 quali verranno adesso elencati:

5 **String nomeCognome**: nome e cognome del committente dell'evento avverso.
6 **StringProperty evento2**: ha la stessa funzionalità di evento con la differenza che il tipo di dato (StringProperty) è leggibile dinamicamente da una ListView.
7 **StringProperty severita2**: ha la stessa funzionalità di severita con la differenza che il tipo di dato (StringProperty) è leggibile dinamicamente da una ListView.
**8** **StringProperty nomeCognome2**: ha la stessa funzionalità di nomeCognome con la differenza che il tipo di dato (StringProperty) è leggibile dinamicamente da una ListView.

**NB:** Tutti i campi StringProperty vengono inizializzati automaticamente dal costruttore creando un nuovo oggetto EventoAvversoTMP

**Indirizzo:**

1.6 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.

**LoginBox:**

1.5 La classe statica **LoginBox** è adibita alla gestione della sessione. E per farlo memorizza le informazioni nei seguenti attributi statici:

1. static String **nome**: nome dell'utente che ha appena fatto il login.
2. static String **cognome**: cognome dell'utente che ha appena fatto il login.

3. static String **nomeCecntroVaccinale**: nome del centro vaccinale sul quale si è appena fatto il login.
4. static String **codiceFiscale**: CF dell'utente che ha appena fatto il login.
5. static List<UtenteVaccinato> **listaVaccinazioni**: lista contenente una serie di oggetti "UtenteVaccinato" che avranno i dati anagrafici appena elencati identici. Ciò che cambia all'interno di questi oggetti sono i dati relativi alla vaccinazione e di conseguenza quelli del relativo evento avverso.
6. static BooleanProperty **isLogin**: variabile booleana contenente true se l'utetne è loggato, false se non è loggato. Su questa variabile è stato fatto un listener che permette di modificare i dati nelle activity in qualsiasi momento viene eseguito il login.

Questa classe gestisce il login e logout tramite due metodi statici, rispettivamente login() e logout():

1 **login (String email, String psw, String nomeCentro)**: vengono passati come parametri:
   - email: la mail con cui l'utente si è registrato al centro vaccinale;
   - psw: la password con cui l'utente si è registrato al centro vaccinale;
   - nomeCentro: il nome del centro a cui l'utente si è registrato.

   Questo metodo richiama direttamente il metodo login presente in DatabaseHelper e se il login va a buon fine nel DB allora setta gli attributi statici della classe LoginBox di modo che tutti i controller ne possano fare uso.

2 **Logout ()**: metodo usato per effettuare il logout dal sistema.
   Metodo che non interpella il database. Semplicemente effettua una cancellazione dei dati precedentemente salvati in fase di Login.

**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.

**UtenteCredenziali:**

1.7 La classe **Utente Credenziali** serve a salvare le informazioni degli utenti che effettuano la registrazione. Al suo interno presenta 4 tag:

1. **UserID:**
2. **IDvaccinazione:** id usato per legare l'utente registrato con i resto del database.
3. **IndirizzoEmail:** indirizzo mail dell'utente usato in fase di login per accedere al sistema.
4. **Password:** password scelta dall'utente per accedere al sistema.

## 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 enumarativa 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:**
  Per la definizione dei vaccini disponibili in commercio. Presenta i seguenti tag:
  1 Pfizer
  2 AstraZeneca
  3 Moderna
  4 JeJ

## Interfacce:

Viene presentata ora l'interfaccia utilizzata da DatabaseHelper:

- **ICittadiniMetodi**:

  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 sei metodi:

  - **String AggiungiEventoAvverso**(Evento e, Severita s, String note, String CF, String data) throws RemoteException, SQLException:
    Metodo utilizzato dal client Cittadini per aggiungere il loro evento avverso all'interno del DataBase.
    NB: per ogni vaccinazione sarà possibile inserire un solo evento avverso
  - **List<UtenteVaccinato> Login**(String email, String psw, String nomeCentro) throws RemoteException, SQLException:
    Metodo utilizzato per permettere ad un utente di effettuare un login presso un centro vaccinale, restituisce una lista di utenti vaccinali perché ogni istanza di UtenteVaccinato conterrà gli stessi dati anagrafici dell'utente ma dati relativi a vaccinazione ed evento avverso diversi
  - **String Registrazione**(String nome, String cognome, String nomeCentro, String CF, String mail, String psw) throws RemoteException, SQLException:
    Metodo che permette la registrazione di un utente ad uno specifico centro vaccinale.
    NB: un utente può essere in grado di iscriversi ad uno o più centri vaccinali solo se è già stato vaccinato in uno di questi.
  - **List<CentroVaccinale> ScaricaCentri()** throws RemoteException, SQLException:
    Metodo utilizzato dalla Home dell'applicazione per scaricare una lista di centri vaccinali. Questi vengono poi mostrati nella list view nella schermata principale.
  - **List<UtenteVaccinato> ScaricaVaccinati**(String nomeCecntroVaccinale) throws RemoteException, SQLException:
    Metodo utilizzato dalla finestra "CentroVaccinaleRG" per settare i dati presenti all'interno dei grafici e inoltre per mostrare i dati relativi agli eventi avversi nella ListView principale.
  - **List<String> ScaricaVaccinazioni(String cf)** throws RemoteException, SQLException:
    Scarica tutte le vaccinazioni relative ad un utente. Necessaria per caricare le vaccinazioni nel DropDown presente nella scheda "EventoAvverso". Queste informazioni vengono poi paragonate con quelle inserite in LoginBox al mometo del login per vericare la possibilità di poter aggiungere o meno un evento avverso nel Database.