CENTRI VACCINALI

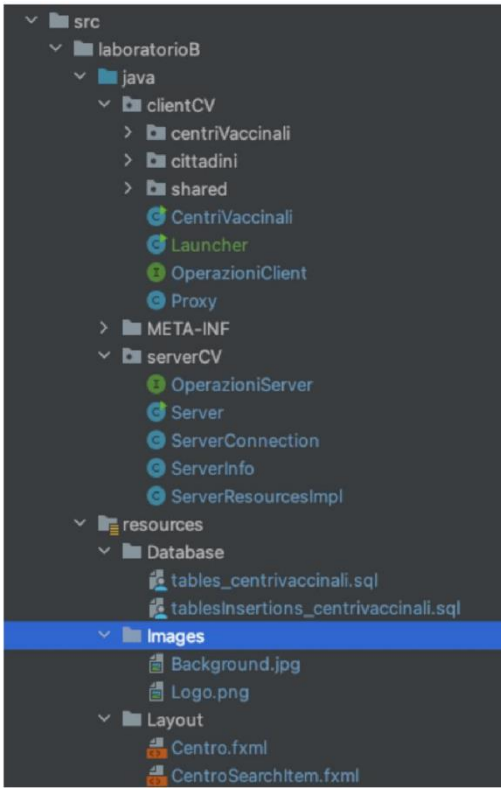
MANUALE TECNICO

Fernando Marcelo Edelstein Fernandez

Eliana Monteleone

INDICE

Struttura Applicazione

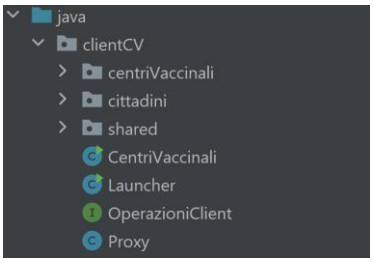


L'applicazione è suddivisa in due packages principali: clientCV e serverCV.

Nel primo package è presente tutta la parte lato Client dell'applicazione, mentre il secondo contiene quella lato Server.

Inoltre vi è la cartella resources dedicata al contenimento dei layout (JavaFX – XML) delle varie interfacce grafiche, gli stili, le immagini e le queries implementate per porter popolare il database al primo avvio dell'applicazione con i dati di default.

Package clientCV



Dentro questo package troviamo le diversi classi che compongono il lato client del sistema. Ci sono diversi packages che lo compongono: Ad esempio all'interno del package CentriVaccinali troviamo gli adapters, che controllano i file fxml per l'interfaccia basata su JavaFX ed i models che contengono gli oggetti necessari per la creazione delle entità utilizzate dal programma.

Il package Cittadini contiene, a sua volta, gli elementi base per la modellazione degli oggetti relativi agli utenti registrati.

Il package Shared contiene la classe Check e ServerInfo: la prima è dedicata all'implementazione dei metodi di Utility utilizzati in tutta l'applicazione, mentre la seconda contiene le informazioni per la connessione al Server.

In seguito troviamo le classi: CentriVaccinali, Launcher, OperazioniClient e Proxy.

- CentriVaccinali
  Questa classe contiene il 'main' dell'applicazione.

- OperazioniClient
  Questa interfaccia descrive tutte le operazioni che vengono implementate dal Proxy
  - close
  - login
  - filtra
  - registraNuovoCentro
  - inserireInDb
  - riceviVaccinati
  - riceviSintomi
  - riceviSegnalazione
  - riceviValoriIndividuali

- Proxy
  Questa classe implementa l'interfaccia OperazioniClient ed è costituito da un costruttore che istanzia un Socket per poter trasferire e ricevere i dati al Server, utilizzando un BufferedReader per la lettura ed un PrintWriter per la scrittura.

Close: Chiude il collegamento con il proxy.

Login: Metodo per poter eseguire il login nell'applicazione. Questo metodo riceve come parametri una Query ed un Utente, entrambi di tipo String, per poter andare a ricercare l'esistenza oppure no di un determinato User e ritornare un oggetto Utente o Cittadino.

Filtra: Metodo per ricercare un Centro Vaccinale in base alla tipologia.

registraNuovoCentro: Metodo per registrare un nuovo centro vaccinale. Riceve in input un parametro di tipo String, che fa riferimento al Nome del Centro, ed esegue un'operazione di CREATE all'interno del Database, creando una tabella di nome "vaccinati_NOMECENTRO"

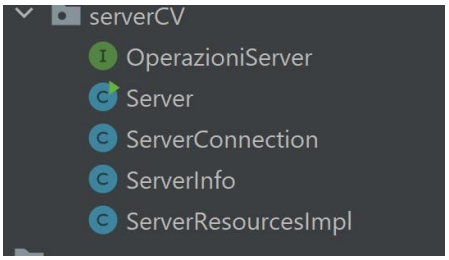
inserireInDb: Inserisce i dati passati come argomento sul Database.

riceviVaccinati: Metodo che riceve in input un parametro di tipo String, corrispondere alla query da eseguire, per poter ottenere la lista di vaccinati. Infatti questo metodo ritorna un ArrayList<Vaccinati>.

riceviSintomi: Metodo che riceve in input un parametro di tipo String, corrispondere alla query da eseguire, per poter ottenere la lista di sintomi. Infatti questo metodo ritorna un ArrayList<Sintomi>.

riceviSegnalazione: Metodo che riceve in input un parametro di tipo String, corrispondere alla query da eseguire, per poter ottenere la lista di segnalazioni. Infatti questo metodo ritorna un ArrayList<Segnalazioni>.

Package serverCV



Il package 'serverCV' è costituito dalle classi e dalle interfacce dedicate al funzionamento dell'applicazione lato server.

Queste sono (come illustrato nell'immagine):

- OperazioniServer:
  Questa interfaccia descrive tutte le operazioni da comandare al DB per l'invio e la ricezione delle informazioni

- Server:
  La classe Server contiene il metodo main che serve ad avviare il Server. Crea un semaforo inizializzato a 100 (per permettere al massimo 100 utenti allo stesso tempo, limite prestabilito da Postgresql) in questo modo, nel caso ci siano più di 100 utenti che 2 provano ad accedere al sistema, questi vanno messi in attesa fino a che non si libera una risorsa nel semaforo. Dopo di che saremmo richiesti delle credenziali di acceso al Db, queste sono segnate sulla classe ServerInfo. Una volta inserite, il programma esegue il metodo tryConnection, il quale prova a connettersi al Db di nome "centrivaccinali" tramite l'IP Server "localhost" e la porta 5432 con le credenziali inserite, nel caso queste siano corrette, viene accettata la connessione ed il Server si avvia, caso contrario, ci saranno richiesti ancora le credenziali di accesso.

- ServerConnection:
  Questa classe è il Thread che si connette al Db e riceve le richieste del programma. Al suo interno abbiamo il metodo fetchFromDb, il quale si interfaccia con la classe "ServerResourcesImpl" (verrà spiegata in seguito). FetchFromDb contiene un riferimento a questa classe alla quale li si sono inviate le possibili richieste da parte del Client, queste sono:
  - login
  - inserireInDb

- filtra
- riceviVaccinati
- riceviValoriIndividuali
- registraNuovoCentro
- riceviSintomi
- riceviSegnalazione

- ServerInfo:
  Questa classe contiene le informazioni relative alla connessione al Db, questi sono il username, password, numero di porta e nome del Db. Questi dati vengono utilizzati dalla classe "Server" all'ora di inizializzare il Server. Contiene anche al suo interno, le informazioni necessarie per la connessione fra il Client e Server, questi sono il IPServer e Port

- ServerResourcesImpl
  Questa classe implementa l'interfaccia OperazioniServer, su cui sono elencate tutte le possibili richieste al Server. Contiene al suo interno un costruttore che prende come argomento un BufferedReader, un PrintWriter e un Connection.

  L'implementazione dei metodi fa uso del PrintWriter per restituire i risultati nel formato adatto alla richiesta.

  Login: Contiene al suo interno due stringe, "query"(la quale viene letta dal BufferedReader) e "userId" le quali formano la query necessaria per richiedere il possibile utente al Db. Contiene anche uno Statement e un ResultSet, utilizzato per eseguire la query. 3 Il resultSet prende questa query e la esegue, se il risultato non è nullo, si verifica se l'utente è un cittadino o un operatore. Per fare ciò, si esegue una seconda query: "query1" la quale cerca se l'utente ricercato è presente nella tabella "utenti", in tal caso, l'utente ricercato è un operatore. In seguito, si restituisce il nome, cognome, cf, userid e password del utente. Se l'utente è un cittadino, si restituiscono anche l'email e l'id vaccinazione.

  Close: Chiude la connessione al server tramite il metodo socket.close();

  riceviSintomi: Utiliza una string "query" che viene letta dal BufferedReader. Il ResultSet esegue la richiesta e restituisce il "sintomo", "idsintomo" e "descrizione" e per ultimo la string "exit" la quale indica la fine del risultato restituito dalla richiesta.

inserireInDb: Contine al suo interno la query letta dal BufferedReader che viene eseguita dal Statement.

registraNuovoCentro: Metodo per registrare un nuovo centro vaccinale, prende dal BufferedReader il nome del centro da creare e la query per creare la sua rispettiva tabella. Prima di tutto, si verifica che la tabella da creare non esista già. Per fare ciò, si usa il riferimento all'oggetto DatabaseMetaData, posteriormente, utilizzando un ResultSet, si verifica l'esistenza di una tabella col nome del centro richiesto. Se il risultato del ResultSet è nullo, allora sappiamo che il centro non è stato ancora registrato, per cui, utilizzando lo Statement.executeUpdate, si esegue la query di creazione della tabella.

riceviValoriIndividuali: Questo metodo è utilizzato per richiedere un valore unico dal Db. Utilizzando la string "query" e "columnLabel" il ResultSet esegue la query e i risultati sono filtrati da un metodo resultSet.getString(columnLabel) per individuare il valore desiderato.

riceviVaccinati: Questo metodo serve per ricevere una lista dei vaccinati registrati. Utilizzando la query, il resultSet ci restituisce il nome del cittadino, cognome, cf, vaccino ed id vaccinazione.

filtra: Questo metodo è utile per ricavare indirizzi. Questo metodo esegue una Query mediante il resultSet e restituisce il nome del centro, tipologia, qualificatore, strada, numero civico, comune, provincia e cap.

riceviSegnalazione: Questo metodo serve per ricevere una segnalazione effettuata da un cittadino. Prende una query dal BufferedReader e restituisce il centro vaccinale di competenza, userid del utente che ha creato la segnalazione, sintomo, severità e descrizione.

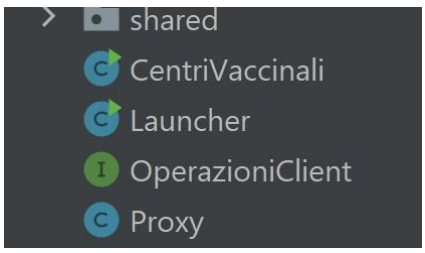
Scelte progettuali

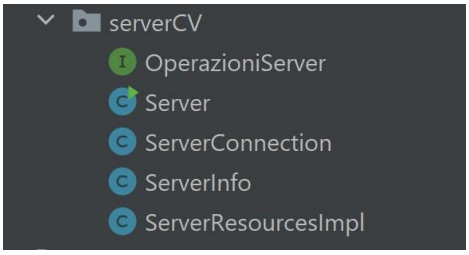
Design Pattern utilizzati

L'interazione tra Client e Server viene favorita dall'implementazione del Design pattern **Proxy**, permettendo al Client di poter richiedere al Server una determinata informazione presente nel Database tramite un'istanza del proxy contenente tutte le funzionalità necessarie.

Per il lato Server, invece, ogni volta che viene accettata un connessione viene istanziato un nuovo ServerConnection per gestire le richieste del Client, in tal modo, si evita di la comunicazione diretta tra Server e Client, ma vengono sfruttati gli intermediari.

Inoltre le funzionalità presenti lato Client vengono gestite tramite il Design pattern **Model View Controller (MVC)**.






```
public class Proxy implements OperazioniClient{

    private final Socket socket;
    private boolean operatore = false;

    private BufferedReader in = null;
    private PrintWriter out = null;

    /**
     * Proxy Constructor
     *
     * @throws IOException
     */

    public Proxy() throws IOException {

        socket = new Socket(ServerInfo.getIPSERVER(), ServerInfo.getPORT());

        try {
            in = new BufferedReader(
```

```
public class ServerConnection extends Thread{

    private Semaphore sem;
    private Socket socket;
    private BufferedReader in = null;
    private PrintWriter out = null;
    private String username, password;

    /**
     * ServerConnection Constructor
     *
     * @param socket
     * @param sem
     * @param username
     * @param password
     */

    public ServerConnection(Socket socket, Semaphore sem, String username, String password) {
        this.username = username;
        this.password = password;
        this.socket=socket;
        this.sem=sem;

        start();
```

Database

Scelte implementative

Il database è composto dalle seguenti tabelle:

- sintomi
- segnalazioni
- centrivaccinali
- cittadinivaccinati
- idunivoci
- utenti
- vaccinati_*nomeCentro*

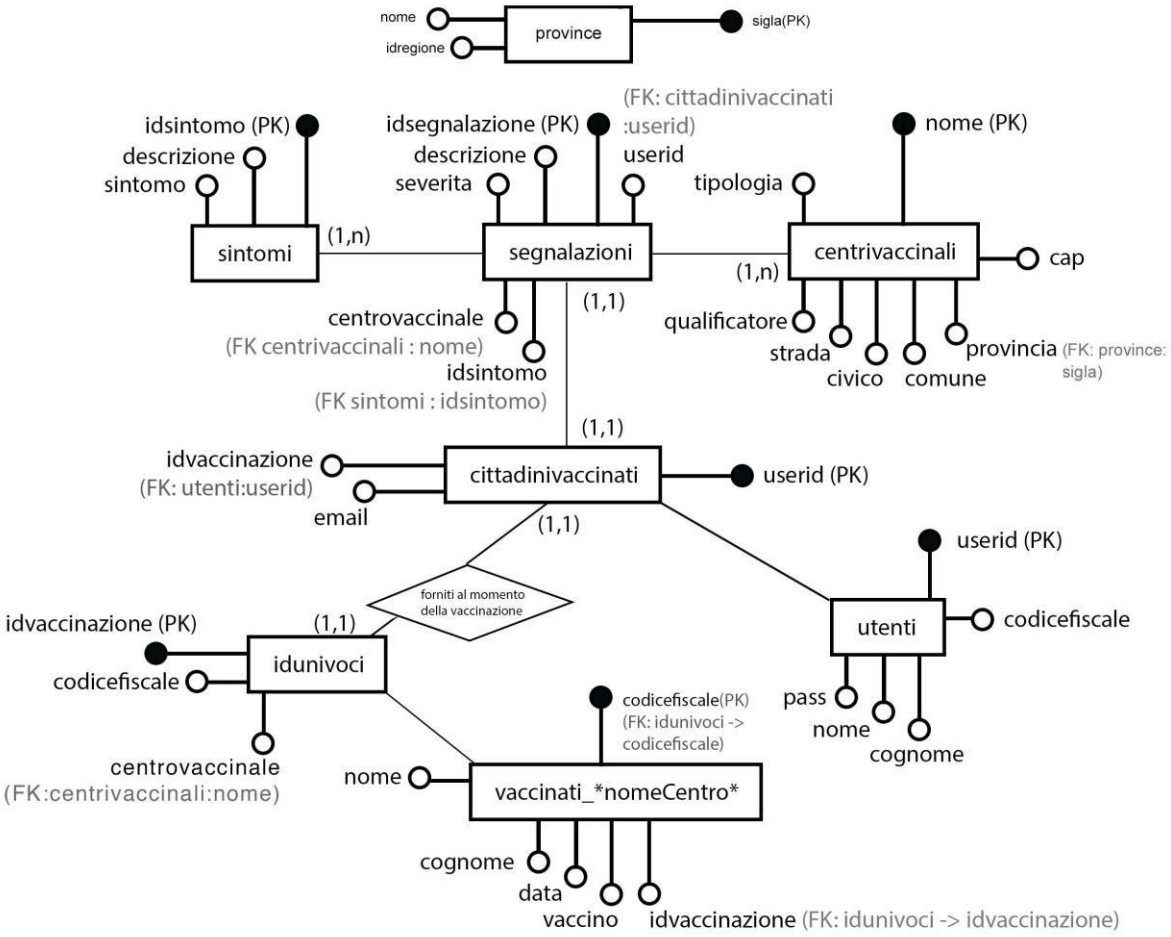


Tabella SINTOMI

Questa tabella è stata creata per poter tenere traccia di tutti i vari sintomi, presenti in un evento avverso al vaccino e qualora ci fosse il bisogno di aggiungerne altri, questi verranno inseriti nel database. Ciò dovrebbe favorire la riduzione degli errori ortografici in quanto molti dei sintomi comuni sono già presenti nella tabella e l'utente non deve far altro che scegliere.

Tabella SEGNALAZIONI

Questa tabella contiene tutte le segnalazioni dei diversi centri vaccinali. Al suo interno troviamo le colonne "centrovaccinale" Foreign Key della tabella centrivaccinali, "idsintomo" Foreign Key della tabella sintomi, "userid" Foreign Key della tabella cittadinivaccinati,con l'aggiunta dei campi 'descrizione' (di massimo 256 caratteri) e 'severita' (da 1 a 5).

Tabella CENTRIVACCINALI

Su questa tabella vengono memorizzati tutti i centri vaccinali registrati, aventi come Primary Key il nome, in modo da evitare i duplicati, dato che non ci potranno essere centri vaccinali con lo stesso nome.

Tabella IDUNIVOCI

La tabella 'idunivoci' viene utilizzata sia in fase di registrazione del vaccinato che in fase di registrazione del cittadino. Serve per rilasciare un codice identificativo che sia univoco tra tutte le tabelle 'vaccinati_'. Quando viene registrato un vaccinato, viene inserita nella tabella 'idunivoci' una coppia Idvacc (ossia l'identificativo univoco di vaccinazione generato randomicamente dall'applicazione) e il codice fiscale (unique) della persona che è stata vaccinata. Dopo ciò, l'identificativo (idvacc) viene utilizzato come chiave esterna nella tabella 'vaccinati_'. In fase di registrazione del cittadino, vengono richiesti l'identificativo (idvacc) e il codice fiscale, che vengono poi confrontati con quelli presenti nella tabella 'idunivoci'.

Tabella UTENTI e CITTADINIVACCINATI

I cittadinivaccinati sono degli utenti, che condividono gli stessi attributi, con l'aggiunta di 'idvaccinazione' e 'l'email'.
Gli utenti che non sono cittadinivaccinati equivalgono agli Operatori.

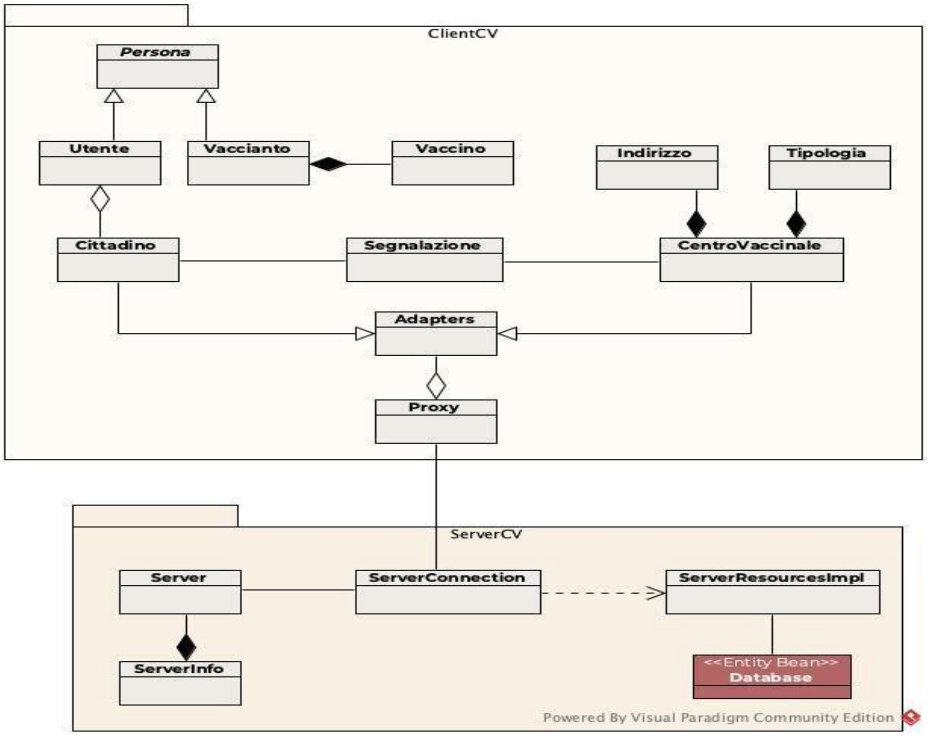
Tabella VACCINATI_*NOMECENTRO*

Queste tabelle vengono generate per ogni centro vaccinale ed al suo interno vengono salvati i dati rispettivi alle vaccinazioni dei cittadini per il relativo centro, in questo modo si evita che un cittadino possa creare una segnalazione per un centro nel quale non è stato vaccinato.

Scelte implementative APP

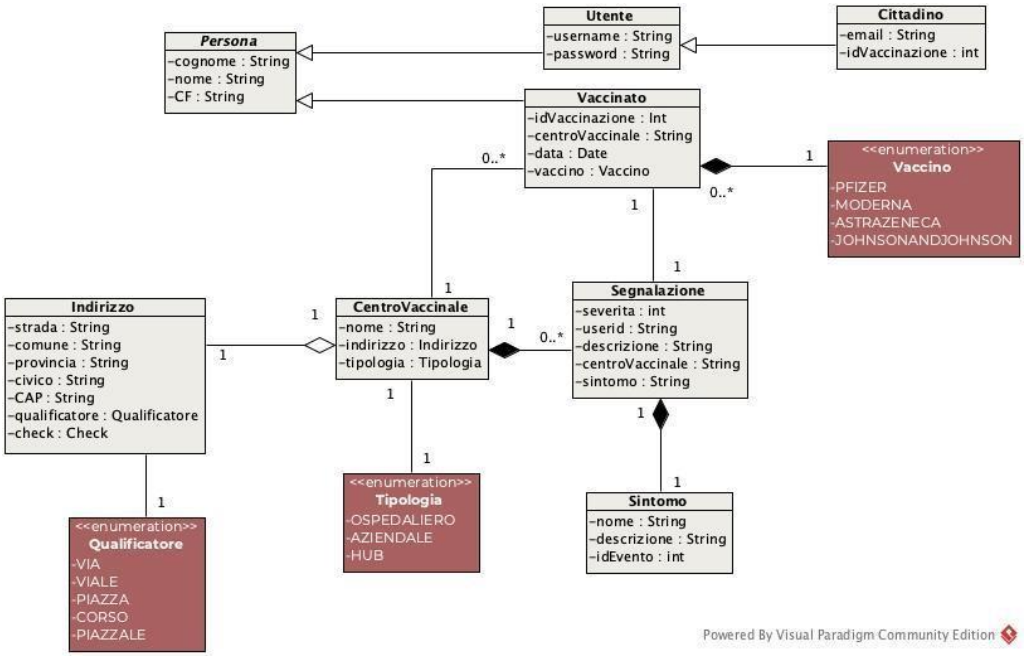
Diagrammi

Struttura del progetto



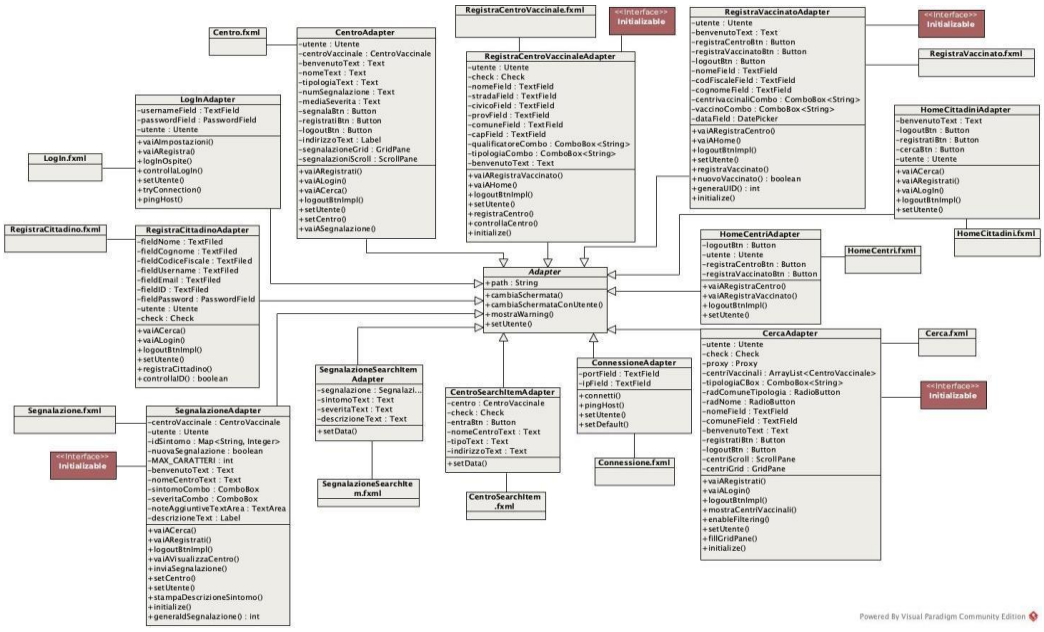
Modelli

I modelli (entità) che vengono salvati e scaricati dalla base di dati riportati all'interno del package clientCV sono organizzate nel seguente modo:



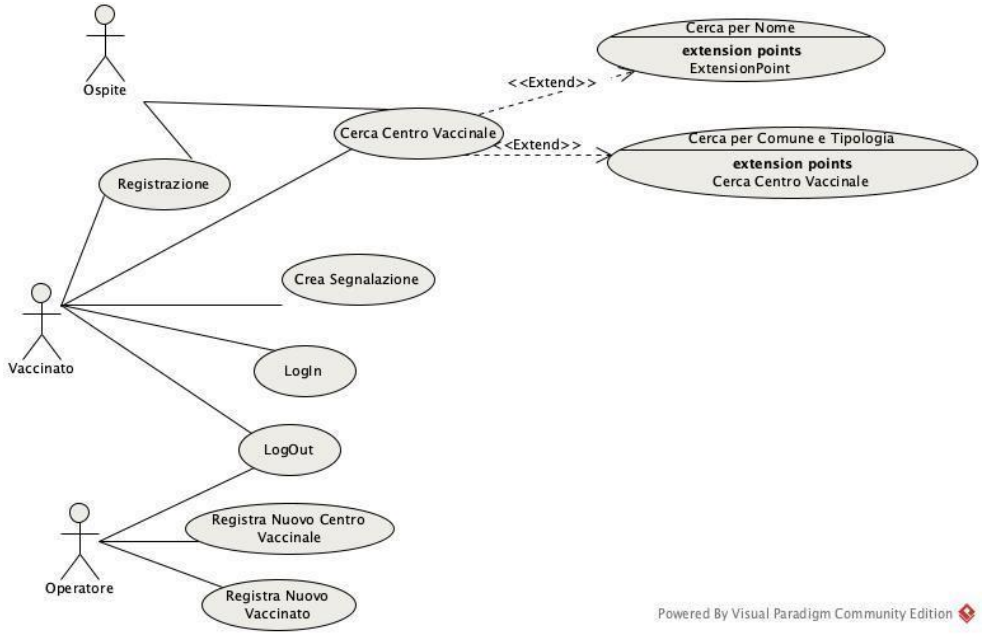
Interfaccia grafica

Per l'interfaccia grafica, è stato utilizzato JavaFx, per cui, ogni schermata è stata sviluppata in un file fxml associato ad ogni classe Adapter. Oltre a tutti gli adapter specifici delle schermate, abbiamo la classe astratta Adapter che contiene dei metodi che servono a tutte le altre classi.



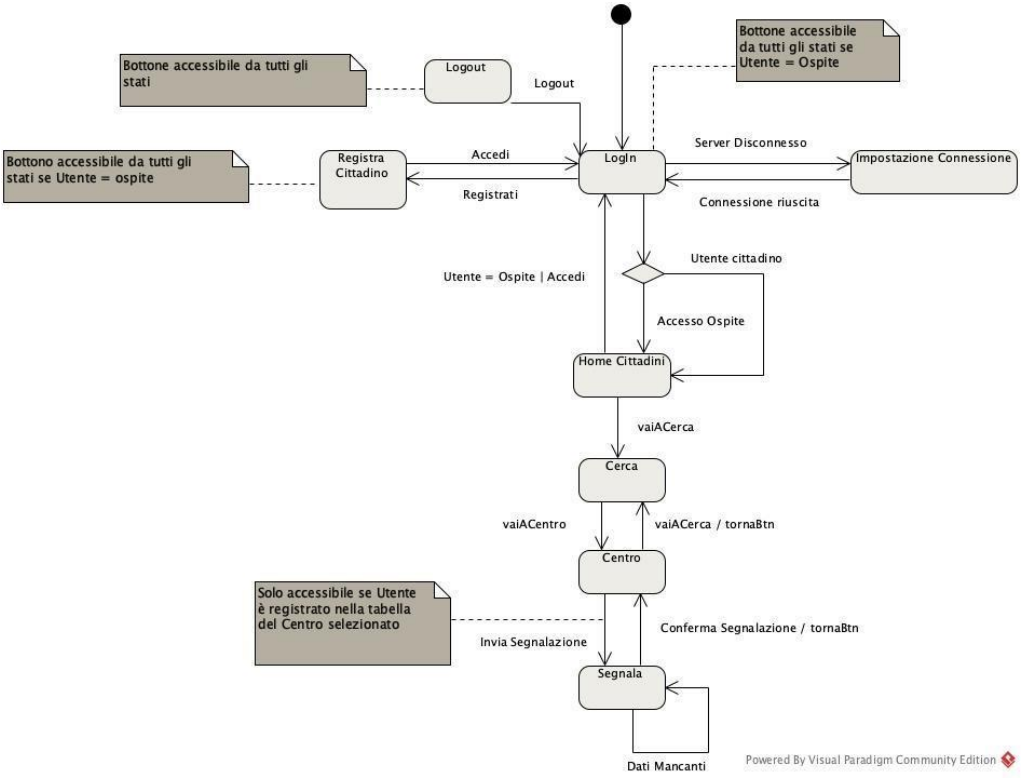
Funzionalità disponibili

Le funzionalità disponibili per i cittadini ed operatori sono:

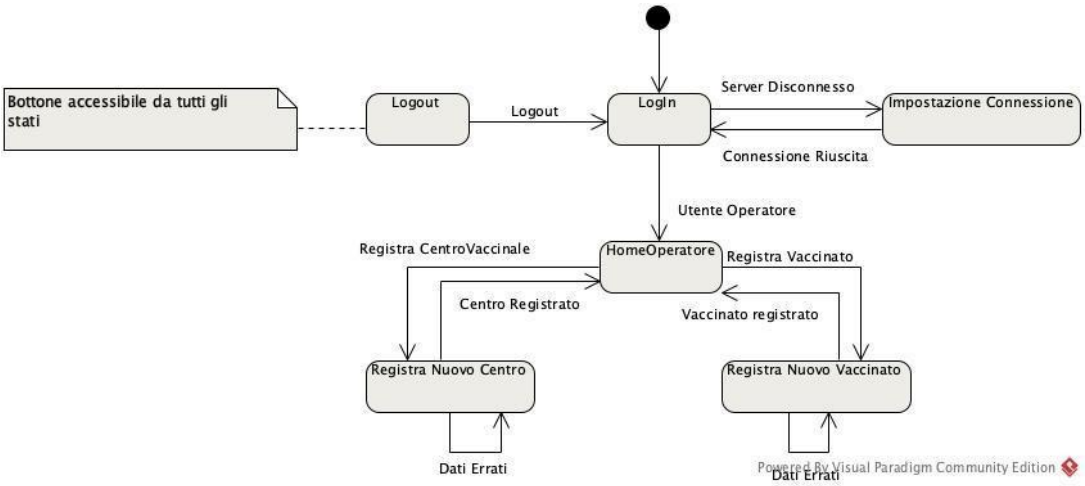


Transizioni o stati

Per il cittadino le situazioni che si possono verificare sono:

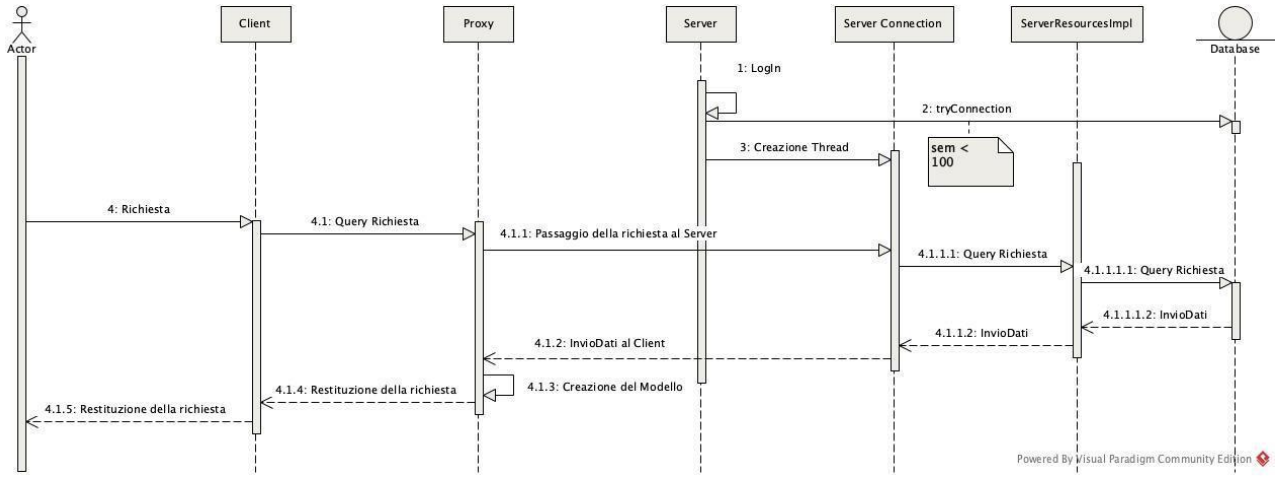


Per l'operatore le situazioni che si possono verificare sono:



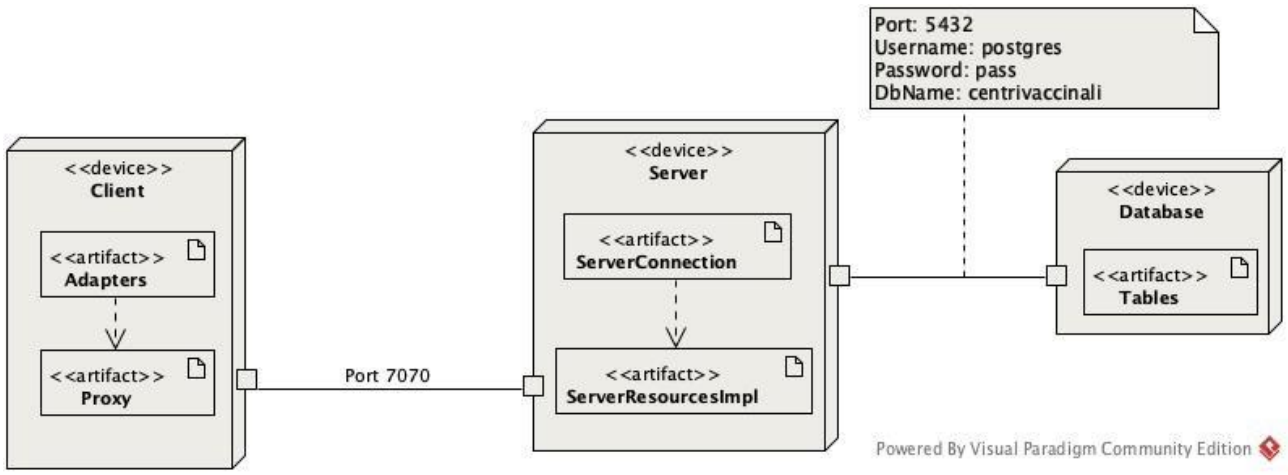
Interazioni Client-Server

Le interazioni Client-Server avvengono nel seguente modo:



Deployment Diagram

Tramite questo diagramma si mostra come le componenti software sono distribuite rispetto alle risorse hardware disponibili sul sistema.



Scelte architetturali

L'applicazione è compatibile con tutti i 3 sistemi operativi : Windows, Mac (architettura x86, può essere eseguito su architettura M1 su intelliJ con la versione beta di JavaFx 17) e Linux. Essa utilizza JavaFx e il driver JDBC.

Scelte algoritmiche

L'applicazione ha come operazione più complessa le richieste al database, che essendo semplici operazioni CRUD, hanno complessità asintotica O(n). Tuttavia la gestione della concorrenza quando il server viene avviato, è stato scelto l'implementazione di un semaforo inizializzato a 100
(Limite di connessioni di Postgresql).Questo viene passato e modificato dallo ServerConnection (classe che estende il Thread) attraverso una acquire iniziale e una release finale.

Strutture dati

Le strutture dati utilizzate per la realizzazione del progetto sono state prevalentemente liste (ArrayList) spesso sfruttate per memorizzare i dati salvati nella base dati, e mappe (Map) per l'interazione tra la parte grafica e di logica del programma.

Gestione file

L'applicazione è stata sviluppata con JavaFX, perciò ogni schermata presenta il file di layout (in XML, reperibili al percorso resources/Layout) e il relativo adapter(controller) (in Java, reperibili al percorso src/laboratorioB/java/clientCV/centriVaccinali/adapters). L'applicazione finale consiste in due files .jar, uno per il server e uno per il client, con i relativi files .bat/.sh per avviare i .jar con tutte le dipendenze associate.