COVID REPORTER

MANUALE TECNICO

Progetto Laboratorio B

Caspani Daniele, Corti Cristian, Marceca Manuel

Sommario

FUNZIONALITA'

Inserimento Centro Vaccinale

L'inserimento di un centro vaccinale viene implementato nella classe `RegistraCentro`, che estende la classe `Registrazioni`, all'interno del package `client.centrivaccinali`. Dopo aver controllato l'effettivo riempimento delle varie caselle di testo e la corretta formattazione di sigla, numero civico, comune e Cap, viene eseguita una semplice query di update per il centro vaccinale impostato secondo i valori delle varie `JTextField` e `JComboBox`.

Inserimento Vaccinato

L'inserimento del vaccinato viene implementato nella classe `RegistraVaccinato` che estende `Registrazioni`, presente all'interno del package `client.centrivaccinali`. Dopo aver trovato il centro corretto per mezzo della funzionalità di ricerca, viene controllato l'effettivo riempimento dei vari campi e la corretta formattazione del codice fiscale. Infine, vengono inviati i dati inseriti al database per mezzo di una query di update.

Registrazione Cittadino

L'inserimento del cittadino viene implementato nella classe *`RegistraCittadino`* che estende `Registrazioni`, presente all'interno del package `client.cittadini`. Dopo aver trovato il centro corretto per mezzo della funzionalità di ricerca, viene controllato l'effettivo riempimento dei vari campi e la corretta formattazione del codice fiscale e della mail. Infine, vengono inviati i dati inseriti al database per mezzo di una query di update.

Login

Operazione che permette l'accesso ad un cittadino registrato utilizzando le credenziali username e password.

Inserimento Evento Avverso

Operazione definita nella classe `RegistraEventiAvversi`, che estende la classe `Registrazioni` appartenente al package `client.cittadini`. Permette l'inserimento di un evento avverso, scelto tra una serie di opzioni prestabilite. Ogni evento avverso non può essere inserito due volte dallo stesso utente. Dopo aver effettuato l'accesso e aver ricercato il centro di vaccinazione viene controllata la corretta formattazione dei campi. Infine viene eseguita una query di update che inserisce l'evento avverso selezionato.

Ricerca

Operazione implementata nelle classi `RicercaNomeCentro` e `RicercaComuneTipologia` che permette di ricercare un dato centro attraverso l'inserimento del comune del centro e la relativa tipologia oppure per mezzo di una stringa che deve essere contenuta nel nome del centro.

Il costo dell'operazione è a carico del DBMS, che effettua la ricerca indipendentemente.

Visualizza Informazioni

Visualizza le informazioni relative ad un dato centro vaccinale e agli eventi avversi segnalati in esso, i quali saranno inseriti in una tabella contenente il numero di segnalazioni per evento, la relativa media, la standard deviazione.

Il costo dell'operazione è a carico del DBMS, che effettua la ricerca indipendentemente ed effettua i calcoli su numero segnalazioni, media e standard deviazione per ogni evento per mezzo dei metodi `COUNT()`, `AVG()` e `STDDEV_POP()`.

È anche calcolata una media totale mostrata in output e dal costo pari a O(n), ovvero il costo necessario a sommare tutti gli indici di ogni evento avverso (n eventi avversi).

CLASSI DI SUPPORTO

Utility

Contenente vari metodi utili alla logica dell'applicazione, spesso dedicati alla gestione dei risultati delle query al database.

Quadrupla

Una struttura dati generica utile alla gestione dei dati relativi alle segnalazioni.

CLASSI ENUMERATIVE

Vaccino

Contenente i valori enumerativi che rispecchiano i vari vaccini somministrati ai cittadini

Qualificatore

Qualificatore di un indirizzo (VIA, PIAZZA O VIALE)

Tipologia

Tipo di centro vaccinale (Hub, Ospedaliero o Aziendale)

EventoAvverso

Tipo di di evento avverso

INTERFACCIA GRAFICA

L'interfaccia grafica è stata implementata attraverso l'utilizzo della classe JFrame estensione della classe Frame.

COMPONENTI UTILIZZATI

- JTextField
- JComboBox
- JButton
- JLabel
- JPanel
- JLayeredPane
- JFrame
- JList
- ImageIcon
- AbsoluteLayout
- GroupLayout

CLASSI IN APPOGGIO ALL' INTERFACCIA GRAFICA

Registrazioni

Permette di utilizzare una serie di dimensioni e oggetti utili e condivisi nelle interfacce grafiche presenti nei package cittadini e centri vaccinali

ServerGraphics

Permette di utilizzare una serie di dimensioni e oggetti utili e condivisi nelle interfacce grafiche presenti nel package server

SwingAwt

Implementa alcuni metodi utili per migliorare l'interfaccia grafica.

PTextField

Estende JTextField e viene utilizzata per migliorare l'interfaccia grafica.

LIBRERIE ESTERNE

Dipendenze / Maven

Questo programma è stato sviluppato con l'aiuto della libreria esterna '*postgresql 42.5.1*', che permette la comunicazione tra la JVM e il DBMS PostgreSQL.

Le dipendenze sono state gestite con Maven, uno strumento che permette l'automazione di alcuni processi nello sviluppo del software, quali gestione delle dipendenze e compilazione del programma.

Un'installazione di Maven è già inclusa nel progetto, come anche lo script 'maven-clean-install.bat' che quando eseguito, utilizzando tale installazione, crea un database denominato 'ProgettoB' (lo cancella e lo ricrea nel caso esso sia già presente), compila il progetto e sposta i file .jar per l'esecuzione del programma nella cartella /jar/.

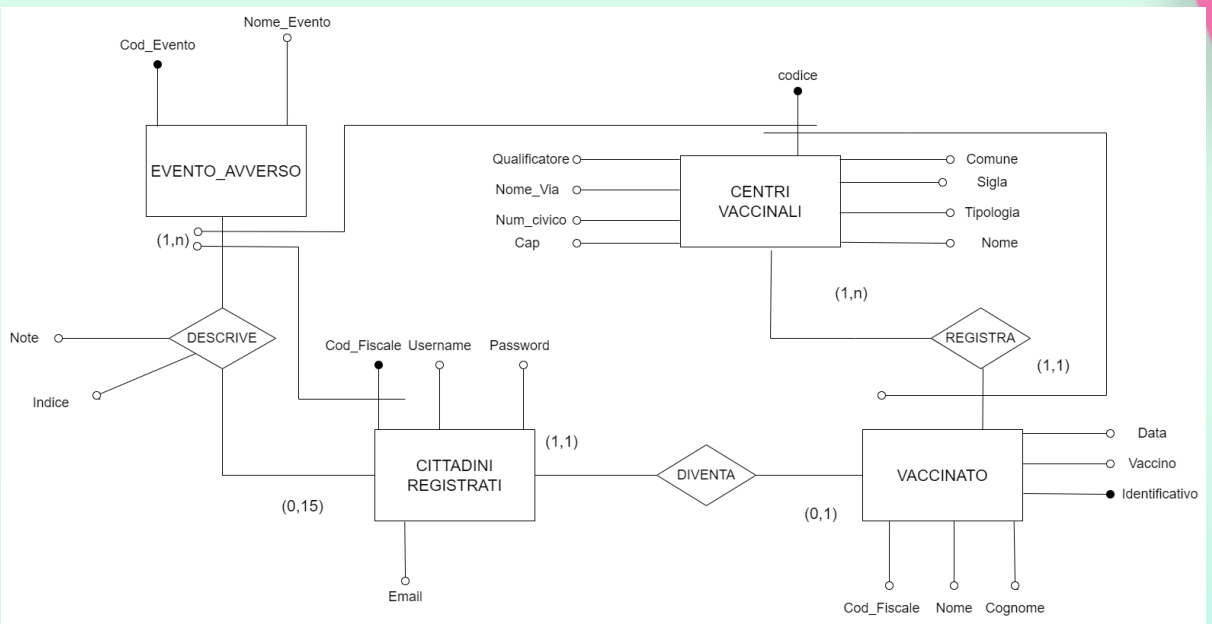
DATABASE

PROGETTAZIONE

PROGETTAZIONE CONCETTUALE

Nella progettazione concettuale vengono implementate quattro relazioni:

Tre di queste diventeranno tabelle future del database. La relazione eventi, invece, non verrà utilizzata poiché non ha degli attributi che possono essere generalizzati, ma cambiano ad ogni istanza di ogni evento.



PROGETTAZIONE LOGICA

Nella progettazione logica vengono effettivamente costruite logicamente le varie tabelle e i loro attributi, chiavi primarie e chiavi esterne.

TABELLE

Nella documentazione e nel codice è possibile trovare l'implementazione del database con estensione .SQL

Centri_Vaccinali

Tabella che contiene le informazioni relative ad un dato centro vaccinale. Contiene le seguenti colonne:

- `Codice`: chiave primaria generata in modo SERIAL
- `Nome`
- `Comune`
- `Sigla`
- `Tipologia`: tipo di centro {Aziendale, Hub, Ospedaliero}
- `Qualificatore`: tipo di indirizzo {Via, Viale, Piazza}
- `Nome_via`
- `Num_civico`
- `Cap`

Vaccinati

Tabella che contiene le informazioni relative ad ogni vaccinato. Contiene le seguenti colonne:

- `Cod_Fiscale`
- `Nome`
- `Cognome`
- `Data`
- `Identificativo`: chiave primaria generata in modo SERIAL
- `Vaccino`
- `Cod_Centro`: chiave esterna

Cittadini_Registrati

Tabella che contiene le informazioni relative ad un dato cittadino registrato. Contiene le seguenti colonne:

- `E-mail`
- `Username`
- `Password`
- `Cod_Fiscale`: chiave primaria

LogEventiAvversi

Tabella che contiene le informazioni relative ad un dato centro vaccinale con i campi

- `Cod_Evento`: chiave primaria generata tramite SERIAL
- `Nome_Evento`
- `Indice`
- `Note`
- `Cod_Centro`: chiave esterna per il centro vaccinale
- `Cod_Fiscale`: chiave esterna per un cittadino registrato

CLASSI CHE SI INTERFACCIANO CON IL DATABASE

ClientToServerRequests

Contenente i metodi che si interfacciano con il client e che utilizzano l'oggetto condiviso sul server.

SelectQuery

Contenente tutte le query utilizzate per il programma

DBManager

Classe del server che implementa i metodi di accesso al database, i quali vengono sovrascritti nello stub (interfaccia di tipo DBInterface) del client.

UML

I diagrammi UML sono disponibili nella cartella UML.

GESTIONE DEI PATH

I path delle immagini sono stati gestiti rendendole delle risorse di compilazione, le quali vengono individuate attraverso il metodo ClassLoader().GetSystemResource().

Per quanto riguarda i file viene utilizzato il termine ./ per rendere ogni percorso relativo e multi Platform.

GESTIONE CLIENT-SERVER

Per separare la gestione del database dall'utilizzo effettivo del programma, abbiamo creato un semplice server attraverso l'utilizzo del package *java.rmi* e delle sue funzionalità. Ciò permette al client di utilizzare i metodi di un oggetto esposto in remoto, il quale viene sovrascritto su un'interfaccia preparata appositamente. L'implementazione dei metodi RMI evita al client di accedere direttamente al database, rendendo l'applicazione più sicura.

TIPO DI SOLUZIONE ADOTTATA

RMI VS SOCKET

RMI si tratta di una tecnica utilizzata in ambiente distribuito che, rispetto alle *socket*, ha il vantaggio di essere facilmente implementabile poiché evita la creazione degli stream di input e output, gestisce automaticamente la multiutenza (in teoria) e permette l'interazione in modo trasparente, attraverso l'utilizzo di oggetti che invocano metodi, metodologia affine alle caratteristiche del nostro programma e in generale della logica di java. Inoltre, anche se in questo programma non è stato sfruttato, è possibile proteggere il server in modo efficace attraverso l'implementazione della classe *RMISecuritymanager*, coadiuvata da un apposito file di permessi. D'altro canto, RMI presenta principalmente due svantaggi:

- `Scarsa flessibilità e portabilità:` in quanto RMI funziona attraverso l'utilizzo di oggetti e metodi chiamati da essi, diventa estremamente difficile utilizzare diversi linguaggi di programmazione all'interno dello stesso programma, caratteristica che invece risulta più favorevole alla tecnologia socket.
- `Minore efficienza:` RMI, soprattutto rispetto ad una socket risulta più lento e soprattutto occupa più risorse in termini di spazio.

Collection utilizzate

Abbiamo deciso di utilizzare la Collection `ArrayList` data la comodità della sua dimensione dinamica e adattabile alla quantità di dati processati, che è molto variabile.

COMPLESSITÀ STIMATE

Le operazioni più onerose sono affibbiate al DBMS, nelle operazioni di ricerca. Attraverso la memorizzazione nei ResultSet la complessità risulta essere O(m) dove m sono il numero di tuple restituite per quanto riguarda le query di select mentre O(1) per quanto riguarda le query di insert.

L'applicazione non contiene operazioni con costo maggiore di O(n), ovvero quando è necessario scorrere liste per inserire o rimuovere dati.

RUOLI E MODUS OPERANDI

Per strutturare e velocizzare il lavoro abbiamo utilizzato GitHub che, attraverso lo scambio di codice e opinioni, permette un approccio più efficiente ed efficace grazie ad una buona gestione del versioning. le varie funzionalità sono state gestite efficientemente da IntelliJ, Ambiente di sviluppo utilizzato per questa seconda parte di progetto.

Per quanto riguarda i ruoli sono riportati nelle righe sottostanti:

- **Manuel Marceca** **Project Manager**
- **Daniele Caspani** **Software Developer**
- **Cristian Corti** **Software Developer**

LIMITI DEL PROGETTO

- Alcune finestre non sono resizable (non possono essere ingrandite) perché altrimenti non sarebbero bilanciate per qualsiasi schermo.
- È possibile che un utente inserisca caratteri che scatenano eccezioni SQL nel server.

Bibliografia

Classe `PTextField` -->
https://stackoverflow.com/questions/11200585/adding-aprompt-text-property-to-jtextfield

Funzione `ControllaCodiceFiscale`, pattern del codice fiscale -->

Controlliamo la correttezza di un Codice Fiscale | MRW.it

https://www.alus.it/pubs/CodiceFiscale/index.php?lang=it

Funzione `mailSyntaxCheck`, pattern email -->

Java RegEx: introduzione alle espressioni regolari | MRW.it