Università degli Studi di Padova

Corso di laurea triennale in Informatica

Corso: Programmazione ad Oggetti

Prof. Francesco Ranzato

Relazione Progetto

LinQedIn

David Tessaro

1047319

Anno accademico 2014/2015

Indice:

1.Introduzione

Il progetto LinQedln consiste nel ricreare un ambiente con le principali funzionalità di Linkedln sviluppato in Qt.

La gestione delle notifiche e news è stato ignorato per questioni pratiche.

Come amministratore è possibile tramite interfaccia, gestire il database e le caratteristiche del singolo utente, caricare un nuovo database oppure farne il backup, aggiungere ed eliminare utenti ed apportare eventuali modifiche al profilo.

Come utente è possibile gestire il proprio account, visualizzare la rete di contatti secondo le metodologie social e visualizzare le loro informazioni.

Il salvataggio è automatico, quindi ogni modifica amministrativa è significativa, in caso di errore è possibile caricare un backup del database.

Per la grafica si è cercato di seguire il pattern MVC. La GUI è stata scritta senza l'ausilio di QtDesigner, in modo da poter approfondire il più possibile le dinamiche di funzionamento delle librerie grafiche di Qt-Widget.

In seguito si farà una veloce presentazione delle principali componenti del progetto.

2.File

Il progetto utilizza tre file xml, rispettivamente per Database(Database.xml), login admin(PwAdmin.xml) e login utente (PwUser.xml). Componenti grafici quali icone e sfondi sono contenuti rispettivamente nel percorso "GUI/Icon" e "GUI/Img" mentre le immagini profilo standard sono in "database/ImgProfile". I backup sono in "database/Backup".

Ulteriore divisione è stata data ai file (cpp,h) contenuti in tre cartelle:

Model: database, excerror, info, profilo, user, versioneinterfacciauser

,versioneinterfacciauser

GUI_ControL: controllerUser, controllerAdmin.

GUI_View: editUserProfile, homeAdminView, homeUserView, loginView, userViewSearch.

3.Classi

La scelta del contenitore del database si è portata sul QMap, quest'ultimo infatti offre un notevole vantaggio grazie alla presenza dell'indicizzazione tramite key (gestione dell'unicità e ricerca) e del tempo d'accesso logaritmico.

Le classi sono:

3.a Class DB: contiene un Qmap(IdLogin,User*) che viene utilizzato come database caricato in memoria. load(QString) e save(QString) permettono il salvataggio e caricamento del Db e sono anche usati per il backup e il cambio database. ControlAdmin() e controlPw() controllano la password corrispondente ad admin e user.

Non utilizzando un database di grandi dimensioni non si è posto il problema del elevato numero di dati si è preferito quindi diminuire il meno possibile gli accessi a disco caricando il database in ram, utilizzando puntatori per modificare o leggere eventuali elementi in esso. Questo ha portato al non utilizzo di smart-pointer in quanto non si presenta la necessità di eliminare eventuali copie del user presente in ram. La copia è unica e viene eliminata solo tramite il delete da parte del admin. In una ottica reale di social network questo ultimo sistema necessiterebbe di un load parziale del database e di un controllo più severo delle risorse.

3.b Class info: classe adibita alla gestione delle informazioni per il profilo utente, separata da profile per questioni di praticità di lettura.

Contiene metodi per get e set, e metodi quali delesp(), addesp() che permettono di eliminare e inserire esp al vector di riferimento. Di interesse anche editEsp() in due modalità:

“const Qvector<Esperienza>” che controlla tutto il Vector con quello nuovo e modifica solo le differenze.

“constEsperienza&,const int” che controlla solo l’esp[int] del QVector.

3.c Class profile: classe di unificazioni delle varie classi per la gestione del profilo. Contiene QString Image che indica il percorso dell’immagine.

3.d Class user:

idLogin{} per l’id del utente(email) con metodi get e set.

Network{} classe per la gestione del network utente contiene Nlink= numero di utenti in lista. Net è un vector di QString contenente le key della rete del user.

user è classe astratta, contiene tre funzioni virtuali pure, dichiarate nelle classi figlie BasicUser, BussinessUser , ExcutiveUser:

myViewYou permette di vedere le informazioni di un utente non presente nella propria rete contatti, in base alla propria tipologia di account

typesUser ritorna la tipologia di account

research(vector<QString>) inizialmente doveva essere una funzione a funtori, in seguito causa tempo e difficoltà di implementazione si è optato per la sua forma embrionale. Permetterebbe la ricerca tramite filtri multipli, opzione non implementata nella versione finale. La ricerca è gestita tramite due vector<QString>, il primo contiene nell’indice i-esimo il numero del filtro da controllare. Il secondo nello stesso indice i-esimo la stringa di confronto di quel dato filtro.

Si è scelto una derivazione con classe padre astratta e tre figli a discapito di una gerarchia di ereditarietà a scala, in modo da semplificare la modifica della singola tipologia e in caso di necessità di aggiunta di un’ulteriore tipologia.

3.e Class VersionInterface: Contiene le funzioni utilizzabili dall’utente o dal admin, diviso in due file per semplicità di lettura.

adminVersion: contiene le funzioni relative alla gestione di utenti e del database. Dispone di un Adb in cui viene caricato un database e di una ricerca propria uguale a quelle del user.

userVersion: contiene le funzioni relative del user, che ha accesso solo ai suoi dati o tramite ricerca a quelli di altri utenti, comunque senza possibilità di modifica.

4. GUI

L'interfaccia grafica è stata divisa in due modalità, User e Admin, costruite su tre tipologie di classi:

Model che si occupano delle funzioni di reperimento dati

View che si occupa del interazione con l'utente

Control che gestisce i dati raccolti da Model e decide come View deve mostrarli

Control viene avviato al avvio del programma, quest'ultimo lancia LoginView.cpp che permette la raccolta delle informazioni di login, se non corrette, rilancia LoginView.cpp, altrimenti, in base alla email inserita (se contiene o meno l'indirizzo "LinQedln") lancia homeAdminView o homeUserView. Per permettere l'invio e raccolta parametri nei public slots si sono usate le lambda expression con c++11.

4.a controller: Gestisce le interazioni tra Model e View. Consiste in un classe contenente tre puntatori principali rispettivamente a homeAdminView, HomeUserView, loginView.

Ogni richiesta della grafica passa per il controller che comunica con il Model per risolverla e chiama le funzioni delle home per visualizzare i risultati. Ogni home ha come puntatore lo stesso oggetto controller, questo permette di comunicare tramite puntatori ed evitare passaggi per valore usati solo per le QString o info. Per evitare modifiche indesiderate si utilizza il const nei metodi di sola lettura.

4.b homeAdminView: Composto da quattro GroupBox e un QTextEdit.

A sinistra la lista delle azioni possibili da parte del Admin, in basso la zona risultati, in alto la ricerca con a fianco un QCombo per scegliere il filtro di ricerca.

Modifica utente preleva l'id utente da modificare e tramite Control crea un utente fittizio e si avvia EditProfileView.cpp con tale utente come destinazione.

4.c HomeUserView: composto sempre da quattro GroupBox con stessa disposizione e presenza di immagine profilo.

Tramite ricerca nel database è possibile vedere immagine profilo, nome e informazioni in base alla propria tipologia account e aggiungerlo negli amici. Si è utilizzato una QScrollArea per permettere un elevato numero di risultati che sono QWidget. Il tasto "Visualizza Info" chiama dal Control userViewSearch che crea una semplice finestra contenente le informazioni del utente selezionato.

Profilo permette di accedere tramite tasto "Modifica Profilo" alla modifica delle proprie info (immagine, password, dati personali, lavoro, formazione, esperienze) che devono essere confermate premendo il pulsante salva sotto l'immagine profilo. Non è possibile cambiare le info con info vuote. Si era pensato di aggiungere la chiama "Modifica tipo account", azione che non sarebbe possibile al user se non tramite conferma di pagamento o admin, quindi si è omesso.

Un'ultima parola su Esperienze, permette di vedere un elenco di esperienze inserite, rimuoverle o aggiungerle. L'aggiunta è gestita da una piccola finestra creata da editUserProfile e non è indipendente dal salva. La modifica della singola esperienza, per non sovraccaricare la già carica interfaccia, non è possibile, sarà comunque possibile caricarne una nuova e eliminare la vecchia.

La GUI è totalmente separata dalla parte Model in quanto l'unico suo scopo è interfacciarsi all'utente. La grafica è molto sobria e priva di menù, in modo da rendere il più intuitivo possibile le operazioni soprattutto da user, un'interfaccia con un design più da Web che da Desktop.

L'inserimento e modifica dei valori avviene in tempo reale con l'aggiornamento della finestra.

5. Conclusione

Una delle più grandi difficoltà ritrovate nel progetto è la gestione della memoria. Sebbene nella parte model la gestione dei puntatori è più agevolata, nella GUI si sono trovate notevoli difficoltà, in quanto tutte le componenti necessitano di oggetti in heap. Per risolvere in parte il problema tutte le finestre che non siano le principali home vengono distrutte in caso di non utilizzo.

Un altro problema è stato le eccezioni. Inizialmente il sistema doveva funzionare con una struttura di eccezioni da pop-up in GUI, ma in questo modo si rischiava di unire la parte Model con la Control. Una soluzione sarebbe stata creare un controller adibito allo scopo di ascoltare le eccezioni da Model e inviarle a View. Causa tempo non mi è stato possibile testare questo sistema, ogni errore interno al model è visibile in console.

In definitiva il progetto è stato soddisfacente al livello di creazione meno a livello di progettazione. Indubbiamente una progettazione iniziale più profonda e completa, soprattutto della parte Model avrebbe portato notevoli benefici alla realizzazione finale.

6. Dati e funzionamento

Compilazione:

Da cartella eseguire:

qt-532.sh
qmake LinQedin.pro
qmake
make
L'eseguibile è disponibile nella cartella (./LinQedin).

Consigli:

- Lasciando vuoto la ricerca si ottiene l'intero database come risultato
- Si salvi sempre le modifiche da "modifica profilo" prima di cambiare finestra

Dati accesso- Admin:

Admin@linQed.net	Admin
MarcoUltron@linQed.net	Admin1

Dati accesso-User:

SUNU@gmail.it	MaII
Kikko@nuovoTest.com	test
MarcoNovelli@gmail.com	test
DavidTessaro@gmail.com	mamma2
ElisaM@gmail.com	mamma3
Test@gmail.com	test
test2@gmail.com	test
test3@gmail.com	test
MarcoRosso@gmail.com	Carne6
MurettoLolllO@hotmail.com	Nessuno
MakioMummo@gmail.com	Mummo4