RELAZIONE PROGRAMMAZIONE AD OGGETTI 2014/2015

Ad opera di Alberto De Agostini (579021)

Introduzione

Il progetto tratta lo sviluppo di un applicazione desktop con scopi simili al social network LinkedIn.

Specifiche progettuali e ambienti di lavoro

Il progetto è stato sviluppato in ambiente Windows 7 con compilatore minGW 4.8 (32 bit) e poi provato sui computer di laboratorio Paolotti in ambiente Linux. Editor usati notepad++ e Qt Creator (non è stato usato il Designer). Versione testata Qt 5, non è stata usata nessuna funzionalità di C++11.

Note per la compilazione e testing

Nella cartella consegnata si trovano tutti i file .h e .cpp come richiesto da specifica, un file .json contenente un database di prova (questo file è necessario), una cartella styles con un file di stile style.qss (non obbligatorio per il funzionamento) , una cartella images contenente tutte le immagini necessarie per la corretta visualizzazione dell'applicazione e il file .pro creato usando il comando 'qmake-qt532 – project'. Per la compilazione basterà eseguire il comando 'make'.

Se si vuole ricreare il file .pro è necessario aggiungere a mano in tale file la riga 'QT += widgets' all'interno.

Nel database di prova sono contenuti 3 profili completi (uno per ogni tipo di user), le credenziali di accesso sono:

- user: fede passw: angi
- user: federica passw: buggi
- user: boss passw: boss

Le credenziali di accesso per l'utente admin invece sono:

- user: admin passw: admin

Parte logica del progetto

Per la parte logica è stato seguito quanto consigliato dal professor F.Ranzato in classe in partenza, con diverse divergenze successive.

Gerarchia Utenti

User

|

BasicUser

|

BusinessUser

|

ExecutiveUser

Esiste una superclasse User (classe astratta) contenente tutto quello di cui ha bisogno un utente: un account, una rete di amici (linkednet) e un funtore per la ricerca. Esiste poi il file realuser che implementa 3 sottoclassi di User (rispettivamente BasicUser, BusinessUser e ExecutiveUser) (si veda ereditarierà sopra)

User contiene un puntatore a Account che conterrà tutte le informazioni legate all'account: dati personali, experienze lavorative, abilità, avatar e credenziali di accesso (tutte queste informazioni suddivise in diverse classi come Username, Info, Experience...) e un puntatore a LinkedNet che conterrà la rete di contatti a cui è connesso e medoti per operazioni su di essi (aggiunta e rimozione di un contatto).

Le differenze tra i vari tipi di utente si trovano sostanzialmente nella ricerca dei contatti con privilegi crescenti: BasicUser ha 10 risultati massimi e può cercare solo per nome o cognome, BusinessUser ha 50 risultati max e può cercare anche contatti tramite skills (ad esempio cerco qualcuno che sa programmazione), ExecutiveUser non ha un massimo numero di risultati, può cercare anch'esso tramite skills e può cercare persone tramite parte del nome o cognome (i.e. 'la ricerca di 'teo' tornerà tutte le persone di nome 'matteo' o 'teocole'), questo user inoltre durante la visita di uno dei suoi contatti può vedere la rete di utenti a cui quest'ultimo è collegato. (Si volevano estendere più differenze tra i vari tipi di utenti ma non è stato possibile, si veda appendice 'problemi riscontrati' per informazioni).

La parte logica del progetto è stata progettata completamente separata dalla parte GUI e si è tentato di renderla più estendibile possibile. È stata creata anche la classe SmartUser per la gestione automatica della memoria per i vari Utenti (puntatore smart a User).

Classe Account

La classe Account all'interno di User contiene tutte le informazioni relative all'utente. Contiene una variabile AccountType che contiene l'informazione del tipo di account, questa variabile non era necessaria ma la sua presenza limita l'uso di RTTI da effettuare durante le varie operazioni percui velocizza l'applicazione. AvatarPath contiene semplicemente il path in cui è contenuta l'immagine relativa all'account, è poi presente un oggetto di tipo Username , tale classe contiene solamente le credenziali di accesso 'username' e 'password' per effettuare il login. Infine c'è un puntatore a Info, questa classe contiene tutte le informazioni 'personali' dell'utente, ovvero tutti i dati anagrafici, le abilità (skills), gli interessi, le lingue parlate e le esperienze lavorative passate, anch'esse definite tramite una classe Experience.

La suddivisione in classi per ogni 'pezzo' che compone l'utente aiuta a rendere più estendibile tutto il concetto di Utente.

LinkedNet (rete di contatti)

La rete di contatti a cui un utente è 'legato' è definita tramite la classe LinkedNet. Questa classe contiene una lista di smartUser (la lista sembrava il contenitore più veloce e meno dispendioso). Sono presenti i vari metodi per la gestione di questi contatti. Avere proprio gli oggetti User anzichè ad esempio una lista di Username rende più veloce la manipolazione delle informazioni nella parte Gui cosicchè non è necessario andare a leggere le infomazioni ogni volta sul database.

LinkedDB (database)

Questa classe si occupa di gestire a livello logico tutta la parte 'database' del progetto, ovvero esegue letture e scritture su file permanente (db.json), questo oggetto è il primo a venire allocato in memoria poichè serve a popolare tutta la struttura di utenti e di nets tramite una lista di SmartUser. Si è scelta la lista anche qui poichè sembrava il contenitore più adatto, inserimento in coda in O(1) , mentre con una vectore sarebbe necessario a volte eseguire l'ampliamento del contenitore senza avere nessun pregio in cambio.
Inizialmente si era pensato di gestirlo tramite una map ma poi è risultato superfluo e cambiato in list.

Questa classe presenta metodi per l'aggiunta o rimozione di utenti, un metodo di ricerca utenti, un metodo per la gestione dell'account Admin e infine i due metodi più importanti: Load e Save.

Questi metodi servono rispettivamente per caricare in ram tutto il database leggendo da file (load) e per scrivere su file il corrente stato di tutti gli utenti (save). Il metodo Save riscrive tutto il database ad ogni chiamata, sovrascrivendo il tutto anche se non ci sono state modifiche, questo lo rende meno efficiente di ciò che si potrebbe fare ma ho trovato difficoltà nel provare a far si che si scriva solo ciò che si modifica.

E' stato scelto di scrivere il file in formato JSON poichè da me molto apprezzato, credo sia un tipo di rappresentazione dati molto leggero, di facile lettura e non verboso a differenza di XML. Inoltre risulta essere di più facile interfacciamento in applicazioni web mediante javascript o altri linguaggi.

Interfacciamento logic-GUI

Sono presenti due classi di 'mediazione' tra la parte logica e quella grafica, queste classi sono LinkedClient e Admin. Entrambe contengono un puntatore a LinkedDB per permettere la modifica del database e posseggono tutti i metodi necessari all'interfacciamento tra la parte Gui e la parte logica dell'utente rappresentato per linkedclient e dell'admin per admin.

N.B. Admin è stata realizzata e poi non utilizzata per dimenticanza, c'è doppio codice percui tra admin e Gui_admin poichè in Gui_admin doveva essere contenuto un puntatore ad Admin per la gestione della parte logica (per rispettare un pattern mvc) ma ahimè avevo scordato di averla fatta (me ne sono reso conto quando ho scritto la relazione finale)!!!

GUI

Per la parte Gui non è stato usato mai QtDesigner.

Inizialmente appare una schermata di login, da essa si può aprire una schermata per la registrazione di un nuovo utente, in entrambe queste schermate i vari errori vengono gestiti tramite 'QmessageBox' come popup contenente l'informazione dell'errore (catturando le eccezioni lanciate dai metodi nella parte logica).

Al momento dell'ok di una nuova registrazione o alla pressione del button Login vengono controllate le credenziali (nel caso di login) e viene lanciata la schermata relativa all'utente.

Se le credenziali di login erano quelle dell'admin viene lanciata Gui_Admin. Questa finestra contiene una lista di tutti gli utenti registrati nel database (visualizzati tramite Username) e affianco dei bottoni per la gestione di essi: è possibile aggiungere un utente, rimuovere un utente, cambiare tipo di account ad un utente o tornare alla login. (è presente una textbox per filtrare gli utenti nel caso ne fossero presenti una grande quantità)

Se le credenziali di accesso non corrispondono all'utente admin ma ad un utente 'normale' allora viene aperta la finestra relativa alla classe Gui_User.

La finestra Gui_User è una semplice finestra strutturata in vari layout (almeno 1 per ogni sezione (i.e. skills)), nella prima colonna si trova l'avatar (per modificare l'avatar basta cliccarci sopra) , le informazioni personali e un bottone per abilitare la modifica di esse (quando il bottone viene premuto le Qlabel vengono nascoste e al loro posto appaiono delle Qtextedit). Le sezioni per le skill, interest languages e experiences sono tutte strutturate nello stesso modo, 1 qtextedit con annesso bottone per aggiungerne una e per ogni entry un bottone 'x' per l'eliminazione di essa (la mappatura viene eseguita tramite una vector di layout). Lo spazio della finestra è autogestito dai layout il che significa che aggiungendo widgets (i.e. una skill) tutti gli elementi della colonna in cui si aggiunge si spostano per fare spazio a quello nuovo, trovo questo comportamento non elegante e poco 'adatto' ad una applicazione desktop poichè se inserisco un numero elevato di entry il tutto si rimpicciolisce (la window ha una grandezza fissata e non ci sono contenuti scrollabili), tutto questi problemi sono descritti nell'ultimo paragrafo di questa relazione.

Nella colonna di sinistra sono presenti i propri contatti (cliccando su una label di un contatto si può visitare il profilo relativo) e una textedit in cui si può inserire il pattern per la ricerca di altri contatti. Quando si effettua

una ricerca le Qlabel dei propri ‘amici’ vengono sostituite da dei layout contenenti label e bottone per aggiungere il relativo contatto (anche questa mappatura di layout per collegare bottoni add ai vari utenti da aggiungere è fatto tramite vector di layout).

Ad ogni modifica del profilo (ogni aggiunta o rimozione di skill interessi o altro) il database viene salvato di volta in volta, col senno di poi forse era meno dispendioso per l’applicazione mettere un bottone save o salvare quando un utente effettua il logout.

Tutte le eccezioni sono ‘esternate’ all’utente tramite Qmessagebox contenenti i messaggi d’errore relativi.

Questa classe è usata anche per la visita di altri profili (tramite un booleano nella creazione si determina se è una userwindow ‘normale’ o una visita).

Problemi presenti nel progetto o difficoltà riscontrate

Praticamente la totalità dei problemi che sono elencati sotto deriva dal fatto di mancanza di tempo, essendo io uno studente lavoratore sebbene abbia avuto mesi per la realizzazione del progetto tra lavoro e la preparazione di altri esami ho avuto meno tempo di quanto avessi voluto per la realizzazione di questo progetto che ho trovato molto interessante e utile.

Uno dei problemi più grandi del progetto è la poca differenziazione degli utenti, avrei voluto aggiungere perlomeno gruppi e messaggi tra utenti con privilegi crescenti in base alla tipologia di utente (questa era la mia idea di partenza).

La parte Gui presenta codice non molto efficiente e scelte progettuali non ottimali (i.e. presenza di codice ripetuto per la gestione dei layout in skills interests ecc..).

L’avatar deve essere 200x200 px per essere visualizzato correttamente.

Dopo una visita di un altro profilo il bottone di logout non funziona!! (invece di tornare alla finestra di login esce dal programma).

Quando cambio il tipo di utente dalla pagina utente devo riloggare per avere le modifiche effettuate.