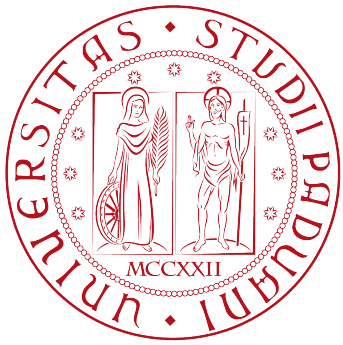
Università degli Studi di Padova

Corso di laurea triennale in Informatica



Corso: Programmazione ad oggetti

Prof. Francesco Ranzato

Relazione del Progetto **LinQedIn**

Studente: Alessandro Filira
Matricola: 1023662

Anno Accademico 2014/2015

Introduzione e scopo del progetto

Il progetto LinQedIn vuole riprodurre in maniera minimale e semplificata il sistema del principale social network per relazioni professionali: LinkedIn.
Tale applicativo anziché essere un servizio web è stato sviluppato in C++ con l'ausilio di Qt, una libreria multipiattaforma per lo sviluppo d'interfacce grafiche.

Il sistema sviluppato, come quello originale, offre ad ogni utente la possibilità di modificare il proprio profilo e aggiungere o rimuovere collegamenti dalla propria rete sociale.
Inoltre è presente un sistema di ricerca che, in base alla tipologia di account (Basic, Business, Executive), permette di visualizzare più o meno integralmente i profili degli utenti iscritti.

È stata sviluppata anche una sezione di amministrazione per l'intero sistema che permetterà di fare le seguenti operazioni: inserimento nel database di un nuovo utente, rimozione dal database di un utente esistente, ricerca nel database di un utente e cambio della tipologia di account. Tutte le modifiche apportate saranno ovviamente salvate su un "database" strutturato appositamente.

Scelte logiche progettuali

La prima scelta affrontata è stata quella riguardante il supporto da utilizzare per salvare i dati relativi agli utenti. Non avendo a disposizione un database si è deciso, come suggerito dal Professore, di optare per il salvataggio dei dati su file per la cui gestione esistono varie librerie c++ e qt di facile utilizzo.
Si è quindi creato un file XML che accogliesse in modo ordinato e comprensibile i diversi dati di cui si ha bisogno: nella prima parte sono presenti gli utenti con i loro dati del profilo e nella seconda parte per ogni utente è visibile la propria rete sociale.
Si suppone che ogni utente abbia username univoco all'interno del sistema.
È stata anche creata una **classe Username** i cui campi dati sono nome e password; l'operatore di uguaglianza per lo Username viene fatto semplicemente sulla stringa "name".
Inoltre il metodo getPassword reso pubblico è utilizzato solo dalla classe database per salvare il dato nel file XML.

La seconda valutazione è stata su come organizzare in maniera efficiente ed estendibile la gerarchia della principale entità modellata: l'utente.
Per fare ciò è stato deciso di prendere in esame diversi possibili modelli gerarchici, anche se è necessario da subito specificare che tali modelli non sono stati implementati.
1) Una classe Utente astratta che poi fosse resa concreta dalle sue classi derivate poste nello stesso piano una rispetto alle altre.
2) Una classe Utente astratta come nel primo caso ma che, anziché essere base di tre classi era base di un'ulteriore classe BasicUser la quale fosse a sua volta base di BusinessUser che veniva estesa da ExecutiveUser. Questa scelta sarebbe stata dettata dal fatto che effettivamente un utente Business non è nient'altro che un utente Basic con qualche funzione in più e il ragionamento analogo è accettabile anche nella relazione Executive- Business.
La scelta che è stata fatta e portata avanti quindi è la seguente:
3)Una classe base User astratta che è estesa da BasicUser e da BusinessUser e infine una classe ExecutiveUser la cui base è BusinessUser.
Questa scelta della gerarchia Utente è stata dettata dal fatto che si volevano **tenere separate le** due principali **tipologie di utente**, quello *free* da quello a pagamento.

Un'ulteriore possibilità sarebbe stata quella di creare altre due classi astratte FreeUser e PayUser entrambe con base Utente che venissero poi opportunamente derivate con le basicUser per la gerarchia free e le classi businessUser ed executiveUser per la parte PayUser. Tale scelta avrebbe implicato una maggiore attenzione nello sviluppo del metodo che si occupa del cambio della tipologia di utente.

Un'analisi fondamentale del progetto è stata riguardo la rappresentazione interna del database. Il database ha la funzione di contenere tutti gli utenti iscritti al sistema e qualora richiesto fornire informazioni su di essi.
Si trattava di scegliere un **opportuno contenitore** per tale classe.
Sostanzialmente la classe **database è una collezione di oggetti di tipo Utente**. Ma quale contenitore della libreria standard rappresenta in maniera corretta ed efficiente la base dati di cui si ha bisogno?
Sono stati analizzati i vari contenitori presenti nella libreria standard di C++ e il dubbio principale era tra Map e Liste.
Sappiamo che le Map(Dizionari) permettono di avere una chiave (Username) e un relativo valore (Utente) il cui tempo di ricerca rispetto alla chiave è molto rapido ma in questo caso il nostro sistema prevede la possibilità di ricerca anche per il valore e per campi dati di classi interne al valore, come il nome o il cognome.
In questo modo avendo una lista si ha la possibilità di accedere a tutti i campi del nodo Utente e poter implementare la ricerca come meglio si crede.
Se da un lato l'aspetto negativo del sistema con le liste è che per estrarre e cercare un elemento è necessario scorrere tutta la lista con tempo medio O(n); d'altro canto per inserire un nuovo elemento il tempo necessario è costante O(1), con evidente efficienza durante il caricamento inziale del database.
Alla fine di questa rapida analisi si è deciso di usare come contenitore una lista.

Sviluppo progetto

Sono state sviluppate alcune classi molto semplici che potessero essere di ausilio per raggiungere in maniera diretta lo scopo del progetto.
Il primo passo è stato quello di prendere in esame un caso reale di Utente LinkedIn il quale è composto da: informazioni personali, titoli di studio, esperienze lavorative, lingue conosciute e competenze.
Ognuna di queste caratteristiche è in realtà una collezione di singole voci, ad esempio una persona può avere più competenze, più titoli di studio e conoscere più lingue.
Ciò ha permesso di pianificare e organizzare la struttura del profilo di un singolo utente.

Sono state perciò create le classi:

- PersonalInfo: i cui campi sono nome, cognome e data di nascita;
- Skills: formata da una sola stringa per le competenze;
- Experience: per gestire le esperienze lavorative. Formata da nome, data inizio, data fine e descrizione.
- Languages: per gestire le lingue formata due campi dati string nome e livello
- Education: contiene campi dati string nome del titolo di studio, voto e un campo dato QDate anno.

Tutte queste classi sono provviste di metodi per *settare* e prelevare i dati.

Una volta sviluppate le classi che contengono i dati base per il profilo, è stata sviluppata la classe profilo.
La **classe profilo** è formata da un oggetto personalinfo, e da una serie di liste contenti puntatori per le competenze, per le lingue, per i titoli di studio e per le esperienze lavorative. Questa classe fornisce i metodi di *add* e *remove* per le varie liste, questi metodi vengono poi portati a livello più alto verso l'utente, classe nella quale il profilo è uno dei campi dati.

La classe Utente è formata da puntatori a Username, Profilo e Rete con adeguati costruttori: si suppone che un Utente esista se ha lo username.

La classe Utente è poi derivata dalle classi BasicUser e BusinessUser il cui costruttore non fa altro che richiamare il costruttore di utente.
Metodo polimorfo reso disponibile da tale classe è *getAccountType()* che torna una stringa con la tipologia di account (Basic, Business, Executive). Tale metodo poteva essere evitato utilizzando **RTTI** e in particolare il **dynamic_cast** ma la ragionevolezza di sapere quale tipo di account ha un utente è sembrata una buona motivazione per rendere tale metodo disponibile.
Altro metodo polimorfo per le classi derivate da utente è il metodo *userSearch()* utilizzato per effettuare la ricerca lato utente. La specifica del progetto prevede che il risultato di ricerca di un utente LinQedin mostri i dati visualizzati in base alla tipologia di account.
Ad esempio Basic visualizza solamente informazioni personali e titoli di studio, Business visualizza anche esperienze e lingue ed in fine Executive visualizza l'intero profilo.
Per sviluppare questa funzionalità si è utilizzato un funtore e la seguente idea di partenza: *UserSearch* scorre tutta la lista degli utenti e quando trova utenti che hanno nome, cognome o mail uguale al parametro di ricerca fa una copia dell'utente e lo inserisce in una nuova lista di smartuser.
La copia serve perché al funtore viene passata per riferimento la nuova lista creata e, per "mascherare" le informazioni, svuota la lista dei campi dati da nascondere attraverso i metodi privati della classe Utente.
Si rende necessaria la copia dell'utente affinché l'operazione di "*mascheramento"* non vada a modificare il vero utente del database, cosa che altrimenti comprometterebbe l'intero sistema.

Per la **gestione della memoria** è stata sviluppata la classe SmartUser. Tale classe crea oggetti che hanno come campo dati un puntatore a Utente il quale a sua volta ha una "*dichiarazione di amicizia"* nei confronti della classe smartUser in modo che possa aggiornare il campo riferimenti per contare i puntatori.
Così facendo quando i riferimenti saranno uguali a 0 l'utente viene eliminato evitando di avere *memory leak*.
Sono stati ridefiniti i metodi per accedere all'utente dello smartpointer con l'operatore di dereferenziazione (*) e accesso a membro (->).
Sono stati inoltre ridefiniti il costruttore di copia, il distruttore e l'assegnazione in modo da poter gestire pienamente la memoria.

La **classe rete**, che è un campo dati privato della classe Utente, è una lista di SmartUser che raccoglie i collegamenti della rete sociale dell'utente. Essa fornisce dei metodi per aggiungere e rimuovere collegamenti e cercare se un collegamento è presente nella rete.

La classe **LinQedinUser** e la classe **LinQedinAdmin** sono le classi che permettono di sviluppare le due tipologie di utilizzo dell'applicativo.
LinQedinAdmin permette infatti, attraverso i metodi interni, di avere una gestione completa del database e di poter inserire, creare e rimuovere utenti dalla classe database.

È stata sviluppata anche la funzionalità per cambiare tipologia di utente, come previsto dalla specifica del progetto. Tale metodo effettua una copia di un utente (come per il funtore) e lo assegna ad un puntatore diverso della gerarchia, ad esempio da basicUser viene assegnato a executiveUser. Successivamente viene chiamata la funzione *replace* che sostituisce nel database un utente con un altro in questo modo si mantiene l'ordine nella lista e infine si invoca la *delete* sul vecchio utente.

LinQedinUser include le funzionalità per la modalità utente. La classe è dotata di un puntatore a utente e un puntatore a database. Tutti i metodi di *set* e *get* vengono eseguiti sull'utente memorizzato nella variabile *currUser*.
In fase di login viene costruito l'oggetto Linqedinuser con puntatore a utente nullo, successivamente viene invocato il metodo *verifyLogin* che scorre il database e individua se le credenziali inserite corrispondono ad un utente esistente.
Se tale metodo torna *true*, il login viene effettuato e *currUser* viene settato con l'utente riconosciuto.
Le diverse credenziali per il login sono indicate nell'appendice 2.

Sviluppo grafica principale

Per tenere separata la parte grafica da quella logica non è stato utilizzato il design pattern MVC ma un sistema più semplice che non prevede uso di controller e view.
Le classi grafiche e quelle logiche sono state separate in due diverse directory "lib" e "gui"; in questo modo è più facile riconoscere le due tipologie e poter intervenire senza il rischio di aggiungere funzioni logiche nella classi grafiche.
Per sviluppare la grafica non è stato utilizzato il tool *QtDesigner* in quanto si è preferito scrivere il codice totalmente a mano.
Tale strumento è stato utilizzato solo in fase iniziale per capire quali componenti (Widget) la libreria Qt rendesse disponibili ed utilizzabili. È stato invece molto utilizzato il tool qtAssisant che contiene la documentazione delle varie classi Qt.

Poiché il progetto prevedeva due interfacce diverse per utente e amministratore, si è deciso di far avviare l'applicativo su una *QDialog* che gestisce il login, la registrazione di un nuovo utente e permette, in base ai dati inseriti, di creare la *MainWindow* per l'admin oppure la *ManiWindow* per l'utente.
In entrambe le *mainWindow* viene caricato il relativo file logico che gestisce l'amministratore o l'utente *LinQedin*.
Queste *mainWindow* sono dotate poi di campi dati che permettono la visualizzazione grafica di un profilo come le classi showProfile la quale crea oggetti di tipo showeducation, showexperience, showlanguages, , showpersonainfo, showskill e shownetwork e le relative classi per modificare tali campi come editProfile la quale crea oggetti di tipo editeducation, editexperience, editanguages, , editpersonainfo, editskill e editnetwork.

È stato inoltre sviluppato il QWdiget showUsersList che genera la lista degli utenti presenti nel database.

Appendice 1

Per compilare da terminale portarsi nella cartella principale del progetto ed eseguire le successive istruzioni:

1. qt -532.sh
2. qmake LinQedin.pro
3. qmake
4. make

Una volta che la compilazione termina si ha il file eseguibile nella directory principale

Appendice 2

Di seguito sono elencate le credenziali per accedere al sistema LinQedin con gli utenti già presenti nel database.

Amministratore:
Username: admin
Password: admin
Ricordarsi di mettere il check come amministratore.

Utenti presenti:

Email	Password	Nome	Cognome
colombo@navigazioni.it	caravel3	Cristoforo	Colombo
a.filira@altramarca.net	ale	Alessandro	Filira
trabucchi@uni.it	traba	Alberto	Trabucchi
vale@motogp.it	vrox	Valentino	Rossi
saviano@feltrinelli.it	RobySav	Roberto	Saviano
andreotti@governo.it	giulio	Giulio	Andreotti
johnny_d@universal.com	J_Deep	John	Deep
m.frittella@rai.it	Marco58	Marco	Frittella
cristoforetti@spazio.it	universo	Samantha	Cristoforetti
hack@astrofisica.net	Stelle	Margherita	Hack
donbosco@diocesito.it	giovani	Giovanni	Bosco
generalediaz@esercito.it	GenDiaz	Armando Vittorio	Diaz
montalcini@pas.va	Rita7862	Rita	Levi-Montalcini
renzorosso@jeans.it	tessutiyeah	Renzo	Rosso
jm@calcio.com	pallacuoio	Josè	Mourinho
h.clinton@usa.it	45cl98	Hilary	Clinton
miyazaki@stghibli.com	totoro	Hayao	Miyazaki
galderisi@calciatore.net	nanupd	Giuseppe	Galderisi
direttore@louvre.fr	gioconda	Claudia	Ferrazzi
zaccaria@rettoreunipd.it	palazzoBo	Giuseppe	Zaccaria
vettel@ferrari1.it	volante	Sebastian	Vettel
giov.arco@repubblica.fr	freccia	Giovanna	D'Arco