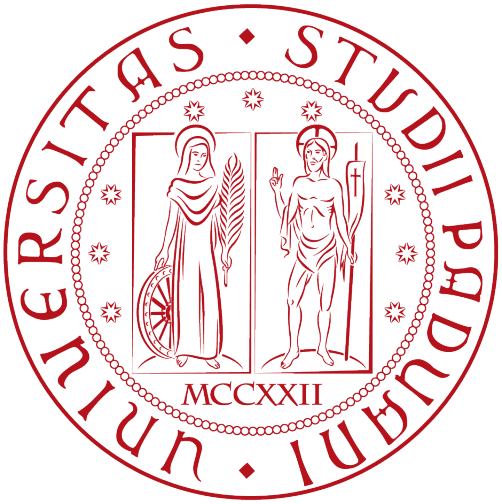

LinQedln

Relazione progetto LinQedln v1.0

Sommario
Lo scopo del progetto LinQedln è quello di creare un' applicazione in c++ munita di interfaccia grafica (GUI), utilizzando le librerie Qt.

Nome del documento	Relazione progetto LinQedln
Versione	1.0
Data di redazione	2015-07-06
Redazione	Mattia Favaron - 1028487

Indice

1 Analisi dei requisiti

1.1 Attori

Al sistema parteciperanno due attori:

- Utente;
- Amministratore.

Un utente potrà accedere al sistema e utilizzarne le sue funzionalità, solo dopo essere stato aggiunto dall'amministratore.

1.2 Requisiti

Vengono elencati i requisiti obbligatori indicati nel documento *specifica del progetto* e i requisiti dedotti da esso che il progetto mira a soddisfare.

L'amministratore del sistema deve avere la possibilità di:

- inserimento nel DB di un nuovo utente. I dati per l'inserimento saranno nome, cognome e username univoco dell'utente.
- Ricerca nel DB di un utente mediante username o nome/cognome.
  - la ricerca restituirà username, nome e cognome degli utenti trovati;
  - per ogni utente trovato si avrà la possibilità di vederne il profilo completo.
- Rimozione dal DB di un utente.
- Cambio di tipologia di account (Basic, Business, Executive) per un utente.
- Salvataggio e lettura su file del DB degli utenti LinkedIn.

Ogni utente deve avere la possibilità di:

- modificare i dati associati al proprio profilo quali:
  - dati personali;
  - titoli di studio;
  - esperienze professionali.
- inserire e rimuovere un utente dalla sua rete di contatti;
- ricercare un altro utente nel DB. La ricerca e i relativi risultati ottenuti cambieranno in base alla tipologia di account. La ricerca in base all'account sarà così implementata:
  - basic: cercherà nel database altri utenti solo per usernane, nome e cognome.
  - business: ricercherà altri utenti per username, nome, cognome, email, luogo di nascita, residenza, diploma o laurea.
  - executive: ricercherà altri utenti per username, nome, cognome, email, luogo di nascita, residenza, diploma o laurea e in più nome azienda e titolo (es. Manager).

  La ricerca restituirà una lista di utenti caratterizzata da username, nome e cognome. Di ogni utente se ne potrà visualizzare il profilo. La visualizzazione del profilo cambia per ogni utente.
  - basic: visualizzerà solo username, nome e cognome.
  - business: visualizzerà username, nome, cognome, email, data di nascita, luogo di nascita, residenza, diploma ed eventuali lauree.

  - executive: visualizzerà il profilo completo, che sarà costituito da tutto quello che visualizza l'utente business e basic più eventuali esperienze lavorative.

2 Design pattern MVC

Nel modellare l'applicativo, si è deciso di utilizzare il design pattern MVC (Model-View-Controller).

2.1 Model

Il model è costituito dalle seguenti classi e relativi campi:

- DatiAnagrafici
  - nome : `QString`
  - cognome : `QString`
  - email : `QString`
  - dataNascita : `QDate`
  - luogoNascita : `QString`
  - residenza : `QString`
- TitoliStudio
  - diploma : `QString`
  - laurea : `vector<QString>`
- Lavoro
  - nomeazienda : `QString`
  - titolo : `QString`
  - citta : `QString`
  - inizio : `QDate`
  - fine : `QDate`
- CompetenzeLavorative
  - esperienze : `vector<Lavoro>`
- Username
  - login : `QString`
- Profilo
  - datiPersonali : `DatiAnagrafici`
  - studi : `TitoliStudio`
  - curriculum : `CompetenzeLavorative`
- Rete
  - follow : `set<QString>`
- Utente
  - login : `Username`
  - info : `Profilo`

  - rete : `Rete`

- DB
  - dbUtenti : `map<QString,Utente*>`

All'interno del progetto sono stati utilizzati tre diversi contenitori:

- **`vector`**: è stato utilizzato per contenere una lista di lauree e una lista di esperienze lavorative per ogni utente. Si è deciso di utilizzare un `vector` per semplicità di utilizzo e per la sua proprietà di inserimento e rimozione in coda in tempo costante.
- **`map`**: è stato utilizzato per contenere una lista di puntatori ad Utenti nel database. Si è deciso di utilizzare un `map` per la sua proprietà nel garantire un ricerca e un inserimento con complessità O(log(n)) con liste molto grandi.
- **`set`**: è stato utilizzato per contenere la lista di username di utenti seguiti dall'utente. Si utilizza un `set` data la sua particolarità di essere una collezione ordinata priva di doppioni e per la ricerca per valore con complessità logaritmica.

2.1.1 Gerarchia tra le classi

2.1.1.1 `Utenti`

Per implementare gli utenti di LinQedln, si è scelto di adottare una gerarchia verticale.

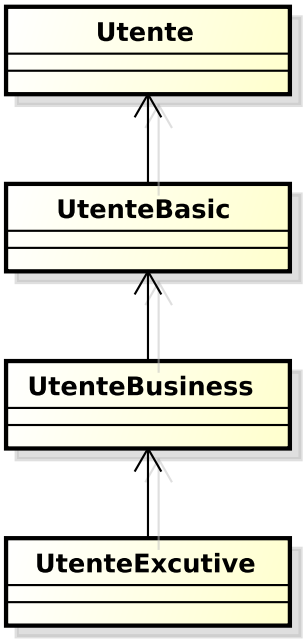

Figura 1: Gerarchia verticale utenti

La classe base `Utente`, è astratta. Si è deciso di utilizzare una gerarchia verticale dato che ogni utente ha privilegi e funzionalità di ricerca sempre maggiori. Ogni utente potrà ricercare altri utenti inscritti al sistema. Si partirà dall'account Basic, che avrà la possibilità di cercare solo per username nome e cognome fino e ad arrivare all'account Executive che avrà tutte le funzionalità di ricerca degli altri utenti (basic e business), più avrà la possibilità di ricercare per nome aziende per cui un'utente ha lavorato e mansione ricoperta.

2.1.1.2 `Controller`

Anche le classi controller del progetto sono organizzate in una gerarchia di classi di tipo orizzontale. La classe `Controller` virtuale pura e le due classi derivate `UserController` e `AdminController`. Ogni classe derivata gestisce la relativa utenza che utilizza il servizio: utente o amministratore. Entrambe le classi contengono un puntatore al DB nella parte privata e un `map` per contenere gli utente trovati dalle funzioni di ricerca. `UserController` inoltre contiene nella sua parte pubblica un puntatore all'utente che sta utilizzando l'applicativo.

2.1.2 Rete di contatti

Per gestire la rete di contatti si è deciso di implementare una lista `set<QString>` per ogni utente. Ogni lista conterrà gli username di tutti gli utenti seguiti dall'utente. Per gestire la rete di contatti tra utenti, si è deciso che quando l'utente A segue un utente B, anche quest'ultimo seguirà il primo senza che questa debba accettare l'amicizia o quant'altro. La stessa cosa vale anche nel caso di rimozione dalla rete di contatti.

2.2 View

Le classi che compongono la view partono tutte dalla `MainWindow`. In base all'utente che farà login nel sito, che sia admin o utente, inizializzerà determinate view quali: `UserWindow`, `AdminWindow`.

2.2.1 `UserWindows`

La `UserWindow` fa da "contenitore" per altre 5 classi:

- `UserInfoWindow`;
- `UserSchoolWindow`;
- `UserWorkWindow`;
- `UserReteWindow`;
- `SearchWindow`.

Tutte queste classi costituiscono il lato client.

2.2.1.1 `UserInfoWindw`

La classe `UserInfoWindow` è l'interfaccia della classe `DatiAnagrafici`. Questa offre la possibilità di poter visualizzare e aggiornare il profilo dell'utente che attualmente sta utilizzando l'applicativo.

2.2.1.2 `UserSchoolWindow`

La classe `UserSchoolWindow` è l'interfaccia della classe `TitoliStudio`. Questa offre la possibilità di visualizzare e aggiornare il diploma di scuole superiori ottenuto dall'utente. Permette inoltre di visualizzare, aggiungere, modificare e eliminare eventuali lauree ottenute. Per visualizzare tutte le lauree, si appoggerà ad un'ulteriore classe `laureaWindow`.

2.2.1.3 `UserWorkWindow`

La classe `UserWorkWindow` è l'interfaccia della classe `CompetenzeLavorative`. Questa offre la possibilità di visualizzare, aggiungere, modificare e eliminare eventuali esperienze lavorative effettuate dall'utente. Per visualizzare correttamente tutte le esperienze lavorative, si appoggia alla classe `lavoroWindow`.

2.2.1.4 `UserReteWindow`

La classe `UserReteWindow` è l'interfaccia della classe `Rete`. Questa offre la possibilità di visualizzare username,nome e cognome di tutti i contatti seguiti dall'utente. Offre inoltre la possibilità di visualizzare il profilo parziale o completo (in base al tipo di account che l'utente possiede) degli utenti seguiti appoggiandosi alla classe `ResultWindow`.

2.2.1.5 `SearchWindow`

Visto che la classe `SearchWindow` è condivisa da entrambe le view, sarà spiegata nelle prossime sottosezioni.

2.2.2 `AdminWindow`

La classe `AdminWindow` costruisce interamente l'interfaccia del lato Admin. Questa permette l'amministratore di aggiungere un nuovo utente, eliminare un utente iscritto al servizio o cambiarne il tipo di account (Basic,Business,Executive). Inoltre permette all'amministratore di cercare qualsiasi utente presente iscritto al servizio, appoggiandosi alla classe `SearchWindow`.

2.2.3 `SearchWindow`

La classe `SearchWindow`, come detto in precedenza, viene utilizzata sia dal lato client, sia dal lato Admin. Questa permette all'utente o all'amministratore di ricercare un'utente iscritto al servizio. Il client avrà diverse funzionalità di ricerca che cambieranno in base al tipo di account dell'utente. L'amministrato invece potrà ricercare gli utenti solo tramite username, nome e cognome. La lista di utenti trovati sarà visualizzata sfruttando al classe `ResultWindow`. Questa classe oltre a permettere la visualizzazione del profilo, parziale o completo all'utente (in base al tipo di account) e completo all'amministratore, offrirà inoltre la possibilità all'utente di aggiungere o rimuovere gli utenti trovati alla propria rete.

3 Salvataggio

Nel gestire il salvataggio, si è deciso di salvare il database sul disco, ogni qualvolta avvenga una modifica:

- modifica informazioni profilo;
- modifica diploma;
- aggiunta, eliminazione, modifica laurea;
- aggiunta, eliminazione, modifica esperienza lavorativa;
- aggiunta o rimozione di un utente alla rete di contatti;
- aggiunta, rimozione o modifica account di utente.

In questo modo si riduce al minimo la perdita di informazioni, ma nel contempo, provoca un gran dispendio di risorse nel caso il database sia molto grande. Per ovviare a questo problema si potrebbe valutare l'implementazione in un futuro di un metodo di salvataggio più intelligente (es. Salvare su disco ogni n operazioni) oppure lasciare l'operazione nelle mani dell'utente (pulsante che richiama funzione di salvataggio).

4 Ambiente di sviluppo

Il progetto è stato sviluppato in ambiente Linux Kubuntu 14.10 utilizzando l'IDE Qt Creator 3.2.1 basato su Qt 5.3.2.

5 Utilizzo e compilazione

5.1 Credenziali d'accesso

Per accedere nell'applicativo come amministratore, basterà inserire la password "admin" all'apertura del programma, nella sezione dell'amministratore. Per accedere invece come utente, si dovrà prima registrare un utente tramite l'apposita funzione nel lato amministratore e poi basterà inserire l'username scelto nella sezione dell'utente all'apertura del programma.

5.2 Compilazione

Per la compilazione dell'applicativo con Qt5.3.2 seguire i seguenti comandi:

```
qmake-qt532 -project
```

Nel file .pro generato aggiungere nell'intestazione:

```
QT += widgets
```

Dopo avere salvato le modifiche apportate, da terminale eseguite:

```
qmake-qt532
```

```
make
```

Troverete così l'eseguibile del prodotto.