*Programmazione ad Oggetti*

*Progetto “QForum”*

*Relazione di* **Mirko Stella**

Matricola **1201184**

Compagno di lavoro **Lorenzo Matterazzo**

*a.a. 2020/2021*

**COMPILAZIONE ED ESECUZIONE**

É stato inserito il file .pro in quanto sono necessari dei flag (QT+=widget e QT+=xml) non presenti di default.

Per compilare ed eseguire l'applicazione bisogna eseguire i seguenti comandi:

qmake -> make -> ./utenti da terminale

**AMBIENTE DI SVILUPPO**

Sistema Operativo: Windows 10 Home

Versione QT: Qt 5.15.2

Compilatore:

**PRESENTAZIONE DEL PROGETTO**

Il progetto riguarda lo sviluppo di un’ applicazione che promuove l’interazione tra utenti per trovare delle risposte in ambito lavorativo o di studio su problemi che possono presentarsi.

Ogni utente possiede un profilo modificabile e visibile agli altri utenti (in modo piú o meno dettagliato in base al tipo di account che possiedono) in modo da illustrare agli interessati le informazioni che lo qualificano e la sua email.

Ogni utente puó porre le proprie domande e ricevere delle risposte dagli altri utenti per poi valutare la risposta piú soddisfacente ed attribuirgli un like.

L'applicazione divide i suoi utenti in tre piani: **Basic, Gold e Premium**.

Ogni utente indipendentemente dal piano che possiede puó cercare domande e utenti, aggiungere o rimuovere amici dalla propria rete di contatti, modificare il proprio profilo, scrivere commenti alle domande, dare dei like e cambiare piano.

Il piano Basic è gratuito ma ha meno privilegi rispetto agli altri che sono a pagamento.

I privilegi si basano sulla possibilitá di ricercare i profili degli utenti in modo piú o meno dettagliato e sul numero di domande che l’utente in questione puó formulare.

A seconda del piano scelto la ricerca di un utente fornisce dettagli aggiuntivi: possedendo il piano **Basic** la ricerca di un utente produrrá come risultato il suo username, nome, cognome ed email; con il piano **Gold** il suo username, nome, cognome, email, i titoli di studio e le competenze professionali; con un piano **Premium** il suo username ,nome ,cognome, email, titoli di studio, competenze e la sua rete di amici e seguaci.

A seconda del piano scelto l’utente possiede un numero iniziale di punti e per ogni domanda posta ne vengono sottratti un certo numero fissato.

Al termine dei punti a disposizione non sará piú possibile porre altre domande.

Per ottenere punti si puó scegliere di cambiare piano o di rispondere alle domande degli altri utenti e nel caso in cui la risposta riceva un like da parte dell’utente che ha posto la domanda verranno accreditati i punti in base al piano che si possiede.

È stata aggiunta la possibilitá di effettuare l’accesso come amministratore.

Per motivi pratici di valutazione non è richiesta alcuna password per accedere come amministratore mentre per il lato utente è richiesto l'username e la password di accesso.

Dalla pagina dell’amministratore si possono aggiungere utenti, togliere utenti, modificare i piani e visualizzare tramite una tabella gli utenti presenti nel database e i loro relativi campi dati.

Parte logica:

Seguono delle brevi descrizioni delle principali classi utilizzate nella parte logica con la spiegazione delle loro funzioni polimorfe.

**DESCRIZIONE DELLA GERARCHIA DI TIPI**



**UTENTE**: Classe base astratta con le principali funzionalitá di un generico utente caratterizzata dai campi privati Profilo,Accesso,Domande,Amici,Seguaci.

Profilo ed Accesso contengono rispettivamente le informazioni personali e le credenziali per effettuare l'accesso al proprio account.

Domande é un contenitore di puntatori a domande ed ogni puntatore punta ad una domanda posta dall'utente.

Amici e seguaci sono dei contenitori di puntatori ad utente che puntano agli amici e ai seguaci dell'utente.

La classe contiene i metodi virtuali puri: **cerca\_utente,get\_punti\_domanda,cerca\_domanda,fai\_domanda,clone**.

Questi metodi dovranno essere implementati nelle sottoclassi e ne definiranno i privilegi rispetto alle altre.

**PAGAMENTO**: classe astratta derivata da UTENTE aggiunta per differire la ricerca delle domande tra un utente Basic e un utente con un piano a pagamento (Gold,Premium).

**cerca\_domanda**: cerca le domande di tutti gli utenti presenti nel database e restituisce una lista di domande dove in testa ci saranno le domande degli amici in ordine decrescente di prioritá e a seguire quelle degli altri utenti in ordine decrescente di prioritá.

La classe pagamento offre un metodo astratto puro **get\_punti\_bonus.**

**BASIC**: classe derivata da UTENTE istanziabile.

Non ha funzionalitá aggiuntive rispetto alla classe utente ma si limita a dare la definizione dei metodi virtuali puri di UTENTE (cerca\_utente,get\_punti\_domanda,cerca\_domanda,fai\_domanda,clone).

**cerca\_utente**: crea un FUNTORE costruito passando il parametro 1 e su quel funtore applica l'operatore () passando un puntatore ad utente ed un contenitore di stringhe vuoto per riferimento. In questo caso la chiamata dell'operatore () inserirá nel contenitore l'username,nome,cognome,email dell'utente cercato.

**cerca\_domanda**:restituisce un contenitore di puntatori a domande (in ordine decrescente di prioritá) che matchano con almeno un numero di parole stabilito con la domanda da cercare.

La ricerca viene fatta solo tra le domande degli amici.

**get\_punti\_domanda**: aggiunge punti all'utente che ha ricevuto il like ed incrementa il numero di like ricevuti.

**fai\_domanda**: inserisce il puntatore alla domanda nel contenitore di puntatori a domande dell'utente. Se la prioritá della domanda é maggiore di 1 sottrae dai punti dell'utente un supplemento.

**clone**: restituisce una copia profonda di un utente BASIC.

**GOLD**: classe derivata da PAGAMENTO istanziabile.

Denifinisce il metodo **get\_punti\_bonus**.

**get\_punti\_bonus:** somma ai punti dell'utente i punti bonus stabiliti dalla variabile statica PuntiBonus.

Definisce i metodi virtuali puri di UTENTE che non sono ancora stati definiti per poter istanziare la classe. (cerca\_utente,get\_punti\_domanda,fai\_domanda,clone).

**cerca\_utente**: crea un FUNTORE costruito passando il parametro 2 e su quel funtore applica l'operatore () passando un puntatore ad utente ed un contenitore di stringhe vuoto per riferimento. In questo caso la chiamata dell'operatore () inserirá nel contenitore l'username,nome,cognome,email,le competenze e i titoli di studio dell'utente cercato.

**get\_punti\_domanda**: aggiunge punti all'utente che ha ricevuto il like ed incrementa il numero di like ricevuti. Inoltre se il numero di like ricevuti supera una certa soglia vengono attribuiti i punti bonus utilizzando get\_punti\_bonus.

**fai\_domanda**:inserisce il puntatore alla domanda nel contenitore di puntatori a domande dell'utente. Se la prioritá della domanda é maggiore di 2 sottrae dai punti dell'utente un supplemento altrimenti porta la prioritá a 2.

**clone**:restituisce una copia profonda di un utente GOLD.

**PREMIUM**: classe derivata da PAGAMENTO istanziabile.

Denifinisce il metodo **get\_punti\_bonus**.

**get\_punti\_bonus:** somma ai punti dell'utente i punti bonus stabiliti dalla variabile statica PuntiBonus.

Definisce i metodi virtuali puri di UTENTE che non sono ancora stati definiti per poter istanziare la classe. (cerca\_utente,get\_punti\_domanda,fai\_domanda,clone).

**cerca\_utente**:**cerca\_utente**: crea un FUNTORE costruito passando il parametro 3 e su quel funtore applica l'operatore () passando un puntatore ad utente ed un contenitore di stringhe vuoto per riferimento. In questo caso la chiamata dell'operatore () inserirá nel contenitore l'username,nome,cognome,email,le competenze,i titoli di studio,gli amici e i seguaci dell'utente cercato.

**get\_punti\_domanda**: aggiunge punti all'utente che ha ricevuto il like ed incrementa il numero di like ricevuti. Inoltre se il numero di like ricevuti supera una certa soglia vengono attribuiti dei punti bonus utilizzando get\_punti\_bonus.

**fai\_domanda**:inserisce il puntatore alla domanda nel contenitore di puntatori a domande dell'utente. Se la prioritá della domanda é maggiore di 3 sottrae dai punti dell'utente un supplemento altrimenti porta la prioritá a 3.

**clone**:restituisce una copia profonda di un utente PREMIUM.

**CONTENITORE**:

Il contenitore templatizzato che abbiamo scelto di implementare é una lista doppiamente linkata.

Sono presenti le principali funzionalitá di una lista ovvero l'inserimento in testa,l'inserimento in coda,la rimozioni in testa,la rimozione in coda,il metodo per verificare se una lista é vuota, un metodo che ritorna il numero di elementi presenti all'interno della lista, le classi iteratore che permettono di scorrere gli elementi e i metodi e operatori che utilizzano tali iteratori.

Abbiamo scelto di inserire delle funzioni aggiuntive che secondo noi potevano fare al caso nostro come ad esempio l'operatore di indicizzazione e una funzione insertionSort per ordinare le domande in base alla loro prioritá.

Il contenitore presente nella classe database contiene dei DeepPtr che utilizzano come parametro dei puntatori ad utente.

**PATTERN DI COMUNICAZIONE VISTA/MODELLO**

Il pattern utilizzato é **MVC** (Model-View-Controller).

Sono presenti due controller: uno per la comunicazione tra vista utente e modello e uno per la comunicazione tra vista amministratore e modello.

Il primo controller contiene un puntatore alla vista utente e uno all'account dell'utente; il secondo un puntatore alla vista amministratore e uno al database.

**INPUT/OUTPUT**

All'esecuzione dell'applicazione la costruzione della vista Login crea un database e chiama il metodo import definito in database.

Import cerca nella directory di livello superiore al file progetto il file database.xml e il file database\_domande\_e\_amici.xml.

Import utilizza delle funzioni della libreria QT che sono in grado di dedurre attraverso i vari tag da cui é formato il file xml da leggere i vari campi degli utenti,domande e commenti.

Viene chiamato per primo l'import su database.xml che crea gli utenti perché se cosí non fosse potrei costruire delle domande o commenti relative ad utenti che non sono ancora stati creati.

La stessa cosa vale per le relazioni di amicizia tra utenti (non potrei inserire nel contenitore amici o seguaci di un utente degli utenti che non sono ancora stati creati).

L'export, invece, viene chiamato quando un utente fa il logout o chiude l'applicazione e dalla vista amministratore quando si salva il database.

L'export si preoccupa di generare il file creando i vari tag che contengono le informazioni corrispondenti.

**DESCRIZIONE INTERFACCIA UTENTE E AMMINISTRATORE**

Vista Login:



Vista visibile al lancio dell'applicazione.

Per accedere come utente si devono inserire username e password e premere su accedi o premere invio da tastiera.

Premendo su registrati si aprirá la finestra per registrare un nuovo utente e verranno richiesti i campi dati dell'utente che si vuole aggiungere.

É possibile accedere come amministratore cliccando su accedi come admin (non sono richiesti né username né password).

Un utente é Mirko con password 0000.

Vista Utente:



Vista visibile dopo aver effettuato l'accesso come utente dalla vista Login.

Da qui é possibile visualizzare e successivamente (se si desidera) modificare il proprio profilo premendo su "Profilo" in alto a sinistra,cercare delle domande con la LineEdit "cerca domanda" ,cercare un utente con la LineEdit "cerca utente" (premere il tasto "Cerca" alla destra di ciascuna di esse o il tasto invio da tastiera dopo aver inserito il testo da cercare ).

Subito sotto é visibile una tabella al cui interno é presente una scrollArea dove si possono vedere i commenti degli amici o le proprie domande cliccando su "Domande Personali".

Ogni domanda é visualizzata con il nome dell'autore, un textEdit in sola lettura contenente il testo della domanda e un pulsante "vedi commenti" che se premuto renderá visibili i commenti relativi alla domanda.

Infine il tasto Logout riporta alla vista Login.

Vista amministratore:



Vista visibile dopo aver fatto l'accesso come amminmistratore dalla vista Login.

Da qui é possibile gestire gli utenti presenti nel database.

Per cambiare piano ad un utente scrivere il nome dell'utente di cui si desidera cambiare il piano nella LineEdit sulla destra della Label "cambio piano utente" e scegliere dal menú a tendina a fianco alla Label "nuovo piano" il nuovo piano che si desidera modificare. Premere poi su conferma subito sotto a "nuovo piano".

In basso é presente una tabella da cui si vedono gli utenti presenti nel database con le loro informazioni.

**ATTENZIONE**: una volta apportate le modifiche desiderate verranno salvate solamente premendo il tasto "Salva database".

Premendo su "Esci" si torna alla vista Login.

Il resto della vista é molto intuitivo e ne ometto la descrizione.

**ORE LAVORATIVE E SUDDIVISIONE DEL LAVORO PROGETTUALE**

|  |  |
| --- | --- |
| **Attività:** | **Ore** |
| Scelta tematica del progetto | 1,00 |
| Definizione della gerarchia e delle sue funzionalità | 3,00 |
| Implementazione codice: gerarchia, container, Controller,deepPtr. | 14,00 |
| Apprendimento libreria QT(tutorato e documentazione) | 7,00 |
| Implementazione GUI: vista login,vista profilo,vista utente,vista crea utente,vista domande,widgetCampoDati,foglio di stile | 11,00 |
| Debug/testing | 8,00 |
| Relazione | 5,00 |
| **Totale:** | 49,00 |

Il progetto é stato seguito a grandi linee da ciascun componente del gruppo ma la principale suddivisione del lavoro si basa sull'implementazione della GUI.

Sono state richieste piú ore del previsto nel risolvere alcuni problemi di scrittura del codice.

In ogni caso sono riuscito a rimanere all'interno del monte ore previsto forse per il fatto che c'é stata una leggera suddivisione del lavoro in difetto per quanto mi riguarda (53 ore di lavoro per Lorenzo contro le mie 49). Penso che la differenza di ore impiegate riguardi soprattutto l'implementazione delle funzioni import/export di cui si é occupato principalmente Lorenzo.

Entrambi abbiamo lavorato simultaneamente sull'implementazione della gerarchia, del container e del controller.