UNIVERSITÀ
DEGLI STUDI
DI PADOVA

UNIVERSITY

PROGETTO DI PROGRAMMAZIONE AD OGGETTI

MARCO DELLO IACOVO

2019-2020

Sommario

University è un programma che permette di gestire e visualizzare le informazioni di una lista di utenti dell'università.

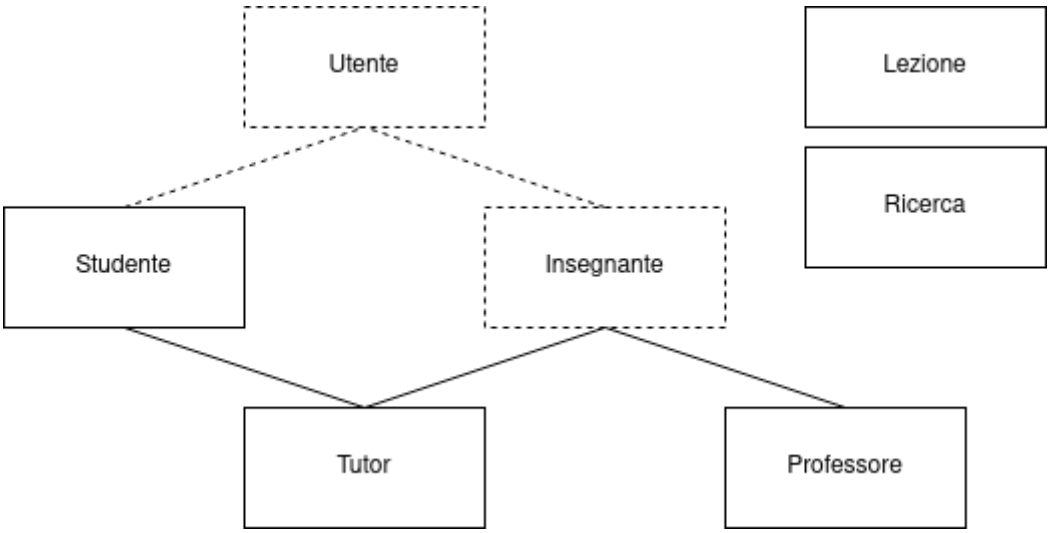
Per la compilazione del progetto viene fornito il file University.pro

Progetto svolto con Matteo Budai.

Il design-pattern utilizzato è Model-View.

Gerarchia

Per rappresentare i vari utenti viene utilizzata la seguente gerarchia:



La classe base astratta è Utente, che permette la memorizzazione dei dati comuni per i vari utenti, come ad esempio le generalità.

Da Utente ereditano le classi Studente e Insegnante. Studente contiene i campi dati specifici di uno studente universitario, come la matricola e il corso di studi.

Insegnante, invece, è una classe astratta che dà la possibilità di memorizzare un elenco di lezioni, con i relativi orari, aule e materie. Per fare questo utilizza un contenitore di classi Lezione.

Tutor rappresenta uno studente che è anche in grado di insegnare ad altri studenti, ma non ha campi aggiuntivi oltre a quelli ereditati dalle due classi Studente e Insegnante.

Professore invece memorizza alcuni dettagli specifici di un professore universitario, come ad esempio il tipo, e contiene anche una lista delle ricerche pubblicate, implementata con un contenitore di classi Ricerca. Ereditando da Insegnante ha anche una lista di lezioni.

Chiamate Polimorfe

Nella gerarchia degli Utenti sono presenti i seguenti metodi virtuali puri:

Utente:

- *virtual utente* clone() const = 0*
  usato come costruttore di copia polimorfo
- *virtual tipoutente getTipoUtente() const =0*
  ritorna una stringa con il tipo di utente ("Professore", "Studente" o "Tutor")

Vengono anche reimplementati i seguenti metodi virtuali di classi Qt:

QAbstractTableModel:

- *int rowCount(const QModelIndex &parent = QModelIndex()) const*
  Ritorna il numero di righe della tabella
- *int columnCount(const QModelIndex &parent = QModelIndex()) const;*
  Ritorna il numero di colonne della tabella
- *QVariant data(const QModelIndex &index, int role = Qt::DisplayRole) const*
  Fornisce alla View il dato presente all'indice index per inserirlo nella cella corrispondente
- *QVariant headerData(int section, Qt::Orientation orientation, int role) const*
  Ritorna, per ogni colonna, il nome del tipo di dati che contiene
- *Qt::ItemFlags flags(const QModelIndex &index) const*
  Ritorna le flags richieste dalla view per ogni indice della tabella
- *bool setData(const QModelIndex &index, const QVariant &value, int role = Qt::EditRole)*
  Imposta all'indice index della tabella il dato value
- *bool insertRows(int position, int rows, const QModelIndex &parent)*
  Inserisce una o più righe vuote nel modello
- *bool removeRows(int row, int count, const QModelIndex &parent)*
  Rimuove una o più righe dal modello

QStyledItemDelegate:

- *QWidget *createEditor(QWidget *parent, const QStyleOptionViewItem &option, const QModelIndex &index) const*
  Crea il widget usato per modificare un indice della tabella
- *void setEditorData(QWidget *editor, const QModelIndex &index) const*
  Aggiorna I dati del widget dopo un cambiamento del modello
- *void updateEditorGeometry(QWidget *editor, const QStyleOptionViewItem &option, const QModelIndex &index) const*
  Aggiorna la geometria del Widget editor in base a options

QDialog:

- *void accept()*
  Se non sono stati inseriti tutti i campi necessari impedisce l'inserimento
- *int exec()*
  Usato per inizializzare i campi nascosti nel QDialog di aggiunta utenti

QSortFilterProxyModel:

- *bool filterAcceptsColumn(int source_column, const QModelIndex &source_parent) const*
  Usato per nascondere le colonne nel filtro per tipo utente
- *bool filterAcceptsRow(int source_row, const QModelIndex &source_parent) const*
  Usato per visualizzare le righe che contengono la stringa nel filtro per campi

Ore richieste

- Analisi preliminare del problema - 2 ore
- Progettazione modello e GUI - 1 ora
- codifica modello – 7 ore
- codifica GUI e Modello Qt, apprendimento libreria Qt – 40 ore
- debugging – 10 ore
- testing – 3 ore
- scrittura relazione – 1 ora

Le ore totali sono 64, cioè 14 in più delle 50 richieste nelle specifiche del progetto, principalmente a causa dello studio del funzionamento della libreria Qt. L'apprendimento della libreria Qt e la codifica della GUI sono state accorpate poiché nuove classi venivano studiate solo quando necessario, e non upfront.

Le classi che hanno richiesto più tempo sono QAbstractTableModel, QAbstractTableView e QSortFilterProxyModel.

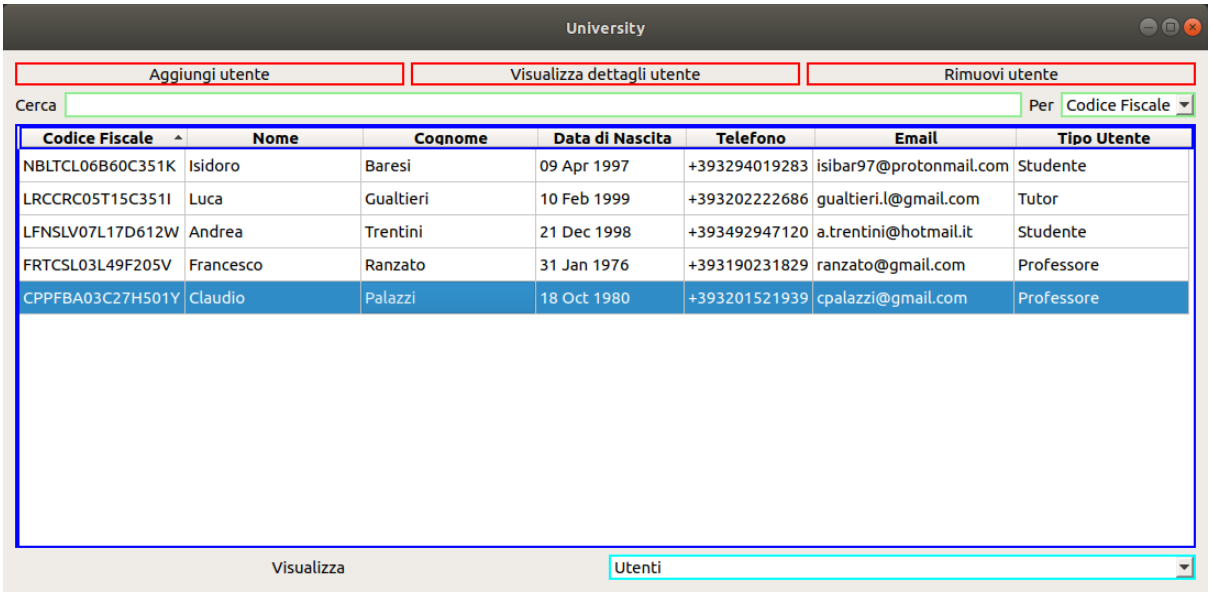
Suddivisione del lavoro progettuale

Marco Dello Iacovo
Matteo Budai

- *Modello*:
  - *Utente*: Marco Dello Iacovo e Matteo Budai
  - *Studente*: Marco Dello Iacovo
  - *Tutor*: Marco Dello Iacovo
  - *Insegnante*: Marco Dello Iacovo
  - *Professore*: Marco Dello Iacovo e Matteo Budai
  - *Lezione*: Marco Dello Iacovo e Matteo Budai
  - *Ricerca*: Marco Dello Iacovo e Matteo Budai
  - *Contenitore*: Marco Dello Iacovo e Matteo Budai
  - *Puntatore Smart*: Marco Dello Iacovo e Matteo Budai

- *View*:
  - *Menu*: Marco Dello Iacovo e Matteo Budai
  - *MenuDatiUtente*: Marco Dello Iacovo e Matteo Budai
  - *MenuLezione*: Matteo Budai
  - *MenuRicerca*: Matteo Budai
- *Model Qt*:
  - *UtentiTableModel*: Marco Dello Iacovo
  - *ProxyModel*: Marco Dello Iacovo
  - *ComboboxDelegate*: Marco Dello Iacovo
  - *LezioniTableModel*: Marco Dello Iacovo e Matteo Budai
  - *RicercheTableModel*: Marco Dello Iacovo e Matteo Budai
- *Gestione eccezioni*: Matteo Budai

Manuale GUI



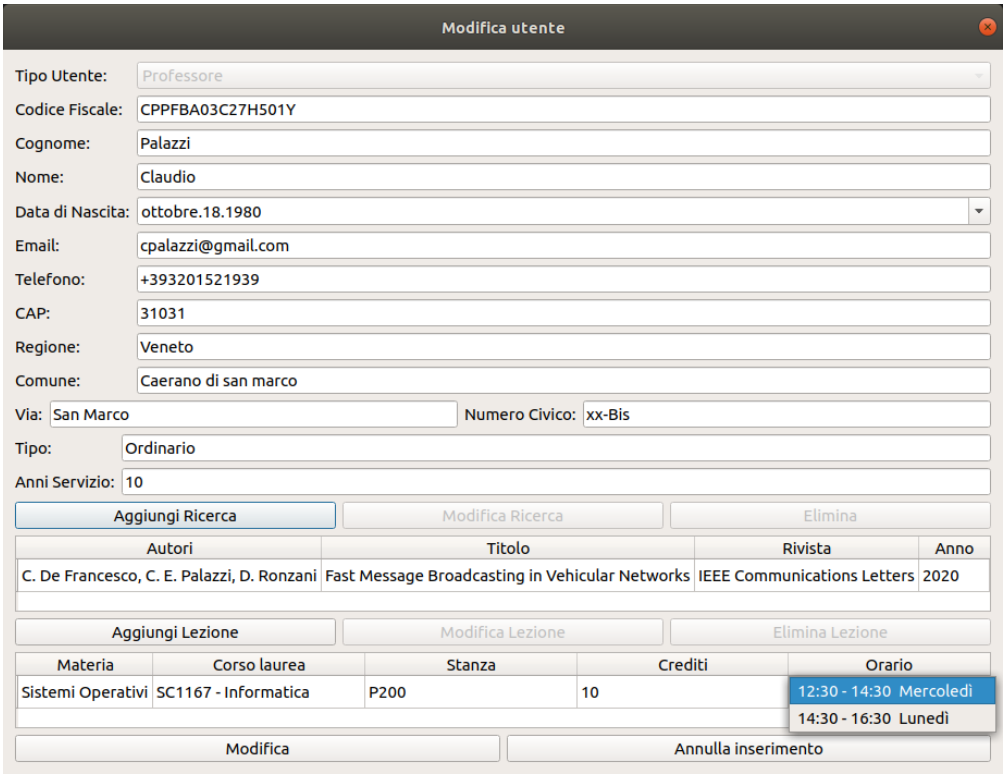
Questo è il menu principale. All'avvio dell'applicazione vengono già inclusi alcuni utenti per poter testare l'applicazione (con dati fittizi).

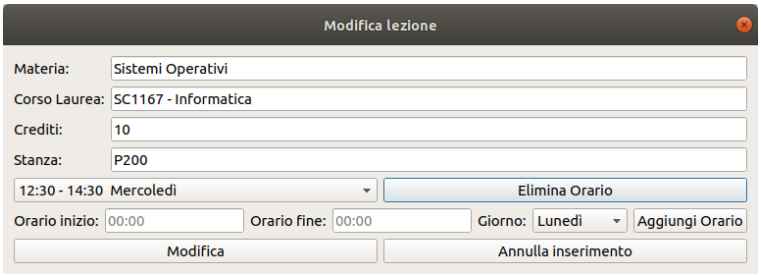
In BLU è visibile la lista degli utenti, con i relativi dati. Ogni colonna è ordinabile alfabeticamente / per data. Per rappresentarla sono state usate le classi TableModelView, QSortFilterProxyModel e QAbstractTableModel.

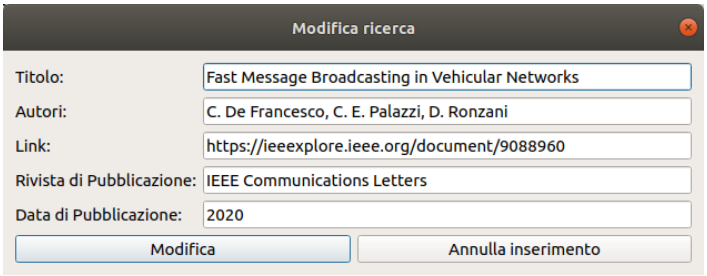
In ROSSO sono presenti i bottoni per l'aggiunta, la modifica e la rimozione degli utenti. I bottoni 'Visualizza dettagli utente' e 'Rimuovi utente' Sono utilizzabili solo se è selezionata una riga della tabella. In caso di selezioni multiple viene visualizzato solo il primo elemento selezionato, mentre vengono rimossi tutti gli elementi selezionati nel caso della rimozione.

In VERDE c'è la sezione di ricerca per colonne. Nella colonna selezionata a destra nella Combobox viene ricercata la stringa inserita a sinistra nella LineEdit.

In AZZURRO è presente la Combobox per il filtro in base al tipo dell'utente. Per ogni tipo di utente vengono visualizzate colonne diverse, riguardanti campi specifici del tipo selezionato. In caso di selezione di tipi astratti vengono visualizzati i tipi derivati istanziabili.







Qui sono mostrate le pagine di visualizzazione e aggiunta dei dati, accessibili tramite i bottoni 'Aggiungi utente' e 'Visualizza dettagli Utente' nella pagina principale.

A seconda del tipo di utente selezionato nella Combobox sono visibili campi diversi. Per rappresentare la lista di lezioni e di ricerche sono presenti due tabelle. Per visualizzare l'elenco delle ore di lezione viene usata una Combobox.

Vengono effettuati dei controlli sui dati inseriti, in modo tale da impedire la creazione di utenti con i campi principali vuoti.

Note

Il progetto è stato sviluppato e testato solamente sulla macchina virtuale di Ubuntu 18.04 64bit fornita dal professore, con versione Qt 5.9.5 e g++ 5.4.0. Su altri sistemi operativi potrebbe non funzionare come previsto.

Abbiamo usato un contenitore basato sulle liste puntate. Per questo progetto sarebbe stato più efficace usare un vettore, a causa dei tempi di accesso alla posizione n di O(1) vs O(n). Abbiamo deciso di usare lo stesso una lista puntata perché è la struttura dati a noi più familiare.