Progetto per il corso di Programmazione ad Oggetti

LT in Informatica, Università degli studi di Padova

QuizRoom

Niccolò Mantovani, 1187325 George Petrea, 1193226

2019/2020

Relazione realizzata da Niccolò Mantovani

Indice

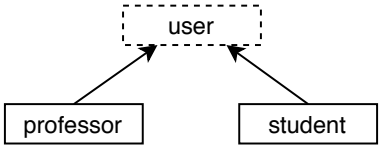
1 Abstract

Il progetto è nato per fornire un aiuto alla didattica a distanza diventata ormai fulcro di discussioni e dibattiti di questi ultimi mesi di pandemia da COVID-19.
QuizRoom propone una piattaforma di quiz che possono essere suddivisi per **corsi**, i quali al loro interno avranno dei **compiti (quiz)** da eseguire. Questi quiz verrano erogati dal professore a tutti gli studenti che sono iscritti al suo corso (infatti uno studente può iscriversi ad un corso tramite un codice d'invito comunicato dal professore stesso).
All'interno dell'applicativo esistono due tipi di utenti: lo **studente** il quale avrà una partecipazione passiva ai quiz, cioè potrà solo svolgerli e visualizzare i risultati. Invece il **professore** potrà creare, modificare ed eliminare corsi, compiti e quiz (quest'ultimi non si potranno modificare perchè sarebbe scorretto nei confronti degli studenti modificare un quiz dopo averlo caricato).
I compiti erogati possono essere **senza valutazione**, con **valutazione**, con **data di scadenza** ed infine con **valutazione e data di scadenza**. Quest'ultimi in fase di valutazione aggiungeranno un bonus o un malus in base a quanto in anticipo/ritardo si ha consegnato il compito rispetto alla data di scadenza.
I quiz invece possono essere di due tipi: **quiz a risposta multipla** (dove ogni risposta errata comporterà ad un malus) e **quiz a combinazione**, cioè dove bisogna abbinare degli elementi con altri (senza malus per una risposta sbagliata).

2 Descrizione delle gerarchie

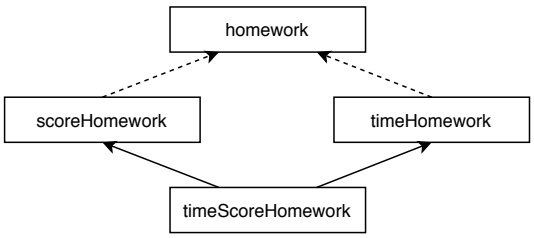
Il progetto presenta nel totale 4 gerarchie nel modello, per rappresentare gli **utenti**, i **compiti**, i **quiz** e infine la **data e l'orario della deadline** di un compito a tempo.
Per quanto riguarda la parte grafica è presente una gerarchia che rappresenta, sostanzialmente, ogni pagina da visualizzare. Inoltre sono presenti altre 2 gerarchie che vanno ad ereditare Widget già ben definiti e strutturati da Qt (QMenu e QMessaggeBox).
L'utilizzo del contenitore C richiesto è stato usato ampliamente su tutto il progetto e non solo nella gerarchia con multiple inheritance. Il contenitore scelto è un **MyVector** perchè garantisce un accesso casuale agli elementi al suo interno. Questo è utile quando un utente vuole accedere ad un determinato corso/compito perchè, appunto, non si devono scorrere tutti i corsi/compiti.

2.1 Gerarchia utenti



La gerarchia consiste in una classe base astratta **user**, resa astratta dal metodo di clonazione e da alcune classi che identificano i permessi di un utente. Ogni utente è caratterizzato da *username, password e un contenitore di corsi*. Le classi che ereditano da user sono **professor** e **student**, che non aggiungono nessun campo dati ma che implementano i metodi virtuali dei permessi. I permessi sono rappresentati attraverso metodi virtuali perchè garantiscono un ulteriore estensibilità del codice e permettono di non sprecare memoria in quanto non sono rappresentanti tramite campi dati.

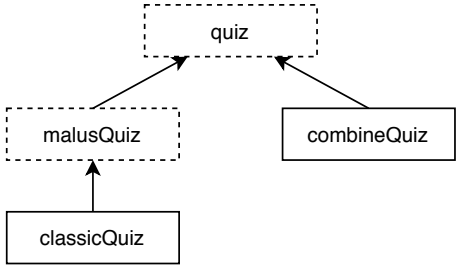
2.2 Gerarchia compiti



La gerarchia rappresenta il tipo di compito che può essere assegnato da un professore agli iscritti del suo corso. I compiti possono essere svolti molteplici volte.

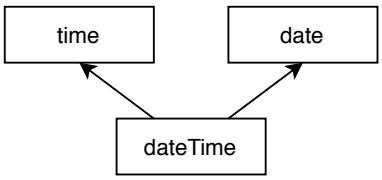
La classe principale è **homework**, che rappresenta un compito descritto da *titolo, descrizione e da un contenitore di quiz*. La classe homework avrà dei metodi virtuali che verrano poi carratterizzati in modo diverso dalle classi figlie.
scoreHomework rappresenta un compito con un punteggio, infatti avrà un *risultato* dipendete dal punteggio effettuato da tutti i quiz all'interno del contenitore.
timeHomework rappresenta un compito con una scadenza, perciò avrà un campo dati *deadline* che mi rappresenta la data limite per svolgere il compito. Inoltre il risultato sarà il numero di giorni di anticipo/ritardo nel quale si è svolto il compito.
timeScoreHomework invece è una multiple inheritance che rappresenta un compito con punteggio e con data di scadenza. Il risultato del compito verrà calcolato in base al punteggio del quiz e verrà applicato un malus se il compito è stato consegnato in ritardo rispetto alla data di scadenza. timeScoreHomework ha un campo dati statico che rappresenta il *numero di giorni di ritardo* per determinare il peso del malus da assegnare.

2.3 Gerarchia quiz



Questa gerarchia descrive il funzionamento dei vari quiz.
quiz è la classe base astratta che è caratterizzata da *domanda del quiz, i punti delle domande corrette, i punti fatti e total_point*, cioè la variabile statica che indica i punti totali del quiz singolo. Quiz avrà al suo interno alcuni metodi virtuali che identificano la caratterizzazione dei quiz, come la presenza o meno di un malus, il calcolo del punteggio del quiz, il metodo che permette di calcolare il punteggio per ogni risposta esatta e i metodi che permettono di mostrare le soluzioni.
malusQuiz è anch'essa una classe astratta ma che va a definire un particolare quiz con un malus sulle risposte sbagliate. Al suo interno avrà due campi dati: *total_malus e malus_point*. Il primo identifica la somma totale del malus raggiunto, il secondo invece identifica il peso del malus per ogni risposta sbagliata.
classicQuiz identifica un quiz a risposta multipla. E' caratterizzato da due vettori, uno contenente *tutte le risposte*, e uno contentente solo le *risposte corrette*. Ovviamente il punteggio del quiz viene calcolato in base al punteggio delle risposte correte più un eventuale malus se è stata data una risposta sbagliata (per evitare si possa inserire tutte le risposte, e quindi indovinare quella corretta)
combineQuiz rappresenta un quiz a combinazione, dove l'utente deve abbinare una parola alla corrispettiva controparte. L'unico campo dati di combineQuiz è una mappa che rappresenta le *combinazioni giuste*, come chiave appunto avrà la parola da abbinare e come valore l'elemento della combinazione corretta. Il punteggio del quiz viene calcolato in base alle combinazioni corrette, senza nessun malus.

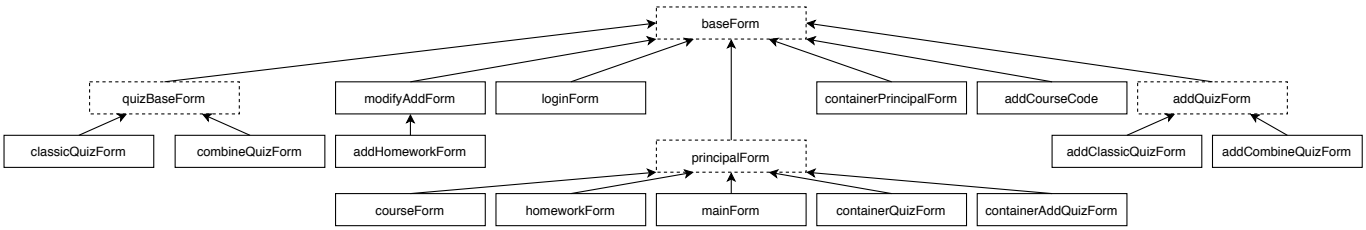
2.4 Gerarchia data e ora



time rappresenta un orario caratterizzata dai *minuti*, e nella visualizzazione dell'orario quest'ultimi vengono convertiti nel formato *ora:minuti*.
date rappresenta una data caratterizzata da *giorno, mese ed anno*. Inoltre ha il metodo per controllare se un anno è bisestile e per controllare il numero di giorni passati dalla data attuale a quella rappresentata dall'istanza della classe date.
dateTime è multiple inheritance che rappresenta una data unita ad un orario. Serve per rappresentare la

deadline del **timeHomework**.
Queste tre classi, inoltre, avranno definito alcuni *metodi di confronto* ($>, <, !=, ==$).

2.5 Gerarchia pagine della vista



Le pagine della view avranno una loro e proprio gerarchia, tutte derivate da QWidget. Qui verrano descritte le più importanti.
Alla base della gerarchia è presente la classe **baseForm**, resa virtuale dai metodi *addForm()* (aggiunta dei widget) e *setStyle()* (impostazioni dello stile della view).
principalForm descrive le pagine principali della vista. Come metodi astratti, oltre ad avere quelli di baseForm, avrà anche *addMenu()*, cioè il metodo che aggiunge una *MenuBar*.
Inoltre sono presenti altre due classi non inserite nella figura perchè facenti parte di un'altra gerarchia: **menuButton** derivata da **QMenu**, che si occupa di mostrare il menu a tendina del bottone per l'eliminazione dei corsi/compiti, e **ErrorMessage** derivata da **QMesaggeBox**, che si occupa di mostrare un messaggio all'utente quando si imbatte in alcuni errori logici dell'aplicazione.

3 Descrizione del codice polimorfo

3.1 Polimorfismo gerarchia utenti

Come metodi polimorfi principali abbiamo il *distruttore*, il quale viene usato per richiamare il distruttore corretto per quando viene distrutto un puntatore ad user. Inoltre abbiamo il metodo *clone()* che serve per creare una copia dell'oggetto.
Infine l'elenco dei metodi super-polimorfi che descrivono i permessi di un user: *CanAddCourse(), CanDeleteCourse(), CanEditCourse(), CanAddHomework(), CanDeleteHomework(), CanEditHomework(), CanAddQuiz(), CanDeleteQuiz()* e *CanEditQuiz()*. Questi metodi sono utilizzati nella view per mostrare o nascondere bottoni con cui l'utente può interargire per modificare, cancellare o aggiungere elementi.

3.2 Polimorfismo gerarchia compiti

Nella gerachia dei compiti è sempre presente il *distruttore* polimorfo e il metodo *clone()*, come abbiamo visto nella gerachia precedente.
Un altro metodo polimorfo è *getResult()*, che ritorna il risultato del compito e che varia in base alla caratterizzazione del compito: voto in base alle risposte dei quiz se il compito è **scoreHomework**, numero di giorni di consegna in anticipo/ritardo se il compito è **timeHomework** e voto in base alle risposte dei quiz più l'aggiunta di un malus in base a quanto in ritardo si è consegnato il compito se esso è **timeScoreHomework**.
Altri metodi polimorfi sono: *isScoreHomework(), isTimeHomework() e haveResult()* che mi vanno a descrivere la caratterizzazione del compito.

3.3 Polimorfismo gerarchia quiz

Come le precedenti gerarchie, è presente il *distruttore* polimorfo e il metodo *clone()*. In aggiunta avrà il metodo polimorfo *getMyPoint()()* che ritorna il totale del punteggio ottenuto sul quiz (polimorfo perchè in **malusQuiz** tiene presente del malus).
Oltre i metodi citati sopra, sono presenti altri metodi super-polimorfi:

- *setPointCAnswer()* che imposta il punteggio di ogni domanda in base al numero di domande corrette contenute e al punteggio totale.
- *CalcPointQuiz()* che calcola il punteggio del quiz.

- *clear_all_answers()* e *resetPoint()*, dove il primo serve per eliminare tutte le risposte (corrette e non corrette) e il secondo serve per fare il reset del punteggio ottenuto (ed eventualmente del malus).
- *showSolution()* e *solutionToString()*, dove nel primo viengono mostrate le soluzioni del quiz a video, invece nel secondo le soluzioni del quiz vengono convertite in una stringa.
- *HaveMalus()* che va a definire se il quiz può avere il malus oppure no
- infine *CalcMalus()*, il quale è presente in **malusQuiz**. Esso va a definire come viene calcolato il malus da aggiungere per una risposta sbagliata di **classicQuiz**.

3.4 Polimorfismo gerarchia pagine della vista

Come nelle gerarchie che abbiamo appena descrito, anche qui è presente il *distruttore* polimorfo e anche un metodo *clone()* in **principalForm** (implementato in modo diverso perchè Qt non permette la costruzione di copia dei Q_OBJECT).
La gerarchia delle pagine contiene 2 metodi polimorfi principali: *addForm()* e *setStyle()*. Il primo è un metodo super-polimorfo che serve per aggiungere gli elementi che vanno a caratterizzare una pagina specifica. Il secondo invece è un metodo che serve per settare le impostazione di stile di una pagina.
Un altro metodo super-polimorfo che si trova in **principalForm** è *addMenu()*, che serve per settare una menu bar (considerata essenziale nelle pagine principali dell'applicazione).
Infine l'ultimo metodo super-polimorfo è *randomize_answer()* in **quizBaseForm**, che serve per disporre in modo casuale le risposte dei vari quiz.
Nella gerarchia della vista, inoltre, sono presenti alcuni **slots** virtuali. Nella gerarchia che parte da **principalForm** sono presenti *confirmAddForm(), to_next_page(), to_previous_page(), to_update_previous_page()*, che servono per capire quale pagina sarà la successiva/precedente rispetto a quella in cui ci si trova, ed eventualmente aggiornarla.
quizBaseForm ha uno slot super-polimorfo, chiamato *getAnswers()*, che serve per ricevere le risposte inserite dall'utente all'interno dell'applicativo.
Infine anche **addQuizForm** avrà un metodo super-polimorfo, cioè *setInformation()*, che serve per inviare i dati del quiz al controller.

4 Descrizione file CSS

L'applicativo utilizza il tipo di file **CSS** per settare lo stile delle varie pagine della vista. I file CSS sono all'interno della cartella *Resources/Style_sheet* e vanno a definire alcune caratterisriche dei QWidget presenti nella pagine, come *color, background, font, margin, padding...*
Questi file vengono principalmente letti (input) all'interno del metodo *setStyle()* delle varie classi della vista.

5 Manuale utente

La compilazione avviene tramite questi comandi: **qmake Progetto_P2.pro** ⇒ **make** ed infine eseguito tramite il comando **./Progetto_P2**.

L'applicazione, all'avvio, precaricherà 3 profili diversi di utenti:

- username: **professor**, password: **professor**.
- username: **student1**, password: **student1**.
- username: **student2**, password: **student2**.

Saranno presenti anche corsi e compiti già precaricati in ogni profilo, più precisamente il professore avrà 4 corsi. Invece i due studenti avranno due corsi dei 4 citati prima ciascuno.
Il professore può apportare modifiche ai corsi, compiti e quiz (cancellare, aggiungere o modificare). Ovviamente le modifiche, essendo i dati precaricati, una volta chiusa l'applicazione verrano perse (si è deciso di non aggiungere la possibilità di salvare i dati su dei file perchè era stato già superato il monte ore concesso).
Gli studenti invece potranno solamente iscriversi a nuovi corsi (creati da un professore) e svolgere i vari compiti.

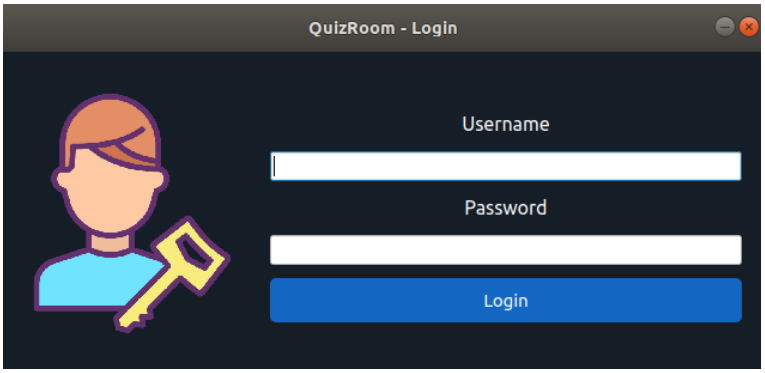


Figura 1: pagina di login

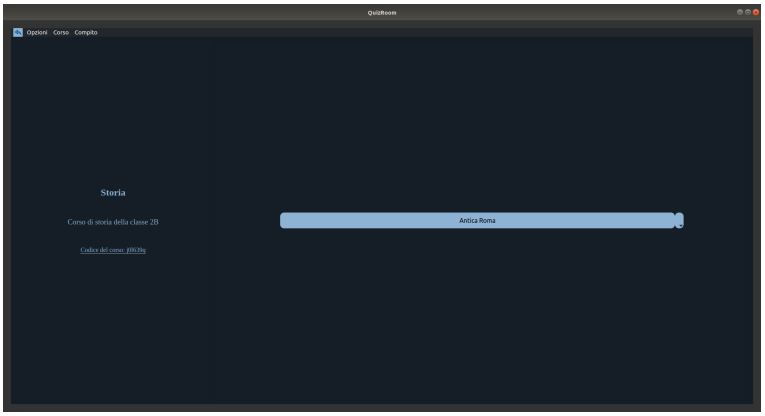


Figura 2: pagina di un corso

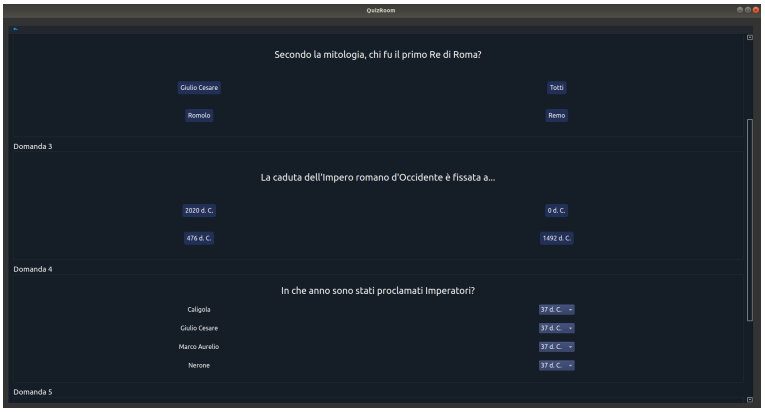


Figura 3: pagina di un quiz

6 Ore richieste

- Analisi preliminare del problema: 1h
- progettazione modello: 3h
- progettazione grafica: 3h
- codifica del modello e controlller: 14h
- codifica della GUI (compreso apprendimento libreria Qt): 31h
- debugging (compreso analisi di memory leak): 3h
- testing: 1h

Il tempo per la realizzazione del progetto è circa di 56h, dovute soprattuto all'apprendimento della libreria Qt e a problemi riscontrati durante la codifica del progetto.

7 Suddivisione del lavoro progettuale

L'analisi preliminare e la progettazione del modello e della grafica è stata fatta sostanzialmente in collaborazione. Ovviamente durante l'avanzamento del progetto, su queste parti, sono state fatte delle modifiche decise singolarmente.

La mia parte del lavoro comprende principalmente:

- Model:
  - Scelta e codifica del contenitore **MyVector**
  - Creazione e codifica della gerachia **user**
  - Creazione e codifica della gerarchia **quiz**
  - Creazione e codifica di alcuni metodi della classe **course**
  - Implementazione del metodo polimorfo *getResult()* di **timeHomework** e **timeScoreHomework**
- Gestione delle exception
- GUI:
  - Ideazione della gerarchia **principalForm**
  - Creazione di **containerQuizForm** e **containerAddQuizForm**
  - Ideazione e creazione della gerarchia **quizBaseForm** e **addQuizForm**
  - Creazione e implementazione della classe **errorMessage**
  - Creazione e implementazione della classe **loginForm**
  - Scelta di alcune immagini da inserire nell'applicazione
- Controller:
  - Ideazione e implementazione dello stack delle pagine principali da visualizzare.
  - Implementazione della gestione di accesso del controller ai quiz.
  - Implementazione della gestione di accesso del controller all'user.

8 Ambiente di sviluppo

- **Sistema operativo:** Windows 10 Home 64 bit/ Virtual Machine Linux Ubuntu.
- **Compilatore:** gcc 7.3.0 (MinGW 7.3.0 64-bit).
- **Qt:** 5.12.5
- **qmake:** 4