Progetto del corso Programmazione ad Oggetti

Relazione di George Petrea

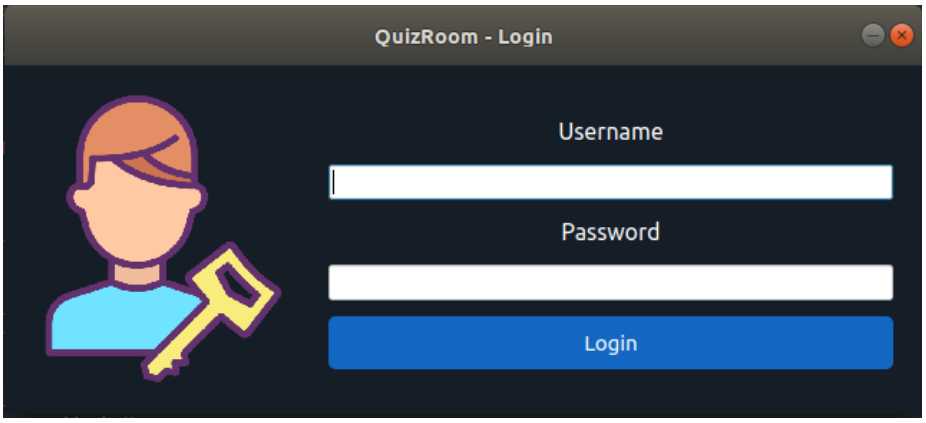
Laurea Triennale in Informatica, A.A. 2019/2020

QuizRoom

Progetto realizzato da **George Petrea** (n° matricola 1193226) e **Niccolò Mantovani** (n° matricola 1187325)

1) Scopo del progetto

Questo progetto nasce per dare un ausilio a rendere la didattica più digitalizzata, tenendo anche conto dell'emergenza COVID-19 con cui abbiamo avuto tutti a che fare in questi mesi. Questo viene fatto proponendo una piattaforma in cui professori e studenti possono interagire attraverso dei **quiz**, contenuti nel corso di una materia, siano essi semplici esercitazioni o veri e propri compiti valutati.
Ogni professore può creare il suo corso a cui gli studenti potranno iscriversi attraverso un **codice corso** che verrà fornito dal professore stesso.
In QuizRoom si avranno due categorie di utenti: il **professore** che può creare, modificare ed eliminare corsi da cui somministrare quiz che egli stesso può creare od eliminare, e lo **studente** che potrà iscriversi ai corsi e svolgere i quiz vedendo alla fine i suoi risultati.
I compiti somministrati possono essere di vario tipo: senza valutazione, con valutazione, con data di scadenza e insieme con valutazione e data di scadenza. Inoltre vi sono a disposizione 2 tipologie di quiz: **classico** e a **combinazione**.



Schermata di login

2) Manuale di utilizzo per l'utente

Questo progetto si compila con "qmake Progetto.pro" e in seguito "make". Il file .pro è incluso nella cartella compressa del progetto.

All'apertura dell'applicazione all'utente verrà chiesto di effettuare il login. Le credenziali per il tipo di utente *professore* sono:

- username: **professor**;
- password: **professor**.

Il professore troverà 4 corsi: italiano, storia, matematica e scienze. Questi avranno già a disposizione alcuni esempi di compiti e quiz che il docente potrà liberamente modificare, eliminare o se desidera aggiungerne di nuovi.

Inoltre vi sono due profili di tipo *studente*:

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

Tali studenti saranno iscritti rispettivamente a italiano e storia il primo, e matematica e scienze il secondo. Gli studenti possono solo svolgere i compiti (vedendone i risultati) o iscriversi a un corso mediante l'apposito *codice corso.*

Le modifiche effettuate vengono perse una volta chiusa l'applicazione, in quanto i dati presenti sono precaricati.

3) **Ambiente di sviluppo**

Anche se il sistema operativo di base è Windows 10 Pro a 64 bit, il progetto nella sua interezza è stato creato e testato sulla **Virtual Machine Linux** fornita dal docente, con **Qt 5.9.5** e **compilatore GCC 7.3.0.**

4) **Ore richieste**

- Analisi preliminare del problema: 1h;
- Progettazione modello: 4h;
- Progettazione grafica: 5h;
- Codifica modello e controller: 9h;
- Codifica della GUI (apprendimento libreria Qt compreso): 28h;
- Debugging: 4h;
- Testing: 2h.

Il tempo impiegato per realizzare questo progetto è di circa 53 ore rispetto alle 50 indicate, il surplus è prevalentemente dovuto alla familiarizzazione con la libreria Qt.

5) Suddivisione del lavoro

Il progetto è stato ideato e realizzato in forte collaborazione tra i due membri del gruppo. Per quanto riguarda la mia parte di lavoro, essa si può dividere in:

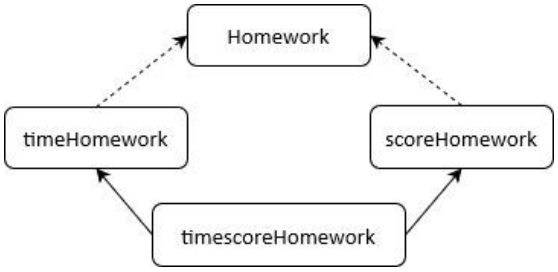
- Model:
  - codifica della gerarchia **datetime**;
  - codifica di alcuni metodi della classe **course**;
  - creazione e codifica delle classi **homework** e **scorehomework**
- GUI:
  - creazione e codifica di **mainform**;
  - creazione e codifica di **menubutton**;
  - creazione e codifica di **courseform, addcoursecode, homeworkform** e **addhomeworkform**;
  - creazione e implementazione delle classi riguardanti i quiz, come **classicquizform**, **combinequizform**, **addclassicquizform** e **addcombinequizform;**
  - creazione file **CSS** per la cura della grafica delle varie pagine;
  - scelta di alcune immagini presenti.
- Controller:
  - implementazione dell'accesso ai **corsi** (creazione, modifica, eliminazione).

6) Gerarchie

Nel progetto vi sono in totale **4** gerarchie:

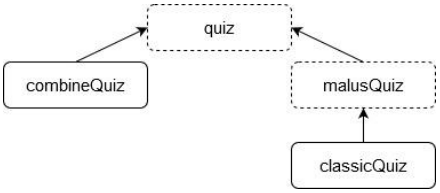
- **compiti**,
- i **quiz**,
- **utenti,**
- **data e ora** della deadline (nel caso di compiti a tempo)

6.1) Gerarchia compiti



Questa gerarchia riguarda le tipologie di compiti che un professore può assegnare, i quali possono essere svolti più volte dallo studente.
homework è la classe principale di questa gerarchia, e raffigura un assignment composto da *titolo, descrizione del compito* e un *contenitore di quiz.* All'interno vi sono dei metodi virtuali usati dalle classi che ereditano da *homework.*
timeHomework rappresenta un compito con *scadenza (deadline)*, il risultato segnerà quanto in anticipo o in ritardo un compito è stato consegnato.
scoreHomework rappresenta un compito con valutazione, il cui voto finale dipende dall'insieme di punti ottenuti nelle varie domande.
timeScoreHomework è una *gerarchia a ereditarietà multipla* delle ultime due classi, in cui il voto finale sarà dato dai punti ottenuti mediante i quiz, più un bonus o penalizzazione stabilito da quando è stato consegnato il compito, sia esso in anticipo o in ritardo.
E' presente un **contenitore MyVector** che, grazie alla possibilità di un accesso casuale agli elementi al suo interno, permette ad esempio di accedere direttamente ad un determinato corso o quiz, invece di dover scorrere tra tutti quelli presenti.

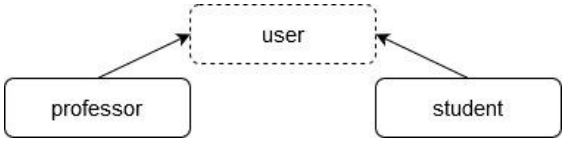
6.2) Gerarchia dei quiz



quiz è la classe astratta base di questa gerarchia. Essa è formata da *domanda, punti delle domande corrette, punti ottenuti*. Vi è inoltre una variabile statica che si occupa di indicare il punteggio di un quiz, chiamata "*total_point*" Questa classe contiene alcuni metodi virtuali utili a determinate tipologie di quiz, ad esempio se ci si trova in presenza di un quiz in cui è presente un malus.
classicQuiz rappresenta un semplice quiz a risposta multipla. Questa classe è composta da due campi dato, il primo chiamato "*answer*" di tipo vettore che contiene *tutte le risposte* e il secondo "*correct_answer*" sempre di tipo vettore che contiene solo quelle *corrette.* Nel calcolo del punteggio si tiene conto di tale malus, altrimenti selezionando tutte le risposte a una domanda si finirebbe per scegliere comunque quella giusta.
malusQuiz è un'altra classe astratta che definisce un tipo particolare di quiz in cui si ha una penalizzazione sulle risposte sbagliate di una domanda. All'interno troviamo due campi dati: *total_malus* che rappresenta la somma dei punti di penalità raggiunta e *malus_point* che indica il peso di tale malus.

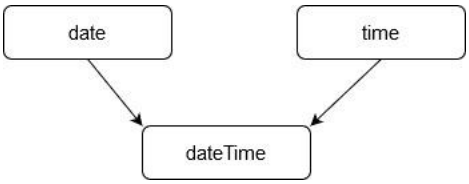
combineQuiz è la classe che definisce un quiz a combinazione, ossia in cui bisogna associare un'opzione alla sua controparte corretta. L'unico campo dati è una mappa che rappresenta le combinazioni giuste.

6.3) Gerarchia degli utenti



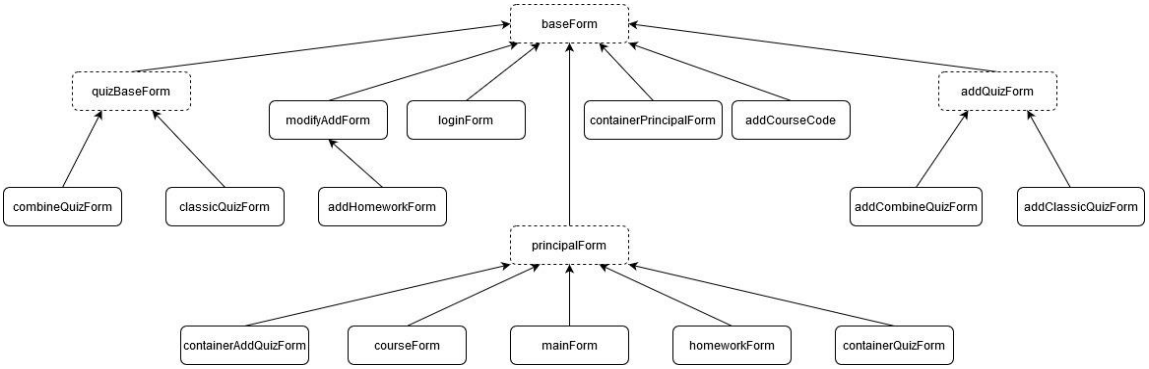
user è la classe base virtuale pura (resa tale dai metodi di clonazione e da altre classi che identificano i permessi), composta da *username*, *password* e un *contenitore di corsi.* Le classi **professor** e **student** ereditano da user e implementano i metodi virtuali dei permessi.

6.4) Gerarchia data e ora



dateTime è una *multiple inheritance* che rappresenta sia una data sia un orario. Questa classe viene usata nel definire la **scadenza** di un quiz, nel caso si stia parlando di un *timeHomework* o *scoreTimeHomework.*
date rappresenta una data nel formato *gg:mm:aaaa.*
time rappresenta un orario definito dal campo dati *minuti.* Nel visualizzare poi questo orario i minuti vengono convertiti nel formato *hh:mm.* In tutte le classi di questa gerarchia vengono definiti dei metodi di confronto per poter verificare se una consegna ha effettivamente rispettato o meno una deadline.

6.5) Gerarchia pagine della vista



La classe **baseForm** è alla base di questa gerarchia. Essa è resa virtuale dai metodi *setStyle() e addForm()* che si occupano rispettivamente di impostare lo stile delle pagine e dell'aggiunta di widget alle stesse. **principalForm** è la classe che descrive le pagine principali che troviamo nella vista. Essa contiene gli stessi metodi di baseForm, oltre all'aggiunta di un metodo *addMenu()* usato per aggiungere una *MenuBar.*
Troviamo poi **ErrorMessage**, derivata da **QMessageBox**, che mostra messaggi all'utente in caso di errori, ad esempio uno sbagliato inserimento delle informazioni di login.
La classe **menuButton** viene usata per raffigurare il menu a tendina del bottone per l'eliminazione dei corsi/compiti ed è derivata da **QMenu.**

7) Descrizione del codice polimorfo

7.1) Polimorfismo gerarchia utenti

Come metodi polimorfi principali vi è un **distruttore** (che viene usato per richiamare il giusto distruttore ogni volta che un puntatore ad user viene distrutto) e un metodo **clone()** per creare una copia di un oggetto. Questi due metodi sono presenti in tutte le gerarchie.
Vi sono poi dei metodi super-polimorfi utili a stabilire i permessi di ogni utente, che nella view si traducono in bottoni che l'utente può vedere o meno e di conseguenza con cui può interagire.
Ad esempio per i corsi troviamo **CanAddCourse(), CanDeleteCourse(), CanEditCourse();**
per i compiti **CanAddHomework(), CanDeleteHomework(), CanEditHomework()**
e per i quiz **CanAddQuiz(), CanDeleteQuiz() e CanEditQuiz().**

7.2) Polimorfismo gerarchia compiti

Troviamo il metodo polimorfo **getResult()** che serve a restituire il risultato di un compito in base alle varie tipologie di quest'ultimo. Altri metodi polimorfi che descrivono la tipologia di compito sono:

- **isScoreHomework();**
- **isTimeHomework();**
- **haveResult()**.

7.3) Polimorfismo gerarchia quiz

Qui è presente il metodo polimorfo **getMyPoint(),** il quale è polimorfo perché tiene conto del malus nella tipologia di quiz che lo prevede, che ritorna il totale dei punti ottenuti su un quiz. I metodi super-polimorfi che troviamo in questa gerarchia sono: **setPointCAnswer(), CalcPointQuiz(), clear_all_answers(), resetPoint(), showSolution(), solutionToString(), HaveMalus()** e **CalcMalus(**).

7.4) Polimorfismo gerarchia pagine della vista

In **principalFrom** troviamo il *distruttore polimorfo* e il metodo *clone().* Questi due metodi sono stati implementati diversamente qui a causa di alcune restrizioni di Qt. I due metodi principali sono **setStyle()** e **addForm()**, che anche qui servono a impostare lo stile di una pagina e all'aggiunta di widget a essa. In *quizBaseForm* troviamo il metodo **randomize_answer()**, che serve a mettere in ordine casuale le risposte di una domanda.
addMenu(), presente in *principalForm*, serve per l'utilizzo di una barra del menù che verrà ampiamente usata nella maggior parte delle pagine dell'applicativo.
Vi sono inoltre vari *slot virtuali*: **confirmAddForm(), to_next_page(), to_previous_page(), to_update_previous_page()** all'interno di *prinicpalForm;* **getAnswers()** in *quizBaseForm* e infine **setInformation()** in *addQuizForm.*

8) Comparto visivo e file CSS

Per rendere la GUI più dettagliata e rifinita, vengono usati dei file di tipo *CSS* che vengono letti in input dal metodo *setStyle()* e servono a definire le **colorazioni**, **bordi** e qualsiasi altro piccolo aspetto grafico da curare ad esempio il **feedback** ricevuto al tocco di un pulsante ecc. Tali file CSS sono collocati in *Resources/Style_sheet.*