UNIVERSITÁ DEGLI STUDI DI PADOVA

LAUREA IN INFORMATICA

A.A. 2016-2017

CORSO DI PROGRAMMAZIONE AD OGGETTI

QPEDIA

MASSIGNAN FABIO

MAT: 1070541

20/06/17

Ambiente di sviluppo:

- Sistema operativo: Windows 10
- Compilatore: GCC 4.9.2
- Libreria QT: QT Creator 4.2.1

Compilazione

- *Qmake QPedia.pro*
- *Make*
- *./QPedia*

Esempio utente corso: *username*: P2 password: psw
Esempio utente studente: *username*: fmassign *password*: fmassign
Per ulteriori dati di accesso consultare il file userDB.xml.

Scopo del progetto:

Il progetto ha lo scopo di implementare un sistema per la gestione di appunti scolastici: un docente ha la possibilità di iscrivere il proprio corso nel sistema, inserendo nel proprio spazio delle note inerenti al corso. Uno studente può accedere al sistema con la possibilità di iscriversi ai corsi disponibili, e di guardare le note pubblicate dal docente nel proprio corso.
Le note possono essere di due tipi: definizione e frammento di codice.
Una definizione è caratterizzata da un titolo e da una descrizione.
Un frammento di codice è caratterizzato da un titolo, dal linguaggio con cui è scritto, e dal corpo del codice.
Per ogni corso si può creare un qualsiasi numero e tipo di note, modificandole in ogni momento, potendole integrare, o eliminare. Vi è anche la possibilità di segnare delle note come preferite, per assegnare una maggiore visibilità ed importanza per il corso e per gli studenti che si iscrivono al corso.
Ogni studente ha la possibilità di iscriversi ad un corso e disiscriversi in qualunque momento, visionandone le note, ma non modificandole o eliminandole.

Funzionalità per l'utente corso

- Inserire una nuova definizione;
- Inserire un nuovo frammento di codice sorgente;
- Modificare una definizione esistente;
- Modificare un frammento di codice sorgente;
- Eliminare una definizione esistente;
- Eliminare frammento di codice esistente;
- Marchiare come preferita nota;
- Visualizzare tutte le note create;
- Visualizzare solamente le definizioni;
- Visualizzare solamente i frammenti di codice sorgente;
- Visualizzare le note preferite;
- Ordinare in ordine alfabetico le note visualizzate;

- Modificare la propria password;

Funzionalità per l'utente studente

- Visualizzare la lista di corsi disponibili;
- Iscriversi ai corsi desiderati;
- Cancellare l'iscrizione ai corsi;
- Visualizzare tutte le note dei corsi a cui si è iscritto;
- Visualizzare tutte le definizioni dei corsi a cui si è iscritto;
- Visualizzare tutti i frammenti di codice dei corsi a cui si è iscritto;
- Visualizzare tutte le note segnate come preferite dai corsi;
- Ordinare in ordine alfabetico le note visualizzate;
- Modificare la propria password;

Progettazione

Per la realizzazione del progetto è stato utilizzato il pattern architetturale MVC.

La componente logica (Model) è composta dalle classi:

- ***dbOnXML***: si occupa si salvare su file i dati degli utenti e le note inserite;
- ***note***: classe base virtuale pura per le note;
- ***definition***: specifica il tipo di nota Definizione;
- **source**: specifica il tipo di nota Frammento di codice;
- **user**: classe base virtuale pura per gli utenti;
- **usrCourse**: specifica il tipo di utente Corso;
- **usrStudent**: specifica il tipo di utente Studente.

La componente grafica (View) è composta dalle classi:

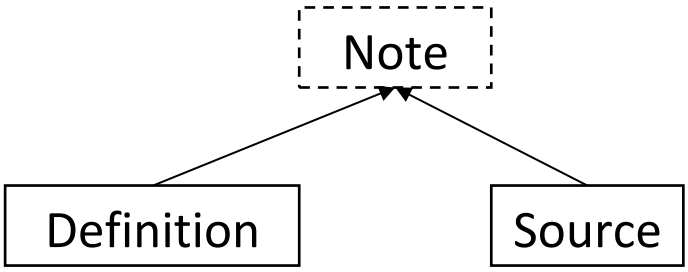
- **viewCourseList**: visualizza la lista di corsi disponibili all'iscrizione;
- **viewDefinition**: visualizza i dati della nota Definizione;
- **viewLogin**: visualizza il form di login;
- **viewReg**: visualizza il form di registrazione;
- **viewSource**: visualizza i dati della nota Frammento di codice;
- **viewUserCourse**: visualizza le funzionalità dell'utente Corso;
- **viewUserData**: visualizza il form per il cambio password;
- **viewUserStudent**: visualizza le funzionalità dell'utente Studente.

La componente che effettua il ponte tra Model e View, (Controller) è composta dalle classi:

- **controllerLogin**: effettua i controlli per il corretto login degli utenti Studente o Corso;
- **controllerUsrCourse**: esegue le funzionalità dell'utente Corso;
- **controllerUsrStudent**: esegue le funzionalità dell'utente Studente.

Gerarchie e classi

Gerarchia Note



- **note**: è la classe base virtuale pura, che funziona da interfaccia comune per tutti i tipi di note.
  Ogni nota è caratterizzata da:
  - unsigned int id: codice univoco per identificare la nota;
  - QString title: titolo della nota;
  - QDateTime creationDate: data di creazione della nota;
  - QDateTime modifyDate: data dell'ultima modifica della nota;
  - bool favourite: indica se la nota è contrassegnata come preferita;
  - QString parent: indica il nome dell'utente che ha scritto la nota.

  e fornisce i seguenti metodi:
  - note(unsigned int=1, QString ="noTitolo", QDateTime=QDateTime::currentDateTime(), QDateTime=QDateTime::currentDateTime(), bool=false, QString = "noParent"): costruttore unico con valori di default per ogni campo;
  - virtual ~*note*(): distruttore virtuale;
  - unsigned int getId() const: ritorna l'id;
  - QString getTitle() const: ritorna il titolo;
  - QDateTime getCreationDate() const: ritorna la data di creazione;
  - QDateTime getModifyDate() const: ritorna la data di ultima modifica;
  - QString getParent() const: ritorna il nome dell'utente che ha scritto la nota;
  - bool isFavourite() const: indica se la nota è segnata come preferita;
  - void setId(const unsigned int&): setta l'id;
  - void setTitle(const QString&): setta il titolo della nota;
  - void setCreationDate(const QDateTime&): setta la data di creazione della nota;
  - void setModifyDate(const QDateTime&): setta la data di ultima modifica della nota;
  - void setFavourite(const bool &): inserisce o toglie il preferito dalla nota;
  - virtual void *saveNote*(QXmlStreamWriter&) const =0: metodo virtuale puro che permette il salvataggio della nota in un database xml;
- **definition**: è una classe concreta, derivata da note, caratterizzata da:
  - QString description: campo dati che contiene la descrizione della definizione;

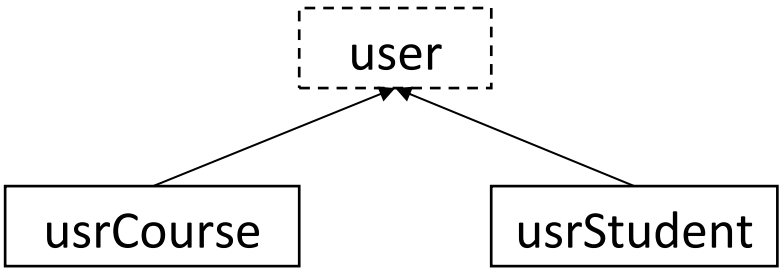
  e fornisce i seguenti metodi:
  - definition(unsigned int=1, QString ="noTitolo", QDateTime=QDateTime::currentDateTime(), QDateTime=QDateTime::currentDateTime(), bool=false,QString = "noParent", QString ="noDescrizione"): costruttore unico con valori di default per ogni campo;
  - QString getDescription() const: ritorna la descrizione della nota;
  - void setDescription(const QString&): setta il valore descrizione;

- void saveNote(QXmlStreamWriter&) const: implementa il metodo virtuale puro che permette il salvataggio della definizione in un database xml.

- **source**: è una classe concreta derivata da note caratterizzata da:
  - QString language: campo dati che definisce il tipo di linguaggio di programmazione utilizzato;
  - QString body: campo dati che contiene il corpo del frammento di codice;

  e fornisce i seguenti metodi:
  - source(unsigned int=1, QString ="noTitolo", QDateTime=QDateTime::currentDateTime(), QDateTime=QDateTime::currentDateTime(), bool=false, QString = "noParent", QString ="noLinguaggio",QString = "noBody"): costruttore unico con valori di default per ogni campo;
  - QString getLanguage() const: ritorna il linguaggio di programmazione utilizzato;
  - QString getBody() const: ritorna il frammento di codice scritto;
  - void setLanguage(const QString&): setta il linguaggio di programmazione utilizzato;
  - void setBody(const QString&): setta il valore del frammento di codice;
  - virtual void *saveNote*(QXmlStreamWriter&) const: implementa il metodo virtuale puro che permette il salvataggio del frammento di codice in un database xml.

Gerarchia utenti



- **user**: è la classe base virtuale pura, che funziona da interfaccia comune per tutti i tipi di utenti.

  Ogni user è caratterizzato da:
  - QString usr: username;
  - QString psw: password di accesso;

  e fornisce i seguenti metodi:
  - user(QString,QString): costruttore unico a due parametri;
  - virtual ~*user*(): distruttore virtuale;
  - QString getUsr() const: ritorna il valore dell'username;
  - QString getPsw() const: ritorna il valore della password;
  - void setUsr(const QString&): setta il valore dell'username;
  - void setPsw(const QString&): setta il valore della password;
  - virtual void *saveUser*(QXmlStreamWriter&) const =0: metodo virtuale puro che permette il salvataggio dei dati di accesso dell'utente del database xml;
  - virtual void *writeOnDb*(QXmlStreamWriter &) const =0: metodo virtuale puro che permette il salvataggio delle note degli utenti in un database xml;
  - virtual const note* searchNote(const unsigned int&) const =0: metodo virtuale puro che permette la ricerca di una nota identificata da un id.

- **usrCourse**: è una classe concreta derivata da user, che si occupa di gestire le note di uno specifico corso. Ogni usrCourse è caratterizzato da:
    - QVector<const note*> dbNote: contenitore di puntatori a note create dall'utente corso;

  e fornisce i seguenti metodi:
    - usrCourse(QString,QString): costruttore unico a due parametri;
    - QVector<const note*> getNote() const: ritorna un contenitore di tutte le note del corso;
    - QVector<const note*> getDefinition() const: ritorna un contenitore di tutte le note di tipo definizione;
    - QVector<const note*> getSource() const: ritorna un contenitore di tutte le note di tipo frammento di codice;
    - QVector<const note*> getFavourites() const: ritorna un contenitore di tutte le note contrassegnate come preferite;
    - virtual void *saveUser*(QXmlStreamWriter&) const: implementa il metodo virtuale puro che permette la scrittura nel database dei dati dell'utente;
    - virtual void *writeOnDb*(QXmlStreamWriter &) const: : implementa il metodo virtuale puro che permette la scrittura nel database delle proprie note;
    - void resetNote(): permette la cancellazione di tutte le note salvate nel contenitore;
    - void addNote(const note*): aggiunge una nota al contenitore;
    - void editDefinition(const unsigned int&, const QString&, const QString&): permette la modifica di una nota di tipo definizione;
    - void editSource(const unsigned int&, const QString&, const QString&,const QString&): permette la modifica di una nota di tipo Frammento di codice;
    - virtual const note* searchNote(const unsigned int&) const: implementa il metodo virtuale puro che ritorna un puntatore alla nota avente l'id richiesto;
    - void clearNote(const unsigned int&): elimina la nota avente l'id richiesto;

- **usrStudent**: è una classe concreta derivate da user, che si occupa della gestione delle note, e dell'iscrizione ai vari corsi di uno studente. Ogni usrStudent è caratterizzato da:
    - QVector<const usrCourse*> course: contenitore di puntatori a usrCourse;
    - usrStudent(QString,QString): costruttore unico a due parametri;
    - void subscribeCourse(const usrCourse*): inserisce nel contenitore un puntatore ad un corso che lo studente intende seguire;
    - void unSubscribeCourse(const QString&): rimuove dal contenitore un puntatore ad un corso che lo studente non intende più seguire;
    - QVector<const usrCourse*> getCourses() const: ritorna un contenitore di tutti i corsi a cui lo studente è iscritto;
    - QVector<const note*> getAllNotes()const: ritorna un contenitore di tutte le note dei corsi a cui lo studente è iscritto;
    - QVector<const note*> getAllDefinitions()const: ritorna un contenitore di tutte le note di tipo definizione dei corsi a cui l'utente è iscritto;
    - QVector<const note*> getAllSources()const: ritorna un contenitore di tutte le note di tipo frammento di codice dei corsi a cui l'utente è iscritto;
    - QVector<const note*> getAllFavourites()const: ritorna un contenitore di tutte le note che sono state contrassegnate come preferite dai corsi a cui l'utente è iscritto;

- virtual void saveUser(QXmlStreamWriter&) const: implementa il metodo virtuale puro che permette la scrittura nel database dei dati dell'utente;
- virtual void writeOnDb(QXmlStreamWriter &) const: implementa il metodo virtuale puro che permette la scrittura nel database dei nomi, che sono univoci, di tutti i corsi a cui l'utente si è iscritto;
- virtual const note* searchNote(const unsigned int&) const: implementa il metodo virtuale puro che ritorna un puntatore alla nota avente l'id richiesto;
- bool isSubscribed(const QString &) const: indica se l'utente è iscritto o meno ad un corso.

Codice polimorfo

Alcuni esempi di come viene utilizzato il polimorfismo all'interno del progetto QPedia sono:

- Nella classe **user**. Compare un metodo virtuale puro:

  virtual void *writeOnDb*(QXmlStreamWriter &) const =0

  che viene implementato in modo diverso dalle classi **usrCourse** e **usrStudent**, costituendo, quindi, il polimorfismo.
  Le due implementazioni si occupano di scrivere all'interno del database xml i dati contenuti nei contenitori che caratterizzano i due tipi di utente, ma il comportamento di tali metodi risulta differente. Per **usrCourse** viene infatti salvato nel database un contenitore di puntatori a note, e nel farlo richia a sua volta un altro metodo polimorfo per la scrittura effettiva delle note. Per **usrStudent** viene salvato nel database un contenitore di puntatori a usrCourse, scrivendo solamente l'username essendo univoco.

- Nella classe **user**. Compare un metodo virtuale puro:

  virtual void *saveUser*(QXmlStreamWriter&) const =0;

  che viene implementato in modo diverso dalle classi **usrCourse** e **usrStudent**.
  Si occupa di scrivere all'interno del database i dati relativi all'accesso (username e password), richiamando gli opportuni metodo di GET, segnalando con un diverso tag xml iniziale se si trattano di dati di un utente di tipo corso o di tipo studente.

- Nella classe **user**. Compare un metodo virtuale puro:

  virtual const note* *searchNote*(const unsigned int&) const =0;

  che viene implementato in modo diverso dalle classi **usrCourse** e **usrStudent**.
  Si occupa di ricercare all'interno dei contenitori che caratterizzano gli utenti le note aventi l'id richiesto, ma essendo i contenitori di natura diversa, il comportamento dei due metodi risulta differente.

- Nella classe **note**. Compare un metodo virtuale puro:

  virtual void *saveNote*(QXmlStreamWriter&) const =0;

  che viene implementato in modo diverso dale classi **definition** e **source**
  Si occupa di scrivere all'interno del database i campi identificativi dell'oggetto di invocazione tramite i vari metodi GET, è necessario quindi, un'implementazione diversa per il salvataggio di tutti i campi delle note.

Manuale utente

- **Login**:
  Si tratta di una schermata che presenta una form che permette il login inserendo obbligatoriamente i campi dati *username* e *password.*
  Premendo il pulsante “Registrazione” si passerà alla schermata di registrazione.

- **Registrazione**:
  Si tratta di una schermata che presenta una form che permette la registrazione, inserendo obbligatoriamente i campi *username* e *password,* di un utente corso se viene premuto il pulsante “Registrazione Corso” o di un utente studente se viene premuto il pulsante “Registrazione Studente”

- **Schermata utente corso**:

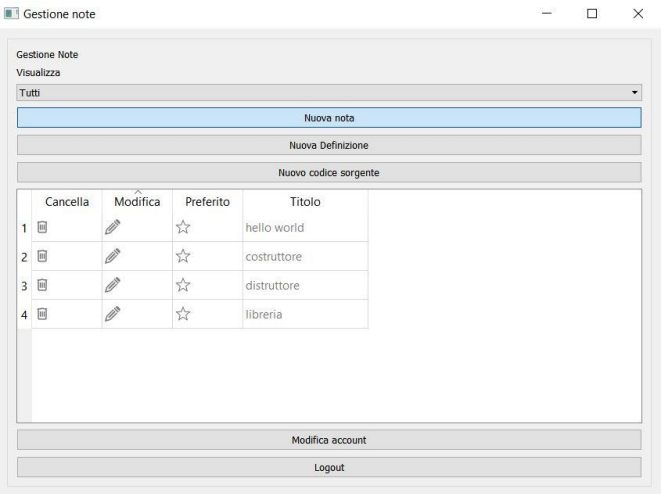


Figura 1 Schermata usrCourse

Visualizza le note del corso in una tabella.
Per visualizzare o modificare una nota è sufficiente selezionare la relativa icona della colonna modifica. Non è possibile modificare l’id della nota, la data di creazione e la data di ultima modifica in quanto vengono aggiornate automaticamente.
Per eliminare una nota, è sufficiente selezionare l’icona della colonna cancella
È possibile segnare come preferita una nota, selezionando l’icona presente nella colonna preferiti
È possibile visualizzare nella tabella, grazie al menu a tendina, tutte le note, solo le definizioni, solo i frammenti di codice o solo le note segnate come preferite.
È possibile ordinare in ordine alfabetico le colonne della tabella, selezionando l’intestazione della colonna che si vuole ordinare.
Selezionando il pulsante “Nuova nota”, compariranno due nuovi pulsanti che danno la possibilità di creare o una nuova definizione “Nuova Definizione” o un nuovo frammento di codice “Nuovo codice sorgente”.
Selezionando il pulsante “Modifica account” è possibile modificare la password dell’utente.
Con il pulsante “Logout” sarà possibile effettuare il logout e tornare alla schermata di login.

- **Inserimento definizione**:
  Presenta un form dove è possibile inserire i dati della definizione escluse la data di creazione e la data di modifica che vengono inseriti automaticamente.
  Il pulsante “Salva” apre una finestra che permette di effetture il salvataggio delle modifiche o la rinuncia alle modifiche effettuate, il pulsante “Annulla” annulla tutte le modifiche effettuate.
- **Inserimento frammento codice**:
  Presenta un form dove è possibile inserire i dati del frammento di codice ad esclusione di data di creazione e data di modifica che vengono inseriti automaticamente. Il pulsante “Salva” apre una finestra che permette di effetture il salvataggio delle modifiche o la rinuncia alle modifiche effettuate, il pulsante “Annulla” annulla tutte le modifiche effettuate.

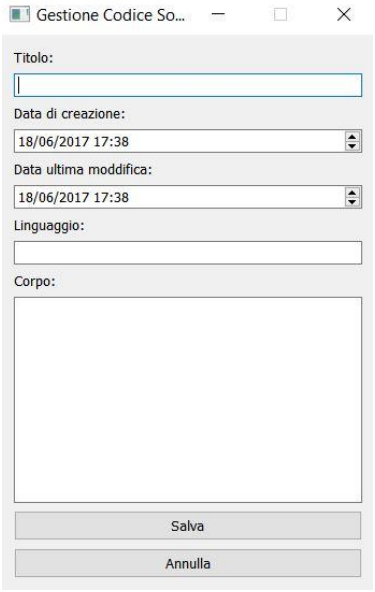


Figura 2 Nuovo Codice

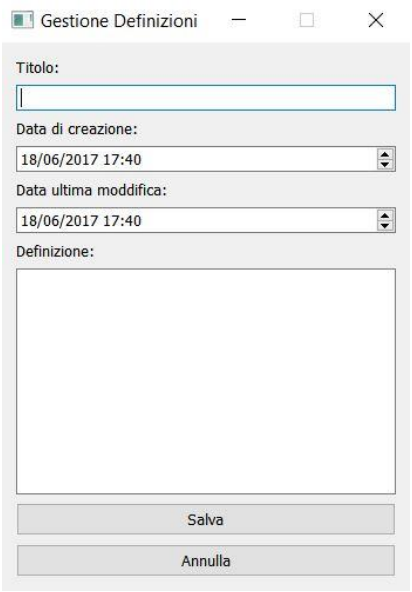


Figura 3 Nuova Definizione

- **Schermata utente studente**:

Figura 4 Schermata usrStudent

Visualizza le note dei corsi a cui lo studente si è iscritto in una tabella.
Per visualizzare una nota è sufficiente selezionare la relativa icona della prima colonna, non è possibile modificare alcun campo dati.
È possibile visualizzare nella tabella, grazie al menu a tendina, tutte le note, solo le definizioni, solo i frammenti di codice o solo tutte le note segnate come preferite dai vari corsi.
È possibile ordinare in ordine alfabetico le colonne della tabella, selezionando l'intestazione della colonna che si vuole ordinare.
Selezionando il pulsante "Corsi" è possibile visualizzare la lista dei corsi a cui l'utente può iscriversi.
Con il pulsante "Logout" sarà possibile effettuare il logout e tornare alla schermata di login
Selezionando il pulsante "Modifica account" è possibile modificare la password dell'utente.

- **Iscrizione corso**:
  Visualizza i corsi a cui lo studente può iscriversi in una tabella.
  Per iscriversi ad un corso è sufficiente premere sulla prima colonna l'icona con il simbolo "+".
  Per cancellare l'iscrizione ad un corso è sufficiente premere sulla prima colonna l'icona con il simbolo "-".
  Con il pulsante "Indietro" sarà possibile tornare alla schermata principale dell'utente

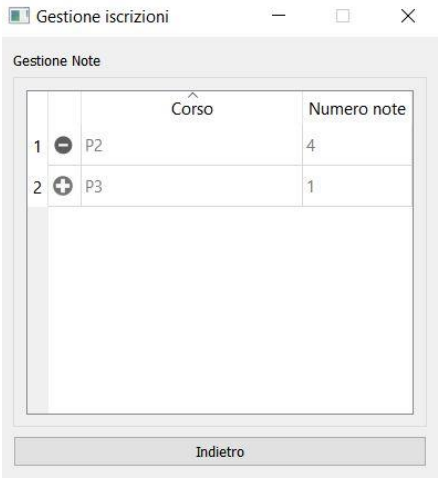


Figura 5 Schermata iscrizione corsi

Ore lavorative

- Progettazione: 10 ore
- Codifica model: 15 ore
- Codifica view: 20 ore
- Codifica controller: 10
- Testing e debug: 4

Totale: 59 ore