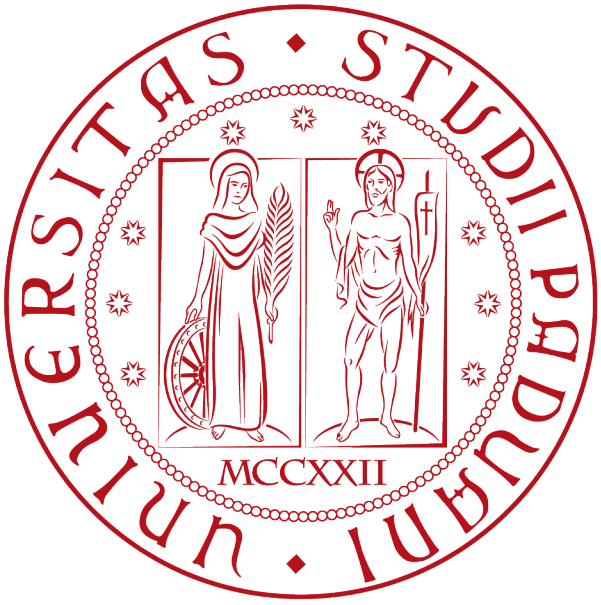

Relazione progetto fine corso

Carlo Sindico - 09 Luglio 2017

Username	Password
admin	admin

Indice

1 Scopo del progetto

Il progetto ha lo scopo di fornire un semplice ed intuitivo sistema per l'amministrazione, attraverso una interfaccia grafca, di una biblioteca. La biblioteca contiene opere che sono caratterizzate da un titolo e da un identificativo univoco, ed ogni opera può essere di due tipi: libro o rivista.

- Un libro è una particolare opera caratterizzata da un autore, mentre invece una rivista è caratterizzata da un anno di uscita.
- Le opere della biblioteca possono venire prestate agli utenti, ogni opera contiene l'informazione relativa al suo stato se essa disponibile al prestito e se presente nella biblioteca.
- Per quanto riguarda i libri non vi è alcun vincolo sul prestito, invece le riviste che hanno un anno di uscita superiore a 20 anni non possono essere prestate nè all'utente basic nè all'utente pro.
- La biblioteca fornisce un'area utente, attraverso la quale ogni utente pu prendere in prestito le opere. (possibilità di visualizzare le opere presenti nella biblioteca, e contemporaneamente quelle prese in prestito). Lutente attraverso questa interfaccia pu ricevere e restituire le opere.
- Un utente caratterizzato da un nome, un cognome, un codice fiscale, un identificativo univoco, una password e il numero di opere che ha in prestito.

2 Compilazione ed esecuzione

Per compilare il progetto è necessario posizionarsi all'interno della cartella dello stesso, eseguire il comando **qmake**, eseguire il comando **make** e successivamente **./biblioteca-p2**. Vengono consegnati anche due file, databaseutenti.xml e database.xml, contenenti dei dati esempio.

3 Correzioni effettuate

In seguito all'appello orale del 30-06-2017, sono state apportate le seguenti modifiche:

- per quanto riguarda gli utenti è stata modificata la gerarchia Utente in particolare:
  - i metodi relativi al prestito e alla ricezione di libri e riviste sono diventati metodi della classe Utente. Questo ha portato ad una riduzione complessiva del codice duplicato in tale gerarchia.
  - sono stati rimossi i campi dati statici della classe utente_basic e utente_pro perchè ritenuti inutili.
  - **i tipi di utente si differenziano sostazialmente per le funzionalità che hanno a disposizione(ricerca, prestito, ricezione di opere)**; infatti il metodo `virtual bool ricerca(const QString,opera*)const=0` fornisce un tipo diverso di ricerca rispettivamente per utente_basic e utente_pro.
  - è stato rimosso il codice duplicato nelle due sottoclassi.

- per quanto riguarda le classi database, è stata modificata la classe database e rimossa la classe database_utenti_opere. Questa scelta è dettata dal fatto che le opere appartenenti alla biblioteca e le opere appartenenti all'utente si differenziano solo per un campo dati.(I metodi della classe database_utente_opere svolgevano le stesse operazioni).Si è deciso di utilizzare quindi un unico database che contiene sia le opere della biblioteca che quelle dell'utente. **In questo modo è stata rimossa la classe database_utenti_opere perchè rappresentava un duplicato della classe database**, riducendo ancora il codice duplicato.
- per quanto riguarda l'autenticazione degli utenti, la GUI non è più responsabile dell'autenticazione di quest'ultimi, ma invece è l'amministratore a svolgere tale compito. Inoltre nella classe homewindow è stato rimosso il doppio dialog riducendo in questo modo la quantità di codice duplicato.
- è stata posta più attenzione all'usabilità richiamando i segnali di chiusura finestra in tutte le classi che permettono di inserire, modificare opere e utenti, rimuovendo anche i controlli inutili.
- sono state rimosse le classi inserisci_utente_basic, inserisci_utente_pro e i relativi controller. Le form di inserimento utente_basic e utente_pro richiedevano di inserire le stesse informazioni, questo rappresenta duplicazione di codice nella GUI. È stata aggiunta un unica classe per inserire l'utente con un menù a tendina per scegliere il tipo di utente da inserire.
- nelle classi listalibri e listariviste della VIEW, **sono stati eliminati i metodi aggiorna_vista_libri_prestito() e aggiorna_vista_riviste_prestito(). Le loro funzionalità sono state incapsulate nel metodo aggiorna_vista(). I metodi precedenti infatti svolgevano lo stesso compito e differivano solo per un controllo**. Questo ha portato ad una riduzione del codice duplicato.

4 Funzionalità disponibili all'amministratore

- l'amministratore attraverso il sistema deve essere in grado di eseguire semplici azioni quali:
    1. Visualizzare tutte le opere presenti nella biblioteca
    2. Visualizzare i dettagli di tutte le opere presenti nell'archivio, ed eventualmente di modificarne i parametri quali titolo autore o anno di uscita, a seconda che sia un libro o una rivista
    3. Visualizzare tutti gli utenti registrati alla biblioteca
    4. Visualizzare i dettagli di tutti gli utenti presenti nell'archivio, ed eventualmente di modificarne i parametri quali nome e cognome
    5. Aggiungere un'opera per aumentare l'offerta concessa dalla biblioteca
    6. Rimuovere un'opera
    7. Aggiungere un utente all'archivio della biblioteca
    8. Rimuovere un utente solo se ha restituito tutte le opere prese in prestito

5 Funzionalità disponibili all'utente

- l'utente attraverso il sistema deve essere in grado di eseguire semplici azioni quali:
  1. utente_basic può ricercare opere in base al Titolo, ricevere in prestito e restituire libri e riviste
  2. utente_pro può ricercare opere in base al Titolo o al codice identificativo, ricevere in prestito e restituire libri e riviste

6 Descrizione della gerarchia costituita da opera e dai suoi sottotipi

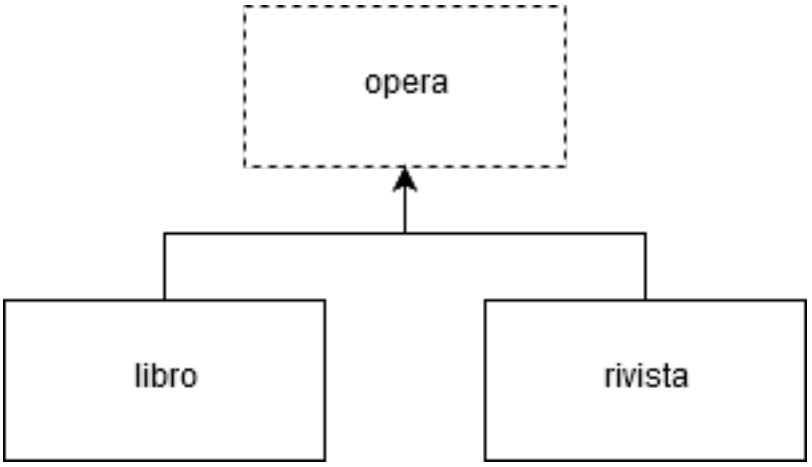

Figura 1: Gerarchia opera

6.1 Classe base astratta opera

- **La classe opera alla base della gerarchia è caratterizzata dai seguenti campi dati:**
- `QString titolo`: titolo dellopera
- `Bool statoPresenza` indica se l'opera presente nella biblioteca oppure in prestito, (1 indica la presenza dellopera nellarchivio) (0 indica la non presenza ossia che è in prestito).
- `Int id`: identificativo univoco dell'opera, attribuito a ciascuna opera al momento della costruzione dell'opera stessa.
- `Int maxid`: il massimo id dell'opera, l'opera con id più elevato di tutte le altre e viene memorizzato in questa variabile.
- `Int appartenenza`: variabile che indica a chi appartiene l'opera (-1 valore attribuito quando l'opera appartiene alla biblioteca), (idutente valore attribuito quando l'opera appartiene ad un utente).

Inoltre sono presenti dei metodi non virtuali quali il costruttore:

- `Opera(const QString& , bool =0)`: costruttore a 1,2 argomenti che consente di costruire un'opera inserendo il solo titolo e segnalando di default che l'opera presente nella biblioteca; il corpo del costruttore consente di assegnare in automatico un ID univoco all'opera prendendo il massimo ID nora assegnato e facendo +1; in questo modo si assicura l'univocità degli ID.

Metodi get:

- `QString GetTitolo()const`: metodo costante che consente di ritornare il titolo dell'opera.
- `int Get Id() const`: metodo constante che consente di ritornare l'Id dell'opera.
- `int Get MaxId() const`: metodo constante che consente di ritornare l'Id massimo assegnato ad un'opera no a questo momento.

Metodi di set:

- `void Set Id(const int n)` : metodo che consente di modi care l'ID di un'opera facendolo diventare = n.
- `void Set maxid(const int)`: metodo che consente di aggiornare il campo dati maxId in modo che resti sempre il massimo Id assegnato ad un'opera.
- `void Setappartenenza(const int)`: metodo che consente di aggiornare il campo dati appartenenza in modo che si ideni chi a chi appartiene quell'opera.
- `int Getappartenenza() const`: metodo constante che consente di ritornare l'appartenenza di quell'opera.
- `void Set Titolo(QString)`: metodo che consente di aggiornare il campo dati maxId in modo che resti sempre il massimo Id assegnato ad un'opera.
- `void Riscatta()`: metodo che simula il ritorno in biblioteca di un'opera (settando statoP=1).

Metodi virtuali:

- `virtual bool disponibile()const`: metodo virtuale che ritorna true se l'opera di invocazione pu essere prestata, false altrimenti (in particolare questa una versione base che ritorna true se l'opera presente negli archivi della biblioteca).
- `virtual void Write opera(QXmlStreamWriter& xmlWriter) const =0`: metodo che consente di scrivere nel le XML l'opera di invocazione. Esso puro in quanto Libro e Rivista hanno campi dati diversi.
- `virtual void PrestaOpera()`: metodo che consente di prestare un'opera, è stato scelto di segnarlo come virtuale in quanto vi sono delle regole diverse nei due sottotipi.
- `virtual info opera info tot() const =0`: metodo virtuale puro che consente di ritornare un oggetto di tipo info contenente tutte le informazioni relative all'opera di invocazione; il metodo virtuale puro in quanto le informazioni presenti nei sottotipi sono diverse.

- `virtual QString Get tipo() const =0`: metodo virtuale puro che ritorna una stringa che rappresenta il tipo dell'opera di invocazione quindi se l'opera un libro il metodo ritorna la stringa libro mentre se una rivista ritorna rivista. (il metodo viene utilizzato solo in lettura)

6.2 Classe libro

Metodi get:

- `QString GetAutore() const`: metodo constante che consente di ritornare l'autore dell'opera
- `void SetAutore(QString)`: metodo che consente di aggiornare il campo dati Autore di un libro.

Metodi virtuali:

- `virtual void Write Opera(QXmlStreamWriter& xmlWriter) const`: implementazione del metodo virtuale puro della classe base che consente la scrittura su database di un libro.
- `virtual info opera info tot()const`: metodo che ritorna un oggetto di tipo info contenente tutte le informazioni riguardo all'oggetto di invocazione: titolo, autore, Id, prensenza in biblioteca e disponibilit al prestito.
- `virtual QString Get tipo()const`: metodo che ritorna la stringa libro.

6.3 Classe rivista

Metodi get:

- `int GetMaxAnni() const`: metodo constante che consente di ritornare il numero massimo di anni entro il quale un'opera deve essere uscita per permettere il prestito.
- `int GetAnnoUscita() const`: metodo constante che consente di ritornare l'anno di uscita della rivista di invocazione.

Metodi virtuali:

- `virtual bool disponibile() const`: metodo che ritorna true se l'opera in biblioteca e se la sua data di uscita minore o uguale a maxAnni. Nel caso in cui ritorna true allora significa che l'opera può venire prestata.
- `virtual void Write opera(QXmlStreamWriter& xmlWriter) const`: implementazione del metodo virtuale puro della classe base che consente la scrittura su database di un'opera.
- `virtual info opera info tot()const`: metodo che ritorna un oggetto di tipo info contenente tutte le informazioni riguardo all'oggetto di invocazione: titolo, anno di uscita, Id, prensenza in biblioteca e disponibilità al prestito
- `virtual void PrestaOpera()`: metodo che testa la disponibilità della rivista di invocazione e presta l'opera se essa diponibile al prestito.

- `virtual QString Get tipo()const`: metodo che ritorna la stringa rivista.

7 Descrizione della gerarchia costituita da utente e dai suoi sottotipi:

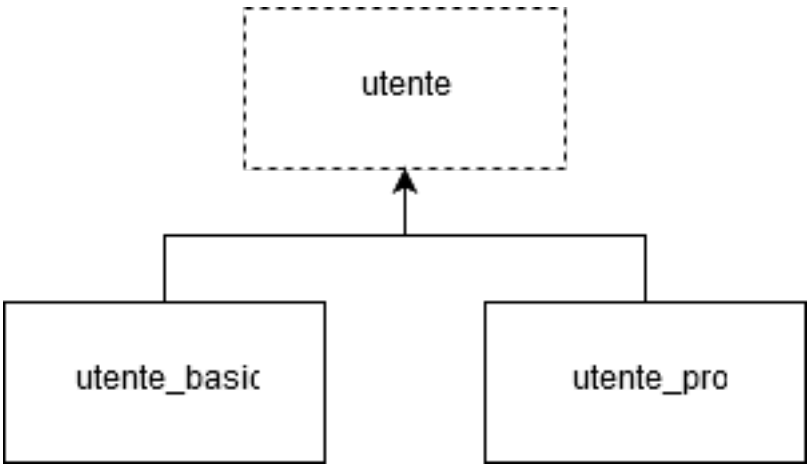

Figura 2: Gerarchia utente

7.1 Classe base astratta utente

- **La classe utente è alla base della gerarchia ed è caratterizzata dai seguenti metodi:**

  Vengono descritti i metodi più significativi

- `database* GetopereBiblioteca() const`: metodo che permette all'utente di interfacciarsi con il database delle sue opere e quelle della biblioteca.
- `void ricevi libro(unsigned int)`: metodo che permette all'utente di ricevere in prestito un libro.
- `void ricevi rivista(unsigned int)`: metodo che permette all'utente di ricevere in prestito una rivista.
- `void restituisci libro(unsigned int)`: metodo che permette all'utente di restituire un libro.
- `void restituisci rivista(unsigned int)`: metodo che permette all'utente di restituire una rivista.
- `virtual bool ricerca(const QString,opera*) const =0`: metodo virtuale puro che fornisce al tipo di utente diverse modalità di ricerca delle opere.

7.2 Classe utente pro

La classe utente_pro implementa il metodo `virtual bool ricerca(const QString,opera*) const =0` permettendo una ricerca approfondita (anche per stringhe contenute), per titolo e id di un'opera. Eredita tutte le funzionalità di prestito e restituzione di un opera.

7.3 Classe utente basic

La classe utente_basic implementa il metodo `virtual bool ricerca(const QString,opera*) const =0` permettendo una ricerca approfondita (anche per stringhe contenute), per titolo di un'opera. Eredità tutte le funzionalità di prestito e restituzione di un'opera.

8 Manuale GUI

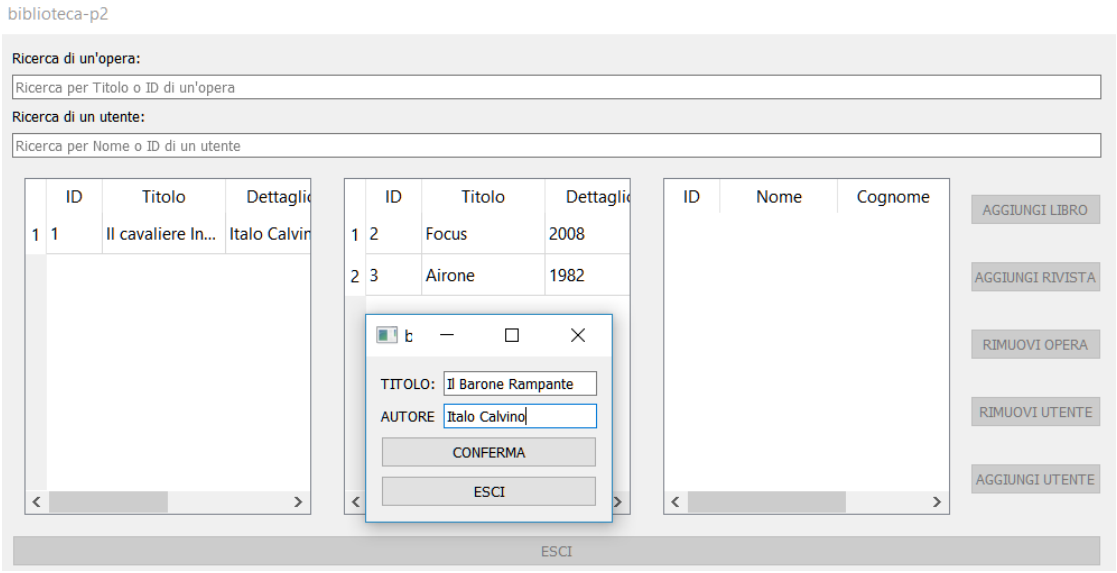

Figura 3: GUI lato amministratore

L'applicazione salva i dati attraverso file XML. Nel caso in cui tali file non venissero trovati, verranno generati automaticamente dall'applicazione stessa. Dopo aver inserito le credenziali, se corrette:

- l'amministratore può accedere alla gestione delle opere e degli utenti e del menu laterale contente i pulsanti di aggiunta e rimozione. Per modificare un opera o un utente l'amministratore dovrà fare doppio click su un opera o un utente che compare in tabella. Disponibile all'amministratore anche la ricerca.
- L'utente invece può accedere all'interfaccia che gli permette di ricevere in prestito e restituire le opere. Anche l'utente ha a disposizione un metodo di ricerc diverso in base al tipo di utente.