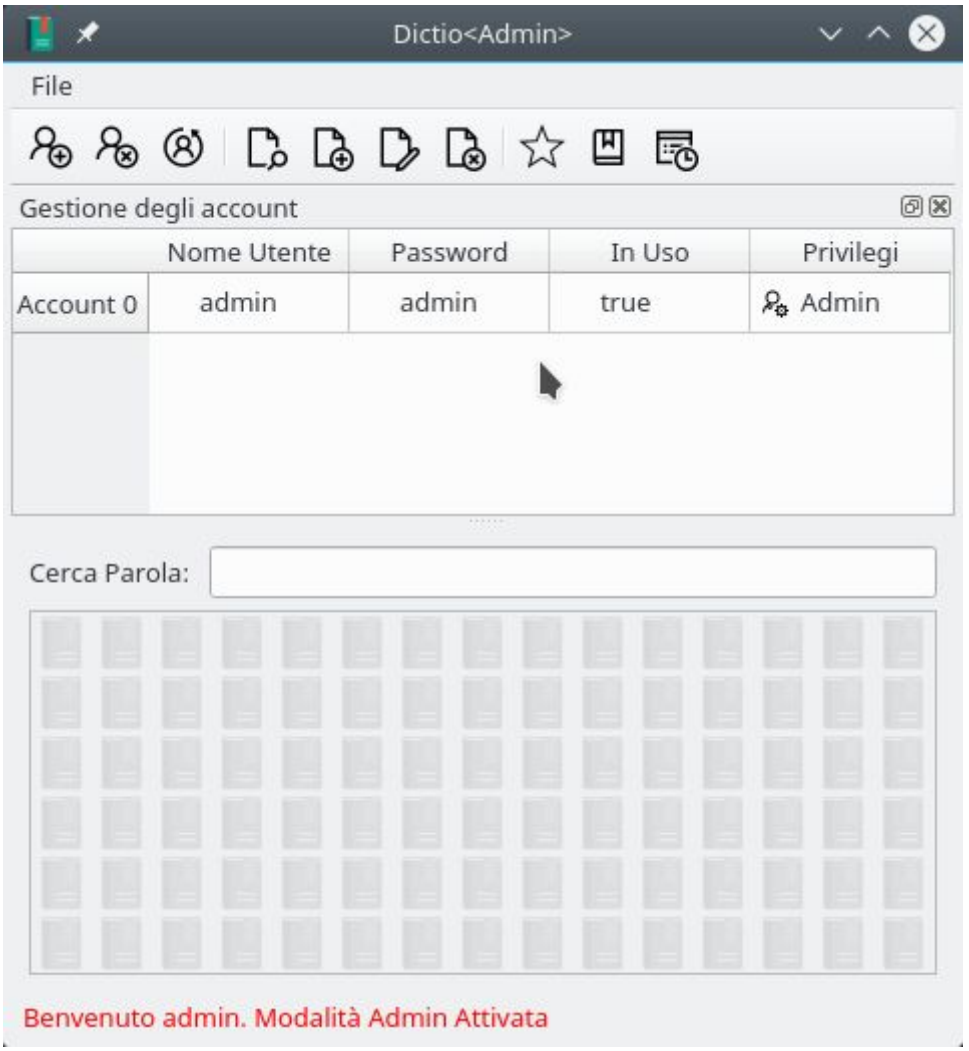
Riccardo Damiani
Matricola: 1125327
Anno Accademico: 2016-2017

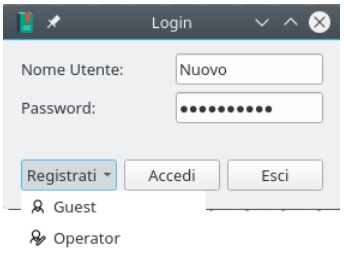
Progetto del corso di Programmazione ad Oggetti



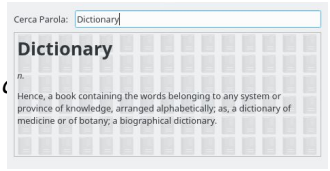
ogetto:

Il progetto in esame consiste in un dizionario monolingue inglese, è costituito da tre interfacce diverse in base alla tipologia di utente registrato.
Un utente iscritto può essere “Amministratore”, “Operatore” oppure “Visitatore”. Il visitatore ha la possibilità di accedere al dizionario in sola lettura, l’operatore anche in scrittura e l’amministratore ha il controllo totale (anche sugli altri utenti). Una *dialog* accoglie l’utente e gli permette di registrarsi (come operatore oppure visitatore) o di accedere se già registrato. Nel caso in cui non ci fossero degli account già presenti in memoria, è sempre disponibile un account di default non modificabile con i privilegi di amministratore e con le credenziali nome_utente: “admin”, password: “admin”.

ia e delle funzionalità:



Finestra di login:
Attraverso questa l'utente può accedere al dizionario tramite un utente già esistente o registrarsi con delle nuove credenziali, non è possibile registrarsi come amministratori utilizzando questa interfaccia.



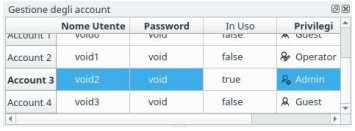
wi

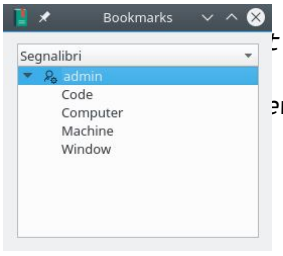
Gestione Vocabolario:
Tramite questo vengono visualizzate le parole cercate attraverso l'uso di codice HTML.

Finestra di modifica:
La che permette (solo agli amministratori e agli operatori) di aggiungere e modificare parole all'interno del dizionario.

QDockWidget

Gestione degli Account:
Con questa classe derivata da è possibile (solo se l'utente attualmente in uso è amministratore) accedere sia in lettura che in scrittura a tutti i dati relativi ai vari utenti memorizzati. Non è possibile modificare l'utente corrente (è necessario fare il logout) né modificare l'account amministratore standard.



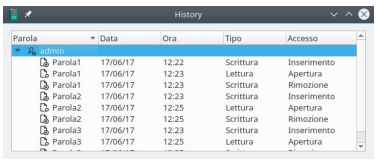


tion

lbar

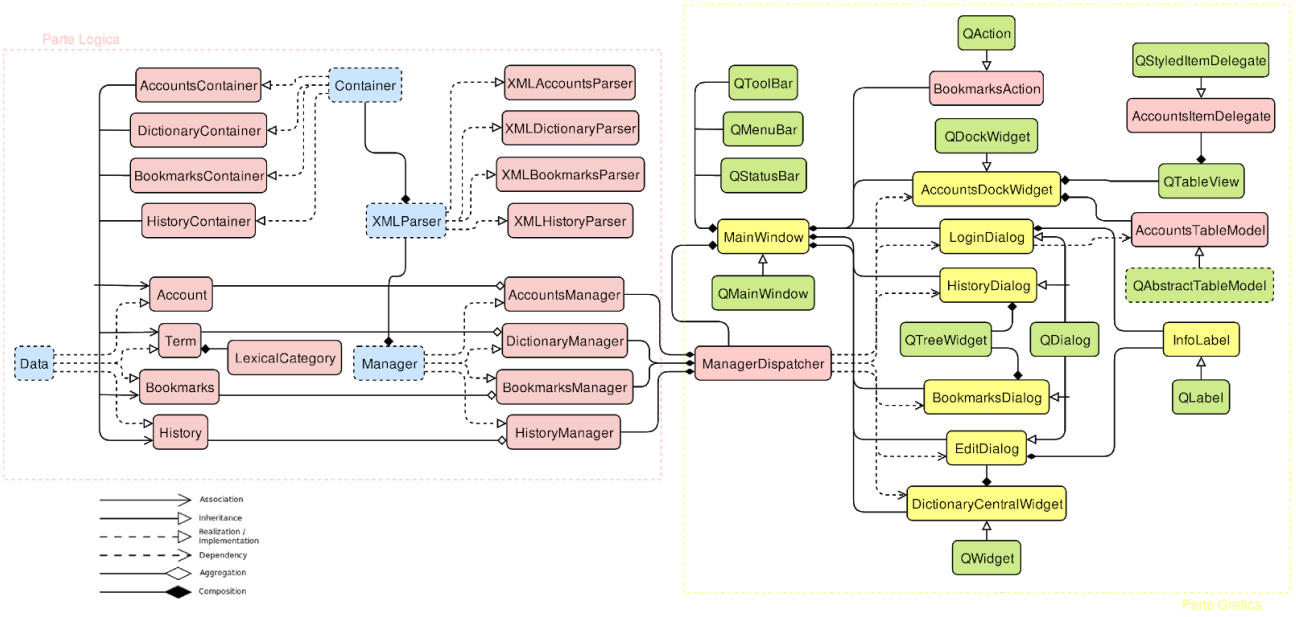
ew

Finestra dei Segnalibri:
Dopo aver aggiunto dei segnalibri (con la collegata alla e al menu) si possono gestire utilizzando questa finestra, nella ad albero è possibile ordinare in ordine alfabetico le parole memorizzate. Un amministratore può accedere ai segnalibri di qualsiasi utente, amministratori compresi.



iew

Finestra degli Accessi:
Grazie a questa finestra vengono mostrate all'utente varie informazioni relative agli accessi al vocabolario eseguiti da tutti gli utenti dal momento della loro registrazione (se non sono stati cancellati). Se non si è amministratori si possono vedere solo le informazioni relative al proprio account. Anche in questa si possono ordinare i dati per colonna.



Nome	Tipo	SuperClassi	Descrizione
	Logica		Rappresenta un , contiene 2 stringhe: username e password, un campo booleano che indica se l'account è attualmente in uso, e un enum che memorizza il tipo di
	Logica		Contiene tutti gli , e permette alle altre parti del programma di accedere tramite puntatori alle varie entry
	Grafica		Contiene il e la relativi agli e ne media l'interazione, fa parte della e può essere riposizionata all'interno dei vari
	Logica		Rappresenta il del paradigma nella logica del framework Qt, permette di visualizzare ed editare correttamente il campo della classe
	Logica		E' l'interfaccia principale di accesso agli , contiene un puntatore alla struttura dati che riceve tramite il . Tramite un campo booleano stabilisce se la struttura dati è stata modificata durante l'esecuzione del programma e se è necessario aggiornare la memoria
	Logica		Nella logica è il layer che sovrasta la memoria, permette di gestire i dati e li rende accessibili alle eventuali che interagiscono con questa interfaccia per visualizzare in maniera

			corretta e aggiornata i dati contenuti nella memoria
	Logica		Classe che estende , permette all'icona nella e nel associata a questa azione custom di cambiare a seconda del fatto che la parola visualizzata sia o meno presente nei segnalibri
	Logica		Rappresenta i segnalibri associati ad un determinato , contiene una stringa e una lista di stringhe: una contiene lo username (univoco) dell'account e l'altra le parole memorizzate
	Logica		Contiene tutti i , e permette alle altre parti del programma di accedere tramite puntatori ai vari segnalibri
	Grafica		Permette all'utente di vedere l'elenco dei segnalibri, dei propri e di quelli di tutti gli altri utenti, se si è amministratori. Quando una parola viene selezionata viene visualizzata all'interno del
	Logica		E' l'interfaccia principale di accesso ai , contiene un puntatore alla struttura dati che riceve tramite il . Tramite un campo booleano stabilisce se la struttura dati è stata modificata durante l'esecuzione del programma e se è necessario aggiornare la memoria
	Astratta		E' il prototipo dell'interfaccia utilizzata da per permettere alle sue classi derivate di accedere alla memoria durante il parsing, e dai vari per ottenere un puntatore alla struttura di dati e permetterne la gestione
	Astratta		Classe astratta con la funzione di wrapper: Fornisce alle altre classi dei metodi puri da implementare e una classe base che riveste gli oggetti appartenenti alle 4 tipologie di dato principali (le sue sotto-classi: , , e)
	Grafica		Permette a tutte le tipologie di utenti di visualizzare le parole contenute nel vocabolario, attraverso il del dizionario recupera la parola da visualizzare che memorizza all'interno di un puntatore, e la mostra all'utente dopo averla codificata in HTML

	Logica		Contiene tutti i , e permette alle altre parti del programma di accedere tramite puntatori alle varie parole
	Logica		E' l'interfaccia principale di accesso ai , contiene un puntatore alla struttura dati che riceve tramite il . Tramite un campo booleano stabilisce se la struttura dati è stata modificata durante l'esecuzione del programma e se è necessario aggiornare la memoria
	Grafica		Estende e tramite questa finestra modale l'utente abilitato a modificare il dizionario può creare nuovi o modificare quelli già esistenti
	Logica		Contiene le informazioni raccolte durante l'utilizzo del dizionario, in particolare registra con dei campi se gli accessi sono in lettura/scrittura, se la parola in questione è stata visualizzata, aggiunta, modificata o eliminata, e lo username dell'utente corrente e altre informazioni
	Logica		Contiene tutti i record della , e permette alle altre parti del programma di accedere tramite puntatori ai vari record
	Grafica		In questa finestra sono raccolti tutti i record e mostrati per mezzo di un , sono ordinabili per colonna, e come per la visualizzare le informazioni sugli accessi al dizionario di tutti gli utenti è consentito solo agli amministratori
	Logica		E' l'interfaccia principale di accesso alla , contiene un puntatore alla struttura dati che riceve tramite il . Tramite un campo booleano stabilisce se la struttura dati è stata modificata durante l'esecuzione del programma e se è necessario aggiornare la memoria
	Grafica		Una custom molto semplice che emula il comportamento di una statusbar dove non è possibile usare una vera e propria (nelle o in un)
	Grafica		Questa classe definisce la finestra del cambio di utente, lanciata dalla all'inizio dell'esecuzione del programma e ogni volta che viene invocata la di logout, accede agli account in memoria tramite il e non il in modo tale che la resti sempre aggiornata

	Logica		Questa classe contenuta in ogni oggetto di classe rappresenta una lista di definizioni semantiche appartenti ad una certa categoria lessicale (aggettivo, verbo transitivo, ecc.)
	Grafica		Il cuore dell'interfaccia grafica. Da qui vengono gestite tutte le finestre e viene permessa la corretta rappresentazione di tutte le informazioni fornite dalla parte logica del programma. Attraverso segnali ed eventi le finestre dialogano tra loro e la fa da collante per il loro corretto funzionamento. Gestisce il lifetime di tutti i moduli in cui è strutturato il codice
	Astratta		Interfaccia che descrive genericamente le operazioni principali di gestione della memoria, contiene al suo interno un puntatore polimorfo al parser associato alla sotto-classe
	Logica		Classe che fa da ponte tra la parte logica e quella grafica del programma, viene gestita dalla mainWindow che utilizza i suoi metodi per ottenere l'accesso alla memoria per mezzo dei diversi manager polimorfi che all'occorrenza vengono distribuiti. E' anche responsabile di dare il via al processo di lettura dei dati e di scrittura
	Logica		Con questa classe viene rappresentata ogni parola contenuta nel vocabolario, contiene internamente una serie di oggetti
	Logica		Tramite le classi e è possibile il parsing dei file xml in cui sono contenuti tutti i dati degli account registrati, solleva eccezioni che vengono gestite internamente e le informazioni relative ad eventuali errori sono mostrate all'utente con delle
	Logica		Analoga alla classe precedente
	Logica		Analoga alla classe precedente
	Logica		Analoga alla classe precedente
	Astratta		Classe astratta contenuta per composizione nel , al suo interno si trova un puntatore polimorfo al tipo di container dinamico a cui è associata, è l'anello

			centrale del meccanismo di chiamate polimorfe che permette il parsing

Il polimorfismo è qui realizzato attraverso le interfacce (classi astratte) e l'overriding dei metodi virtuali puri le cui signature sono qui elencate.

Classe Astratta	Signature dei Metodi Virtuali

	`bool (int, QVariant&) =0;`
	`virtual QVariant getData(int) const =0;`
	`virtual int getDataCount() const =0;`

	`virtual Data* getPtr(int) const =0;`
	`virtual void* getPtr() const =0;`
	`virtual int size() const =0;`
	`virtual void insert(const Data&) =0;`

	`virtual void open4reading() =0;`
	`virtual void open4writing() =0;`
	`virtual bool parseXML() =0;`
	`virtual bool writeXML() =0;`

	`virtual void insert(const Data&) =0;`
	`virtual void remove(const QString&) =0;`
	`virtual int find(const QString&) const =0;`
	`virtual int count() const =0;`
	`virtual void setModified(bool) =0;`
	`virtual bool isModified() const =0;`
	`virtual Data* getPointer(int) const =0;`

	`virtual int columnCount(const QModelIndex&) const =0;`
	`virtual int rowCount(const QModelIndex&) const =0;`
	`virtual QVariant data(const QModelIndex&, int) const =0;`
	`virtual QVariant headerData(int, Qt::Orientation, int) const =0;`
	`virtual bool setData(const QModelIndex&, const QVariant&, int) =0;`
	`virtual Qt::ItemFlags flags(const QModelIndex&) const =0;`

Tempistiche: Monte ore totali approssimativo == 240h... di cui
Studio del framework Qt == 50%
Scrittura del codice == 40%
Debugging && Testing == 10%

Sistema e tools di sviluppo: openSUSE Leap 42.2 - Qt 5.8.0 - GCC 4.8.5