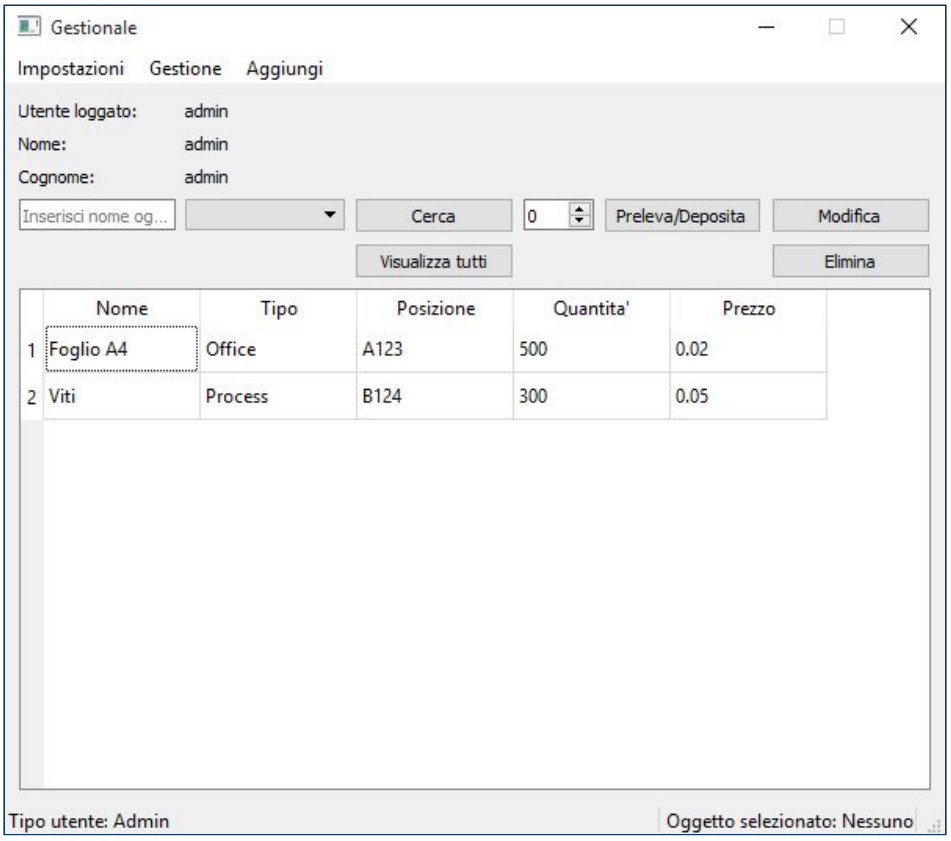
Cristian Maschio
Matricola: 1123398
20/6/17

Università degli Studi di Padova

Relazione progetto P2



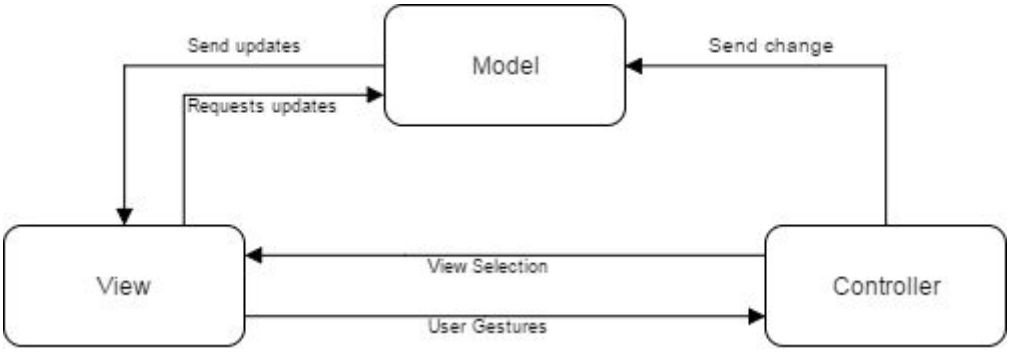
Sommario

CRISTIAN MASCHIO
MATRICOLA: 1123398
20/6/17

1- Scopo del progetto

Il progetto si prefigge lo scopo di creare un'applicazione di supporto per la gestione di un magazzino, la quale facilita il ritrovamento degli utensili e permette di eseguire, aggiunte, prelievi e modifiche su di essi. Così facendo sarà possibile avere una visione aggiornata e completa di tutti gli utensili.

Per far ciò si è deciso di adottare il design pattern Model View Controller (MVC), permettendo così una maggiore astrazione tra model e view, agevolando quindi la localizzazione di possibili errori e semplificando implementazioni future.



2- Descrizione delle gerarchie di tipi usati

Nel progetto si è deciso di utilizzare una sola gerarchia, quella riguardante gli utenti, poiché gli item\utensili utilizzati non avevano caratteristiche diverse da poter rendere utile una gerarchia su di essi.

Per contenere item e utenti si sono create due classi, *warehouse* e *allusers* le quali si occupano della gestione(eliminazione, creazione e modifica), scrittura e della lettura, su file.xml appositi.
Inoltre utilizzano come contenitore la **std::list** fornita dalla libreria STL.
Questa scelta di contenitore è dovuta al fatto di far risultare le operazioni di rimozione più efficienti rispetto a quelle effettuate su un **vector**, che necessita di essere "ricompattato" dopo ogni rimozione.

Nei due punti successivi verranno descritte le classi che modellano item e utenti :

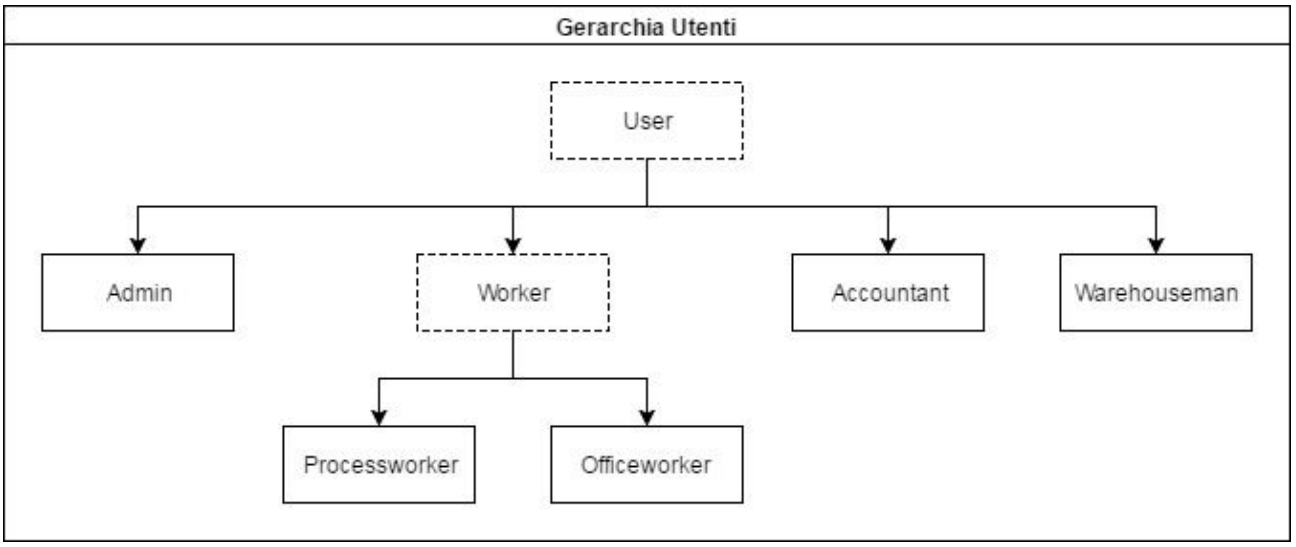
2.1- Descrizione item

La classe item, modella il concetto di oggetto materiale, con 5 campi dati:

- nome
- prezzo
- quantità
- posizione
- tipo, il quale accetta solo stringhe composte da "All", "Office" e "Process".

La classe dispone dell'operatore di uguaglianza (operator ==) e di opportuni metodi per il salvataggio e caricamento dell'item nel file xml a loro dedicato.

2.2- Gerarchia utenti



La gerarchia degli utenti è composta da una classe base astratta denominata *User*, contenente: username, password, nome, cognome, skill

Da essa derivano *Admin*, *Worker*, *Accountant* e *Warehouseman* ognuna delle quali possiede un campo dati **itemAccess** di tipo static const string il quale specifica il tipo di item a cui l'utente può accedere.
L'utente *admin* ha la caratteristica di poter gestire l'intera applicazione, compresi gli utenti stessi.
Accountant, ossia il personale contabile, può invece accedere a tutti i tipi di oggetti e di modificarne il prezzo.
Warehouseman, il personale responsabile del magazzino, può accedere a tutti i tipi di oggetti e può modificarne la posizione.
Per finire il *Worker*, classe astratta, da cui derivano *Processworker* e *Officeworker*;
Processworker, il personale operaio, può accedere solo ad oggetti di tipo process, mentre *Officeworker* , il personale che figura come impiegato, può accedere agli oggetti di tipo office.

Tutti gli utenti sono abilitati a modificare la quantità degli oggetti da loro visualizzati.

3- Descrizione dell'uso del codice polimorfo

Nella classe User:

- **virtual void loadUser(QXmlStreamReader&), virtual void saveUser(QXmlStreamWriter&)const** metodi utilizzati per la lettura e scrittura di file.xml, permettono una maggiore manutenibilità del codice per implementazioni future;
- **string getType() const**, restituisce il tipo di utente, permettendo al fine di evitare dynamic cast;
- **string getItemAccess() const**, restituisce una stringa contenente i permessi riguardanti all'accesso degli oggetti;
- **User* clone()const**
- e una sequenza di booleani che permettono di rintracciare le funzionalità date all'utente.

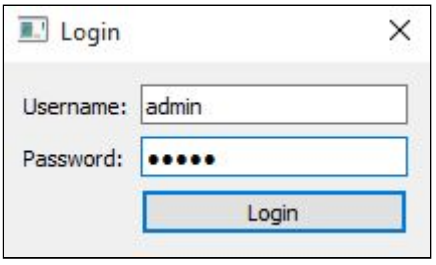
Altro utilizzo di codice polimorfo avviene nella classe **logmainview**, dove si ha il metodo virtuale QMainWindow: **virtual void closeEvent (QCloseEvent* event)** con il quale si effettua l'overriding dell'evento di chiusura della classe QMainWindow, dando la possibilità all'utente di annullare l'operazione di chiusura oppure di uscire con o senza salvare.

4- Manuale utente

L'applicativo, al fine di avere un funzionamento corretto, necessita del proprio file **.pro** oppure dell'inserimento manuale della seguente linea di codice QT **+= core gui** e QT **+= widgets** nel file.pro generato con il comando **qmake -project**, al fine di poter utilizzare la libreria grafica di QT.

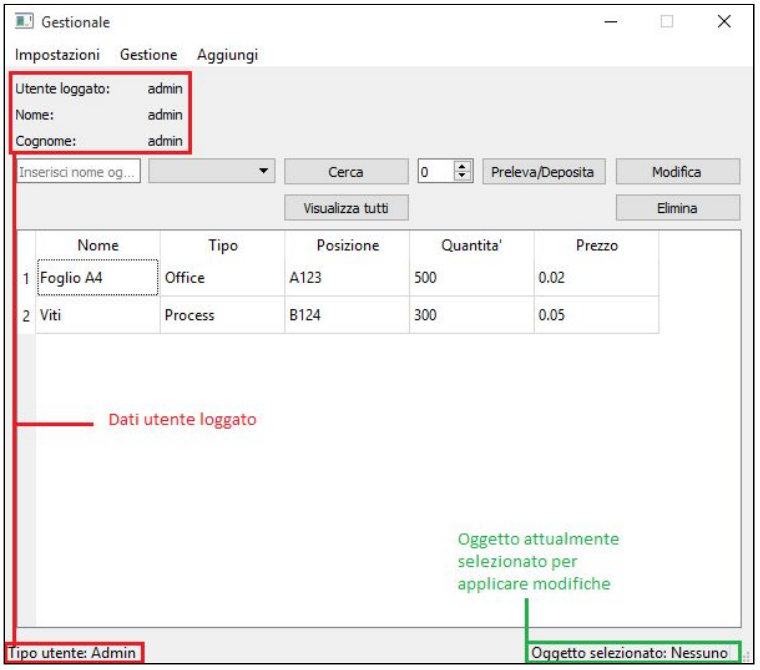
All'esecuzione il programma cerca di leggere due file: *users*.xml e *items*.xml, il corretto funzionamento non è influenzato dalla presenza di questi file.

Successivamente si aprirà la finestra di login; nel caso in cui non venissero trovati l'applicativo permette l'autenticazione di un utente amministratore di default con **username: admin** e **password: admin,** il quale verrà creato in mancanza di admin nel database degli utenti.



Solo l'admin ha la possibilità di creare, eliminare utenti o oggetti.

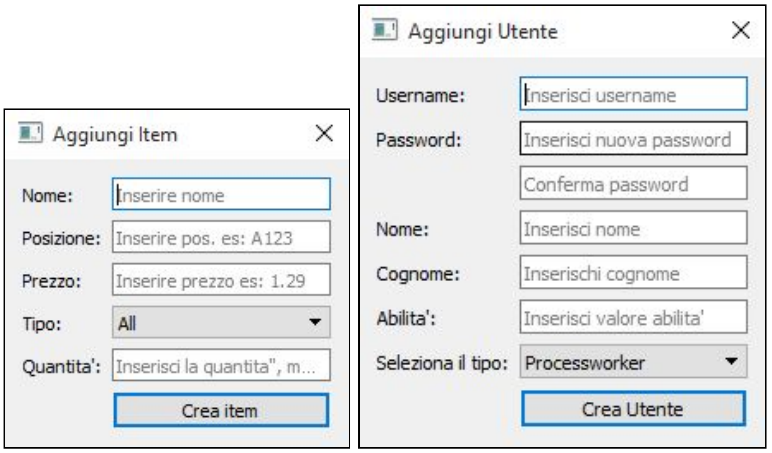
Una volta autenticati si presenterà la finestra principale (la cui immagine vieni riportata qui sotto) con al suo interno, una tabella che ha come unico scopo di mostrare a video gli oggetti e utenti cercati.



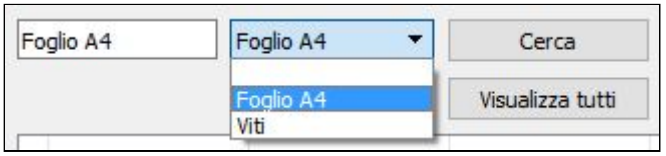
Nella voce "**Impostazioni"** del menu viene data la possibilità di modificare il proprio account, disconnettersi ed eventualmente di salvare le modifiche apportate.

Per la gestione degli utenti è sufficiente cliccare sulla voce "**Gestione"** del menù e successivamente la voce "**Utenti**". Verrà quindi mostrata l'interfaccia grafica per la gestione utenti, simile a quella dedicata agli oggetti.

Nella voce "**Aggiungi**" del menu superiore è possibile aggiungere utenti e oggetti, cliccando su "**nuovo utente"** o "**nuovo item**".



Per poter modificare un oggetto o utente, bisogna prima selezionarlo; per far ciò bisogna inserire il nome nel editor di testo e cercarlo oppure selezionarlo dalla tendina affianco.



Attenzione, il programma non salva in automatico tutte le modifiche, quindi sarà necessario salvare prima della chiusura.

5- Ore utilizzate

Progettazione: 4h
Progettazione grafica: 5h
Scrittura codice: 13h
Scrittura gui: 29h
Test: 3h
Debug: 1h
Documentazione: 12h
TOTALE: 67h

6- Ambiente di sviluppo

Sistema operativo: Windows 10 Pro 64-bit
Compilatore: MinGW 5.3.0 32bit for C++
QT: Qt 5.7.1 MinGW 32bit
Qmake: 3.0