RELAZIONE PROGETTO PROGRAMMAZIONE AD OGGETTI

Anno accademico 2020/2021

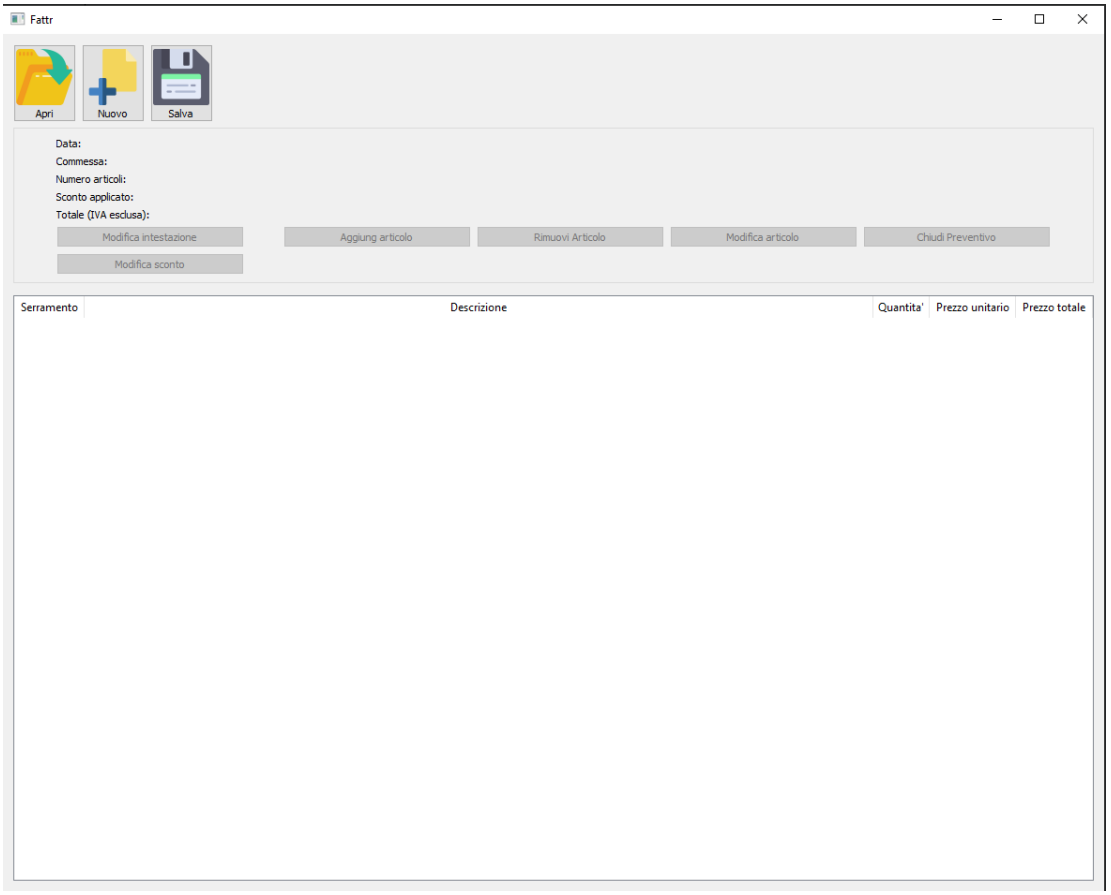
Michele Gatto 1224458

1. Introduzione

Fattr è un applicativo usato per la gestione di preventivi di serramenti in legno, pensato per la falegnameria dove lavoro. Il programma è pensato per essere estendibile, permettendo la possibilità di aggiungere nuovi tipologie di serramenti e/o materiali.

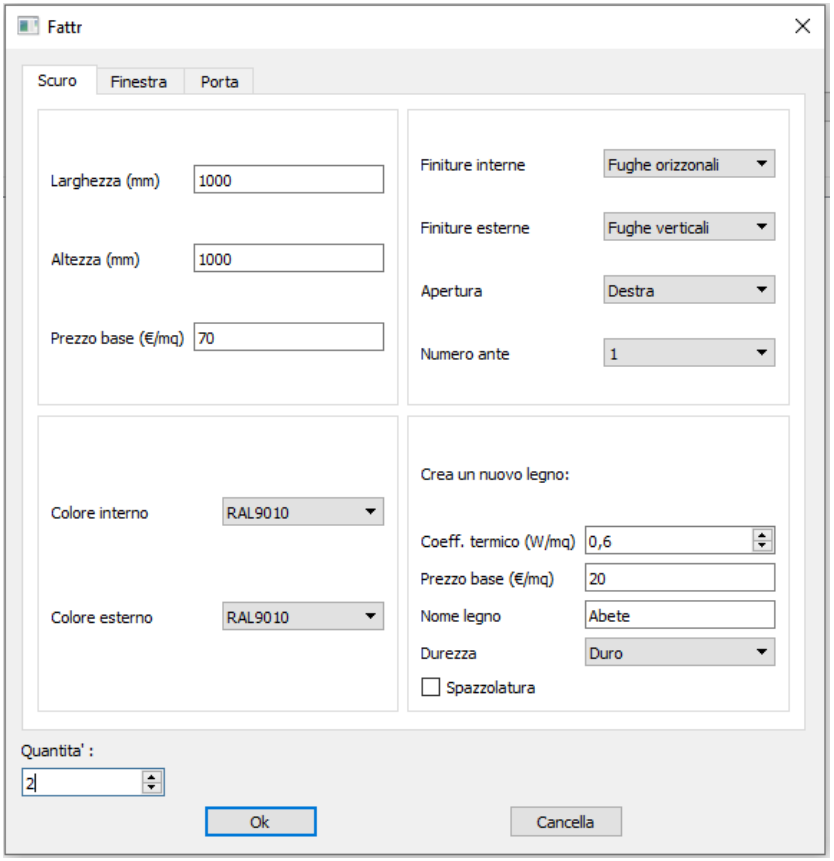
2. Manuale d'uso GUI

All'avvio del programma ci sarà la seguente schermata di avvio, che riempirà tutto lo schermo (si è deciso così per migliorare la visualizzazione, anche se la schermata è modificabile fino ad una certa misura).



Nella toolbar in alto a destra si possono notare i 3 bottoni per la gestione del preventivo, "Apri" che permette di aprire un preventivo salvato in memoria (saranno salvati in formato

Json), “Nuovo” che permette di creare un nuovo preventivo vuoto e “Salva” che permette di salvare il preventivo corrente in memoria in formato Json.
I bottoni per la modifica del preventivo sottostanti sono disabilitati fintanto che non c’è nessun preventivo da gestire.
Dopo aver creato un nuovo preventivo con “Nuovo”, possiamo notare come la data venga inizializzata alla data di creazione e la commessa a “Nuovo preventivo del *data creazione* ”. Ora che i bottoni sono abilitati cominciamo ad aggiungere articoli premendo “Aggiungi articolo”. Ci comparirà un dialogo simile a questo:



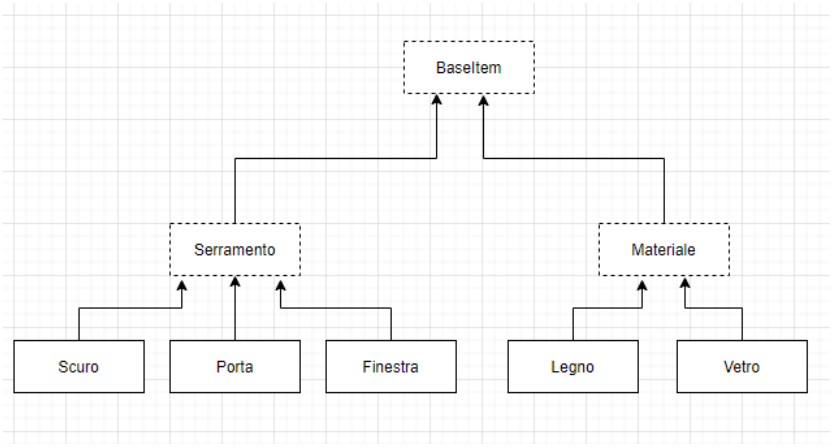
Si può sfogliare i vari prodotti disponibili e, trovato quello che desideriamo, procedere ad inserire le informazioni richieste. A lato un esempio con il prodotto “Scuro”. Quando si è terminato si può premere “Ok”, quindi dopo aver verificato la sanità dei dati inseriti, l’articolo (serramento + quantità) verrà aggiunto al preventivo corrente.

Nella schermata sottostante si può vedere l’articolo correttamente aggiunto con alcune sue caratteristiche base visualizzate.
Per visualizzare dettagli aggiuntivi basta fare click sulla cella “Descrizione” dell’articolo a cui si è interessati.

Il resto è molto intuitivo, “Rimuovi articolo” e “Modifica articolo” richiedono l’indice del articolo su cui effettuare l’operazione (lo si può recuperare nell’intestazione verticale della tabella), “Chiudi preventivo” appunto elimina il preventivo corrente ma non salva i dati, (ci sarà un messaggio per ricordarlo all’utente), “Modifica sconto” permette di applicare lo sconto sul totale finale del preventivo (prezzi unitari e totali dei singoli articoli non vengono modificati) e “Modifica intestazione” permette di cambiare commessa o data del preventivo.

3. Gerarchia del Modello

La gerarchia del modello è relativa agli elementi gestiti dal programma, in particolare la classe base astratta “BaseItem” definisce solo il distruttore virtuale della gerarchia e due metodi virtuali puri di base: clone() (per il funzionamento del “DeepPtr” e della copia profonda) e Serialize(QJsonObject&) per la gestione della serializzazione dei tipi della gerarchia.
Da “BaseItem” derivano 2 classi (astratte anche loro): “Serramento” e “Materiale” che rappresentano rispettivamente gli infissi in legno prodotti dall’azienda e i materiali principali utilizzati.
“Serramento” definisce i metodi virtuali puri PrezzoFinale(), NomeProdotto(), TrasmittanzaTermica() e MetriQuadri(), mentre “Materiale” definisce i metodi virtuali puri CoeffTermico() e Prezzo(). Infine ci sono 5 classi concrete istanziabili “Scuro”, “Porta”, “Finestra” che rappresentano i serramenti prodotti dall’azienda (e derivano da “Serramento”) e “Legno”, “Vetro” che rappresentano i materiali usati per la costruzione dei serramenti (e derivano da “Materiale”). Da notare che “Serramento” (e tutte le sue classi derivate) contengono un puntatore a “Legno” mentre “Finestra” contiene anche un puntatore a “Vetro” (relazione has-a), questo viene allocato e gestito internamente alla gerarchia con copia, assegnazione e distruzione profonda. La gerarchia è estendibile in quanto basta aggiungere nuovi serramenti (in legno) ed eventuali materiali ed implementare i metodi richiesti.



4. Chiamate polimorfe

-clone(): usato per il funzionamento del DeepPtr;

-Serialize(QJsonObject&): usato per serializzare i dati della gerarchia, quindi nel salvataggio del preventivo viene fatta scorrere la lista degli articoli contenuti nel preventivo e chiamato il metodo (nel modello);
-PrezzoFinale(): usato per il calcolo del prezzo finale unitario del serramento, ogni serramento aggiunge specifici sovrapprezzi a seconda di determinate caratteristiche desiderate (utilizza MetriQuadri() internamente);.
-MetriQuadri(): usato per la visualizzazione dei metri quadri, poiché spesso il calcolo può variare a seconda del prodotto (ad es. per la finestre), in generale se minore di 1,5 mq viene messo a 1,50 (almeno per i serramenti contemplati);
-NomeProdotto(): usato per restituire una stringa che identifica il prodotto (usato per la visualizzazione degli articoli nella GUI);
-TrasmittanzaTermica(): calcolo della trasmittanza termica (dispersione di calore calcolata in W/mq), dove ogni serramento ha un calcolo e riduzioni (o aumenti) percentuali diversi (NB: i calcoli sono MOLTO semplificati e adattati in sede di progetto);
-CoeffTermico(): il corrispettivo della trasmittanza per i serramenti (viene usato internamente da TrasmittanzaTermica() poiché anche il materiale rientra del calcolo della stessa), anche qui varia profondamente a seconda del materiale considerato;
-Prezzo(): calcola il prezzo unitario totale del materiale in questione applicando tutti i sovrapprezzi dovuti, è separato da PrezzoTotale() in quanto è considerato per mq.

5. Gerarchie e struttura parte grafica

Nella GUI c'è una piccola gerarchia “WidgetItemGenerico” (classe base astratta) dalla quale derivano 3 classi concrete: “WidgetScuri”, “WidgetPorte”, “WidgetFinestre”. Questa gerarchia esiste semplicemente per il riutilizzo di codice (in particolare per la generazione di warning) e per fornire una struttura base da seguire per la creazione delle tabs dei prodotti della gerarchia del modello.
La finestra principale della vista è costituita da “MainWindow” che costruisce la toolbar superiore con i vari bottoni, l’intestazione del preventivo e la tabella (QTableWidget) dove vanno messi gli articoli. “MainWindow” contiene 3 puntatori inizializzati con i 3 QDialog principali “DialogProdotti”, “DialogModificaIntestazione” e “DialogMostraDettagliSerramento”.
“DialogModificaIntestazione” si occupa della gestione della finestra che modifica l’intestazione, usando il controller per la modifica della stessa.
“DialogMostraDettagliSerramento” è una semplice finestra che mostra del testo, in questo caso mostra tutti i dettagli relativi al prodotto cliccato.
“DialogProdotti” è la più complessa e gestisce la finestra di aggiunta o modifica degli articoli. Essa contiene una QTabWidget le cui tab sono costituite dagli QWidget descritti sopra. Essi gestiscono la verifica dei dati e la generazione della nuova istanza del serramento selezionato, mentre “DialogProdotti” gestisce la scrittura del nuovo serramento tramite il controller.
Questi QDialog sono sempre riutilizzati e vengono nascosti o mostrati (e i campi riempiti o cancellati) dinamicamente a seconda della funzione che devono svolgere. Tutti i QDialog descritti sopra sono modali, cioè l’utente non può interagire con la finestra sottostante (quella principale in questo caso) fino a che non ha finito di operare con il QDialog.

6. Note sul pattern utilizzato

Per l'applicativo è stato usato un pattern simile al MVC. Utilizza la classica suddivisione in model (che contiene tutti i dati), view (che raccoglie l'input da utente e si occupa di visualizzare il model) e controller (che riceve i comandi dell'utente dalla view e applica le dovute modifiche al model). Il pattern è stato un po' modificato secondo le esigenze del progetto. In particolare la view prende i dati da visualizzare direttamente dal model, tramite un puntatore costante ad esso, const Model*, il `const` si è reso necessario per rafforzare la promessa che la view non modifica il model direttamente, così facendo la view potrà chiamare solo metodi costanti del model che userà per ottenere i dati da visualizzare direttamente, senza passare per il controller. Il controller a sua volta è il componente che comunica direttamente con il model e che può modificarlo. Quindi la view dovrà mappare l'input dell'utente con i comandi del controller per la modifica del model, garantendo la massima separazione.

7. Note sul contenitore sviluppato

Il contenitore usato per il programma è la linked list singola per garantire un ridotto uso della memoria permettendo di avere una lista dei prodotti flessibile con possibilità di rimozione e aggiunta degli elementi. La lista è templatizzata ad un DeepPtr (anche esso templatizzato) per la gestione efficiente della memoria.

7. Input/Output

Il programma usa il formato file Json per il salvataggio dei dati. In particolare salva tutte le informazioni relative al preventivo corrente su un JsonObject. Dopo aver salvato tutta la lista degli articoli (indicizzata nel JsoNObject tramite chiavi corrispondenti alla loro posizione nel preventivo), la data e la commessa del preventivo il JsonObject viene scritto su file.
L'apertura di un file .json con i dati del preventivo viene gestita da una classe a parte "PreventicoJsonReader" che genera il preventivo dal file selezionato e lo carica nel programma.

8.Tempo impiegato

Di seguito il monte ore impiegato per lo sviluppo del programma:
-analisi del problema: ~3h
-progettazione modello e GUI: ~10h
-apprendimento libreria Qt: ~3h (già usata in precedenza)
-codifica modello e GUI: ~35h
-debugging, testing: ~3h
-scrittura relazione: ~2h
TOTALE: ~56h
Il tempo è stato leggermente sforato (~6h) a causa di un refactoring in corso d'opera con il taglio di alcune feature che erano previste ma che non potevano essere sviluppate in tempo, il tutto ha causato la necessità di ristrutturare il modello e le gerarchie e infine una tentata (e abbandonata) ottimizzazione prematura che ha generato problemi a cascata.

9. Istruzioni di compilazione

Come da richiesta il progetto compila con i seguenti comandi in ambiente Linux (usando il file Fattr.pro fornito):

```
qmake Fattr.pro
make
```

Per eseguire l'applicativo dopo la compilazione basta usare il comando:
`./Fattr`

10. Ambiente di sviluppo

Ubuntu 18.04 (virtual machine)	GCC 7.4.0	Qt 5.9.5
Windows 10 Pro	GCC 6.3.0	Qt 5.9.5

11. Crediti icone

Le icone usate sono state prodotte da diversi autori di https://www.flaticon.com/, tranne il logo che ho fatto io.