chartando

nini-byte/chartando

Introduzione

chartando è un'applicazione che permette all'utente di realizzare grafici relativi all'andamento delle vendite di un centro sportivo.

Vengono monitorate due tipologie di vendite:

- standard, in cui viene venduto un abbonamento che ha un costo mensile e che dura un determinato numero di mesi
- promozionale, in cui al costo mensile viene applicata una percentuale di sconto per un determinato numero di mesi

Informazioni generali

File forniti

Il progetto verrà consegnato come archivio chartando.zip, contenente:

- file ch_*.h e ch_*.cpp
- main.cpp
- chartando.pro
- file *.png contenuti nella cartella ch_icons
- ch_sampledata.xml, ch_bigsampledata.xml

Compilazione ed esecuzione

Per compilare ed eseguire l'applicazione è sufficiente lanciare i seguenti comandi da terminale, all'interno della cartella chartando:

- qmake
- make
- ./chartando

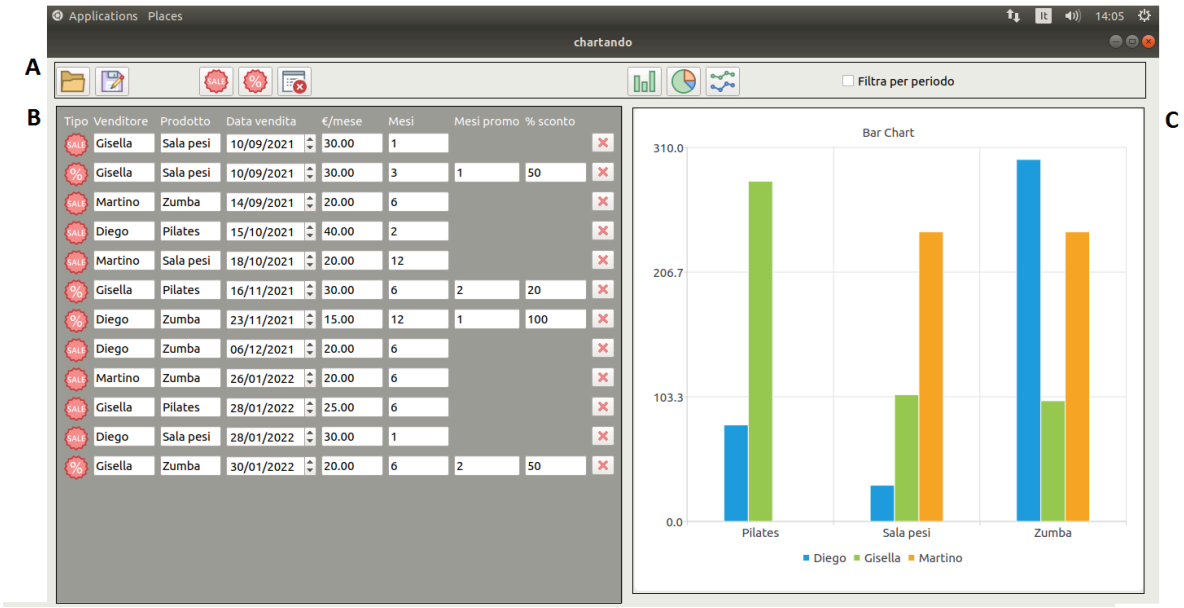
Ambiente di sviluppo

- sistema operativo: Ubuntu 18.04.3 LTS
- compilatore: gcc (Ubuntu 7.4.0-1ubuntu1~18.04.1) 7.4.0
- libreria Qt: QMake version 3.1 Qt version 5.9.5
- strumenti: IDE QtCreator (sì), QtDesigner (no)

Manuale utente

L'applicazione è formata da una finestra unica, che contiene al suo interno:

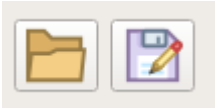
A. menu utente
B. dataset attualmente caricato
C. eventuale grafico visualizzato



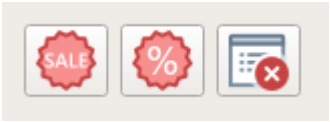
E' consigliata una visualizzazione a schermo intero, ma è stata gestita anche la possibilità di ridimensionare la finestra principale.

Il menu utente è così composto:

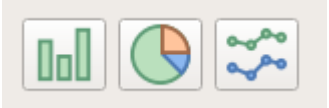
- importazione ed esportazione del dataset



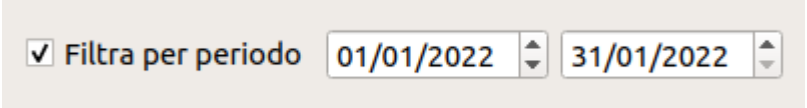
- aggiunta di una nuova vendita, di una nuova vendita promozionale, pulizia dell'intero dataset



- visualizzazione di un grafico a scelta tra bar chart, pie chart, line chart



- possibilità di impostare un range di date in base alle quali costruire il grafico



Sono state fornite infine le funzionalità di modifica ed eliminazione delle singole righe del dataset; ad ogni modifica del dataset o del range di date impostato il grafico verrà aggiornato automaticamente.

Architettura

Pattern MV

Per separare il modello dalla vista è stato scelto il pattern MV.

La GUI è stata realizzata interamente con Qt. Essendo l'applicazione formata da una finestra principale unica, *ch_SaleView*, è proprio quest'ultima che si occupa di monitorare i vari eventi legati all'interazione con l'utente tramite i *signal* e di dare la risposta corretta richiamando gli opportuni *slot*. La maggior parte dei signal utilizzati sono quelli forniti da Qt, mentre gli slot sono stati opportunamente definiti nella classe.
ch_SaleView utilizza la gerarchia *ch_Sale* come modello principale dell'applicazione, la gerarchia *ch_Chart* per la costruzione dei vari grafici, e la classe ausiliaria *ch_SaleParser* che offre le funzionalità di importazione ed esportazione del dataset.

Gerarchie di classi

Per il progetto sono state realizzate due gerarchie di tipi.

ch_Sale
    ch_PromoSale: public ch_Sale
ch_Sale rappresenta una vendita “standard”, di cui interessa memorizzare il nome del venditore e dell’abbonamento venduto, la data dell’operazione, la durata (in mesi) dell’abbonamento e il costo mensile. La classe è dotata di un distruttore virtuale *~ch_Sale()* e del metodo *getSaleTotal()*, anch’esso virtuale, che restituisce il totale della vendita.
ch_PromoSale rappresenta una particolare tipologia di vendita, “promozionale”, in cui in cui al costo mensile dell’abbonamento viene applicata una percentuale di sconto, ma solo per un determinato numero di mesi sul totale.

> *Per esempio, su un abbonamento annuale che costa 20€/mese, i primi due mesi potrebbero avere uno sconto del 50%.*

ch_Chart
    ch_BarChart: public ch_Chart
    ch_PieChart: public ch_Chart
    ch_LineChart: public ch_Chart

ch_Chart è una classe astratta che rappresenta un grafico generico, ed è dotata di un distruttore virtuale *~ch_Chart()* e del metodo *draw(...)*, virtuale puro, che si occupa di calcolare il grafico.
ch_BarChart rappresenta un grafico a colonne, che mostra il totale venduto raggruppato per venditore e prodotto venduto - utile per capire chi vende cosa.
ch_PieChart rappresenta un grafico a torta, che mostra il totale venduto raggrupato per venditore - utile per capire chi sono i venditori che contribuiscono maggiormente al totale delle entrate del centro.
ch_LineChart rappresenta un grafico a linee, che mostra l'andamento del totale venduto, mese per mese - utile per monitorare le entrate complessive.

Chiamate polimorfe

Il codice sfrutta il polimorfismo in diversi punti.

chart->draw(...) è una chiamata polimorfa che viene fatta all'interno dello slot *ch_SaleView::drawChart(...),* riga 371. *chart* è un puntatore a *ch_Chart*, e di conseguenza viene invocata la versione di *draw(...)* corretta in base al suo tipo dinamico, ovvero alla tipologia di grafico selezionata dall'utente.

In questo modo è stato possibile creare un solo slot *drawChart(...)*, il quale viene richiamato per tutte le tipologie di grafico previste; inoltre, nel caso in cui si decidesse di aggiungere un'altra tipologia di grafico nell'applicazione, questo slot non avrebbe bisogno di manutenzione.

*(*it)->getSaleTotal()* è un'altro esempio di chiamata polimorfa, stavolta all'interno dei vari override di *ch_Chart::draw(...), per esempio in ch_BarChart::draw(...), riga 30*. Dal momento che il dataset è contenuto in un vettore di puntatori a *ch_Sale*, il metodo *getSaleTotal()* viene invocato in base al tipo dinamico di ogni *ch_Sale**.

Anche in questo caso, dal momento che il metodo *draw(...)* non dipende dalle sottoclassi di *ch_Sale*, è possibile estendere la gerarchia delle vendite senza che questo richieda un intervento sulla gerarchia dei grafici.

Formati di I/O

Per l'importazione ed esportazione del dataset è stato scelto il formato XML.
Più in dettaglio, la classe *ch_SaleParser* offre i metodi *readFromFile(...)* e *saveToFile(...)* che sfruttano le funzionalità fornite da Qt per la costruzione e interpretazione dei dati in formato .xml, per esempio la classe *QDomDocument*.

Contenitore per il dataset

Il dataset è formato da oggetti della gerarchia di *ch_Sale*, ovvero *ch_Sale* e *ch_PromoSale*. Come contenitore è stato scelto il *QVector<T>* fornito da Qt, quindi tutte le vendite sono contenute in un *QVector<ch_Sale*>*.

Tenendo in considerazione anche le necessità di efficienza dell'applicazione, è stato scelto un vettore. Infatti, nonostante sia prevista la possibilità di eliminare una riga del dataset in qualunque posizione, l'operazione più frequente sarà l'inserimento in coda e pertanto verrà sfruttata la caratteristica di inserimento in coda in tempo ammortizzato costante del vettore.

Robustezza

L'applicazione è stata progettata con una particolare attenzione alla gestione dell'interazione con l'utente, implementando gli opportuni controlli. L'utente non potrà pertanto generare vendite con dati incompleti o incoerenti, creare grafici selezionando un periodo non valido, e così via. In questi casi, l'utente viene informato sul problema riscontrato e sulle possibili soluzioni.

E' stata infine gestita la minima quantità possibile di eccezioni in modo da mantenere la leggibilità del codice prodotto. La classe *ch_SaleParser*, infatti, è l'unica che contiene istruzioni *throw*; in questo caso la gestione delle eccezioni è indispensabile dal momento che, interfacciandosi con il file system, è possibile che si verifichino situazioni imprevedibili.