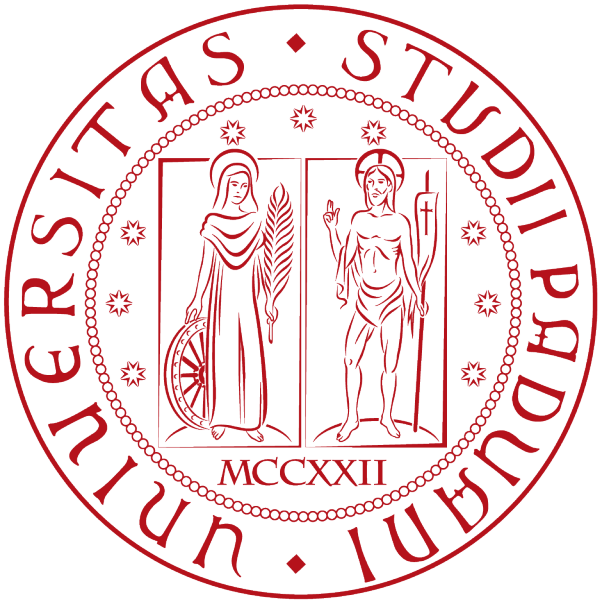

Relazione PAO

Alessia Bragagnolo - 20 Giugno 2017

Tiplogia Utente	Username	Password
Amministratore	admin	admin
Cameriere	cameriere	cameriere
Cuoco	cuoco	cuoco

Indice

1 Scopo del progetto

Il progetto realizzato ha il fine della creazione di un'applicazione semplice per la gestione del personale e dei tavoli all'interno di un ristorante/pub, dando alle diverse categorie di utente permessi differenti.

Gli utenti si dividono in: admin, cuochi e camerieri. I tavoli possono essere di tipo pub oppure ristorante. Le consumazioni possono avere tipo bar oppure cucina. Ogni tavolo può avere delle consumazioni, inserite dall'utente admin o dai camerieri. La creazione di nuovi tavoli e degli utenti è riservata all'utente admin, il quale può anche cambiare i dati personali dei camerieri, dei cuochi e di se stesso.

Ogni utente può però cambiare i proprio dati personali.

2 Compilazione ed esecuzione

Per compilare il progetto è necessario posizionarsi all'interno della cartella dello stesso, eseguire il comando **qmake -makefile**, eseguire il comando **make** e successivamente **./Ristorante**. Vengono consegnati anche due file, databaseutenti.xml e databasetavoli.xml, contenenti dei dati esempio. Nel caso in cui venissero cancellati è previsto che si crei il file denominato databaseutenti.xml con all'interno un utente amministratore con le credenziali di accesso:

- **username**: admin
- **password**: admin

3 Struttura

3.1 Informazioni Generali

Per la modellazione dei database di Utenti e Prodotti è stata utilizzata la struttura dati **list** messa a disposizione della libreria STL. Il database degli utenti è una semplice lista che contiene puntatori agli utenti, che possono essere di tipo differente, mentre il database dei tavoli è una lista di puntatori a tavoli, che possono essere di tipo differente. Ogni tavolo può contenere un puntatore ad una lista di consumazioni. La gestione del garbage è affidata alle singole funzioni di eliminazione.

3.2 Pattern MVC

Durante la realizzazione del progetto si è cercato di seguire il più possibile l'architettura del pattern Model-View-Controller, separando l'implementazione grafica da quella logica. I primi controlli nell'inserimento dati vengono affidati alla View, mentre quelli derivati da vincoli logici al controller. Ogni classe è dotata di opportuni metodi "get" e/o "set".

3.3 Implementazione GUI

Per quanto riguarda l'implementazione dell'interfaccia grafica, poiché non significativamente complessa, la codifica è stata effettuata "a mano", senza utilizzare quindi tool del

Framework di Qt. Al fine di evitare ripetizione eccessiva di codice alcune finestre adattano diversamente il loro layout in base alla necessità. Le varie schermate utilizzano un oggetto **QGridLayout** per disporre in maniera ordinata gli oggetti al loro interno. La gestione della distruzione degli oggetti della GUI di ogni view, viene affidata alla view stessa, poiché ogni oggetto viene aggiunto al Layout della view stessa.

3.4 Gerarchia Utenti

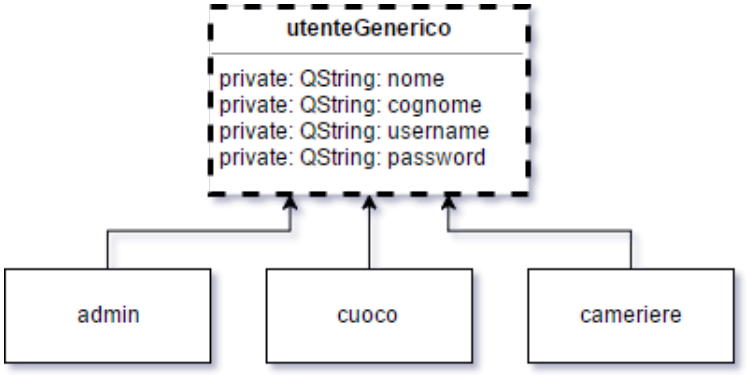

Figura 1: Gerarchia utenti

3.4.1 Base astratta utente generico

Rappresenta l'astrazione del concetto utente, oltre agli attributi riportati nella figura precedente presenta i seguenti metodi:

- `virtual QString getTipo() const=0`: restituisce una QString contenente il tipo dell'utente, utilizzato al fine di non usare eccessivamente dynamic-cast ove non strettamente necessario;

- `virtual bool canDoTavolo()const=0`: restituisce un booleano a seconda che l'utente da cui viene invocato abbia permessi di modifica, aggiunta e/o rimozione sui tavoli;

- `virtual bool canDoConsumazioni()const=0`: restituisce un booleano a seconda che l'utente da cui viene invocato abbia permessi di modifica, aggiunta e/o rimozione sulle consumazioni;

- `virtual bool canDoGestione()const=0`: restituisce un booleano a seconda che l'utente da cui viene invocato abbia permessi di modifica, aggiunta e/o rimozione sugli utenti;

- `virtual void saveUtente(QXmlStreamWriter& xmlWriter)=0`: metodo virtuale puro che si occupa di procede al salvataggio dei dati utenti nel modo corretto, in relazione a come implementato per ogni sottoclasse;

- `virtual void loadUtente(QXmlStreamReader& xmlReader)=0`: metodo virtuale puro che si occupa di procede alla lettura dei dati utenti nel modo corretto, in relazione a come implementato per ogni sottoclasse.

I dati comuni a tutti gli utenti vengono caricati con i
metodi `void loadBaseUtente(QXmlStreamReader& xmlReader)`
e scritti nel file con `void saveBaseUtente(QXmlStreamWriter& xmlWriter)`.

Le classi *admin*, *cameriere*, *cuoco* si occupano di implementare i metodi virtuali della base, conferendo ad ogni utente diversi permessi.

Nel dettaglio:

Ruolo	Funzionalità
admin	• ricerca di tavoli, consumazioni e utenti • gestione(aggiunta, modifica e rimozione) tavoli, consumazioni e utenti • gestione dei propri dati utente
cameriere	• ricerca di tavoli, consumazioni e utenti • gestione(aggiunta, modifica e rimozione)consumazioni • gestione dei propri dati utente
cuoco	• ricerca di tavoli, consumazioni e utenti • gestione dei propri dati utente

Tabella 1: Riepilogo funzionalità

3.5 Gerarchia Tavoli

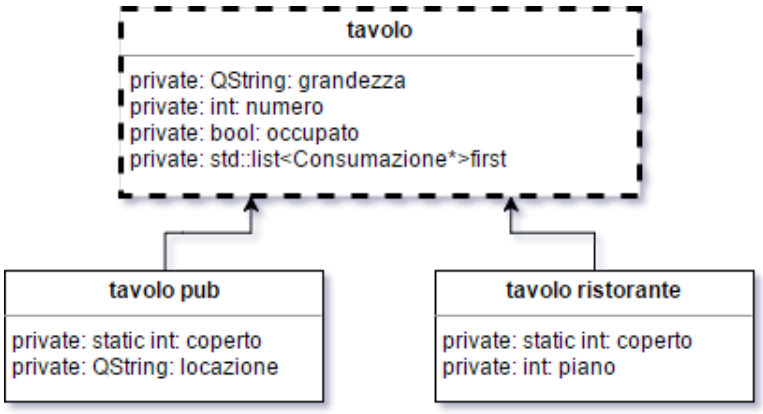

Figura 2: Gerarchia tavoli

3.5.1 Base astratta tavolo

Rappresenta l'astrazione del concetto di tavolo, oltre agli attributi riportati nella figura precedente presenta i seguenti metodi:
• `virtual int getPosti() const=0`: restituisce un intero che rappresenta i posti disponibili in un tavolo a seconda della grandezza dello stesso;
• `virtual double getCoperto() const =0`: restituisce un double rappresentante il prezzo del coperto per il tavolo;
• `virtual QString getTipo()const=0`: restituisce una QString contenente il tipo del tavolo, utilizzato al fine di non usare eccessivamente dynamic-cast ove non strettamente necessario;
• `virtual void saveTavolo(QXmlStreamWriter & w)=0`: metodo virtuale puro che si occupa di procede al salvataggio dei dati del tavolo nel modo corretto, in relazione a come implementato per ogni sottoclasse;
• `virtual void loadTavolo(QXmlStreamReader & r)=0`: metodo virtuale puro che si

occupa di procede alla lettura dei dati utenti nel modo corretto, in relazione a come implementato per ogni sottoclasse.

I dati comuni a tutti i tavoli vengono caricati con i
metodi `void loadBaseTavolo(QXmlStreamReader& r))`
e scritti nel file con `void saveBaseTavolo(QXmlStreamWriter& xmlWriter)`.

Non vengono salvate su file le consumazioni aggiunte ai tavoli.

3.5.2 Classe concreta tavolo pub

Ogni tavolo pub ha un campo dati statico constante che rappresenta il coperto. Questa classe modella la rappresentazione di un tavolo nel ristorante con una locazione che può essere esterna o interna.

3.5.3 Classe concreta tavolo ristorante

Ogni tavolo ristorante ha un campo dati statico constante che rappresenta il coperto. Questa classe modella la rappresentazione di un tavolo nel ristorante che si trova ad un piano indicato dall'utente.

3.6 Classe concreta consumazioni

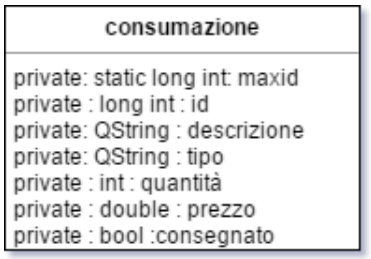

Figura 3: Classe consumazione

Modella gli oggetti che sono contenuti nei tavoli. Ogni consumazione ha un id univoco. Le consumazioni non vengono salvate nel file .xml. Alla terminazione del programma i dati a loro associati andranno persi.

4 Manuale GUI

Nel caso in cui i file non venissero trovati viene visualizzato un messaggio informativo. Dopo aver inserito le credenziali, se corrette, l'utente visualizza la finestra principale dalla quale se è di tipo admin può accedere alla gestione dei tavoli e degli utenti dalla barra del menu, altrimenti può effettuare il logout oppure cambiare i propri dati personali dopo aver selezionato l'opzione "**Opzioni**" dalla barra del menu. L'admin può accedere alle funzioni di modifica cliccando due volte sopra la riga dell'oggetto su cui intende effettuare le modifiche:

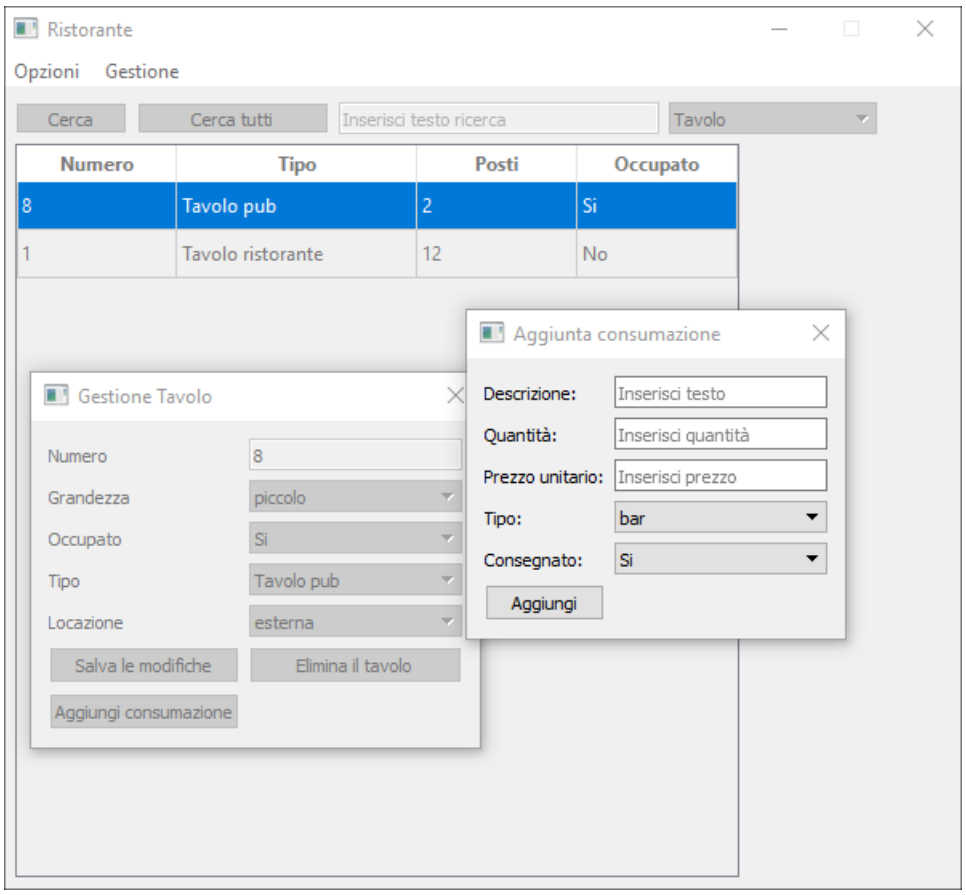

Figura 4: View gestione tavoli di un utente admin

Per gli utenti camerieri questa funzionalità è prevista solo per la modifica e l'eliminazione delle consumazioni, che può essere effettuata cliccando due volte sopra la riga della consumazione effettuata per eliminarla e/o modificarla mentre per l'aggiunta di una consumazione sarà necessario cercare il tavolo a cui si vuole aggiungere una consumazione e cliccare due volte sopra di esso.

Per gli utenti di tipo cuoco queste funzionalità non sono presenti, essi possono effettuare unicamente ricerche. La ricerca di un utente avviene per username oppure si possono visualizzare tutti gli utenti. La ricerca di un tavolo avviene per numero oppure si possono visualizzare tutti i tavoli. La ricerca di una consumazione avviene per numero tavolo oppure digitando la descrizione di una consumazione che si vuole ricercare o parte di essa; la ricerca di una descrizione vuota equivale alla ricerca di tutte le consumazioni.

Nell'inserimento dei dati utente vengono fatti alcuni controlli con espressioni regolari al fine di permettere l'inserimento di sole lettere nei campi **nome**, **cognome**, per **username** è permesso l'inserimento anche di numeri, mentre per **password** non avvengono controlli.

Nell'inserimento dei dati tavoli vengono fatti alcuni controlli con espressioni regolari al fine di permettere l'inserimento di soli numeri nei campi **numero** e **piano**.

Nell'inserimento dei dati consumazioni vengono fatti alcuni controlli con espressioni regolari al fine di permettere l'inserimento di sole lettere nel campo **descrizione**, per **quantità** è permesso l'inserimento di soli numeri, mentre per **prezzo** è permesso l'inserimento di numero con al massimo due cifre decimali.

A Indicazioni conclusive

A.1 Impegno temporale

- **Progettazione model e GUI**:5 ore
- **Realizzazione model e GUI**: 48 ore
- **Debugging**: 4 ore
- **Test**: 4 ore
- **Totale**: 63 ore

A.2 Informazioni tecniche

- **Sistema Operativo**: Windows 10 Home
- **Compilatore**: MinGW 5.3.0 (32bit)
- **Versione Qt**: Qt 5.7