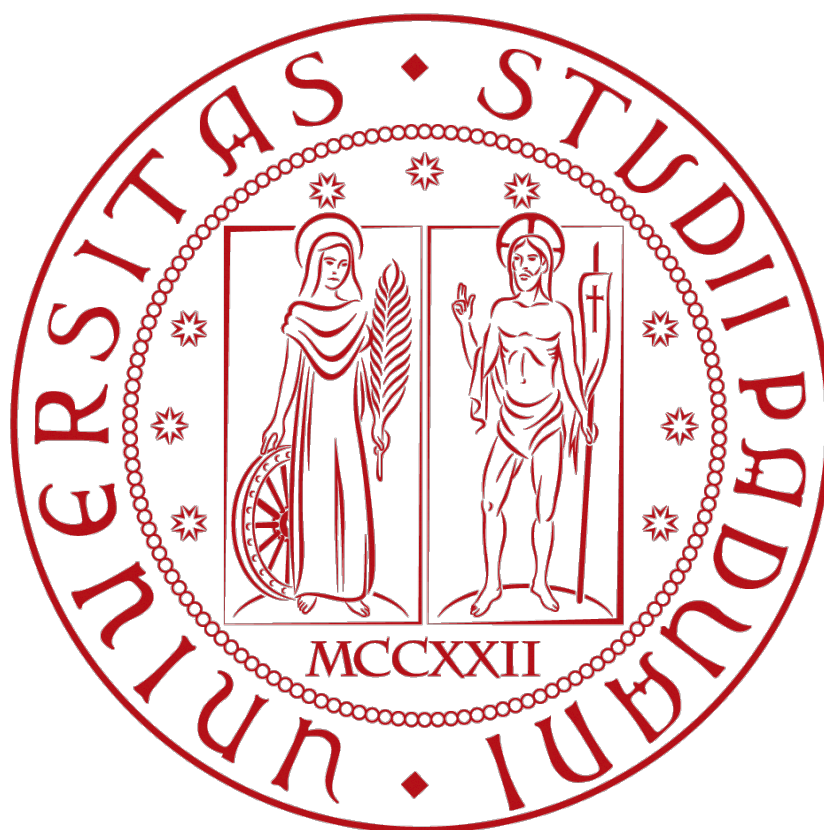


Progetto Programmazione ad Oggetti 2018/2019

Menù piatti ristorante



Vidotto Giovanni 1072624

# 1 Relazione

## 1.1 Introduzione

Il progetto rappresenta un gestionale ideato per la gestione dei piatti del menu di un ristorante. Attraverso un'interfaccia grafica è possibile gestire gli oggetti memorizzati con operazioni di visualizzazione, inserimento, modifica, eliminazione e ricerca. Il progetto è stato sviluppato utilizzando l'ide QtCreator nella versione 5.5.1 su un sistema operativo macOS Mojave e, una volta conclusa la parte di sviluppo, è stato testato anche sulla macchina virtuale fornita durante il corso di programmazione ad oggetti.

## 1.2 Modalità di consegna

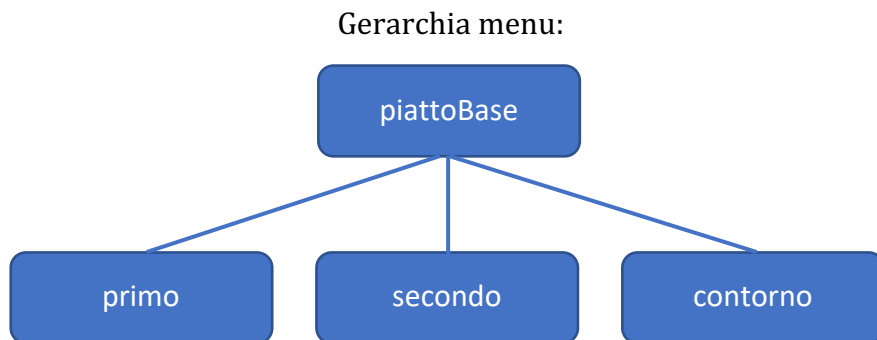
Il progetto è stato consegnato attraverso il comando "consegna progetto-pao-2019". All'interno sono presenti tutti i file necessari e il file.pro che si genera automaticamente da Qt all'avvio del progetto. Il file.pro è necessario per la compilazione attraverso "qmake" perché la compilazione attraverso il comando "qmake-project" genera un file non adatto. Questo per permettere la generazione automatica tramite "qmake" del "Makefile".

## 1.3 Descrizione classi e gerarchia

La gerarchia del progetto è costituita da una classe base astratta chiamata **piattoBase**, essa contiene tutti i campi comuni per ogni classe derivata.

Ci sono tre classi istanziabili che derivano dalla classe base **piattoBase** e sono:

- **primo**: rappresenta un primo piatto (di pasta) di un ristorante;
- **secondo**: rappresenta un secondo di un ristorante;
- **contorno**: rappresenta un contorno;



In seguito la descrizione di ogni classe:

### 1.3.1 Classe **piattoBase**

- **nome**: rappresenta il nome del piatto;
- **vegano**: indica se il piatto è vegano oppure no;
- **glutenFree**: indica se il piatto è senza glutine oppure no;
- **prezzo base**: indica il prezzo base da cui, a seconda del tipo, ogni piatto parte;
- **urlImmagine**: rappresenta il percorso dell'immagine rappresentativa del piatto;

### 1.3.2 Classe primo

- **soia:** indica se il piatto è a base di soia oppure no;
- **tipoPasta:** indica il tipo di pasta usato nel primo;
- **condimento:** indica il condimento del primo (es pomodoro, pesto ecc);
- **ingrediente1:** indica il primo ingrediente, oltre al condimento, se presente;
- **ingrediente2:** indica il secondo ingrediente, oltre al primo, se presente;
- **ingrediente3:** indica il terzo ingrediente, oltre al secondo, se presente;

### 1.3.3 Classe secondo

- **tipoCarnePesce:** indica il tipo di carne o pesce su cui è basato il secondo;
- **tipoPiatto:** indica come verrà cucinato il piatto (es alla griglia, alla piastra ecc);

### 1.3.4 Classe contorno

- **tipoContorno:** indica il tipo di contorno (es tipo di patate);

## 1.4 Qontainer

Per gestire l'inserimento dei dati è stato implementato un template di classe, chiamato container, con la funzione di contenitore. Il contenitore essendo templetizzato può contenere qualsiasi tipo di oggetto della gerarchia.

Il contenitore è stato implementato come una lista di nodi doppiamente linkata dove ogni nodo contiene tre campi: il campo info (templetizzato) che contiene l'informazione dei piatti, il campo prev che contiene il puntatore al nodo precedente ed il campo next che contiene il puntatore al nodo successivo. Nei campi dati della classe Contenitore sono presenti due variabili nodo\* di nome primo e ultimo, le quali contengono rispettivamente il puntatore al primo e ultimo nodo della lista, questo per gestire in modo più efficiente l'operazione di inserimento del nodo nella lista. Le funzioni principali all'interno del contenitore sono la pushBegin e la pushEnd servono per inserire, rispettivamente, all'inizio o fine della lista. Sono state create due classi, iteratore e constIteratore, la cui funzione è quella di iterare la lista in modo costante e non. La classe contenitore inoltre contiene altre funzioni come begin, end, erase le quali sono atte al funzionamento delle classi Iterator e ConstIterator. Nello specifico la funzione erase (utilizzabile solo dalla classe iteratore) andrà ad eliminare e sistemare i nodi della lista in un'operazione di eliminazione.

### 1.4.1 Model view controller

Per lo sviluppo dell'applicazione è stato usato lo standard model-view-controller:

- **model:** si occupa della gestione dei dati del menu;
- **view:** fornisce la parte grafica del menu;
- **controller:** gestisce i comandi utente e carica i dati del model nella view;

### 1.4.2 modello.cpp e modello.h

Nel modello sono presenti le funzioni per il caricamento dei dati (*caricamentoDati()*), il salvataggio dei dati (*salvataggioDati()*) ed una funzione che controlla se eventuali modifiche fatte sono state salvate nel file XML.

### 1.4.3 viewpiatti.cpp e viewpiatti.h

Nella viewpiatti vengono caricati i pulsanti che, una volta premuti, visualizzano la lista di piatti selezionata. Inoltre contiene un campo dato di tipo `listaPContainer*` che permette di visualizzare la lista dei piatti selezionata.

### 1.4.4 controller.cpp e controller.h

Nel controller viene gestito tutto il funzionamento principale dell'applicazione. Esso va a collegare il modello (dove sono memorizzate le informazioni) con la view (dove le visualizza) e visualizza i pulsanti principali nell'interfaccia utente.

## 1.5 Codice estensibile

Il progetto è diviso tra parte grafica e parte gerarchica. Ogni file è diviso in .h e .cpp, i file della parte grafica sono contenuti nella cartella "gui", mentre i file della parte gerarchica sono contenuti nella cartella "gerarchia". La maggior parte del codice è stato commentato per chiarire il funzionamento in modo da rendere più semplice un futuro riutilizzo del codice, anche in caso di ampliamento della gerarchia.

Le operazioni di inserimento, modifica e cerca avvengono tramite l'apertura di una finestra di dialogo, questo permette una buona separazione del codice da quello principale e, nel caso si voglia implementare meglio una di queste funzioni, basterà andare a modificare solo la pagina interessata.

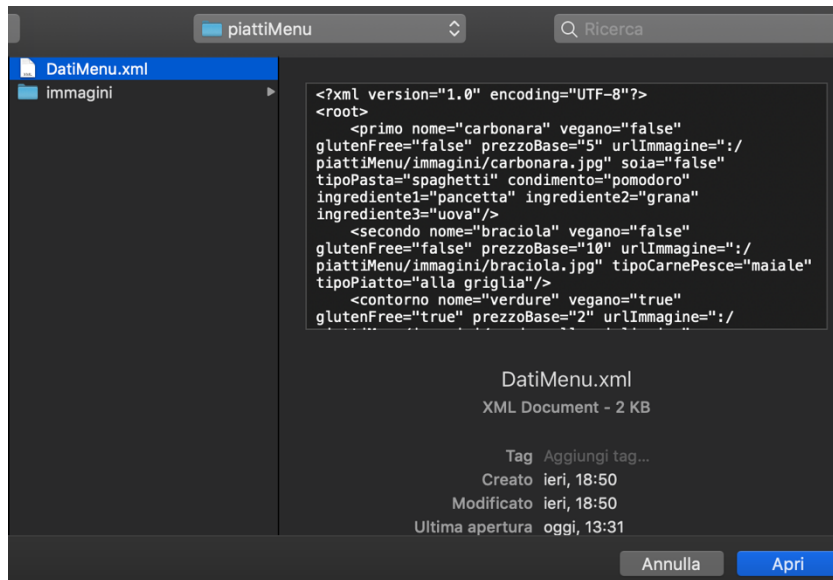
## 1.6 Metodi polimorfi

Sono stati implementati i seguenti metodi polimorfi nella gerarchia:

- **virtual ~ piattoBase()**: distruttore virtuale;
- **virtual double prezzoFinale() const**: metodo virtuale che calcola il prezzo finale del piatto in base al tipo dell'oggetto e dalle sue caratteristiche;
- **virtual string piattoInstring(piattoBase\*)**: funzione che restituisce una stringa completa di tutte le caratteristiche di ogni oggetto, utilizzata nella lista di visualizzazione per mostrare i dati dei piatti.
- **virtual operator==(const piattoBase\*)**: ridefinizione dell'operatore di uguaglianza, implementato in modo da controllare se due oggetti sono uguali e per facilitare la ricerca.

## 1.7 Avvio del progetto

All'avvio dell'applicazione il programma chiede di selezionare un file in formato .xml, questo file contiene tutti i dati che saranno caricati all'interno del nostro contenitore. Inoltre su questo file verranno salvati i dati relativi a nuovi inserimenti, eliminazioni e modifiche dei prodotti. Questo è possibile attraverso l'uso del file xml. Xml è un linguaggio di markup che consente la rappresentazione di documenti e dati strutturati su supporto digitale.

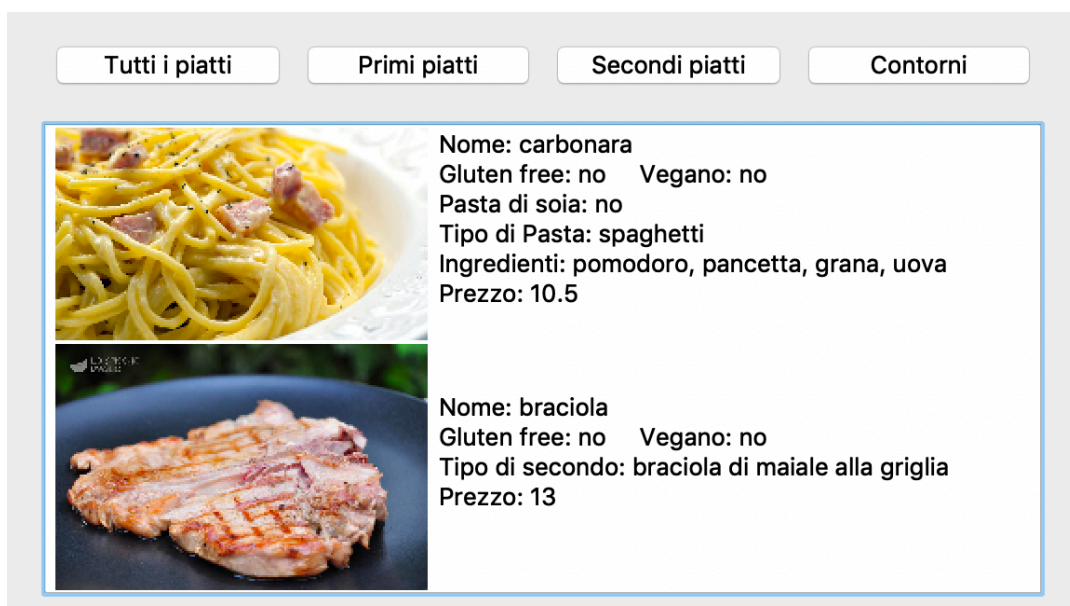


## 1.8 Interfaccia grafica

L'interfaccia grafica è composta una pagina principale contenente tutti i pulsanti e la lista dei prodotti e da tre nuove pagine che si aprono in caso di pressione dei tasti aggiungi, modifica e cerca.

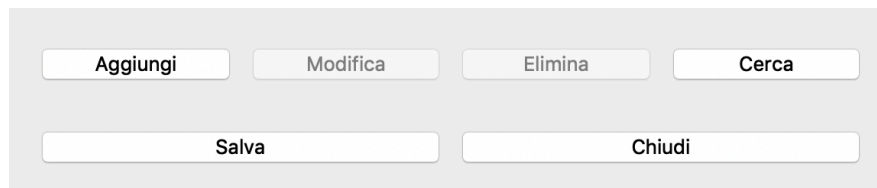
### 1.8.1 View piatti

La view dei piatti contiene i pulsanti necessari per filtrare e visualizzare i piatti in base al tipo.



## 1.8.2 Tasti principali

Subito sotto la view sono presenti i principali tasti, i tasti modifica ed elimina diventano cliccabili nel momento in cui viene selezionato un piatto



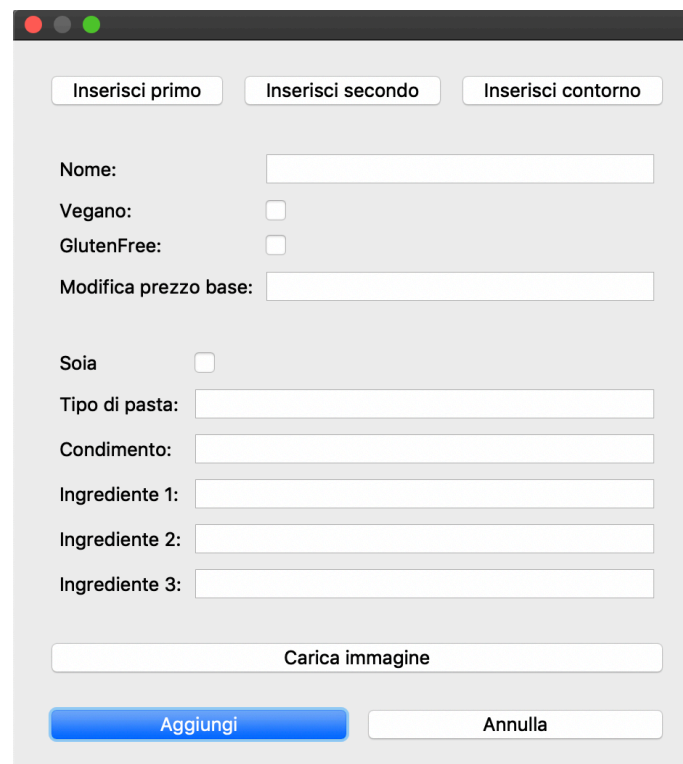
A horizontal bar containing six buttons: 'Aggiungi', 'Modifica', 'Elimina', 'Cerca', 'Salva', and 'Chiudi'.

## 1.8.3 Tasto aggiungi

La pressione del tasto “Aggiungi” provoca la creazione di una finestra di dialogo necessaria per l’inserimento di un nuovo piatto, si potrà selezionare il tipo di piatto che si vuole aggiungere, inserire i dati e caricare un’immagine rappresentativa.

Sono stati aggiunti i seguenti controlli:

- inserimento campi minimi per ogni tipologia di piatto, viene generato un warning nel caso in cui non accada con indicazioni su quali campi vanno obbligatoriamente popolati
- controllo inserimento doppio in base al nome per i primi piatti ed in base al nome e tipo di piatto per i secondi e contorni, non sarà possibile quindi avere due primi con lo stesso nome, due secondi con stesso nome e stessa tipologia di carne e pesce e due contorni con stesso nome e tipologia di contorno (esempio: “nome patate e tipo di contorno chips” è diverso da “nome patate e tipo di contorno alle erbe”, anche se il nome del piatto è comune)
- nel caso in cui non viene caricata un immagine ne viene settata una di default

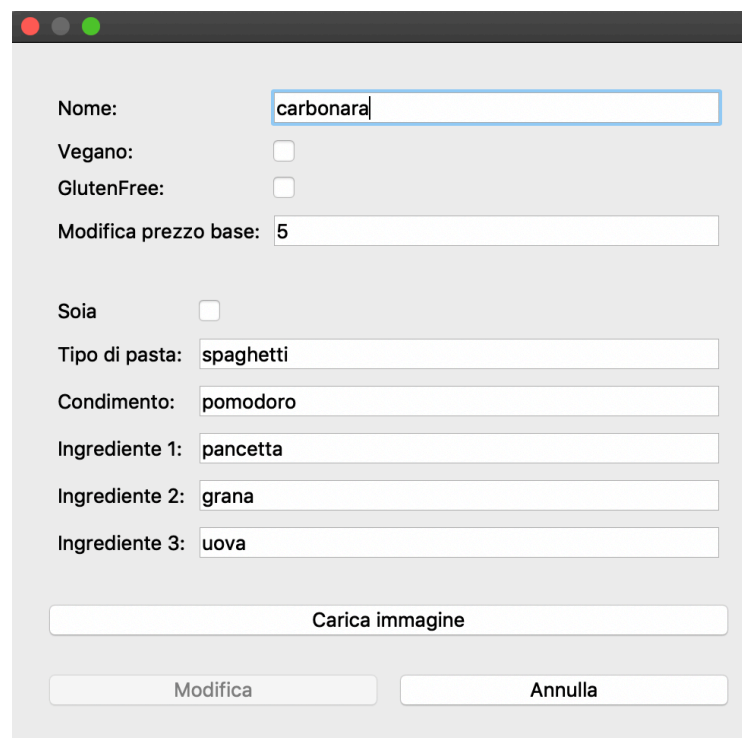


A dialog window for adding a new dish. It features three buttons at the top: 'Inserisci primo', 'Inserisci secondo', and 'Inserisci contorno'. Below these are several input fields: 'Nome:', 'Vegano:' (checkbox), 'GlutenFree:' (checkbox), 'Modifica prezzo base:', 'Soia' (checkbox), 'Tipo di pasta:', 'Condimento:', 'Ingrediente 1:', 'Ingrediente 2:', and 'Ingrediente 3:'. At the bottom, there is a 'Carica immagine' button, an 'Aggiungi' button (highlighted in blue), and an 'Annulla' button.

### 1.8.4 Tasto modifica

La pressione del tasto “Modifica” provoca la creazione di una finestra di dialogo necessaria per la modifica di un piatto già esistente. Sono stati aggiunti i seguenti controlli:

- controllo che i campi minimi per ogni piatti siano presenti anche dopo la modifica
- controllo sul duplicato di un piatto (esempio: prendo il primo “amatriciana” e provo a cambiargli il nome con “carbonara” che è già presente nella mia lista di piatti, questo comportamento provocherà un warning e resetta i dati a com’erano originariamente prima della modifica)
- il tasto “Modifica” diventa cliccabile nel momento in cui un campo viene realmente modificato



Nome: carbonara

Vegano: ☐

GlutenFree: ☐

Modifica prezzo base: 5

Soia ☐

Tipo di pasta: spaghetti

Condimento: pomodoro

Ingrediente 1: pancetta

Ingrediente 2: grana

Ingrediente 3: uova

Carica immagine

Modifica Annulla

### 1.8.5 Tasto elimina

Una volta selezionato un piatto dal menu il tasto elimina si occupa di eliminare dal contenitore il nodo del piatto selezionato e di sistemare i puntatori.

### 1.8.6 Tasto cerca

La pressione del tasto “Cerca” provoca la creazione di una finestra di dialogo necessaria per la ricerca di un piatto già esistente. Anche la ricerca è suddivisa in base al tipo di piatto e per iniziarla bisogna inserire almeno un dato.

In caso di esito positivo il/i piatto/i verranno caricati nella view principale.

### 1.8.6 Tasti salva ed esci

La pressione del tasto “Salva” provoca il salvataggio delle modifiche effettuate nel file xml in modo da rendere non volatili le modifiche fatte una volta chiuso e riaperto il programma.

Il tasto “Esci” permette di chiudere l'applicazione.

È stato aggiunto il seguente controllo:

- se tento di chiudere il programma senza aver salvato le eventuali modifiche viene generato un messaggio che chiede se salvare le modifiche ed uscire, annullare l'uscita oppure uscire senza salvare

## 1.9 Conclusioni ed ore di lavoro

Visto le ore a disposizione non ho curato molto l'aspetto grafico, infatti la grafica è rimasta quella standard generata da QtCreator ma ho preferito soffermarmi più sull'aspetto funzionale, inserendo tutti i tipo di controlli necessari per un menu di un ristorante.

Analisi dei requisiti	1 ora
Pianificazione contenitore e gerarchia	2 ore
Implementazione contenitore e gerarchia	7 ore
Implementazione e apprendimento Gui	40 ore
Debug su macchina virtuale	3 ore
Scrittura relazione	2 ore
Totale	55 ore