



UNIVERSITÀ DEGLI STUDI DI PADOVA

PROGETTO DI PROGRAMMAZIONE AD OGGETTI

SacredHeart

Luca Dal Medico

Matricola: 1099176

A.A. 2016/2017

Indice

1	Scopo del progetto	2
2	Descrizioni classi	2
2.1	Modello	2
2.2	Viste	3
2.3	Controller	4
3	Rappresentazione grafica del database	5
4	Descrizione dell'uso di codice polimorfo	5
5	Indicazione delle ore impiegate	6
6	Manuale utente della GUI	6
7	Ambiente di sviluppo	8
8	Materiale consegnato	8
9	Credenziali per i test	8
10	Risorse utilizzate	8

1 Scopo del progetto

L'idea alla base del progetto è quella di realizzare un'applicazione in *C++/Qt* per la gestione minimale dei dati all'interno di un ospedale. Viene fornita l'interfaccia da cui i dipendenti possono fare il login con le proprie credenziali e consultare i dati a cui hanno accesso. I tre tipi di utente presenti nel progetto sono:

1. **Primario:** ricopre il ruolo di amministratore, può esaminare tutte le visite attuali, consultare i pazienti o i dipendenti e modificarli, aggiungerli o eliminarli. Esso è l'unico utente che può aggiungere dati da file o esportarli in una cartella;
2. **Segreteria:** tipologia d'utente che gestisce gli appuntamenti e i pazienti: può aggiungere, cancellare o modificare i campi solo di questi due tipi;
3. **Medico:** utente base, può conoscere solo i propri appuntamenti e le informazioni dei pazienti delle sue visite.

2 Descrizioni classi

2.1 Modello

- **Database:** contiene tutti i dati riguardanti i pazienti, il personale e gli appuntamenti. Sono state usate differenti strutture per contenere i dati: per il personale ed i pazienti sono stati usati due *std::vector* perché, in seguito all'inserimento iniziale, le aggiunte o le modifiche ai dati sono rare, pensando ad una situazione reale; mentre per gli appuntamenti è stato usato una *std::list* perché è molto probabile che, nell'arco di breve tempo, vengano aggiunti diversi nuovi appuntamenti e modificati o eliminati quelli già esistenti. La classe permette anche di inserire dati da file esterni o di esportarli in una cartella. Inoltre è possibile, dati username e password, ottenere il dipendente che effettua il login. Le strutture dati contengono *std::shared_ptr* per avere una gestione ottimizzata della memoria. È stato deciso di utilizzare i puntatori smart della libreria standard (*shared_ptr* e *unique_ptr*) invece di definirne di nuovi per affidarsi alla correttezza della libreria standard, in particolare si è scelto gli *shared_ptr* per il conteggio del numero di riferimenti, anche se questo potrebbe appesantire il programma rispetto l'uso di *unique_ptr* o di puntatori normali;
- **Evento:** classe astratta per rappresentare gli appuntamenti. Non è stato aggiunto nessun campo per rappresentare il paziente per permettere eventualmente di creare nuove classi in cui non fosse necessario (ad esempio conferenze o riunioni);
- **Persona:** classe base per la rappresentazione di qualsiasi persona (viene utilizzata per un paziente generico);

- **Visita:** classe rappresentante una visita di un paziente con un medico. Per il paziente viene usato il tipo *Persona* in quanto contiene tutte le informazioni necessarie per descriverlo.
- **Donazione:** classe per rappresentare un appuntamento di un donatore presso il centro trasfusionale. Il metodo *durata* è impostato per riportare la distanza tra una donazione e quella precedente del donatore (3 mesi per gli uomini e 4 per le donne);
- **GruppoSanguigno:** classe per descrivere il gruppo sanguigno di un donatore (gruppo + fattore).
- **Medico:** classe che descrivere un medico, deriva da *Personale*. Permette la ricerca degli appuntamenti personali e di tutti gli appuntamenti di un paziente;
- **Primario:** classe che descrive il primario, deriva da *Medico*. Permette la ricerca di tutti gli appuntamenti, sia di un medico che di un singolo paziente;
- **Segreteria:** classe che descrive un lavoratore nel settore di segreteria, deriva da *Personale*. Permette la ricerca degli appuntamenti complessivi o di un singolo paziente.
- **Personale:** classe astratta, prevede due metodi virtuali puri che rappresentano la ricerca in una lista di eventi e la ricerca in una lista di eventi considerando una persona. Contiene la classe ausiliaria *Functor* utilizzata nelle ricerche. Deriva da *Persona*. È stato deciso di non rendere disponibile nessun metodo per mostrare la password, questo per aumentare la sicurezza nell'applicazione.
- **Donatore:** classe che descrive un donatore, deriva da *Persona*.

2.2 Viste

- **AzioniEvento:** classe base per tutte le finestre che fanno azioni sulla lista degli appuntamenti;
- **AzioniPersona:** classe per tutte le finestre che fanno azioni su una persona (può essere un paziente oppure un dipendente);
- **CambioChiavi:** finestra che dato un booleano permette la modifica dell'username (se si dà valore *true*) o della password (se si dà valore *false*) del dipendente che lo richiede;
- **InfoPersona:** finestra che mostra tutte le informazioni di un paziente o un dipendente, nel caso di un dipendente se il secondo campo è *true* permette di conoscere lo stipendio e se il terzo campo è *true* permettere di vedere lo username;

- **LoginWindow:** finestra per l'autenticazione di un dipendente;
- **PersonaleWindow:** finestra astratta da cui derivano le altre viste dei dipendenti. Vengono implementati molti metodi per evitare di ripetere il codice dove non strettamente necessario e per metterli a disposizione di classi per nuovi tipi di utente. Alcuni metodi riguardanti le visite vengono resi virtuali puri perché dipendono strettamente dal tipo d'utente;
- **MedicoWindow:** finestra per un utente di tipo *Medico*, nella tabella delle visite viene tolta la colonna per il medico in quanto si tratta sempre dell'utente;
- **PrimarioWindow:** finestra per l'amministratore, rende disponibili tutte le azioni e permette inoltre di caricare da file o esportare su cartella;
- **SegreteriaWindow:** finestra per un utente di tipo *Segreteria*;
- **ModificaEvento:** finestra per modificare i campi di un evento;
- **ModificaPersona:** finestra per modificare i dati di un paziente o di un dipendente;
- **NuovaPersona:** finestra in cui inserire i dati di un nuovo paziente o di un nuovo dipendente;
- **NuovoEvento:** finestra in cui inserire i dati di un nuovo evento;
- **Window:** classe padre di tutte le finestre, rende disponibili metodi per centrare la finestra su schermo, rendere la finestra visibile e per creare una finestra di errore;

2.3 Controller

- **LoginController:** controller principale, gestisce una finestra di login e crea un controller adeguato a seconda del tipo di utente loggato;
- **PersonaleController:** controller astratto per ogni tipo di utente, fornisce metodi per tutti i controller tranne quelli con azioni specifiche di un tipo di utente;
- **MedicoController:** controller per un utente di tipo *Medico*, gestisce una finestra di tipo *MedicoWindow*;
- **SegreteriaController:** controller per un utente di tipo *Segreteria*, gestisce una finestra di tipo *SegreteriaWindow*;
- **PrimarioController:** controller per l'amministratore, gestisce una finestra *PrimarioWindow*. Nonostante alcuni metodi di questa classe siano identici a quelli di *SegreteriaController*, si è deciso di derivare la classe solo da *PersonaleController* per rispettare la logica dei tipi presente nel modello.

3 Rappresentazione grafica dei dati presenti nel database

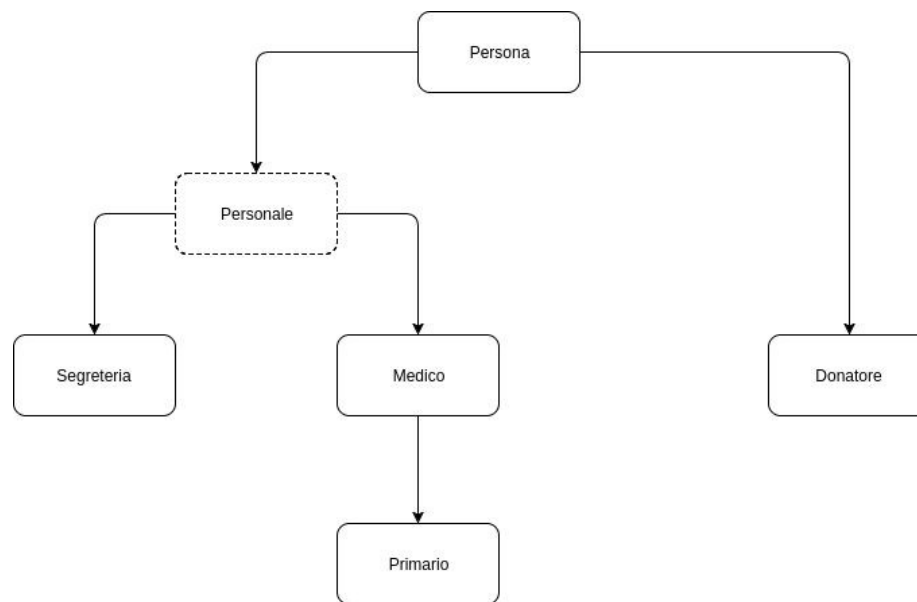


Figura 1: Dipendenti e pazienti

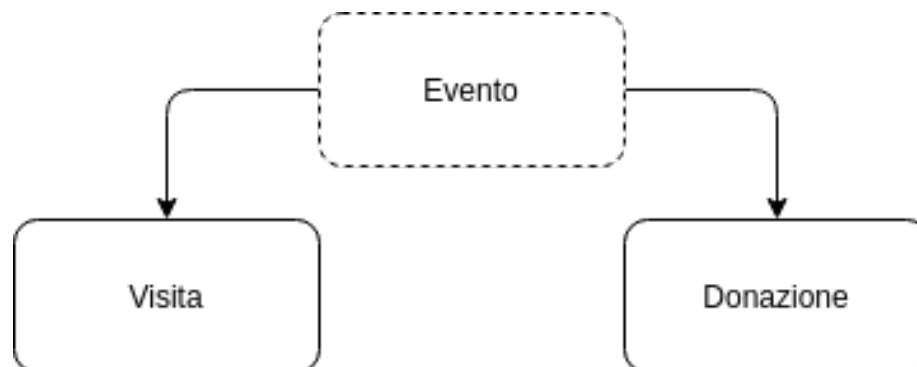


Figura 2: Appuntamenti

4 Descrizione dell'uso di codice polimorfo

- **Personale:** la classe astratta descrive un dipendente dell'ospedale, sono presenti due metodi virtuali puri per la ricerca, implementati poi in ogni

classe figlia a seconda dei permessi dati;

- **Evento:** classe astratta che descrive un appuntamento nell'agenda dell'applicazione, l'operatore di uguaglianza è virtuale puro per essere definito nelle classi derivate così come *Durata*, rappresentante l'arco di tempo di default, questo permette di non aggiungere appuntamenti sovrapposti per i medici o per evitare che un donatore prenoti una donazione prima dell'intervallo adeguato;
- **PersonaleWindow:** la classe fornisce tutti gli strumenti per creare una finestra per un utente, sono presenti metodi virtuali puri, questo perché alcune funzioni sono specifiche del tipo di utente, quindi la definizione viene delegata alle classi figlie;
- **PersonaleController:** come per la finestra, il controller di un utente generico fornisce gli strumenti in modo da creare altri controller per utenti, vengono resi virtuali puri solo i metodi che dipendono dal tipo di utente, vanno quindi definite nelle opportune classi.

5 Indicazione delle ore effettivamente richieste dalle fasi progettuali

- **20 ore:** progettazione modello e GUI;
- **35 ore:** codifica modello e GUI;
- **10 ore:** debugging, testing e relazione.

Il monte ore è stato leggermente superato perché durante la progettazione del progetto si è dovuto ricorrere spesso a risorse online, come il manuale *Qt*, per analizzare quali scelte fossero più opportune nel progetto.

6 Manuale utente della GUI

La finestra con tutti bottoni attivi è come in figura (alcuni elementi potrebbero non essere presenti a seconda del tipo di utente o del tipo di dati mostrati).

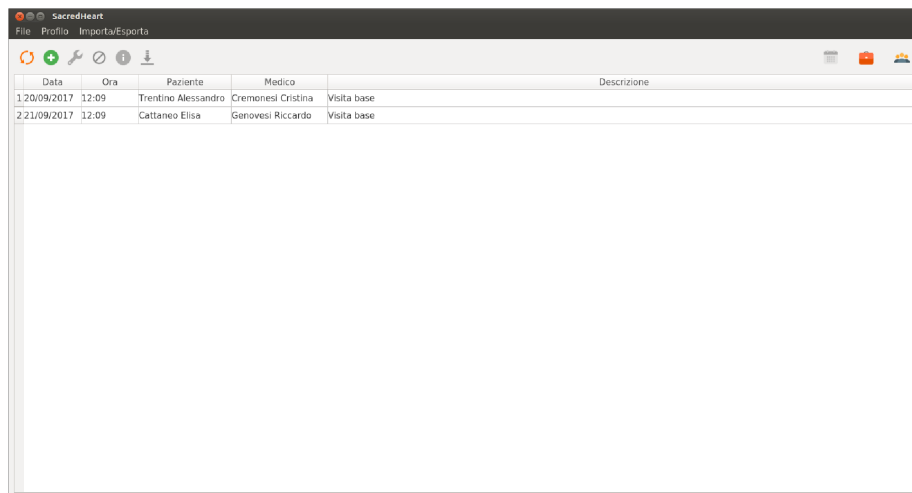











Figura 3: Finestra principale

A sinistra è presente il menù delle azioni, da sinistra a destra le icone rappresentante sono:

-  **Ricarica:** aggiorna le viste caricando i dati da file;
-  **Aggiungi:** aggiunge dati al database;
-  **Modifica:** per attivare questa icona bisogna selezionare l'intera riga da modificare (click sulla cella più a sinistra che contiene il numero dell'appuntamento);
-  **Elimina:** si attiva come *Modifica*;
-  **Informazioni:** l'icona è presente solo nella sezione degli appuntamenti, si attiva quando si seleziona una cella con una persona (paziente o medico) ed apre una finestra con le informazioni della persona selezionata;
-  **Salva:** salva tutte le modifiche sul database;

Sulla destra sono invece presenti le icone per ottenere le informazioni dal database, che sono:

-  **Elenco appuntamenti:** mostra la tabella degli appuntamenti a seconda del metodo di ricerca dell'utente;
-  **Elenco dipendenti:** mostra la tabella con le informazioni di ogni dipendente;
-  **Elenco pazienti:** mostra l'elenco di tutti i pazienti.

Nella barra di stato sono presenti tre menù. Questi, con le relative azioni, sono:

1. File:

- (a) *Logout*: chiude la finestra e ritorno alla finestra di login iniziare;
- (b) *Chiudi*: chiude l'applicazione;

2. Profilo:

- (a) *Cambia username*: permette di cambiare l'username dell'utente;
- (b) *Cambia password*: permette di cambiare la password dell'utente;
- (c) *Informazioni personali*: permette di conoscere i dati dell'utente (tranne la password);

3. Importa/Esporta:

- (a) *Aggiungi pazienti da file*: permette di selezionare un file xml contenente informazioni aggiuntive da aggiungere ai dati dell'applicazione;
- (b) *Aggiungi dipendenti da file*: come (a) ma per i dipendenti;
- (c) *Aggiungi appuntamenti da file*: come (a) ma per gli appuntamenti.

7 Ambiente di sviluppo

- **Sistema operativo:** Ubuntu 16.04 LTS (64-bit);
- **Compilatore:** GCC 5.4.0;
- **Versione libreria *Qt*:** 5.5.1.

8 Materiale consegnato

- file *.h* e *.cpp* necessari per la compilazione del progetto;
- file *progetto.pro* utilizzato per la creazione del *Makefile*;
- file Immagini *.qrc* necessario per utilizzare le icone memorizzate nella cartella */Images*;
- file *xml* nella cartella */Data* necessari per effettuare il login e visualizzare i dati oppure per provare l'inserimento da file.

9 Credenziali per i test

- **Primario:**
 - *username:* admin;
 - *password:* admin;
- **Medico:**
 - *username:* user1;
 - *password:* user;
- **Segreteria:**
 - *username:* user2;
 - *password:* user.

10 Risorse utilizzate

- **Set icone:** <http://graphicburger.com/300-flat-color-icons/>;
- **Logo:** ricerca Google.