Relazione Progetto di Programmazione ad Oggetti 2019/2020

INFOPOINT

Studente: Cocco Mattia

Matricola:1096738

Indice.

1 Introduzione

InfoPoint è un'applicazione sviluppata e pensata per raccogliere informazioni riguardo i Locali presenti sul territorio. Infatti consiste in un contenitore i cui oggetti rappresentano differenti tipi di Locali: Bar, Ristoranti, Discoteche, Discobar ed infine Pasticcerie.

L'applicazione oltre all'inserimento del Locale dispone anche di altre funzionalità: Salvataggio/Caricamento di dati mediante un file .xml, la possibilità di visualizzare la lista dei Locali presenti all'interno del Container, la funzione di modifica, che consente all'utente di modificare alcuni campi del Locale nel caso siano stati scritti sbagliati, ed infine la funzione che consente all'utente di cancellare dal container un Locale.

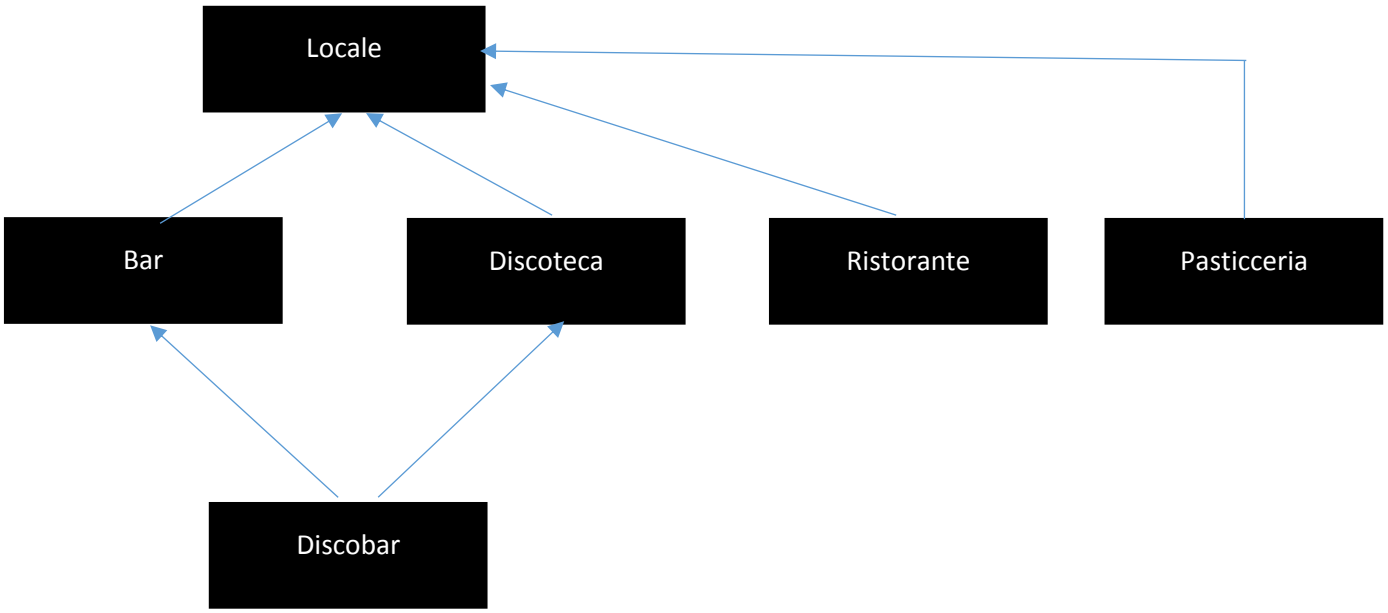
Oltre queste principali funzioni è possibile usare anche la funzione di ricerca per estrapolare dal Container uno o più Locali in base a ciò che si cerca. Questa funziona contiene 3 filtri di ricerca, che verranno specificati in seguito.

Il tutto viene rappresentato mediante un'interfaccia grafica (GUI) in cui l'utente può trovare tutte le informazioni necessarie per l'inserimento dei dati grazie ai placeholder posti nel campo di input del dato.

2 Progettazione

Tutto il progetto è stato creato basandosi sulla configurazione Model-View-Controller. Mediante Qt, e i suoi strumenti, si è stati più facilitati nel tenere separate queste tre caratteristiche, in modo da avere un lavoro più pulito e semplice.

3 Descrizione Gerarchia e Tipi



La classe alla base della gerarchia è la classe polimorfa ***Locale***, che rappresenta un generico Locale, senza alcuna informazione, ma che dichiara gli attributi generici che saranno poi comuni a tutte le classi derivate da essa. I campi presenti sono: *string* rating, nome, citta, descrizione, indirizzo, oraApertura e oraChiusura.

Si è deciso di mettere rating come *string* in quanto si è pensato alla possibilità che il locale possa essere giudicato con un giudizio da 0 a 10 oppure con un giudizio sottoforma di una parola singola.

Anche gli orari di apertura e chiusura sono stati dichiarati come *string* in quanto diamo la possibilità all'utente di poter scrivere sia l'orario in versione standard (0-24) oppure in versione americana, quindi con l'aggiunta di am/pm. Specifichiamo che gli ***Orari di Apertura*** e di ***Chiusura*** sono da inserire nel formato **HH:MM** eventualmente aggiungendo anche **Am/Pm**.

La classe contiene i metodi: ***getTipo()***, virtuale puro, che ritorna la tipologia del locale preso in considerazione, e ***getInfo()*** che ritorna tutte le informazioni riguardanti il locale che si sta visualizzando, ricercando o inserendo. Queste due funzioni verranno implementate da tutte le altre sottoclassi, che essendo diverse ritorneranno informazioni diverse, il tutto tramite chiamate polimorfe.

Oltre questi due metodi, la classe contiene anche altre funzioni più semplici, ***get()*** e ***set()*** dei campi dati dichiarati, il distruttore virtuale ***~Locale()*** e l'overloading dell'operatore ***cout***.

Come nello schema, da ***Locale*** derivano quattro classi istanziabili: ***Bar, Discoteca, Ristoranti*** e ***Pasticceria.***

- La classe ***Bar***, derivata virtualmente da ***Locale***, è composta da un solo campo dato, ovvero *postiASedere*(di tipo Bool). Questo campo, che potrebbe sembrare superfluo, in realtà è utile in quanto esistono, ed esisteranno, realtà di Bar che magari hanno solo il bancone senza alcun posto dove sedersi, una sorta di take-away.
  Come per tutte le prossime classi che verranno trattate, ***Bar*** implementa l'override delle funzioni ***getTipo()*** e ***getInfo()***, il distruttore virtuale ed infine il proprio metodo ***getPostiASedere()*** che ritorna se il Bar è attrezzato per sedersi o meno.

- La classe ***Discoteca***, anch'essa derivata virtualmente da ***Locale***, è caratterizzata da due campi: *ingresso* (di tipo bool) e *prezzo_ingresso*(di tipo unsigned int). Il primo campo dati specifica se c'è un prezzo da pagare di ingresso o meno, il secondo campo invece specifica il prezzo, in euro, di ingresso alla discoteca. Basandoci sulle realtà che si conoscono abbiamo deciso che il *prezzo_ingresso* dev'essere un numero **intero** quindi senza virgole. Nel caso la discoteca abbia ingresso gratuito, *ingresso*==false, allora il *prezzo_ingresso* sarà automaticamente settato a 0 euro. Come per ***Bar*** la classe implementa ***getTipo()*** e ***getInfo()*** ed il distruttore virtuale Inoltre la classe implementa i suoi propri metodi ***getIngresso()*** e ***getP_ing()*** che ritornano rispettivamente se la discoteca è ad ingresso libero, o meno, e il costo di ingresso ad essa.

- La classe ***Ristorante*** invece consiste di un campo dati: *coperto*(di tipo bool). Essendo un progetto destinato alla raccolta dei Locali di un territorio abbiamo pensato di non appesantire troppo l'applicazione inserendo troppi campi dati, spesso superflui agli occhi di chi la usa, quindi abbiamo deciso soltanto di specificare se al ***Ristorante*** è presente il coperto o meno, in quanto è una decisione del gestore del locale se averlo o meno come extra. Implementa i due metodi della classe base ***getTipo()*** e ***getInfo()***, il distruttore virtuale e il proprio metodo ***getCoperto()*** che ritorna un valore booleano in base alla presenza o meno del coperto.

- La classe ***Pasticceria*** contiene due campi dati: *specialita* (di tipo string) e *caffetteria*(di tipo bool). Il primo campo rappresenta la specialità della pasticceria, che può essere un dolce, un pasticcino, un bignè, etc. Il secondo campo invece rappresenta la presenza, o meno, di una caffetteria al suo interno, in quanto alcune realtà non la hanno. Come per le altre classi anche qua vengono implementati i metodi ***getTipo()*** e ***getInfo()***, oltre che al distruttore virtuale e ai proprio metodi ***getSpecialita() e getCaffetteria()***.

L'ultima classe, anch'essa istanziabile, della gerarchia è ***Discobar*** e deriva, mediante ereditarietà multipla, dalle classi ***Bar*** e ***Discoteca***.

Questa classe è stata pensata prendendo spunto da un locale a Sottomarina il quale non è propriamente né un bar né una discoteca, da qui l'idea di inserire nella gerarchia questa tipologia di locale, presente soprattutto nelle zone costiere, prevalentemente sulle spiagge dei litorali.

Derivando dalle due classi sopracitate, si è soltanto pensato di inserire *costo_drink*(di tipo string) che specifica il prezzo base dei costi dei cocktail, in modo tale che l'utente si possa fare un'idea di quanto sia il costo minimo di un drink. Si è scelto di usare un tipo *string* in quanto, al momento dell'inserimento, si possa scegliere se scrivere un **numero** soltanto oppure scrivere un **range** di valori nella forma, per esempio, **"5 ai 10"**, il resto verrà completato dal programma in automatico.

Come le altre classi, anche questa implementa i metodi ***getTipo()***, ***getInfo()*** e il distruttore virtuale oltre che il proprio metodo ***getCosto_drink()*** che ritorna appunto il prezzo base dei costi dei drink nel locale.

4 Descrizione Polimorfismo ed Estensibilità

Come si nota dalla descrizione delle classi, si possono definire le funzioni ***getTipo(), getInfo()*** e il distruttore ***~Locale()*** come funzioni virtuali.

Le chiamate polimorfe, di cui l'applicazione usufruisce maggiormente, sono ovviamente ***getTipo()*** e ***getInfo()*** che permettono di essere chiamate da tutte le classi istanziabili con risultati diversi di classe in classe.

Dato il totale di ore a disposizione, la gerarchia risulta molto semplice e di conseguenza molto facilmente estensibile, soprattutto trattandosi di *Locali*, basterebbe quindi implementare nuove classi, con conseguenti attributi e funzioni, modificare le funzioni riguardanti la lettura/scrittura sui file XML per incorporare anche le nuove classi e implementare le nuove funzioni. Questo porta quindi solo a piccole modifiche sul codice, rendendo più facile lo sviluppo della gerarchia preesistente.

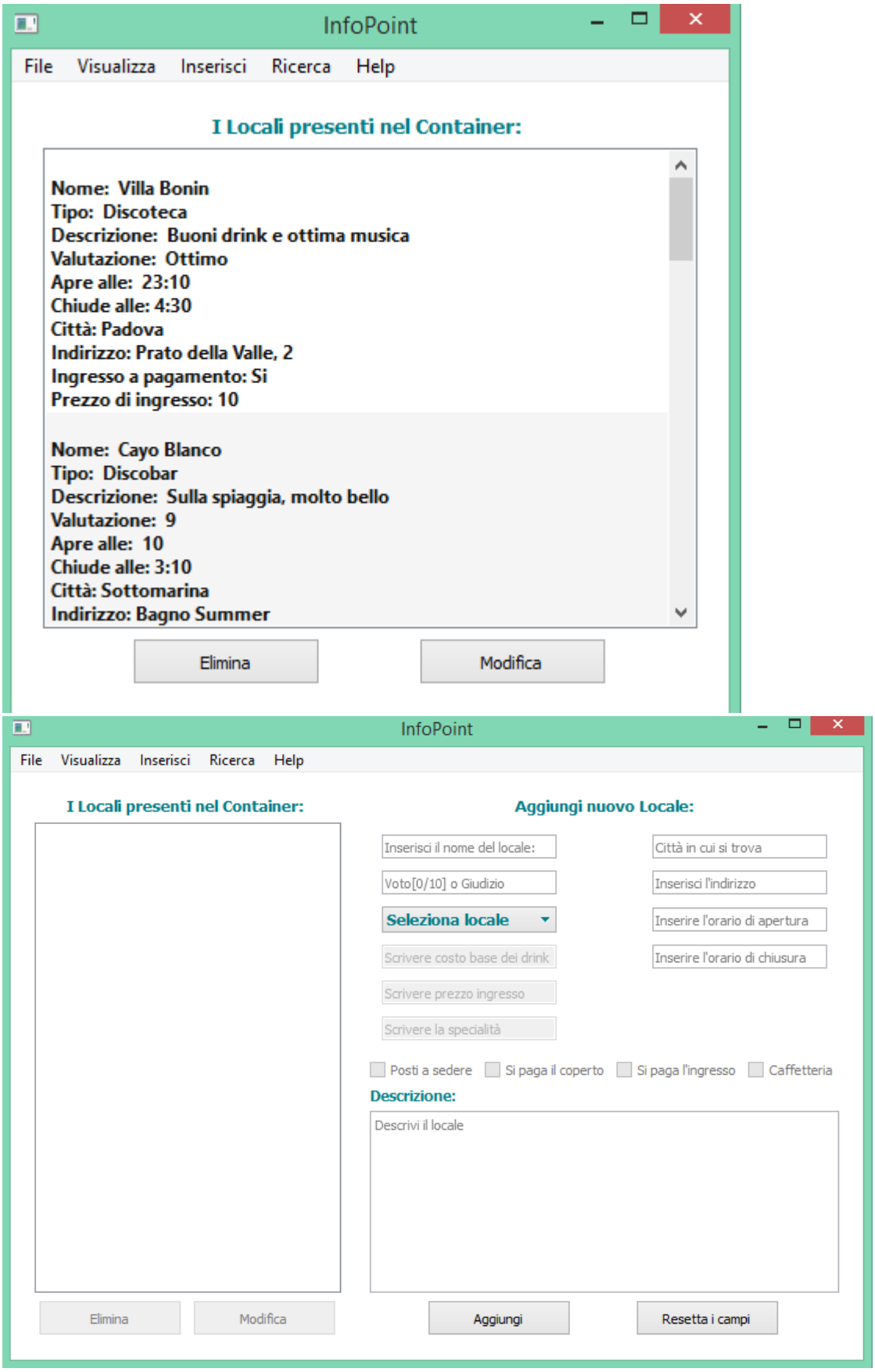
5 Descrizione File Input/Output

Il programma è anche in grado di gestire il caricamente e il salvataggio del container mediante l'implementazione di funzioni di lettura/scrittura su file.xml.

Al momento del lancio dell'applicazione, essa richiederà all'utente di caricare un file pre esistente "*dati.xml*", qualora l'utente non dovesse caricare nulla allora il programma fa uscire un pop-up dove avverte che nessun file è stato caricato, creando così un container vuoto, dando comunque la possibilità all'utente di poter creare una nuova lista di Locali. Invece se l'utente sceglie di caricare il file, il programma caricherà quest'ultimo. Se il file caricato risulta vuoto, il programma avvisa l'utente sempre mediante pop-up e continuerà l'esecuzione come se l'utente avesse deciso di non caricare nulla.

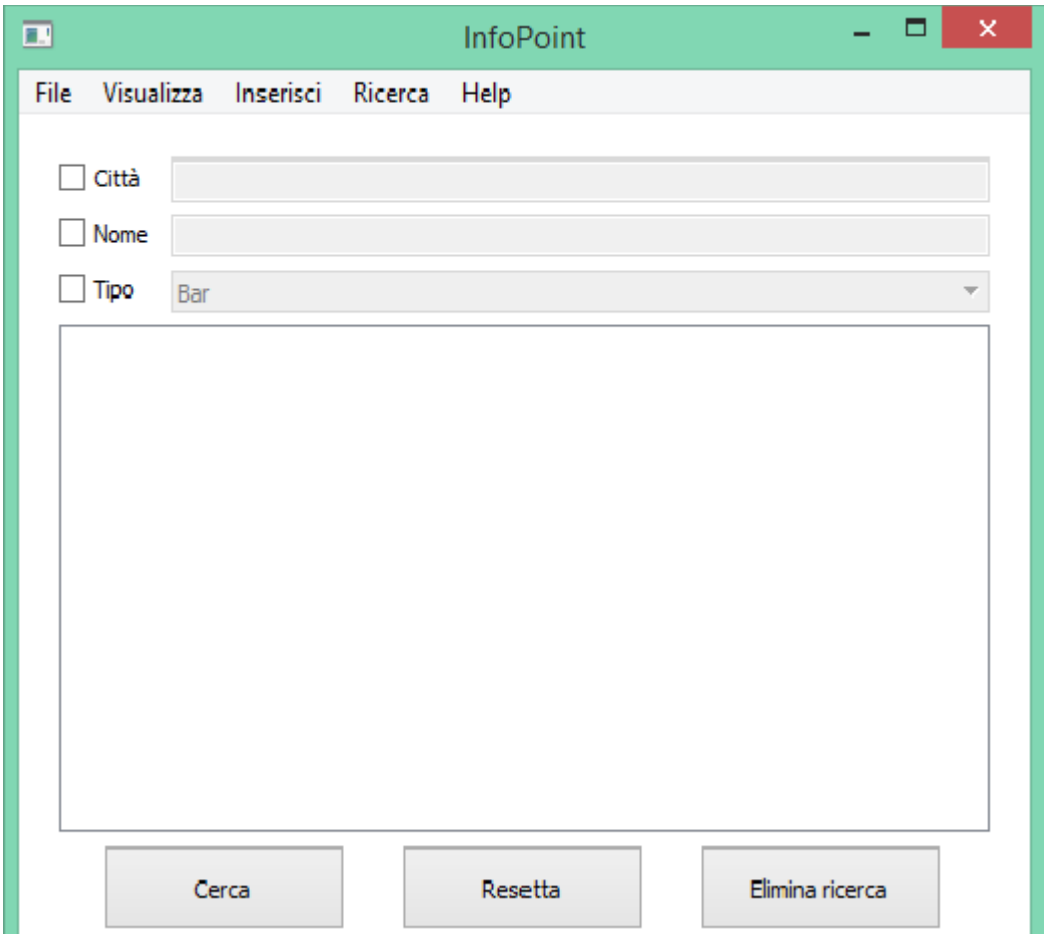
La scelta di utilizzare file XML non è casuale, deriva infatti dallo studio delle librerie Qt, che permettono, mediante funzioni specifiche, di leggere e scrivere da/su file XML. Inoltre l'XML ha tag di facileiinterpretazion, sia per l'utente che per il programma, che rappresentano tutti i campi dati delle classi.

6 Descrizione GUI



Nel caso l'utente dovesse decidere di caricare il file "*dati.xml*" preesistente allora il programma si presenterà come nella **prima figura**, ovvero mostrando la lista dei Locali presenti nel container con tutti i propri campi dati. Merito del ***Layoutview*** che si occupa di gestire e far apparire su display, in una lista con scroll bar, tutti i locali esistenti.

Se invece l'utente non dovesse caricare nulla, o non ci fosse alcun elemento nel container, allora il programma si presenterà già nella sezione ***insertlayout*** (**seconda figura)**, ovvero la sezione nella quale l'utente sarà in grado di inserire un Locale, il tutto agevolato grazie ai PlaceHolder inseriti nei campi da compilare, che guida l'utente passo per passo. L'utente è tenuto a inserire tutti i campi o almeno *nome, indirizzo, città, tipologia e descrizione*, altrimenti il programma lo avvertirà che alcuni campi sono vuoti.



L'ultimo principale layout è il ***searchlayout*** ovvero la parte dell'applicazione nella quale è possibile ricerca uno o più locali. Come si può vedere è suddiviso in 3 filtri diversi:

- ***Città:*** inserendo una città e premendo "*cerca*" il programma stampa a video i risultati ti tale ricerca, sempre un risultato sotto l'altro.
- ***Nome:*** questa è la ricerca più specifica dell'applicazione in quanto il nome deve esistere nel container, quindi è suggerita per l'utente che conosce il nome giusto del locale, maiuscole e numeri compresi.
- ***Tipo:*** quest'ultima tipologia di ricerca è la più generica in quanto basta scegliere la categoria dei locali da ricercare e il programma mostrerà tutti i locali di quella tipologia presente nel container.

Premendo "*Resetta"* si resetteranno i filtri, mentre premendo "*Elimina ricerca"* si elimineranno **tutti** i risultati trovati nel container.

Oltre a queste tre funzioni, il sottomenù *File* offre le seguenti funzioni: *Visualizza*, *Inserisci*, *Ricerca*, *Salva*, *Carica nuovi dati*, *Exit*.

Mediante il bottone, presente in ***Layoutview*** e ***insertlayout***, "*Elimina*" sarà possibile eliminare dal container un elemento. Invece grazie al bottone "*Modifica*" è possibile modificare dei campi dati di un Locale presente nel container. Per ultimo, con "*Resetta i campi*", il programma resetterà tutti i campi, eliminando i dati inseriti fino a quel momento.

Il tasto "*Help->Segnala Errori*" invece mostrerà a schermo un messaggio in cui avviserà l'utente che in caso di bisogno potrà scrivere ai due creatori dell'applicazione, facendo vedere anche le relativi nomi, cognomi, numero di matricola e mail.

7 Suddivisione del Lavoro

I responsabili di questo progetto sono: ***Mattia Cocco matricola 1096738*** e ***Hossain Safdari matricola 1102482***. Inizialmente si era deciso di fare tutto assieme in parallelo ma successivamente si è diviso il lavoro tra noi in modo da ottimizzare i tempi per rimanere dentro il totale delle ore a disposizione. ***Hossain*** ha iniziato la parte riguardante la modellazione del progetto mentre ***Mattia*** la parte riguardante la GUI. Nonostante ciò, entrambi si sono aiutati a vicenda nella risoluzione di alcuni alcuni problemi, quindi entrambi conoscono e saprebbero modificare il codice senza alcun problema. Infatti ***Hossain*** ha scritto la parte riguardante il salvataggio/caricamento su *file xml* mentre ***Mattia*** ha scritto la parte riguardante l'ereditarietà multipla della classe *Discobar*. La classe *container* invece è stata pensata e modellata a da entrambi.

8 Conteggio Ore

Le ore lavorative individuali del progetto sono così conteggiate:

- *Analisi preliminare del problema*: 2 ore
- *Progettazione modello e GUI*: 6 ore
- *Apprendimento libreria Qt*: 16 ore
- *Codifica modello*: 6 ore
- *Codifica GUI*: 16 ore
- *Debugging*: 3 ore
- *Testing* 2 ore

L'ora in eccesso è dovuta al fatto che ci sono stati problemi con la GUI che purtroppo hanno richiesto più lavoro del necessario per risolverli.

9 Note

Per la corretta esecuzione del programma, su macchina Linux, è necessario utilizzare i seguenti comandi da terminale: posizionarsi tramite terminale nella cartella del progetto(su Linux ***cd Desktop, cd InfoPoint***), eseguire il comando ***qmake InfoPoint.pro***, successivamente ***make*** ed infine, per lanciare il programma

eseguire ***./InfoPoint***. In caso si voglia caricare un file allora l'utente è **obbligato** ad usare ***dati.xml*** presente nella cartella principale.

Si **consiglia** di utilizzare il programma in **fullscreen** per avere un'esperienza migliore riguardo l'utilizzo dell'applicazione.

L'intero progetto è stato creato usando **Qt5.9.5** in ambiente Windows 8.1 per quanto riguarda *Mattia* mentre su MAC con MacOS Mojave 10.14 per quanto riguarda *Hossain*.