# Gestione quaderno dei compiti e valutazioni

**Titolo del progetto:** Gestione quaderno dei compiti e valutazioni
**Alunno/a:** Lucas Previtali
**Classe:** I4AA
**Anno scolastico:** 2019/2020
**Docente responsabile:** Ugo Bernasconi

# 1 Introduzione

## 1.1 Informazioni sul progetto

***Progetto svolto da:*** Lucas Previtali
***Mandante del progetto:*** Ugo Bernasconi
***Docente Responsabile:*** Ugo Bernasconi
***Scuola:*** Arti e Mestieri Trevano
***Sezione:*** Informatica
***Classe:*** I4AA
***Data d'inizio:*** 03.09.2019
***Termine della consegna:*** 20.12.2019

## 1.2 Abstract

E' una breve e accurata rappresentazione dei contenuti di un documento, senza notazioni critiche o valutazioni. Lo scopo di un abstract efficace dovrebbe essere quello di far conoscere all'utente il contenuto di base di un documento e metterlo nella condizione di decidere se risponde ai suoi interessi e se è opportuno il ricorso al documento originale.

Può contenere alcuni o tutti gli elementi seguenti:

- **Background/Situazione iniziale**
- **Descrizione del problema e motivazione**: Che problema ho cercato di risolvere? Questa sezione dovrebbe includere l'importanza del vostro lavoro, la difficoltà dell'area e l'effetto che potrebbe avere se portato a termine con successo.
- **Approccio/Metodi**: Come ho ottenuto dei progressi? Come ho risolto il problema (tecniche…)? Quale è stata l'entità del mio lavoro? Che fattori importanti controllo, ignoro o misuro?
- **Risultati**: Quale è la risposta? Quali sono i risultati? Quanto è più veloce, più sicuro, più economico o in qualche altro aspetto migliore di altri prodotti/soluzioni?

Esempio di abstract:

> *As the size and complexity of today's most modern computer chips increase, new techniques must be developed to effectively design and create Very Large Scale Integration chips quickly. For this project, a new type of hardware compiler is created. This hardware compiler will read a C++ program, and physically design a suitable microprocessor intended for running that specific program. With this new and powerful compiler, it is possible to design anything from a small adder, to a microprocessor with millions of transistors. Designing new computer chips, such as the Pentium 4, can require dozens of engineers and months of time. With the help of this compiler, a single person could design such a large-scale microprocessor in just weeks.*

## 1.3 Scopo

Questo progetto nasce per semplificare il lavoro della creazione di un quaderno dei compiti e per velocizzare il processo dell'assegnazione dei punti per la valutazione. Lo scopo è quello di creare un applicativo che si occupi di far visualizzare i criteri di valutazione per un progetto e che dia la possibilità di scegliere quelli adatti a dipendenza del progetto in questione. Oltre a questo per rendere il più completo possibile il quaderno dei compiti deve essere possibile inserire le varie informazioni sul formatore e sul canditato al quale il progetto è assegnato e altre informazioni generiche. Per quanto riguarda la parte relativa alle valutazioni, l'applicativo dovrebbe riuscire ad interfacciarsi con Word per riuscire a prendere i requisiti scelti. Deve poi mostarli e deve essere possibile inserire i punti per ogni requisito (una volta terminato il progetto). Inoltre deve essere possibile anche inserire le motivazioni complete per ogni valutazione data.

**Analisi**

### 1.4 Analisi del dominio

L'applicativo sarà usato principalmente dai docenti che creano i quaderni dei compiti. Non necessariamente questi docenti saranno informatici, quindi l'applicazione deve essere usabile in modo intuitivo e veloce da chiunque, anche da chi non ha competenze informatiche più che basilari. Siccome lo scopo del progetto è quello di rendere più semplice tutto il processo di creazione di un quaderno dei compiti e della successiva assegnazione dei punti per ogni requisito, il programma essere veloce e facile da usare.

### 1.5 Analisi e specifica dei requisiti

In base allo scopo e all'analisi del dominio, oltre che al colloquio con il formatore, ho sviluppato la seguente lista di requisiti. L'obbiettivo dell'applicazione è quello di permettere la creazione e la gestione di quaderni dei compiti tramite interfaccia grafica. Per questo motivo i requisiti principali sono delle view che siano semplici e intuitive da usare e che permettano di creare un nuovo quaderno dei compiti con facilità. La parte front-head del progetto quindi saranno delle maschere con UI WPF, mentre la parte back-head verrà sviluppata con C#.

| **ID: REQ-001** | |
|---|---|
| **Nome** | Ambiente di sviluppo adeguato e funzionale |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** | - |
| **Sotto requisiti** | |
| **001** | Installare la versione migliore di Visual Studio (che sia recente ma allo stesso tempo stabile). Per questo progetto ho scelto Visual Studio 2017 Versione 15.9.6. |
| **002** | Installare tutti i componenti aggiuntivi necessari. |

| **ID: REQ-002** | |
|---|---|
| **Nome** | View per l'inserimento di informazioni generiche |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** | - |
| **Sotto requisiti** | |
| **001** | Sviluppare una view che permetta l'inserimento di tutte le informazioni generiche relative al progetto (dati canditato, dati formatore, date, ecc.). |

| ID: REQ-003 | |
|---|---|
| **Nome** | View per la descrizione |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** | - |
| **Sotto requisiti** | |
| **001** | Sviluppare una semplice view per l'inserimento della descrizione del progetto |

| ID: REQ-004 | |
|---|---|
| **Nome** | View per la scelta dei requisiti |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** | - |
| **Sotto requisiti** | |
| **001** | Creare una view che permetta di scegliere i 7 requisiti A14-A20 da assegnare al candidato |
| **002** | Far viualizzare sia il codice che il titolo (eventualmente descizione) |

| ID: REQ-005 | |
|---|---|
| **Nome** | Interfacciamento tra documento word e view per la scelta dei requisiti |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** | - |
| **Sotto requisiti** | |
| **001** | Fare in modo che la view si interfacci al documento word nel quale sono contenuti tutti i requisiti estesi |
| **002** | Scrivere il titolo del requisito in base al codice dato |

| ID: REQ-006 | |
|---|---|
| **Nome** | View per l'assegnazione dei punteggi. |

| | |
|---|---|
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** | - |
| **Sotto requisiti** | |
| **001** | Creare una view che permetta l'assegnazione del punteggio per ogni requisito scelto in precedenza. |
| **002** | Visualizzare i requisiti in maniera estesa |

| **ID: REQ-007** | |
|---|---|
| **Nome** | Motivazioni dei voti. |
| **Priorità** | 2 |
| **Versione** | 1.0 |
| **Note** | - |
| **Sotto requisiti** | |
| **001** | Prevedere un sistema di copia-incolla per rendere più veloce la parte della motivazione del punteggio assegnato. |

| **ID: REQ-008** | |
|---|---|
| **Nome** | Modifica di un qdc esistente |
| **Priorità** | 3 |
| **Versione** | 1.0 |
| **Note** | - |
| **Sotto requisiti** | |
| **001** | Fare in modo che sia possibile modificare un qdc già esistente. (Modifica delle info generiche, della descrizione e eventualmente dei requisiti scelti). |

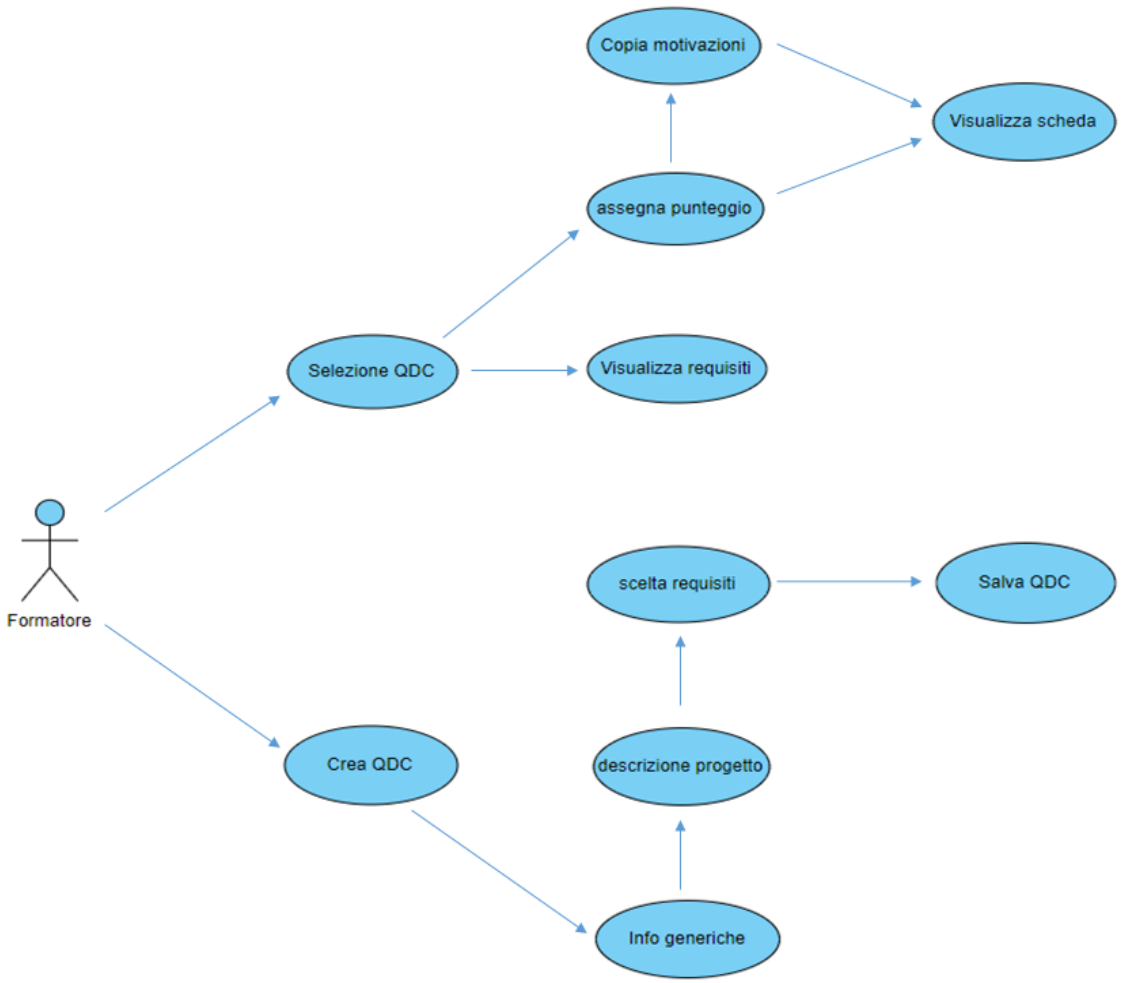
| **ID: REQ-009** | |
|---|---|
| **Nome** | Sviluppo della banca dati |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** | - |

| **Sotto requisiti** | |
|---|---|
| **001** | Ideare una banca dati adeguata ai requisiti |
| **002** | Generare e aggiornare la banca dati direttamente dal codice tramite SQLServer |

| **ID: REQ-010** | |
|---|---|
| **Nome** | Sviluppo dell'applicativo con modello MVVM |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** | - |
| **Sotto requisiti** | |
| **001** | Sviluppare l'intero applicativo usando il modello MVVM.. |

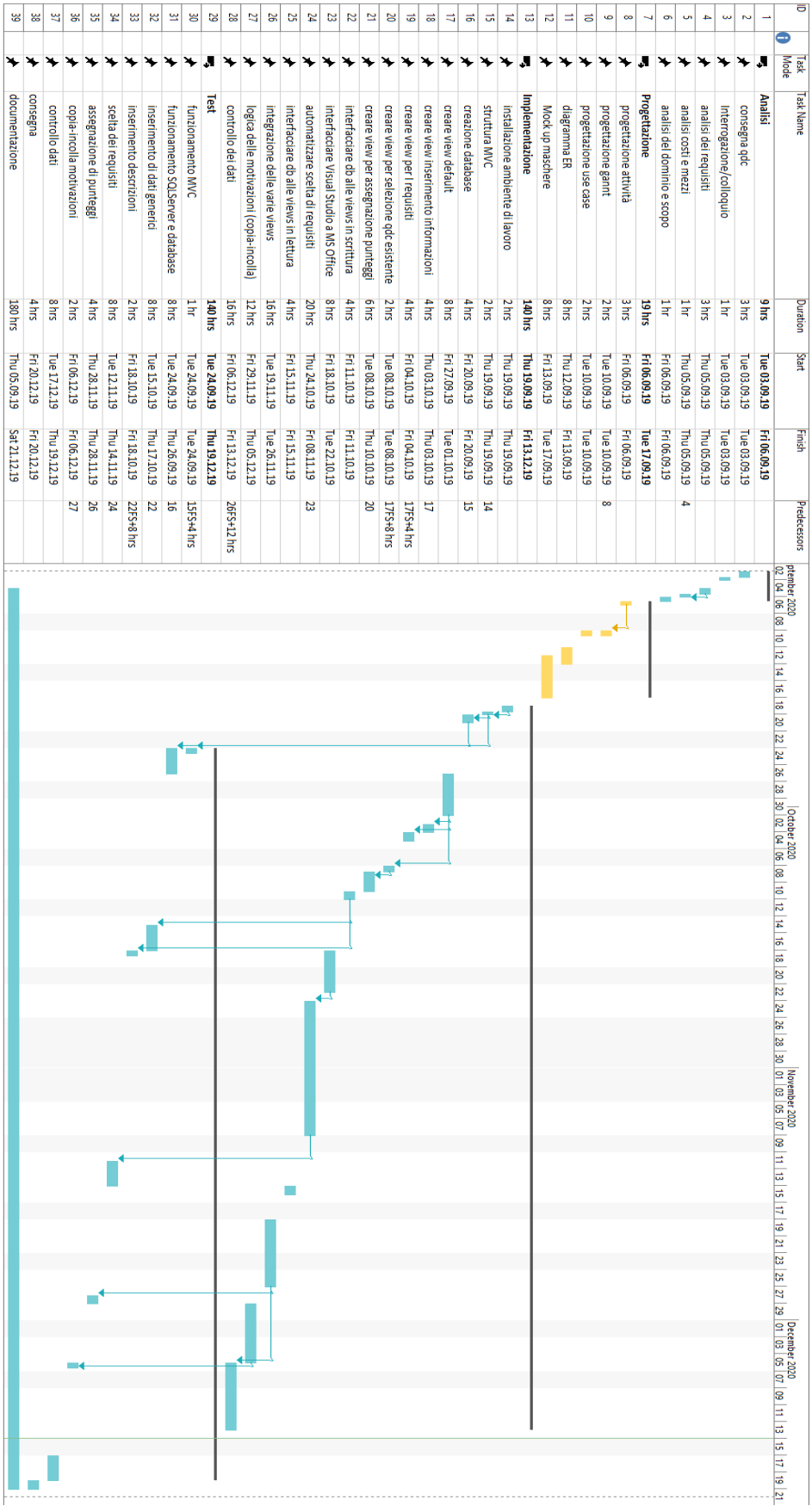
### 1.6 Use case

L'applicativo è pensato per essere usato esclusivamente dai docenti formatori che si occupano di preparare un quaderno dei compiti, per questo motivo nello use case è presente un solo attore, ovvero il *Formatore*. Il formatore quando apre l'applicazione ha la possibilità di scegliere se creare un Quaderno dei compiti nuovo (operazione di default) oppure se selezionarne uno già esistente.
Nel caso in cui scelga la prima opzione si troverà una maschera nella quale potrà inserire le informazioni generiche riguardo al progetto (informazioni formatore, informazioni candidato, titolo, date e altro). Una volta inserite queste informazioni potrà scrivere la descrizione del progetto (o copiarla). Questo avviene in un'altra maschera. Dopodiché si arriva alla parte più importante, cioè quella della scelta dei requisiti. In questa maschera il formatore potrà andare a selezionare i sette requisiti A14 – A20 da assegnare al candidato oppure potrà semplicemente inserire il numero ID del requisito e il programma si occuperà di trovare nel file word contenente tutti i requisiti in versione estesa il requisito giusto. A questo punto sarà possibile salvare il quaderno dei compiti appena creato.
Nel caso in cui invece venga scelta la seconda opzione, il docente dovrà selezionare uno dei qdc creati in precedenza. Al quaderno dei compiti selezionato potranno essere aggiunte le valutazioni per ognuno dei sette requisiti scelti e potranno essere inserite o copiate le motivazioni per il voto assegnato. Infine potrà essere visualizzata la scheda.




1 Use Case

## 1.7 Pianificazione



| ID | Task Mode | Task Name | Duration | Start | Finish | Predecessors |
|---|---|---|---|---|---|---|
| 1 | | **Analisi** | **9 hrs** | **Tue 03.09.19** | **Fri 06.09.19** | |
| 2 | | consegna qdc | 3 hrs | Tue 03.09.19 | Tue 03.09.19 | |
| 3 | | Interrogazione/colloquio | 1 hr | Tue 03.09.19 | Tue 03.09.19 | |
| 4 | | analisi dei requisiti | 3 hrs | Thu 05.09.19 | Thu 05.09.19 | |
| 5 | | analisi costi e mezzi | 1 hr | Thu 05.09.19 | Thu 05.09.19 | 4 |
| 6 | | analisi del dominio e scopo | 1 hr | Fri 06.09.19 | Fri 06.09.19 | |
| 7 | | **Progettazione** | **19 hrs** | **Fri 06.09.19** | **Tue 17.09.19** | |
| 8 | | progettazione attività | 3 hrs | Fri 06.09.19 | Fri 06.09.19 | |
| 9 | | progettazione gannt | 2 hrs | Tue 10.09.19 | Tue 10.09.19 | 8 |
| 10 | | progettazione use case | 2 hrs | Tue 10.09.19 | Tue 10.09.19 | |
| 11 | | diagramma ER | 8 hrs | Thu 12.09.19 | Fri 13.09.19 | |
| 12 | | Mock up maschere | 8 hrs | Fri 13.09.19 | Tue 17.09.19 | |
| 13 | | **Implementazione** | **140 hrs** | **Thu 19.09.19** | **Fri 13.12.19** | |
| 14 | | installazione ambiente di lavoro | 2 hrs | Thu 19.09.19 | Thu 19.09.19 | |
| 15 | | struttura MVC | 2 hrs | Thu 19.09.19 | Thu 19.09.19 | 14 |
| 16 | | creazione database | 4 hrs | Fri 20.09.19 | Fri 20.09.19 | 15 |
| 17 | | creare view default | 8 hrs | Fri 27.09.19 | Tue 01.10.19 | |
| 18 | | creare view inserimento informazioni | 4 hrs | Thu 03.10.19 | Thu 03.10.19 | 17 |
| 19 | | creare view per l requisiti | 4 hrs | Fri 04.10.19 | Fri 04.10.19 | 17FS+4 hrs |
| 20 | | creare view per selezione qdc esistente | 2 hrs | Tue 08.10.19 | Tue 08.10.19 | 17FS+8 hrs |
| 21 | | creare view per assegnazione punteggi | 6 hrs | Tue 08.10.19 | Thu 10.10.19 | 20 |
| 22 | | interfacciare db alle views in scrittura | 4 hrs | Fri 11.10.19 | Fri 11.10.19 | |
| 23 | | interfacciare Visual Studio a MS Office | 8 hrs | Fri 18.10.19 | Tue 22.10.19 | |
| 24 | | automatizzare scelta di requisiti | 20 hrs | Thu 24.10.19 | Fri 08.11.19 | 23 |
| 25 | | interfacciare db alle views in lettura | 4 hrs | Fri 15.11.19 | Fri 15.11.19 | |
| 26 | | integrazione delle varie views | 16 hrs | Tue 19.11.19 | Tue 26.11.19 | |
| 27 | | logica delle motivazioni (copia-incolla) | 12 hrs | Fri 29.11.19 | Thu 05.12.19 | |
| 28 | | controllo dei dati | 16 hrs | Fri 06.12.19 | Fri 13.12.19 | 26FS+12 hrs |
| 29 | | **Test** | **140 hrs** | **Tue 24.09.19** | **Thu 19.12.19** | |
| 30 | | funzionamento MVC | 1 hr | Tue 24.09.19 | Tue 24.09.19 | 15FS+4 hrs |
| 31 | | funzionamento SQLServer e database | 8 hrs | Tue 24.09.19 | Thu 26.09.19 | 16 |
| 32 | | inserimento di dati generici | 8 hrs | Tue 15.10.19 | Thu 17.10.19 | 22 |
| 33 | | inserimento descrizioni | 2 hrs | Fri 18.10.19 | Fri 18.10.19 | 22FS+8 hrs |
| 34 | | scelta dei requisiti | 8 hrs | Tue 12.11.19 | Thu 14.11.19 | 24 |
| 35 | | assegnazione di punteggi | 4 hrs | Thu 28.11.19 | Thu 28.11.19 | 26 |
| 36 | | copia-incolla motivazioni | 2 hrs | Fri 06.12.19 | Fri 06.12.19 | 27 |
| 37 | | controllo dati | 8 hrs | Tue 17.12.19 | Thu 19.12.19 | |
| 38 | | consegna | 4 hrs | Fri 20.12.19 | Fri 20.12.19 | |
| 39 | | documentazione | 180 hrs | Thu 05.09.19 | Sat 21.12.19 | |

Il mio gantt preventivo è diviso in 4 fasi e occupa un totale di 180 ore.
La prima fase è la fase di analisi, che serve principalmente a capire il progetto, parlare con il formatore dei punti più importanti e fare domande sui requisiti meno chiari. La parte più importante di questa fase è sicuramente l'analisi dei requisiti, perché è la partenza effettiva del progetto. Questa fase è stata prevista per la prima settimana di lavoro.
La seconda fase è la progettazione, nella quale si comincia ad analizzare più a fondo il progetto, partendo dalla progettazione delle attività e del gantt, passando per gli use case e arrivando alla progettazione dello schema ER e delle interfacce. Questa fase è importante perché se svolta la meglio prmette di velocizzare la parte di implementazione vera e propria. La parte di progettazione è stata prevista per la seconda settimana e parte della terza.
La terza fase è l'implementazione del progetto, fase cruciale e molto lunga perché è la parte in cui viene sviluppato il progetto, in questo caso l'applicativo. Questa fase comincia la terza settimana e prosegue fino alla consegna dle progetto. La parte iniziale dell'implementazione consiste nel creare le varie maschere. Dopodiché queste maschere dovranno essere interfacciate al database e dovrà essere sviluppato un sistema per reperire delle informazioni direttamente da un file .docx.
L'ultima fase è quella dei test, che va di pari passo con l'implementazione. Quando vengono sviluppate delle parti importanti o sensibili, viene fatto un test per verificare il corretto funzionamento del codice appena scritto. È importante fare i test man mano che vengono implementate le funzionalità per avere a disposizione più tempo per poter fare eventuali modifiche o correzioni.
Oltre a queste fasi c'è anche la parte relativa alla documentazione del progetto, che occupa l'intera programmazione e viene aggiornata giornalmente.

## 1.8 Analisi dei mezzi

Per lo svolgimento del progetto non servono particolari software o componenti hardware al di fuori da ciò che riguarda Visual Studio (con le varie librerie e plugins elencati nei punti successivi).

### 1.8.1 Software

| Software | Versione | Note |
|---|---|---|
| **Windows** | 10 | Sistema operativo usato per lo svolgimento dell'intero progetto |
| **Visual Studio 2017** | 15.9.6 | IDE usato per lo sviluppo dell'aplicativo (linguaggio c#, progettazione secondo standard MVVM) |
| **NuGet** | 4.1 | ? |
| **GitHub Desktop** | 2.2.3 | Programma usato per il salvataggio in cloud del progetto e della documentazione |
| **Microsoft Word** | 2016 | Utilizzato per allestire la documentazione di progetto e i diari giornalieri |
| **Microsoft Project** | 2016 | Utilizzato per allestire i diagrammi di Gannt del progetto. |

### 1.8.2 Hardware

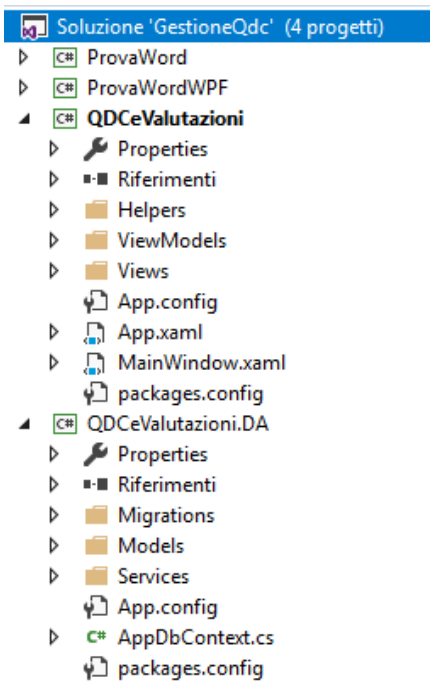
| Hardware | Modello | Note |
|---|---|---|
| **HP** | 7-x185nz | Computer personale usato per lo svolgimento del progetto |

# 2 Progettazione

## 2.1 Design dell'architettura del sistema

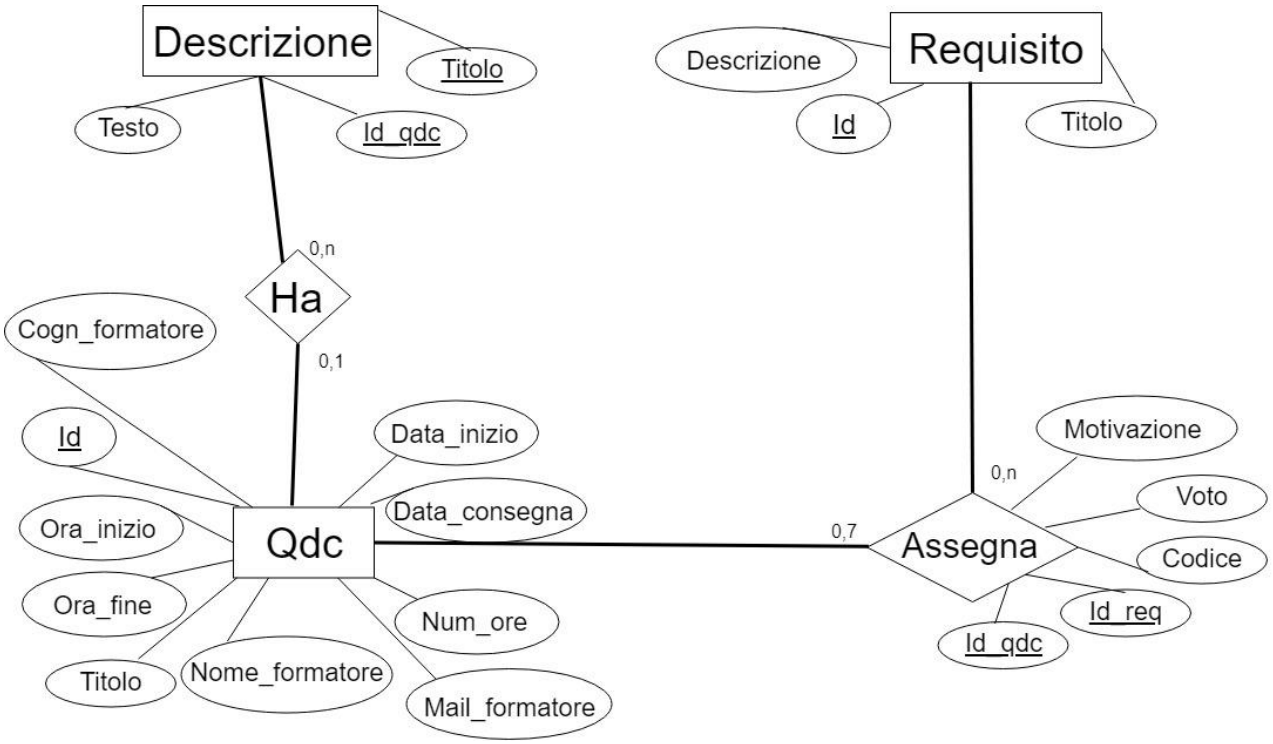
Il progetto verrà svolto su Visual Studio 2017. La struttura del progetto è questa:

- Blank Solution
  - Progetto WPF
    - Views
    - ViewModels
    - Helpers
  - Libreria di classi
    - Models
    - Services
    - Context




Userò il pattern MVVM perché è comodo da usare e perché è uno dei requisiti del progetto. Nella libreria di classi verrà gestita la parte relativa ai dati, mentre nel progetto WPF verranno inserite le maschere e le classi che interagiscono tra View e Models. Inoltre saranno presenti molte classi di aiuto o di supporto.

### 2.2 Design dei dati e database




| **Qdc** | |
|---|---|
| *Id* | PK integer |
| *Titolo* | Varchar |
| *Nome_formatore* | Varchar |
| *Cognome_formatore* | Varchar |
| *Mail_formatore* | varchar |
| *Data_inizio* | datetime |
| *Data_consegna* | datetime |
| *Ora_inizio* | Time |
| *Ora_fine* | Time |
| *Num_ore* | Integer |

Il database è formato dalla tabella principale Qdc, che presenta diversi attributi che contengono le informazioni principali proprio sul quaderno dei compiti. Potranno eventualmente essere aggiunte diverse colonne nel caso fosse necessario.
Id è la chiave primaria ed è autogenerata e incrementale. Il titolo è il titolo scelto per il progetto. Ci sono poi delle informazioni sul formatore e sulle date di inizio e di consegna del progetto, oltre a gli orari di lavoro e al numero totale di ore disponibili per lo svolgimento del progetto.

| **Descrizione** | |
|---|---|
| *Id_Qdc* | PK FK integer |
| *Titolo* | PK Varchar |
| *Testo* | Varchar |

La tabella descrizione contiene appunto la descrizione di un Qdc. È quindi presente come chiave esterna l'Id del Qdc stesso e poi ci sono il campo Titolo e Testo della descrizione.

| **Requisito** | |
|---|---|
| *Id* | PK integer |
| *Titolo* | PK Varchar |
| *Descrizione* | Varchar |

Esiste poi la tabella requisito, che contiene appunto i requisiti, identificati da un id e con gli attributi titolo e descrizione. Il titolo è il titolo del requisito scelto ed è preso dal file word che contiene tutti i requisiti possibili, la stessa cosa vale per la descrizione.

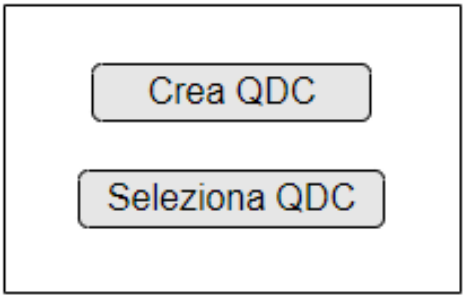
| **Assegna** | |
|---|---|
| *Id_qdc* | PK FK integer |
| *Id_req* | PK FK integer |
| *Codice* | Varchar |
| *Voto* | integer |
| *Motivazione* | varchar |

La tabella fondamentale per il progetto è la tabella Assegna, creata dalla relazione molti a molti tra un qdc e un requisito (infatti un qdc contiene 7 requisiti a scelta e 23 requisiti standard che sono uguali per ogni qdc che viene creato; un requisito può essere associato a diversi qdc). In questa tabella (identificata dai due id delle tabelle alla quali si riferisce), viene assegnato anche un codice (A14-A20). Inoltre sono presenti i campi facoltativi voto e motivazione. Questi campi non sono obbligatori perché vengono assegnati in un secondo momento rispetto alla creazione del qdc e all'assegnazione dei requisiti.

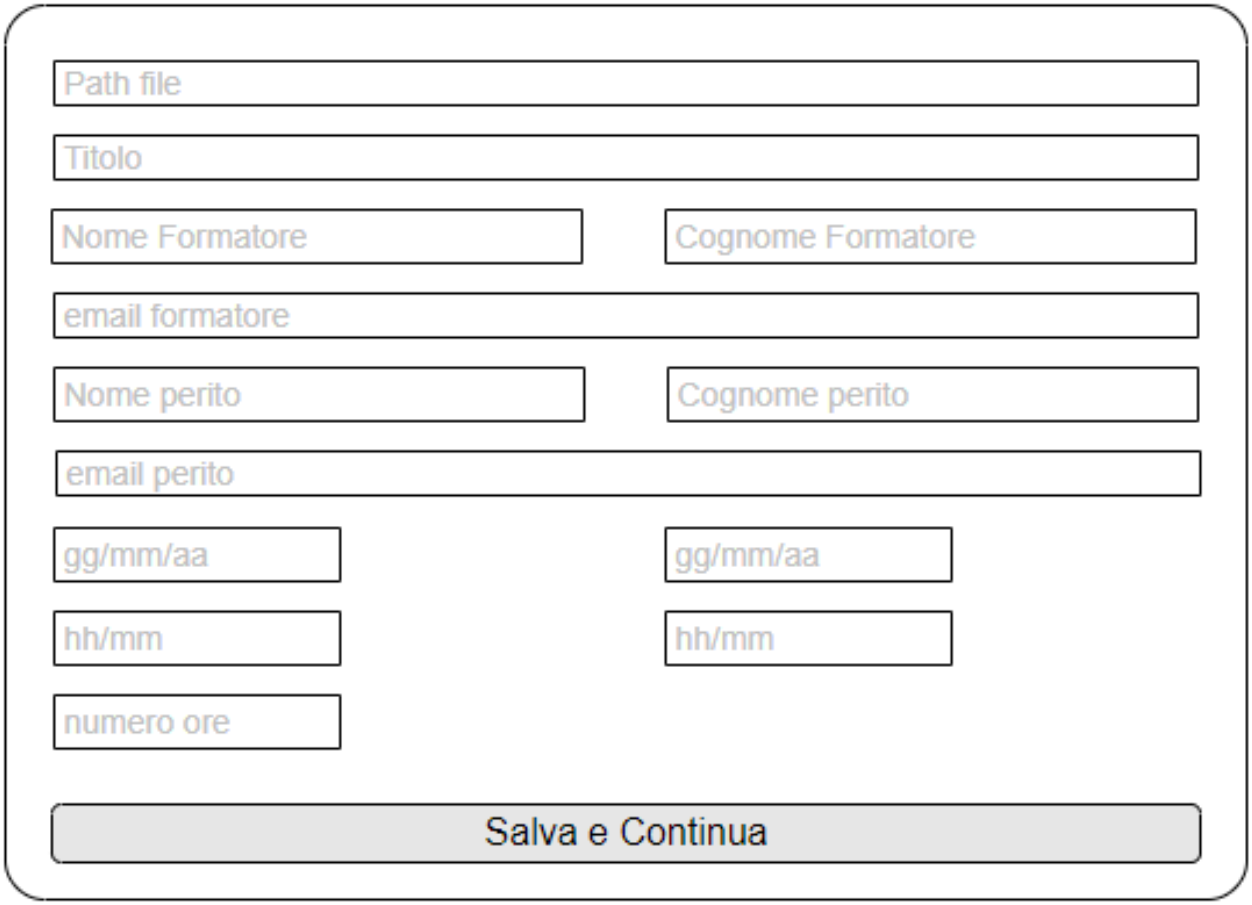
### 2.3 Design delle interfacce

Tutte le maschere che compongono l'applicativo sono sviluppate con l'UI WPF (C#).
La maschera di deafult (compare quando si fa partire l'applicazione) permette di scegliere quale operazione eseguire. A Dipendenza di quale tasto viene premuto si aprono altre maschere che permettono di svolgere il lavoro desiderato.

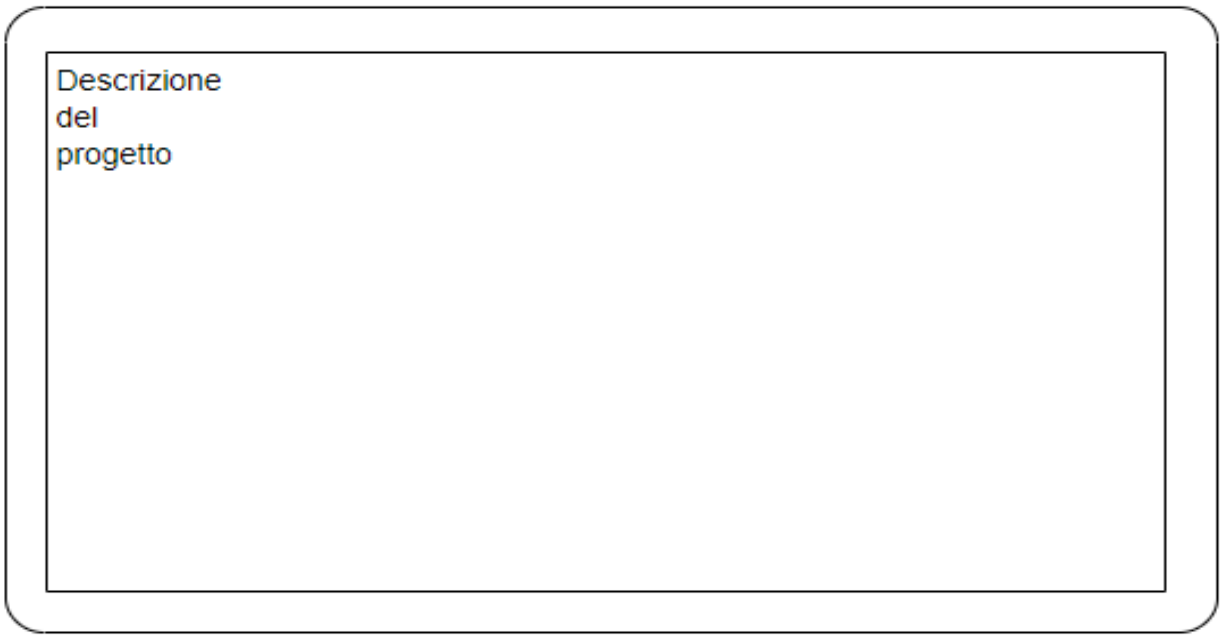
La maschera di default permette di decidere se creare un nuovo quaderno dei compiti oppure di selezionarne uno esistente per assegnare i voti.



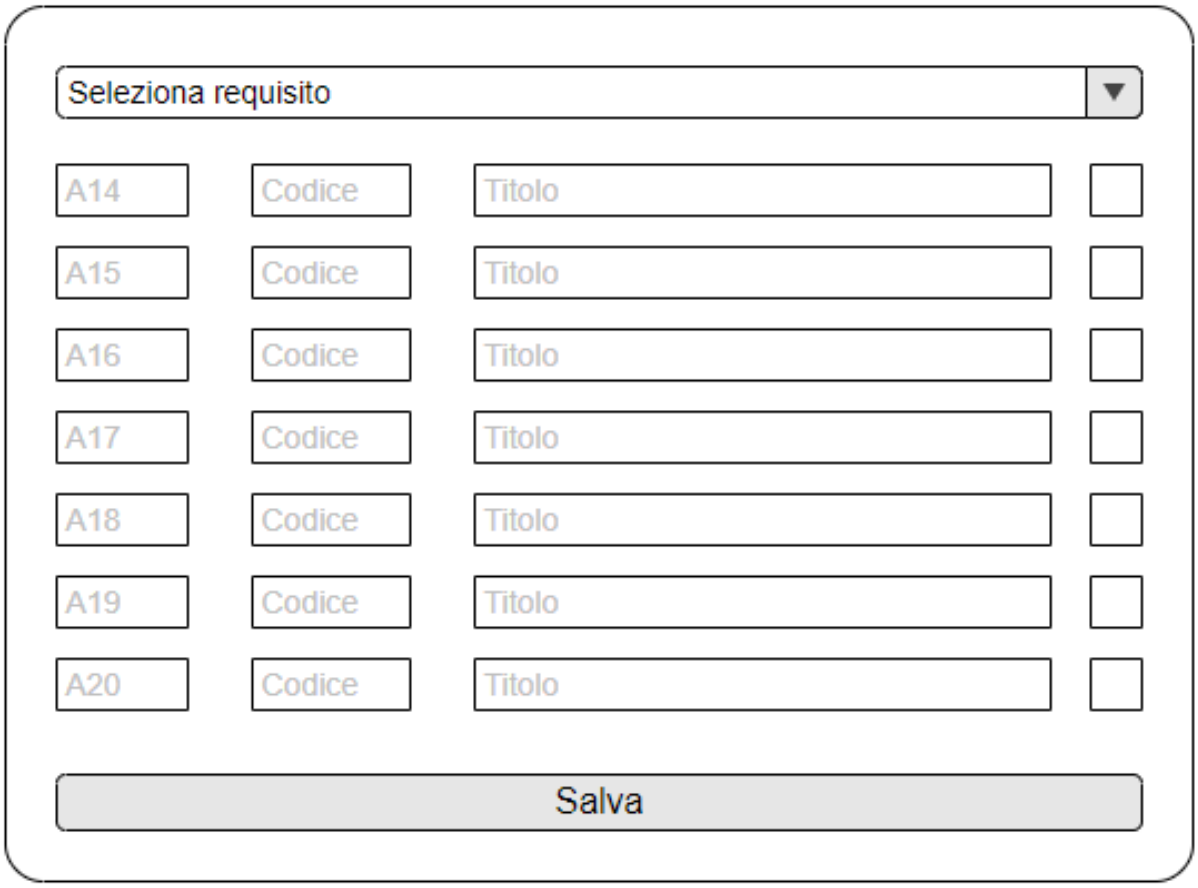

Se viene premuto su Crea QDC nella maschera di default appare questa maschera che serve per inserire le informazioni principali su quaderno dei compiti da creare. Le informazioni che si possono inserire sono il percorso dal quale prendere il file contenente i requisiti estesi, le informazioni sul formatore, sul perito e più generalmente sul progetto (date di inizio e consegna, orario di lavoro, ore totali a disposizione).

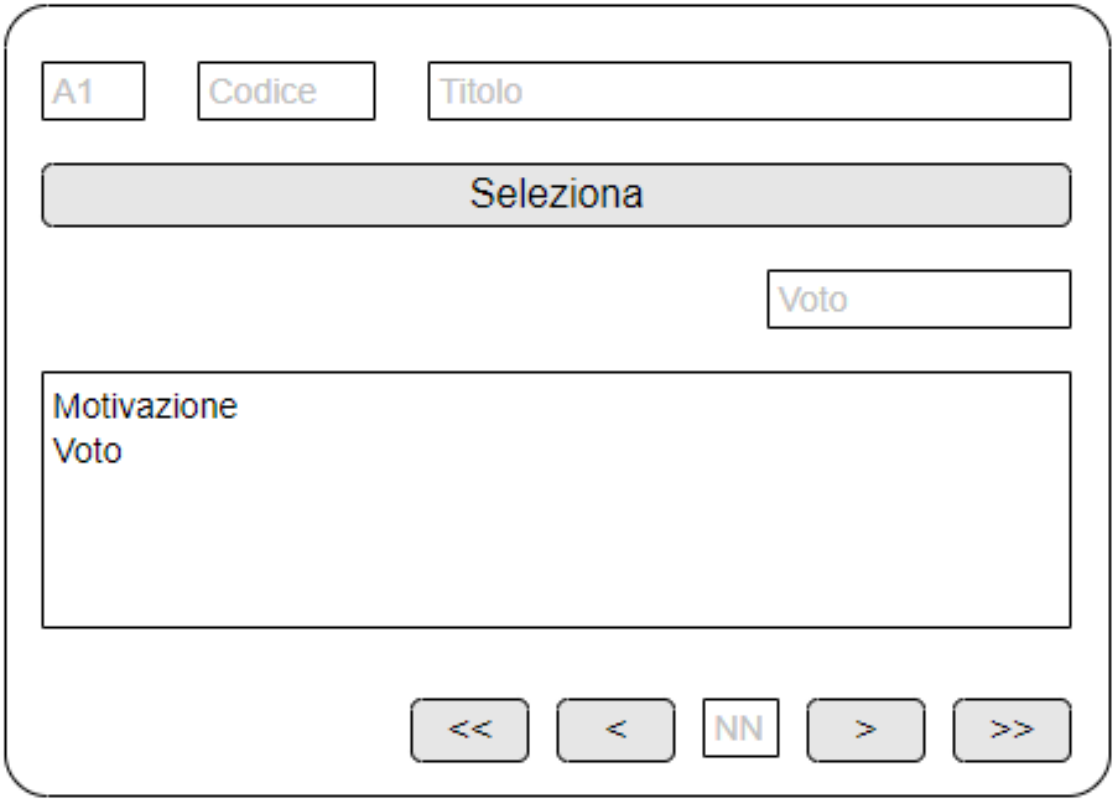



Dopo aver inserito le informazioni generali bisogna inserire la descrizione del progetto. Siccome la descrizione la maggior parte delle volte è abbastanza lunga, in questa maschera ho messo una text area, che permette di inserire testi lunghi e di andare a capo.



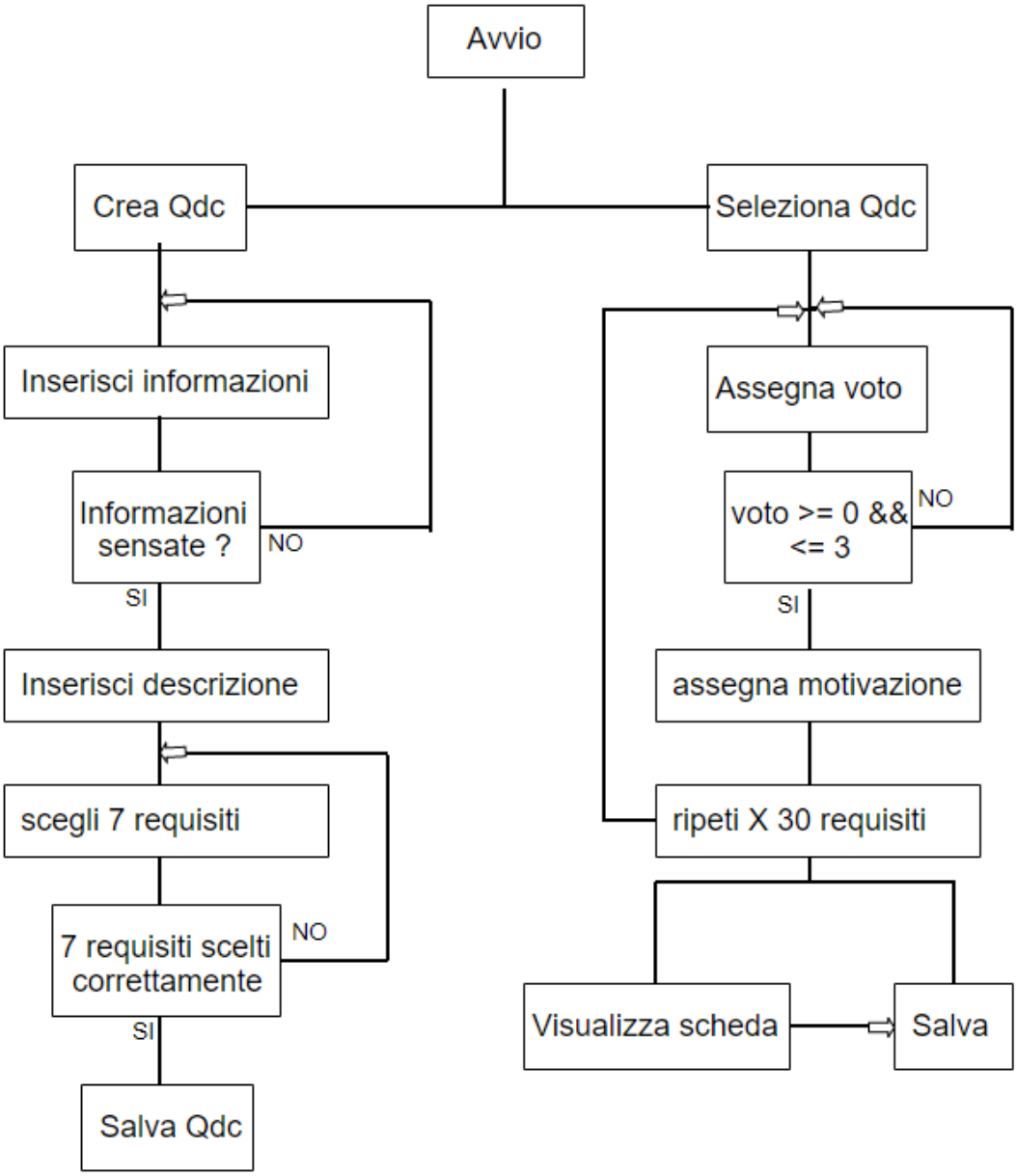
Infine bisogna scegliere i requisiti da assegnare al progetto (7 requisiti da A14 a A20). Per scegliere i requisiti esistono due possibilità. La prima è quella di selezionarli tramite la combo box scegliendo tra tutti i requisiti esistenti. Il requisito scelto viene assegnato al primo campo libero (per esempio A14). La seconda opzione è quella di inserire manualmente il codice del requisito. In questo caso viene automaticamente cercato il requisito corrispondente e verrà scritto il titolo nel campo apposito.
Se ci fosse la necessità di eliminare un requisito scelto, si può premere sul pulsante a destra del requisito.



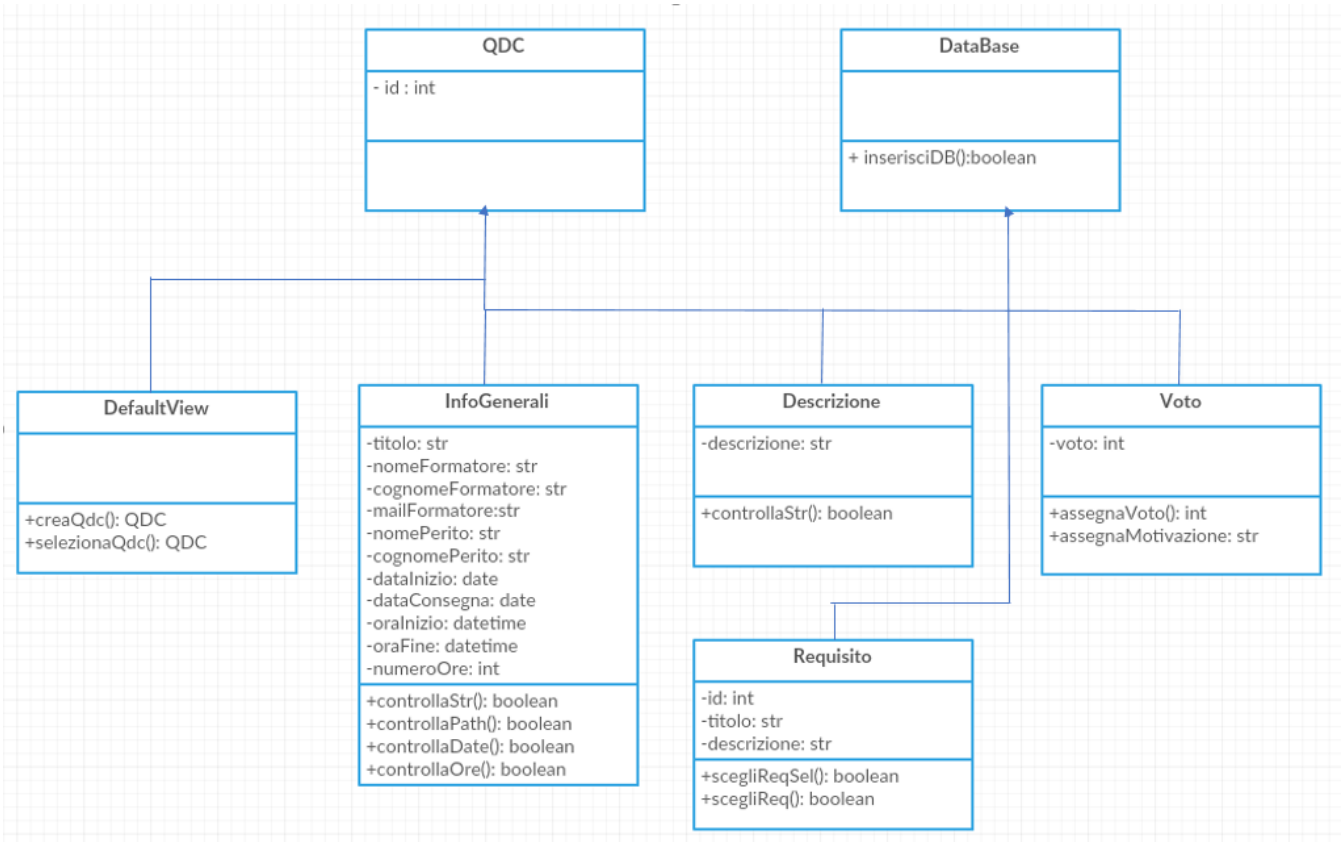



Se viene premuto Seleziona QDC nella maschera iniziale, dopo aver scelto il qdc, appare questa maschera. Questa è la maschera che permette di asseganare i punteggi e le motivazioni per ognuno dei 30 requisiti. Nella prima riga viene mostrato il requisito attuale. Se viene premuto il pulsante seleziona viene mostrato il requisito per esteso nel documento apposito (lo stesso che viene usato per la scelta dei requisiti), in questo modo si possono anche copiare le motivazioni da inserire nella text area. Una volta assegnato il voto si può passare al requisito successivo tramite l'apposito pulsante.

## 2.4 Design procedurale




L'UML si divide in due sezioni ed è simile allo use case. In un caso si sceglie di creare un nuovo qdc e vengono richieste diverse informazioni generiche, una descrizione e 7 requisiti a scelta. Se le informazioni inserite non sono sensate (ad esempio un numero dentro al campo "nome_formatore") viene chiesto di inserire quel campo nuovamente. Lo stesso vale per la scelta dei requisiti (in questo caso vengono richiesti se ad esempio viene scelto due volte lo stesso requisito o ne vengono scelti meno di sette).
Nell'altro caso si sceglie un qdc già creato e viene richiesto di assegnare un voto per ognuno dei 30 requisiti che compongono il progetto e di motivarli. Se il voto inserito non è valido (è un carattere non numerico oppure non rientra nel range da 0 a 3 compresi) viene richiesto l'inserimento per quel requisito. Quando sono stati inseriti tutti i voti è possibile salvare quanto è stato fatto.




Un quaderno dei compiti è rappresentato dalla classe QDC e da varie sottoclassi. La classe QDC fornisce un id che è il numero identificativo del quaderno dei compiti. La classe DefaultView implementa la classe QDC e presenta i due metodi creaQDC() e selezionaQDC() che ritornano un oggetto di tipo qdc. La classe DataBase contiene il metodo inserisciDB() che serve per gestire le query in scrittura sul database. È implementata da tutte le prossime classi (InfoGenerali, Descrizione, Voto e Requisito). La classe InfoGenerali comprende vari attributi generali per il qdc (titolo, informazioni sul formatore, ecc.) e i metodi per controllare che le informazioni inserite siano sensate, come stringhe scritte nel modo giusto, date e orari con il formato corretto e la path dei documenti con un formato adeguato. La classe Descrizione contiene la descrizione e un metodo per controllarla. La classe requisito contiene i requisiti che vengono scelti e i metodi per gestire tale scelta. Infine la classe Voto contiene in voto numerico da 0 a 3 e i metodi per gestire l'assegnazione del voto e della relativa motivazione.

# 3 Implementazione

## 3.1 Ambiente di sviluppo

Dopo aver terminato la progettazione del progetto ho cominciato l'implementazione. La prima cosa che ho fatto è stata l'installazione di Visual Studio 2017. Ho installato la versione 2017 – 15.9.6. Ho scelto i seguenti carichi di lavoro:

- Sviluppo per desktop .NET
- Sviluppo per la piattaforma UWP
- Sviluppo per Office/SharePoint
- Sviluppo multipiattaforma .Net Core

Questo passaggio è stato svolto parallelamente alla progettazione per guadagnare un po' di tempo. Una volta eseguite tutte le installazioni necessarie per cominciare lo sviluppo ho per prima cosa fatto un piccolo progetto di prova per assicurarmi che le dipendenze e i riferimenti generati automaticamente dal tool Menu Samt funzionassero. Durante questo passaggio ho notato che prima di eseguire qualsiasi operazione è necessario compilare la soluzione, altrimenti il riferimento del ViewModel nella View non funziona. Senza compilare potrebbe generarsi un altro errore nel caso il progetto venga chiuso, ovvero che quando lo si riapre non è in grado di caricare automaticamente i files.

### 3.1.1 Struttura progetto (panoramica):

A questo punto ho cominciato con il mio progetto. Ho creato una soluzione vuota (blank solution) chiamata GestioneQDC. In questa ho inserito dapprima un progetto WPF chiamato QDCeValutazioni e poi una libreria di classe chiamata QDCeValutazioni.DA.
Nel progetto WPF ho inserito le cartelle Helper, Model, ViewModel e View. Questa è una breve panoramica del contenuto delle cartelle:

**Helper:** Contiene delle classi di aiuto per svolgere determinate operazioni:
**BindableBase.cs**: implementa InotifyPropertyChanged, serve per generare il metodo OnPropertyChanged che viene richiamato nel ViewModel e si occupa di gestire i Binding.
**RelayCommand.cs**: implementa ICommand, contiene i metodi CanExecute() e Execute(), ha le stesse funzionalità di DelegateCommand e serve appunto per delegare determinate azioni a metodi già esistentei.
**Utility.cs**: implementa EventArgs e serve per gestire gli errori. (se ci fossero degli enum andrebbero inseriti in questa classe.

**Model:** La cartella viene creata automaticamente e ci viene inserita la classe, ma siccome utilizzo anche la libreria di classi, il contenuto del Model.WPF va spostato nel Model.DA

**ViewModel:** Contiene la classe MainViewModel.cs, che contiene vari attributi e metodi per gestire l'interfacciamento tra le view e le classe (funziona come il controller del modello MVC).

**View:** Contiene MainView.cs, la view iniziale del progetto, che contiene sempicemente due bottoni per la scelta tra crea qdc oppure selziona qdc.

nel progetto .DA invece ho inserito le cartelle Model e Service:

**Model:** Contene la classe Main.cs, in precedenza inserita nel Model dell'altro progetto. Verranno inserite le varie classi che rappresentano le tabelle del database.

**Service:** contiene varie classi che offrono delle funzionalità. Principalmente sono classi di supporto per eseguire delle operazioni con i dati.

**Context:** è un file che contiene l'inizializzazione del database.

Dopo aver creato la struttura generale ho installato SQL Server (2017 Express Edition). Con SQL Server posso decidere se salvare il database in locale oppure usare un istanza SQLEXPRESS. Durante il corso del progetto ho deciso di lavorare con un database SQLite, quindi SQL Server non mi serve.

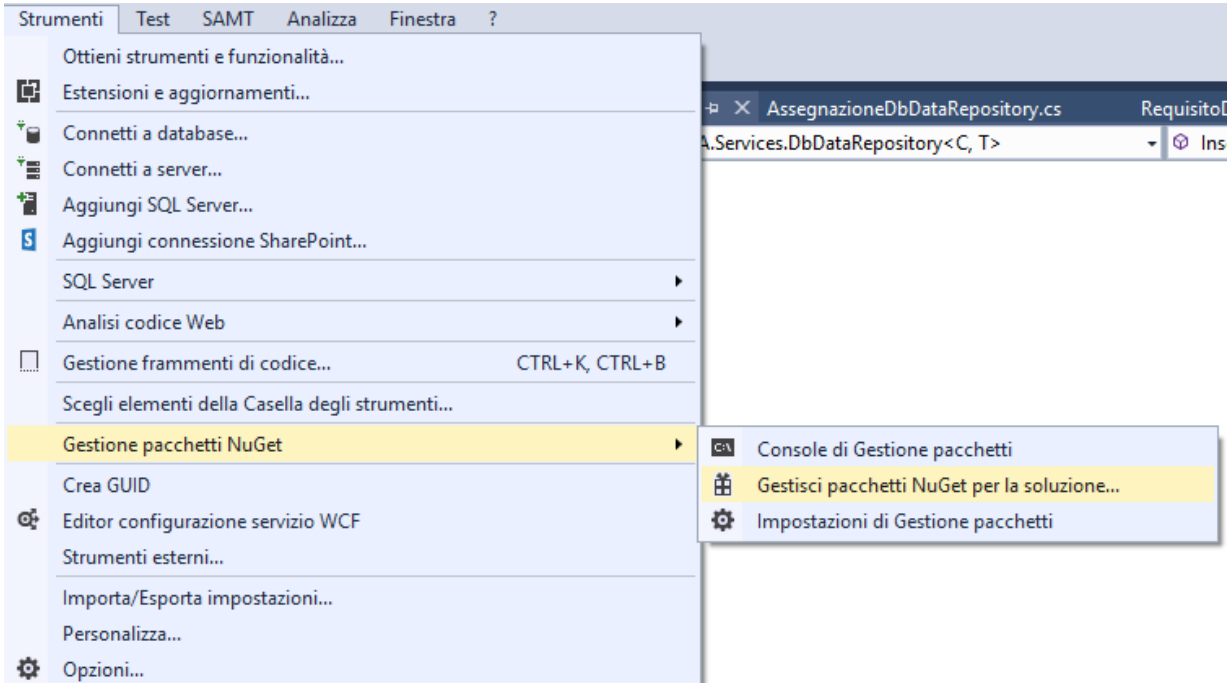
### 3.1.2 Pacchetti NuGet

Per lavorare con i database è necessario l'uso di pacchetti NuGet, ovvero dei pacchetti contenenti dei files DLL con codice compliato che sono condivisi da altri utenti. Esistono molti pacchetti disponibile e alcuni di questi sono resi disponibili dalla Microsoft stessa. Per il mio progetto servono i seguenti pacchetti:

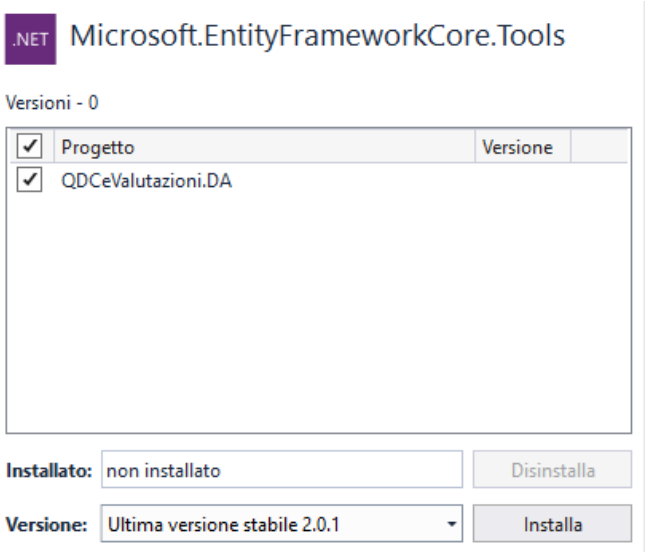
- EntityFramework
- EntityFrameworkCore
- EntityFrameworkCore.Sqlite
- EntityFrameworkCore.Tools

Questi pacchetti permettono di lavorare con dei depositi di dati (quindi anche con dei database)

Per installare un pacchetto Nuget sono andato su Strumenti -> Gestione pacchetti NuGet -> Gestione pacchetti NuGet per la soluzione.

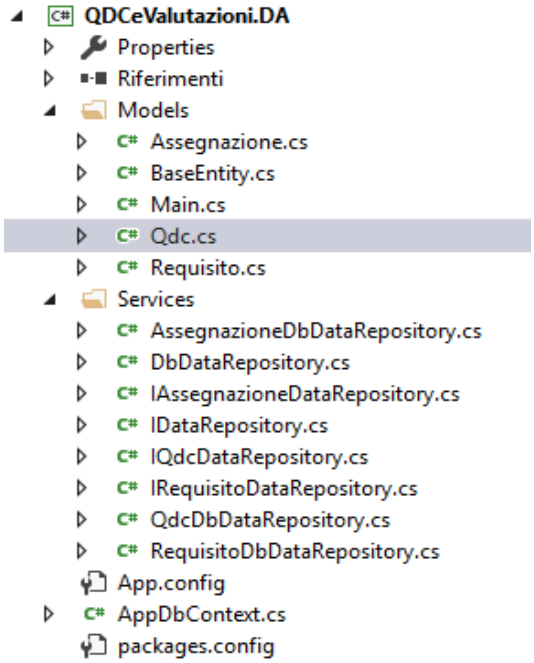



A questo punto si apre un menu per la scelta dei pacchetti da installare. È possibile anche eseguire una ricerca per trovare il pacchetto giusto.Dopo aver scelto il pacchetto desiderato ho premuto semplicemente Installa.



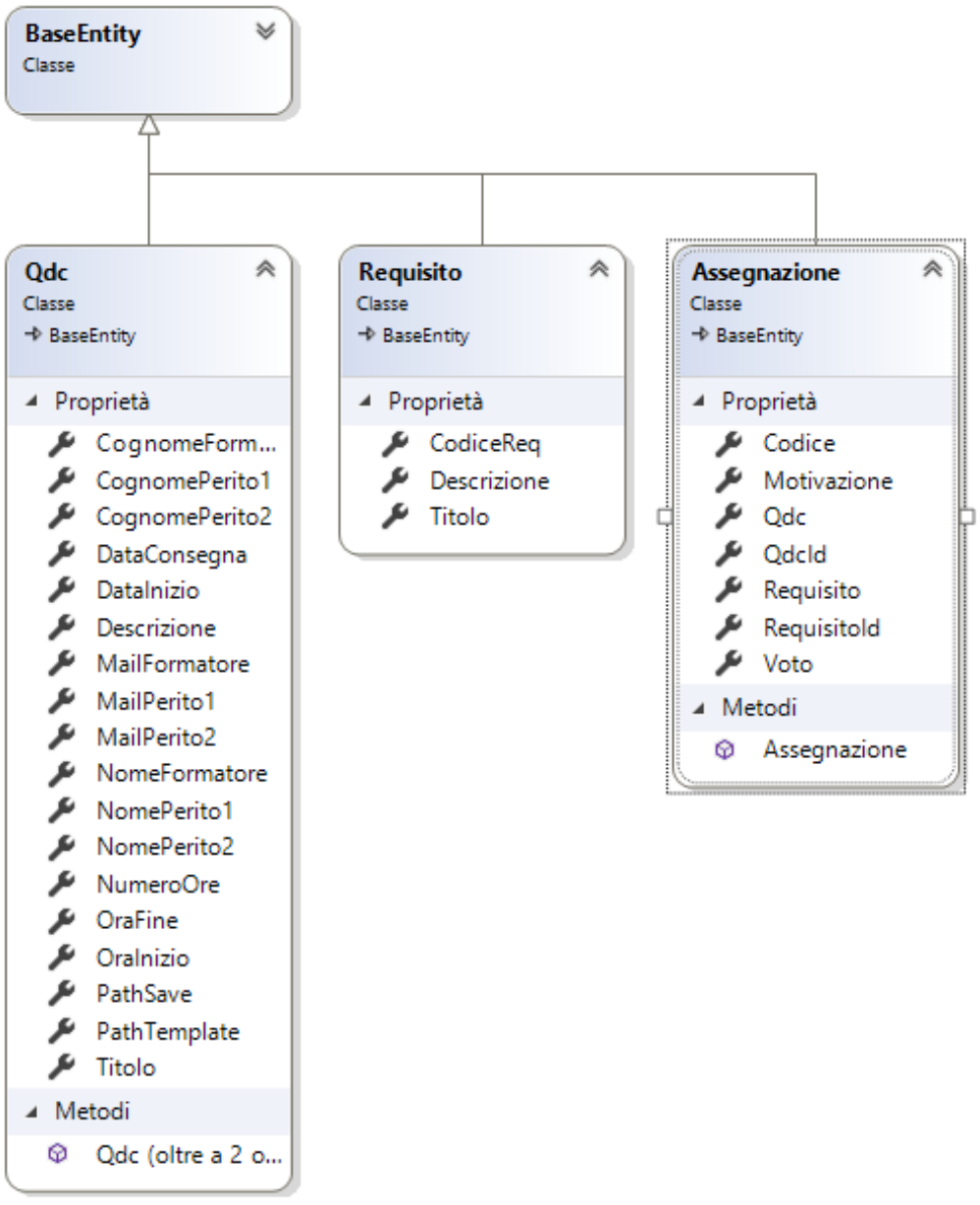
### 3.2 Libreria di classi:

Dopo che l'ambiente di sviluppo è pronto e la struttura MVVM è creata correttamente, ho potuto cominciare dalla libreria di classi. Questo progetto si occupa principalmente della gestione dei dati e di tutto quello che concerne il database. Ho chiamato questo progetto QDCeValutazioni.DA.



### 3.2.1 Models

La directory Models contiene tutte quelle classi che si occupano di rappresentare i dati. In pratica ogni classe rappresenta una tabella del database, ad esempio la classe Requisito.cs rappresenta la tabella Requisito e contiene i campi del database come attributi. Fanno eccezione in questo caso la classe BaseEntity, che contiene semplicemente un id e la classe Main che contiene dei metodi di supporto. Le classi vengono definite in base allo schema ER fatto durante la fase di progettazione.
Lo schema UML è diverso da quello della progettazione:




Le differenze stanno principalemente nel fatto che durante la fase di progettazione non avevo previsto di separare Models e ViewModels. Lo schema dei ViewModels è nel capitolo dedicato.

#### 3.2.1.1 BaseEntity

BaseEntity è la classe che definisce l'id. Siccome tutte le tabella necessitano un id ho creato una classe che funge da superclasse e viene implementata da tutte le altre. Questa classe nonostante sia nel Model non rappresenta nessuna tabella del database vera e propria ma serve esclusivamente come classe di supporto.

```
/// <summary>
/// BaseEntity è la classe che implementano
/// tutte le altre classi.
/// </summary>
public class BaseEntity
{
    /// <summary>
    /// Numero identificativo che serve a tutte le classi
    /// </summary>
    public int Id { get; set; }
}
```

#### 3.2.1.2 Qdc

La classe Qdc rappresenta la tabella Qdc del database. Questa è la classe principale e contiene tutte le informazioni generiche di un quaderno dei compiti. Questa classe implementa BaseEntity, dalla quale prende l'Id.

```
/// <summary>
/// Classe qdc che rappresenta la tabella qdc nel database.
/// Contiene tutti i campi che definiscono un qdc
/// </summary>
public class Qdc : BaseEntity
```

È presente il titolo del quaderno dei compiti e delle informazioni sul formatore che si occupa del progetto.

```
 /// <summary>
 /// titolo del qdc.
 /// </summary>
 public string Titolo { get; set; }

 /// <summary>
 /// nome, cognome e mail del formatore che si occupa del progetto.
 /// </summary>
 public string NomeFormatore { get; set; }

 public string CognomeFormatore { get; set; }

 public string MailFormatore { get; set; }
```

Sono presenti delle informazioni sulle tempistiche del progetto. Al momento dei test queste tempistiche erano facoltative, tramite il ? dopo il tipo di dato (DateTime?).

```
/// <summary>
/// date di inizio e consegna del progetto.
/// </summary>
public DateTime? DataInizio { get; set; }

public DateTime? DataConsegna { get; set; }
```

È presente anche il campo descrizione, che ovviamente rappresenta la descrizione del quaderno dei compiti. Durante la fase di progettazione questo per questo campo avevo pensato a una tabella specifica, poi ho deciso che non valeva la pena di sprecare risorse per un solo campo e quindi l'ho inserito nel Qdc.

```
/// <summary>
/// descrizione del progetto (si aggiunge in un altra view)
/// </summary>
public string Descrizione { get; set; }
```

Oltre a questi campi sono presenti anche due attributi per la definizione dei percorsi dei file Word (un file di template e il percorso nel quale salvare il nuovo file).
Nella classe c'è anche un costruttore, che permette di salvare i nuovi dati sul database. Questo costruttore riceve come parametri tutti i campi che devono essere inseriti:

```
public Qdc(string titolo, string nomeFormatore, string cognomeFormatore, string
mailFormatore, . . . ,string pathSave)
{
    Titolo = titolo;
    NomeFormatore = nomeFormatore;
    CognomeFormatore = cognomeFormatore;
    MailFormatore = mailFormatore;
    . . .
    PathSave = pathSave;
}
```

#### 3.2.1.3 Requisito

La classe Requisito rappresenta la tabella Requisito del database. Questa classe contiene le informazioni riferite a un requisito, ovvero il titolo e la descrizione. Ovviamente contiene anche un Id, che prende dalla classe BaseEntity. Questa classe contiene dei valori statici e predefiniti, cioè tutti quanti i requisiti che possono essere scelti. Questi requisiti sono presi dal file "Criteri di valutazione LPI (estesi).docx".

```
public class Requisito : BaseEntity
{
    /// <summary>
    /// titolo del requisito.
    /// </summary>
    public string Titolo { get; set; }

    /// <summary>
    /// descrizione del requisito.
    /// </summary>
    public string Descrizione { get; set; }

    /// <summary>
    /// codice identificativo di un requisito (A01 - A30).
    /// </summary>
    public string CodiceReq { get; set; }
}
```

L'Id è commentato perché viene usato quello presente in Base Entity.

#### 3.2.1.4 Assegnazione

La classe assegnazione rappresenta la tabella Asseganzione del database. È una tabella generata dalla relazione molti a molti tra Requisito e Qdc, infatti contiene un campo per entrambe le classi. Inoltre è la classe dove vengono gestiti i voti e le motivazioni assegnate a ogni requisito.

```
public class Assegnazione : BaseEntity
{
    /// <summary>
    /// Requisito da assegnare a un qdc.
    /// </summary>
    public virtual Requisito Requisito { get; set; }

    public int RequisitoId { get; set; }

    /// <summary>
    /// qdc al quale vanno assegnati i requisiti.
    /// </summary>
    public virtual Qdc Qdc { get; set; }

    public int QdcId { get; set; }
    /// <summary>
    /// voto (da 0 a 3) per un requisito di un qdc.
    /// </summary>
    public int? Voto { get; set; }

    /// <summary>
    /// codice identificativo di un requisito (A01 - A30).
    /// </summary>
    public string Codice { get; set; }

    /// <summary>
    /// Motivazione per il voto.
    /// </summary>
    public string Motivazione { get; set; }
}
```

Anche questa classe contiene un costruttore, per permettere di inserire gli Id del quaderno dei compiti e dei requisiti e anche il codice numerico del requisito.

```
public Assegnazione(string codice , Qdc qdc, int req)
{
    Codice = codice;
    QdcId = qdc.Id;
    RequisitoId = req;
}
```

#### 3.2.1.5 Main

La Classe main non contiene niente.

```
public class Main
{
}
```

### 3.2.2 Services

La directory Services contiene tutte quelle classi e interfacce che servono per fare delle operazioni sul database. Sono presenti i metodi che si occupano di gestire delle query sui dati, le principali sono quelle che permettono di selezionare (select), inserire (insert), modificare (update) e eliminare (delete) i dati.

#### 3.2.2.1 IDataRepository

IData Repository l'interfaccia principale tra tutti i servizi. Contiene i metodi principali per la gestione delle operazioni su un database (get, insert, update e delete). Il parametro <T> rappresenta il modello di dati (quindi la classe) al quale fa riferimento. La condizione è che questo modello di dati implementi BaseEntity, come infatti fanno tutte le classi presenti nel Model.

```
/// <summary>
/// Interfaccia di base che implementa i metodi relativi alle
/// operazioni principali sui dati di un database.
/// </summary>
/// <typeparam name="T">Modello di dati di riferimento</typeparam>
public interface IDataRepository<T> where T : BaseEntity
{
    /// <summary>
    /// Ritorna un'entità in base all'id passato.
    /// funziona come una select.
    /// </summary>
    T Get(int id);

    /// <summary>
    /// Ritorna tutte le entità presenti.
    /// funziona come una select all.
    /// </summary>
    IQueryable<T> Get();

    /// <summary>
    /// Inserisce una nuova entità.
    /// </summary>
    T Insert(T entity);

    /// <summary>
    /// Update di un'entità.
    /// </summary>
    void Update(T entity);

    /// <summary>
    /// Elimina un'entità.
    /// </summary>
    void Delete(T entity);
}
```

#### 3.2.2.2 DbDataRepository

DbDataRepository è la classe principale tra i servizi, infatti implementa IDataRepository. Serve principalmente da contenitore per i dati da gestire. Infatti presenta una variabile context che rappresenta il contesto dei dati di riferimento.

```
/// <summary>
/// Deposito di dati che implementa i metodi principale relativi al database.
/// </summary>
/// <typeparam name="C">Contesto di dati di riferimento</typeparam>
/// <typeparam name="T">Modello di dati di riferimento</typeparam>
public abstract class DbDataRepository<C, T> : IDataRepository<T> where T : BaseEntity
                                                                  where C : DbContext, new()
{
    /// <summary>
    /// rappresenta i dati del database
    /// </summary>
    protected C context;

    /// <summary>
    /// definizione del contesto di dati del database.
    /// </summary>
    protected DbDataRepository(C ctx)
    {
        context = ctx;
    }
```

La variabile context viene implementata nel costruttore e viene poi usata per eseguire delle operazioni sul database. Infatti in questa classe vengono implementati tutti i metodi presenti nell'interfaccia IDataRepository, come ad esempio l'insert. In questo caso l'insert riceve come parametro l'entità da inserire e tramite un setter applicato alla variabile context viene salvata sul database.

```
/// <summary>
/// Inserisce una nuova entità.
/// </summary>
/// <param name="entity">Entità da inserire.</param>
/// <returns>l'entità inserita</returns>
public virtual T Insert(T entity)
{
    context.Set<T>().Add(entity);
    context.SaveChanges();
    return entity;
}
```

Ovviamente in DbDataRepository sono implementati anche tutti gli altri metodi presenti in IDataRepository.

#### 3.2.2.3 IRequisitoDataRepository

IrequisitoDataRepository è l'interfaccia che si occupa di implementare i metodi principali per le operazioni sul database (Quelli presenti in IDataRepository che viene implementata) in riferimento alla classe Requisito.

```
/// <summary>
/// Interfaccia che implementa i metodi principali relativi al database
/// (presenti in IDataRepository), relativi al modello corrente, cioè Requisito.
/// </summary>
public interface IRequisitoDataRepository : IDataRepository<Requisito>
{
}
```

Esiste un'interfaccia di questo tipo per ogni classe del Model (ad eccezione di Main e BaseEntity che non rappresentano nessuna tabella) secondo il pattern INomeClasseDataRepository.

#### 3.2.2.4 RequisitoDbDataRepository

RequisitoDBDataRepository è una classe che implementa DbDataRepository. Si occupa di definire il contesto dei dati, in questo caso dlla classe Requisito. Possono essere implementati tramite un override i vari metodi di DbDataRepository, come ad esempio il Get, per eseguire delle operazioni sulla tabella specifica.

```
/// <summary>
/// Deposito dati relativo alla classe Requisito
/// </summary>
public class RequisitoDbDataRepository : DbDataRepository<AppDbContext, Requisito>,
IRequisitoDataRepository
{
    /// <summary>
    /// Definizione del contesto di dati.
    /// </summary>
    /// <param name="ctx">Contesto di dati del database.</param>
    public RequisitoDbDataRepository(AppDbContext ctx) : base(ctx)
    {
       //throw new NotImplementedException();
    }

    /// <summary>
    /// Ritorna tutti requisiti ordinati per titolo.
    /// </summary>
    /// <returns>i requisiti ordinati per titolo.</returns>
    public override IQueryable<Requisito> Get()
    {
        return base.Get().OrderBy(s => s.Titolo);
    }
}
```

Esiste una classe di questo tipo per ogni classe del Model (ad eccezione di Main e BaseEntity che non rappresentano nessuna tabella) secondo il pattern NomeClasseDbDataRepository. Ovviamente per ogni classe cambiano i parametri da inserire nella dichiarazione della classe. Per esempio la classe QdcDbDataRepository viene definita in questo modo:

```
public class QdcDbDataRepository : DbDataRepository<AppDbContext, Qdc>, IQdcDataRepository
```

#### 3.2.2.5 AppDbContext

AppDbContext è un'altra classe molto importante. Si occupa di impostare tutte le entità del database e il percorso nel quale salvare i dati. Vengono definiti i modelli di dati per tutte le classi presenti nel Model.

```
/// <summary>
/// Classe che imposta le entità del database e il percorso
/// di salvataggio del db.
/// </summary>
public class AppDbContext : DbContext
{
    /// <summary>
    /// Raccolta di Qdc.
    /// </summary>
    public DbSet<Qdc> Qdcs { get; set; }

    /// <summary>
    /// Raccolta di Requisiti.
    /// </summary>
    public DbSet<Requisito> Requisiti { get; set; }

    /// <summary>
    /// Raccolta di Assegnazioni.
    /// </summary>
    public DbSet<Assegnazione> Assegnazioni { get; set; }
```

Il metodo costruttore si occupa di creare il database. Può essere senza parametri oppure può contenere il nome della stringa di connessione per effettuare il collegamento (presente nel file app.config).

```
public AppDbContext() : base()
{

}

/// <summary>
/// preparazione database (costruttore alternativo).
/// </summary>
/// <param name="options">Configurazioni del database.</param>
public AppDbContext(DbContextOptions<AppDbContext> options) : base(options)
{
}
```

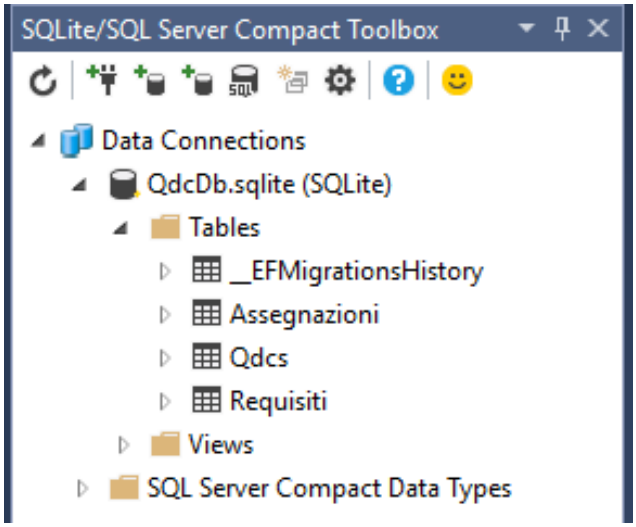
In questo caso non userò nessuno di questi due metodi, bensì ho sovrascitto il metodo OnConfiguring() nel quale ho impostato il percorso nel quale creare e in seguito utilizzare il Database. Ho usato Sqlite (che funziona solo importando il pacchetto NuGet EntityFrameworkCore.Sqlite).

```
/// <summary>
/// Percorso di salvataggio del database e dei dati.
/// </summary>
/// <param name="optionsBuilder">Opzioni del contesto di dati presenti nel sistema.</param>
protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
{
    if (!optionsBuilder.IsConfigured)
    {
        optionsBuilder.UseSqlite("Data Source=C:\\Users\\lucas\\source\\repos\\GestioneQdc"
+
            "\\QDCeValutazioni.DA\\QdcDb.sqlite");
    }
}
```

A questo punto è stato possibile generare il database. Per farlo ho utilizzato la Console di gestione pacchetti di NuGet. I comandi da eseguire sono in ordine:

- Add-Migration Initial
- Update-Database

A dipendenza della versione del framework installato si può utilizzare il comando Enable-Migrations al posto di Add-Migration Initial.
Se l'operazione va a buon fine viene generata la cartella Migrations contenente due file autogenerati. Siccome sto usando un database SQLite, nella finestra "Esplora gli oggetti di SQL Server" non viene visualizzato. Per rendere visibili i database SQLite bisogna installare un tool apposito. Questo tool si chiama SQLite/SQL Server Compact Toolbox e permette di scegliere quale tipo di database visualizzare nel exlporer di Visual Studio.




Con il proseguimento del progetto ho installato e usato un altro Tool esterno a Visual Studio per gestire il mio Database. Il programma che ho utilizzato si chiamo DbBrowser ed è molto più completo e immediato, soprattutto per database SqLite, come nel mio caso.
(https://download.sqlitebrowser.org/DB.Browser.for.SQLite-3.11.2-win64.msi)

## 3.3 Progetto WPF

Questo progetto (App WPF .NET Core) si trova nella stessa soluzione di QDCeValutazioni.DA e si chiama QDCeValutazioni. Il suo scopo è quello di occuparsi di gestire i dati definiti nella libreria di classi e poi mostrarli all'utente tramite delle maschere (View) che può eseguirci delle operazioni.

### 3.3.1 Helpers

In questa Directory vengono inseriti tutte le classi che lavorano in background e che non interessano minimamente all'utente. Queste classi servono come classi di supporto per eseguire determinate operazioni, come ad esempio l'OnPropertyChanged.

#### 3.3.1.1 BindableBase

La classe BindableBase si occupa di implementare InotifyPropertyChanged, che gestisce le notifiche quando vengono appunto cambiati dei valori. Questa classe viene implementata da tutti i ViewModel, in modo che possano delegare a questa classe la gestione del OnPropertyChanged.

```
/// <summary>
/// Classe di base che viene implementata da tutti i ViewModel
/// </summary>
public abstract class BindableBase : INotifyPropertyChanged
{
    public event PropertyChangedEventHandler PropertyChanged = delegate { };

    protected void OnPropertyChanged(string propertyName)
    {
        PropertyChangedEventHandler handler = PropertyChanged;
        if (handler != null)
        {
            handler(this, new PropertyChangedEventArgs(propertyName));
        }
    }
```

#### 3.3.1.2 DelegateCommand

DelegateCommand è una classe che implementa l'interfaccia IDelegateCommand e viene utilizzata dai ViewModel per delegare delle operazioni come ad esempio CanExecute e OnExecute. Questi due campi vengono passati dal Costruttore dalla MainViewModel.

```
/// <summary>
/// Classe che implementa IDelegateCommand e i metodi
/// CanExecute e Execute.
/// </summary>
public class DelegateCommand : IDelegateCommand
{
    Action<object> execute;
    Func<object, bool> canExecute;
    // Evento richiesto da ICommand
    public event EventHandler CanExecuteChanged;
    // Construttore
    public DelegateCommand(Action<object> onExecute, Func<object, bool> canExecute =
null)
    {
        this.execute = onExecute;
        this.canExecute = canExecute;
    }

    // Metodi richiesti da ICommand
    #region ICommand
    public void Execute(object parameter)
    {
        execute(parameter);
    }
    public bool CanExecute(object parameter)
    {
        bool b = canExecute == null ? true : canExecute(parameter);
        return b;
    }
}
```

#### 3.3.1.3 Utility

È una classe che contiene la gestione dei messaggi di errore.

```
public class MessageEventArgs : EventArgs
{
    public string Message { get; set; }
}
```

Nel progetto sono presente altre classi di supporto che aiutano a svolgere determinate operazioni.

### 3.3.2 Views

La cartella Views contiene tutti i file UserControl che rappresentano la parte Front-head dell'applicativo, ovvero ciò che viene mostrato all'utente. A dipendenza della View l'utente potrà svolgere un determinato tipo di azioni sula View stessa e riceverà un determinato Feedback. Questo dipende dallo scopo dell'interfaccia. Tendenzialmente ogni View è associata a un ViewModel che ne gestisce la parte logica.

#### 3.3.2.1 MainWindow

MainWindow è la finestra che si apre quando viene fatta partire l'applicazione. Siccome sto usando il pattern MVVM non utilizzo questa view, che quindi contiene solamente un riferimento a MainView (che è la view principale del progetto).

```
<Grid>
    <view:MainView></view:MainView>
</Grid>
```

#### 3.3.2.2 MainView

MainView è la view principale del progetto ed è associata a MainViewModel. Contiene un Menu che permette di eseguire delle semplici operazioni, come ad esempio chiudere la finestra. In ContentControl viene eseguito un binding che corrisponde alla proprietà CurrentViewModel del ViewModel che serve a definire la View da mostrare (di default MenuView). Quest'operazione viene eseguita in MainViewModel.

```
<Menu DockPanel.Dock="Top">
    <MenuItem Header="_File">
        <MenuItem Header="_Crea QDC" Command="{Binding Path=QdcListCommand}"/>
        <MenuItem Header="_Seleziona QDC" Command="{Binding Path=SelezionaListCommand}"/>
        <MenuItem x:Name="esciMI" Header="_Esci" Click="EsciMI_Click"/>
    </MenuItem>
    <MenuItem Header="_Info">
        <MenuItem Header="_Aiuto"/>
        <MenuItem x:Name="informazioniMI" Header="_Informazioni"/>
    </MenuItem>
</Menu>

<ContentControl Content="{Binding Path=CurrentViewModel}"/>
```

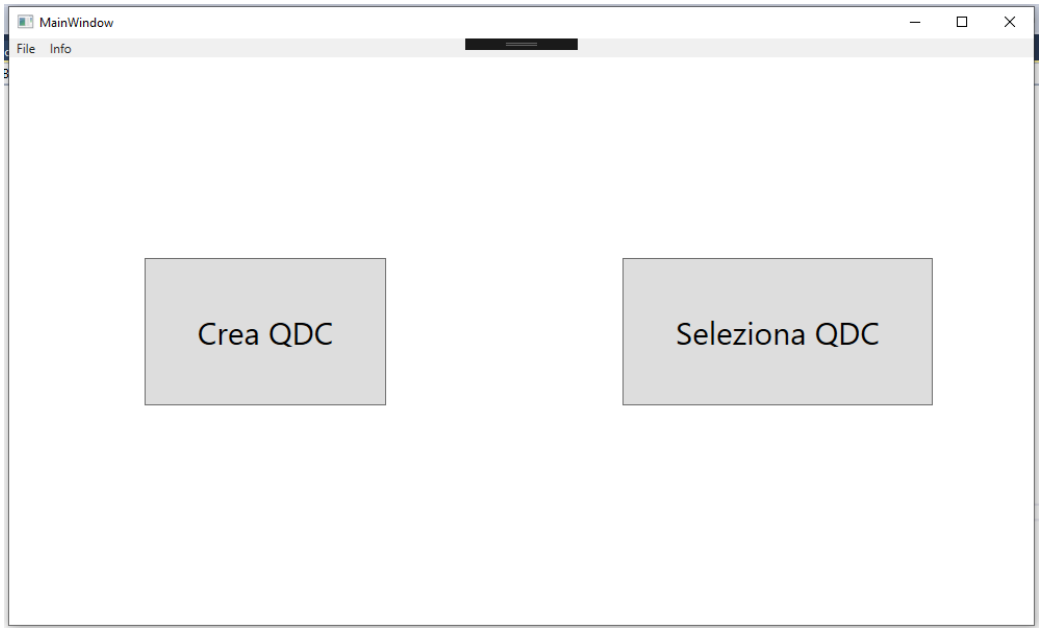
#### 3.3.2.3 MenuView

MenuView è la view che permette di scegliere tramite un pulsante di creare un nuovo Qdc (o eventualmente di selezionarne uno già creato). È associata a MenuViewModel. È una view molto semplice che contiene due pulsanti con un Binding che fa si che quando vengono premuti si passi alla view desiderata (nel caso in cui venga premuto Crea Qdc si passerà a QdcView).

```
<Button
    Grid.Column="0" VerticalAlignment="Center" HorizontalAlignment="Center"
    Content="Crea QDC" Padding="25"
    Command="{Binding Path=QdcListCommand}">
</Button>
```

```
<Button
    Grid.Column="1" VerticalAlignment="Center" HorizontalAlignment="Center"
    Content="Seleziona QDC" Padding="25"
    Command="{Binding Path=SelezionaListCommand}">
</Button>
```

La view si presenta in questo modo:




#### 3.3.2.4 QdcView

QdcView è la view che permette di inserire le informazioni generali di un Qdc (titolo, formatore, periti, date, orari, ecc.). È associata a QdcViewModel. Le informazioni vanno inserite nei TextBox appositi. Tutti i TextBox sono preceduti da delle Labels che aiutano a capire quale tipo di contenuto inserire.

```
<TextBox Grid.Row="6"
         Grid.Column="0"
         Grid.ColumnSpan="2"
         Margin="120,0,20,0"
         x:Name="TitoloTextBox"
         AutomationProperties.HelpText="Titolo"
         Text="{Binding Path=Titolo}" Foreground="Gray"></TextBox>

<TextBox Grid.Row="8"
         Grid.Column="0"
         Grid.ColumnSpan="1"
         Margin="120,0,20,0"
         x:Name="NomeFormatoreTextBox"
         AutomationProperties.HelpText="Nome formatore"
         Text="{Binding Path=NomeFormatore}" Foreground="Gray"></TextBox>
```

Il testo iniziale funzionava come un placeholder, che avevo dovuto gestire “manualmente” perché non c’è una proprietà specifica. Per farlo avevo aggiunto due metodi che mi controllavano quando il cursore entrava nei vari TextBox. Ho tolto questa funzionalità in seguito perché era inutile in quanto non avevo più il placeholder ma delle Labels.

Altri due campi fondamentali di questa View sono i due TextBox che gestiscono i percorsi dei file. Il primo richiede di inserire il percorso del file di template di un quaderno dei compiti (capitolo: File di template), il secondo invece si aspetta il percorso, compreso il nome del file, nel quale salvare il Qdc che si andrà a creare partendo proprio dal template.

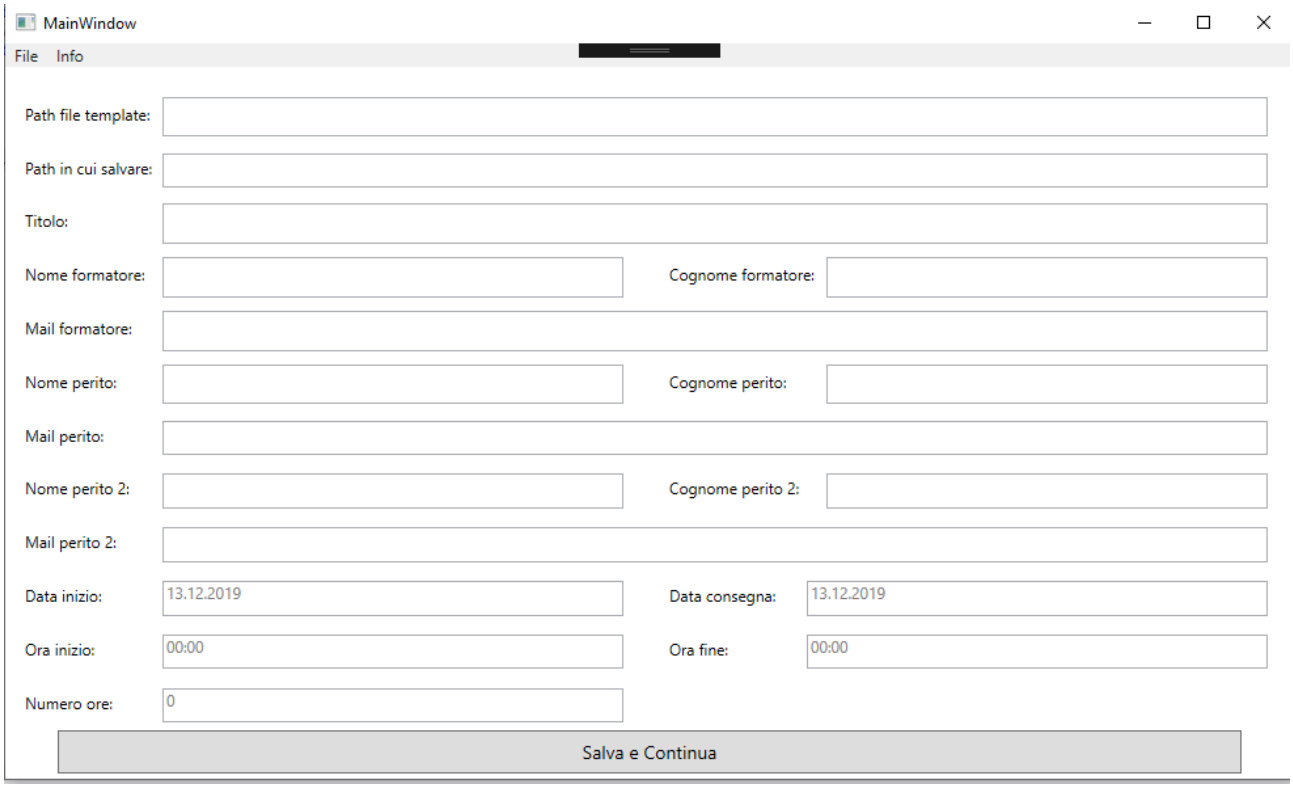
```
<TextBox Grid.Row="2"
         Grid.Column="0"
         Grid.ColumnSpan="2"
         Margin="120,0,20,0"
         x:Name="PathTemplateTextBox"
         Text="{Binding Path=PathTemplate, Mode=TwoWay}" Foreground="Gray" />

<TextBox Grid.Row="4"
         Grid.Column="0"
         Grid.ColumnSpan="2"
         Margin="120,2,20,2"
         x:Name="PathSaveTextBox"
         AutomationProperties.HelpText="Path file"
         Text="{Binding Path=PathSave}" Foreground="Gray"/>
```

Alla fine della View è presente un bottone che ha diverse funzionalità. Si occupa di salvare i dati appena inseriti sul database e anche di scriverli sul file Word in base al percorso inserito nel TextBox apposito. Contemporaneamente gestisce anche la richiesta di passare alla View successiva, ovvero DescrizioneView

```
<Button Grid.Row="25" Grid.Column="0" Grid.ColumnSpan="1"
        Margin="40,4,40,4"
        Content="Salva"
        Command="{Binding Path=SaveInfo}"
        FontSize="14"
        x:Name="SaveInfoButton" Click="SaveInfoButton_Click"/>
```

La view si presenta in questo modo:



#### 3.3.2.5 DescrizioneView

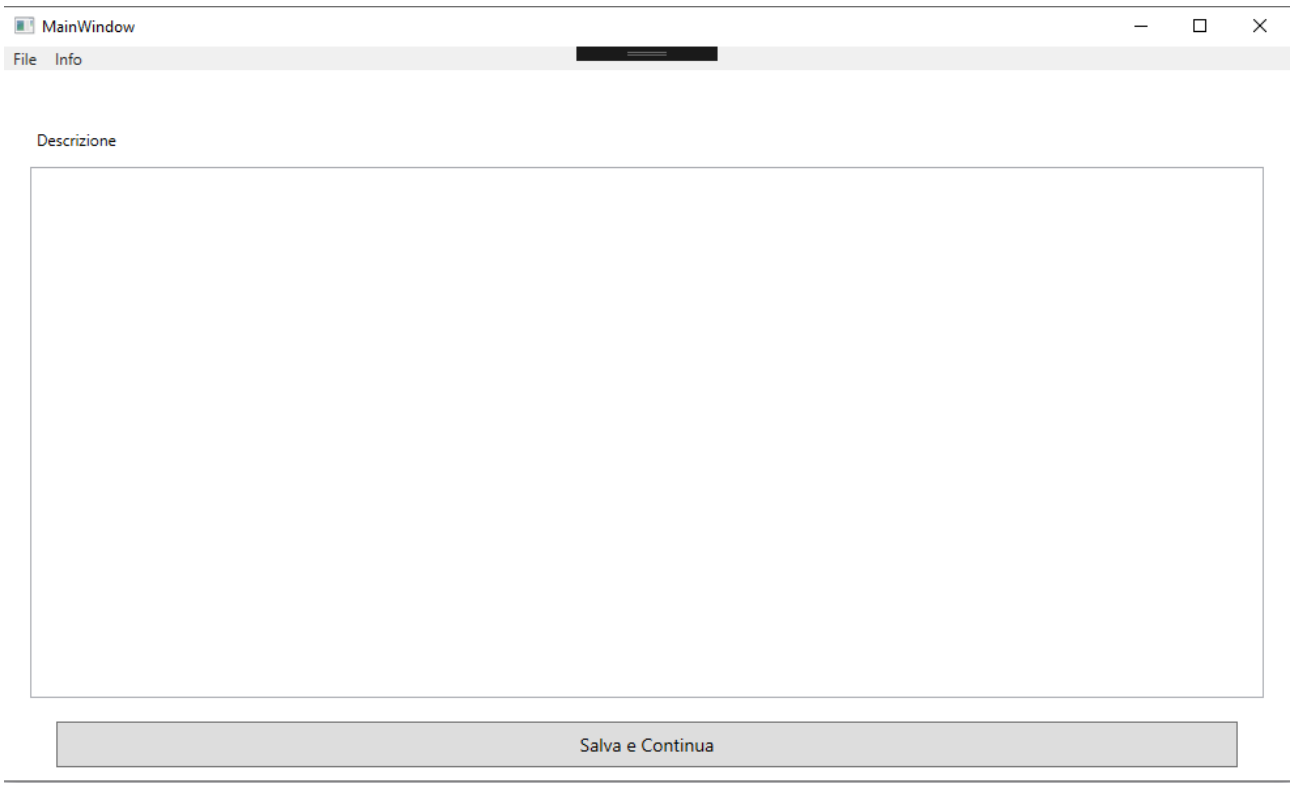
DescrizioneView è la view che permette di inserire una descrizione al quaderno dei compiti. Inizialmente era associata a DescrizioneViewModel. In seguito ho deciso di associarla a QdcViewModel, per il semplice motivo che questa la descrizione è un campo della tabella Qdc, quindi ha più senso che venga gestita nello stesso luogo degli altri campi della stessa tabella. La descrizione va inserita in un TextBox che permette di andare a capo quando viene premuto Enter e che presenta una ScrollBar. Queste funzioni vengono gestite dalle proprietà AcceptsReturn e VerticalScrollBarVisibility.

```
<TextBox
    TextWrapping="Wrap"
    AcceptsReturn="True"
    HorizontalScrollBarVisibility="Disabled"
    VerticalScrollBarVisibility="Auto"
    Grid.Row="2" Grid.Column="0" Grid.ColumnSpan="2"
    Margin="20,8,20,8"
    x:Name="DescrizioneTextBox"
    Text="{Binding Path=Descrizione}"/>
```

Anche in questo caso è presente un pulsante che permette di salvare i dati e accedere alla View successiva, ovvero RequisitoView.

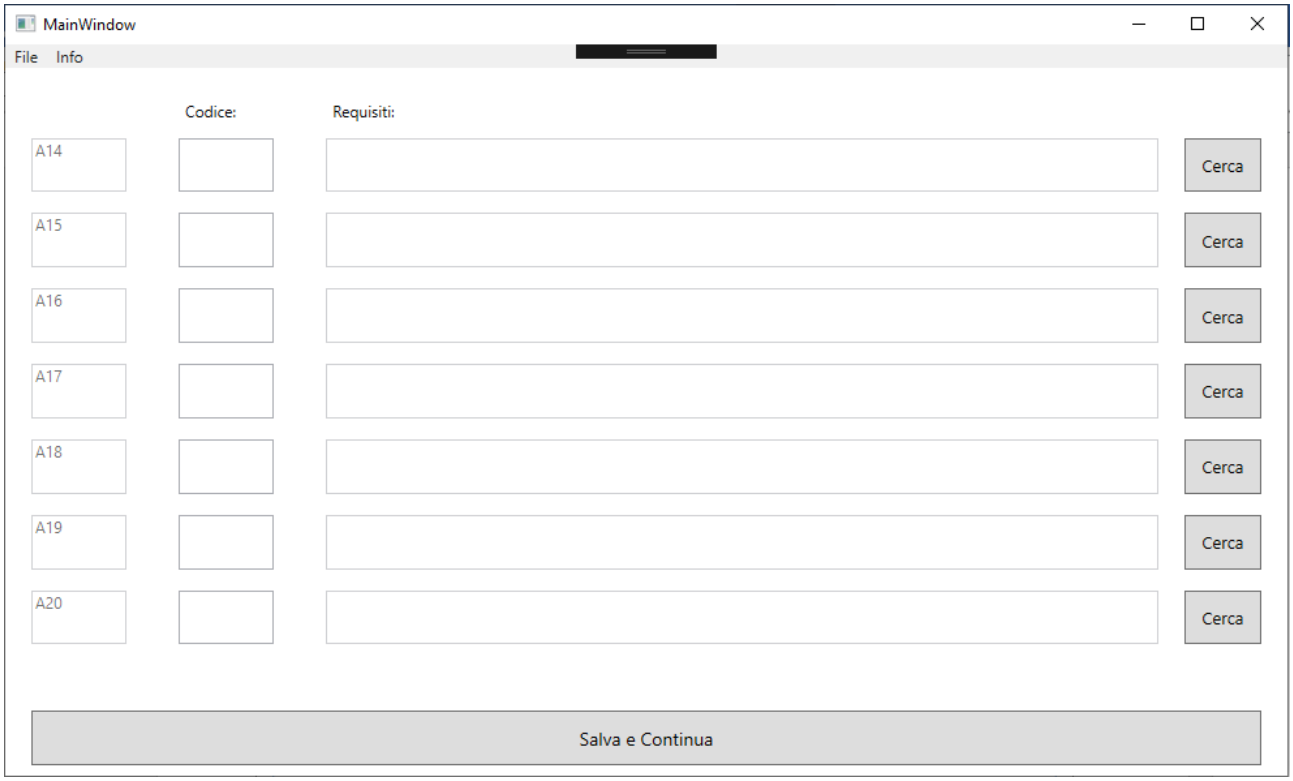
```
<Button Grid.Row="3" Grid.Column="0" Grid.ColumnSpan="1"
        Margin="40,10,40,10"
        Content="Salva"
        Command="{Binding SaveDescrizione}"
        FontSize="14"
        x:Name="SaveDescrizioneButton" Click="SaveDescrizioneButton_Click"/>
```

Questa è la view:



#### 3.3.2.6 RequisitoView

RequisitoView è la view che permette di scegliere i sette requisiti specifici per il quaderno dei compiti (A14-A20). È associata a RequisitoViewModel. I requisiti possono essere scelti inserendo il codice univoco. Il titolo del requisito associato viene stampato in un TextBox non editabile (IsEnabled è settato a false). Quest'ultima funzionalità al momento non è attiva, perché non riesco a scrivere nel TextBox direttamente dal Database. In questo caso è più comprensibile partire dall'aspetto grafico della View, che è il segunete:




La prima colonna è composta da 7 TextBox disabilitati che indicano il codice A14-A20 del requisito da inserire. La seconda colonna è composta sempre da 7 TextBox che permettono all'utente di inserire il codice numerico del requisito specifico:

```
<TextBox
    Grid.Row="2"
    Grid.Column="1"
    Grid.ColumnSpan="1"
    Margin="20,8,20,8"
    x:Name="cod14"
    Text="{Binding Path=Cod14, Mode=TwoWay}"
    AutomationProperties.HelpText="Codice">
</TextBox>
```

Nella terza colonna sono presenti ancora 7 TextBox disabilitati che dovrebbero stampare il titolo del requisito, dopo che viene premuto il relativo pulsante "Cerca" nella quarta colonna. Infine è presento il solito bottone per salvare il tutto e proseguire alla View successiva, Ovvero VisualizzaView.

### 3.3.2.7 VisualizzaView

VisualizzaView è la View "finale" che permette di visualizzare il Qdc appena creato oppure di tornare al Menu principale, per creare un nuovo quaderno dei compiti o eventualmente per selezionarne uno e aggiungerci i voti. È associata a VisualizzaViewModel.
Questa View è molto simile a MenuView, infatti contiene solamente due bottoni che permettono di eseguire le operazioni appena citate.

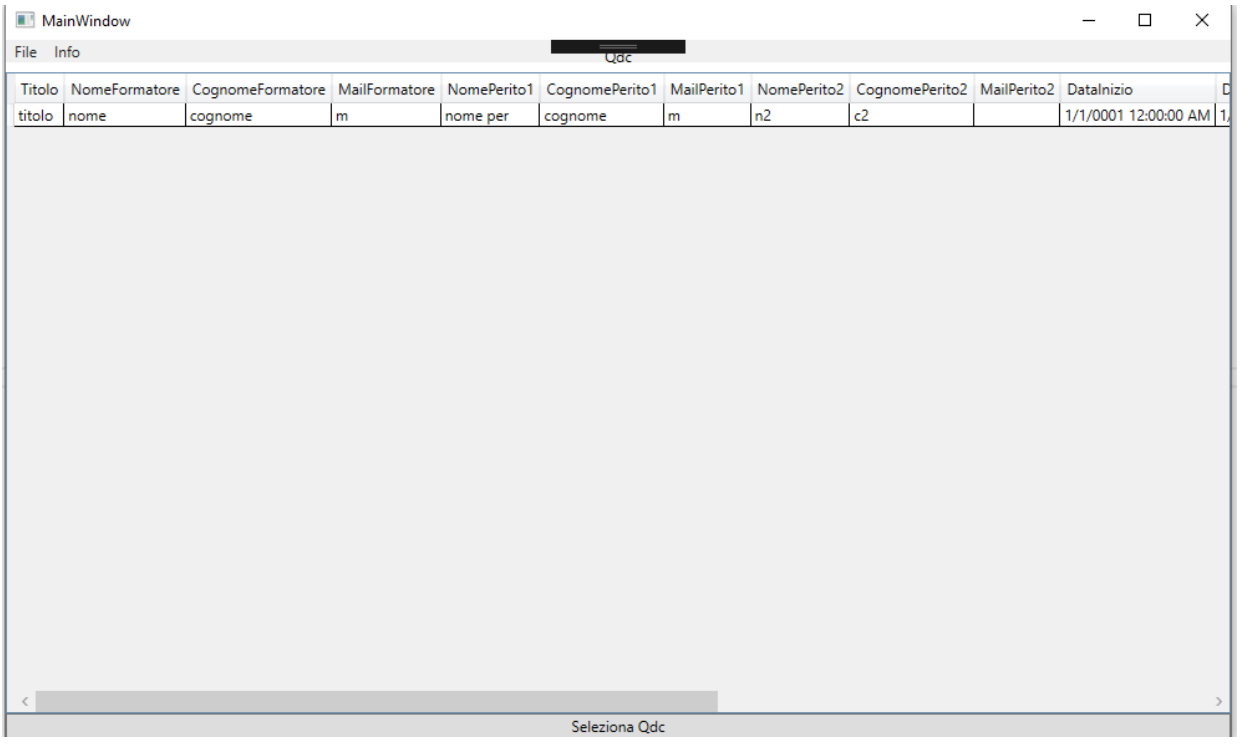
```
<Button
    Grid.Column="0" VerticalAlignment="Center"  HorizontalAlignment="Center"
    Content="Visualizza QDC" Padding="50" FontSize="30"
    Command="{Binding Path=VisualizzaQdc}">
</Button>
<Button
    Grid.Column="1" VerticalAlignment="Center"  HorizontalAlignment="Center"
    Content="Torna al MENU" Padding="50" FontSize="30"
    Command="{Binding Path=MenuCommand}">
</Button>
```

### 3.3.2.8 SelezionaView

SelezionaView è la View che permette di selezionare uno dei Qdc già creati, purtroppo questa funzionalità non è stata implementata. Questa View viene visualizzata se dal Menu viene premuto il pulsante "Seleziona QDC" ed è associata a SelezionaViewModel. Contiene un oggetto DataGrid che mostra tutti i Qdc esistenti e un bottone che porta alla view successiva, ovvero MotivazioneView.

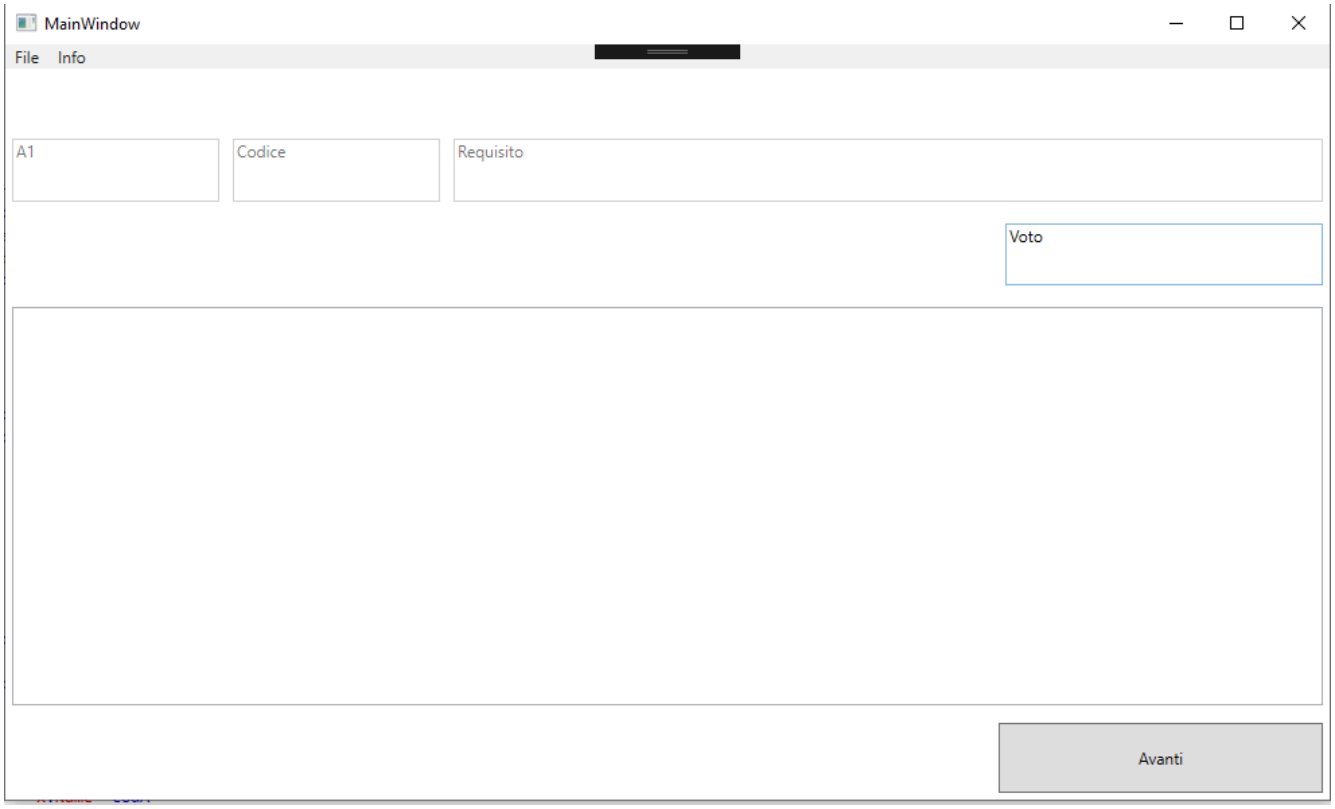
```
<DataGrid x:Name="QdcDataGrid" Grid.Column="0" Grid.Row="1" IsReadOnly="True"
          ItemsSource="{Binding Path=Qdcs}">
</DataGrid>
```

La View si presenta in questo modo:



#### 3.3.2.9 MotivazioneView

MotivazioneView è la view che permette di inserire un voto e una motivazione per ogni requisito (sia per i sette specifici scelti in RequisitoView che gli altri 33 fissi per ogni progetto). La View è formata da 3 TextBox disabilitati che dovrebbero stampare il requisito al quale assegnare il voto e la motivazione. Il voto può essere inserito in un normale TextBox, mentre la motivazione viene inserita in un TextBox che funziona come quello della descrizione.
Questa View al momento non esegue alcuna operazione.
Questo è l'aspetto della View:



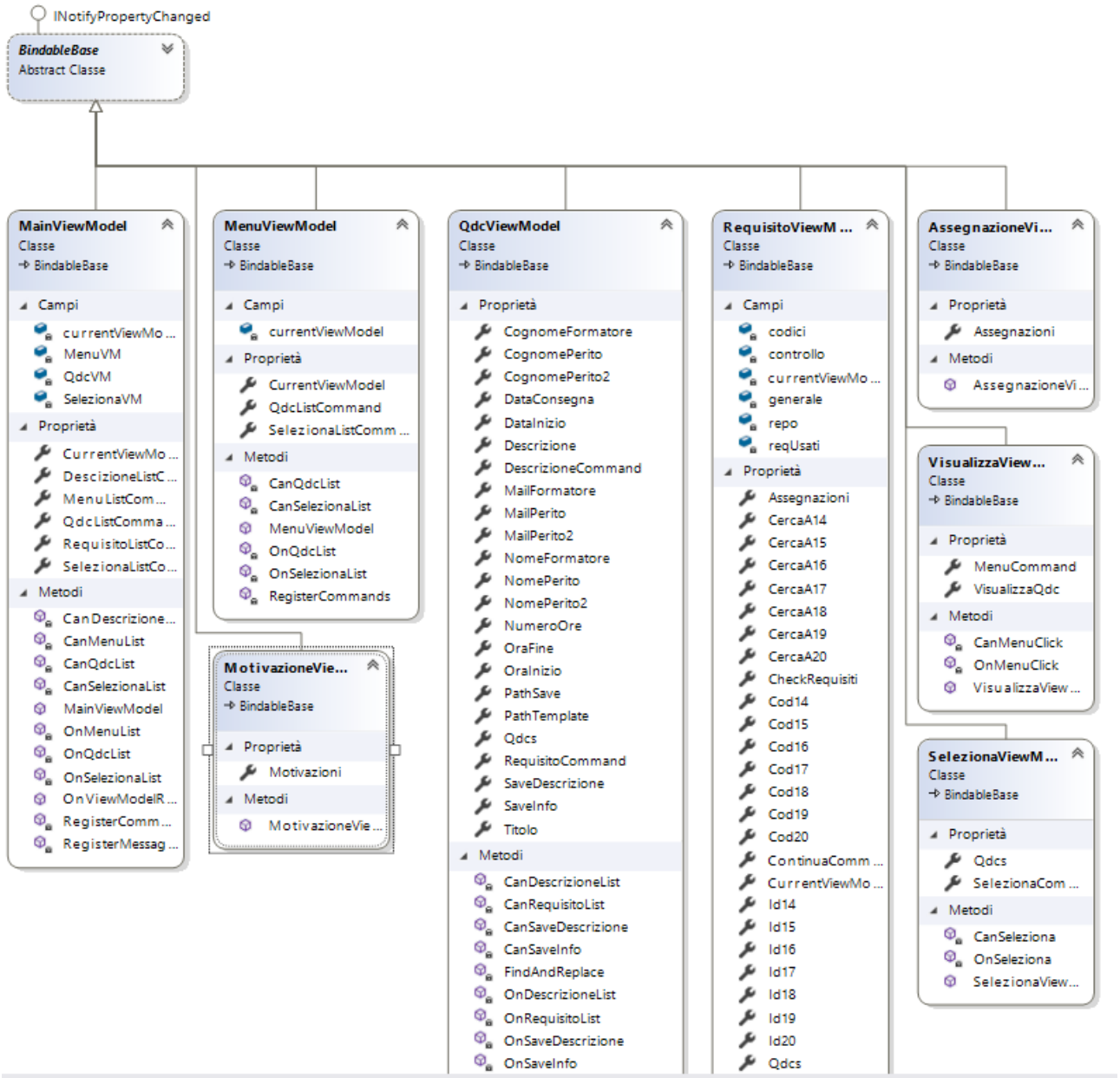

Quando viene premuto il bottone "Avanti" il voto e la motivazione dovrebbero essere aggiunti nella riga del requisito (tabella Assegnazione) e dovrebbe comparire il nuovo requisito in ordine di Id.

#### 3.3.2.10 VisualizzaVotiView

Questa View serve per tornare al menu principale dopo aver aggiunto i voti. È molto simile a VisualizzaView, contiene due bottoni, uno appunto per tornare al menu e l'altro per visualizzare il file dei voti. Inoltre contiene due TextBox per inserire il percorso del file dei voti di Template e quello n cui salvare il nuovo file.

### 3.3.3 ViewModel

La directory ViewModel contiene tutte quelle classe che si occupano di interagire con i dati presenti nel Model della libreria di classi e gestirli all'interno del progetto WPF, rendendoli visibili oppure prendendoli dalle View in modo che possano essere fatte delle operazioni. Nel mio caso praticamente ogni View ha un ViewModel corrispondente.
In fase di progettazione non avevo diviso il diagramma delle classi in due parti. Questa è la parte relativa ai ViewModels:



#### 3.3.3.1 MainViewModel

MainViewModel è la classe principale del ViewModel. Si occupa di gestire tutti quanti i ViewModel. La sua funzione principale è quella di definire quale ViewModel va mostrato in base alla view che si utilizza. Vengono definiti tutti i ViewModel, le istanze di IDelegateCommand e una variabile che si occupa di gestire quale ViewModel mostrare.

```
/// <summary>
/// istanza dei ViewModel
/// </summary>
private QdcViewModel QdcVM;
private SelezionaViewModel SelezionaVM;
private MenuViewModel MenuVM;

/// <summary>
/// istanza di IDelegateCommand per la delega delle operazioni
/// </summary>
public IDelegateCommand QdcListCommand { get; set; }

public IDelegateCommand RequisitoListCommand { get; set; }

public IDelegateCommand DescizioneListCommand { get; set; }

public IDelegateCommand SelezionaListCommand { get; set; }

public IDelegateCommand MenuListCommand { get; set; }

/// <summary>
/// ViewModel da mostrare
/// </summary>
private BindableBase currentViewModel;

public BindableBase CurrentViewModel
{
    get { return currentViewModel; }
    set { SetProperty(ref currentViewModel, value); }
}
```

Nel costruttore viene richiamato il metodo RegisterCommands() che si occupa di delegare la gestione dei due metodi OnExecute e CanExecute (OnList e CanList in questo caso) a DelegateCommand. Inoltre viene settato il ViewModel di defaut e viene richiamato il metodo RegisterMessages()

```
/// <summary>
/// Delega delle operazioni a DelegateCommand
/// </summary>
private void RegisterCommands()
{
    MenuVM = new MenuViewModel();
    QdcVM = new QdcViewModel();
    SelezionaVM = new SelezionaViewModel();
    QdcListCommand = new DelegateCommand(OnQdcList, CanQdcList);
    SelezionaListCommand = new DelegateCommand(OnSelezionaList, CanSelezionaList);
    MenuListCommand = new DelegateCommand(OnMenuList, CanMenuList);
    // la view di partenza è quella del menu.
    currentViewModel = MenuVM;
    RegisterMessages();
}
```

Il metodo RegisterMessages si occupa di gestire i messaggi da parte dei ViewModels, di inviarli al metodo OnViewModelReceived e di impostare il corretto ViewModel.

```
/// <summary>
/// Gestione dei messaggi dei viewmodel
/// </summary>
private void RegisterMessages()
{
    Messenger.Default.Register<BindableBase>(this, OnViewModelReceived);
}

/// <summary>
/// Impostazione del viewmodel in base al messaggio
/// </summary>
/// <param name="viewmodel">ViewModel da impostare.</param>
public void OnViewModelReceived(BindableBase viewmodel)
{
    CurrentViewModel = viewmodel;
}
```

Sono presenti infine i metodi che permettono di assegnare il corretto ViewModel da visualizzare e CanExecute.

```
/// <summary>
/// Attribuzione del ViewModel da mostrare
/// </summary>
/// <param name="obj"></param>
private void OnQdcList(object obj)
{
    CurrentViewModel = QdcVM;
}

private bool CanQdcList(object arg)
{
    return true;
}
```

#### 3.3.3.2 MenuViewModel

MenuViewModel è il ViewModel di default. Infatti quando viene fatta partire l'applicazione è la View associata a questo ViewModel a essere visualizzata. Lo scopo di questo ViewModel è quello di permettere di scegliere di creare un nuovo Qdc (o eventualmente di selezionarne uno), quindi deve permettere di passare direttamente a QdcView (quando viene premuto un pulsante).
Vengono quindi create le variabili associate ai comandi della View:

```
/// <summary>
/// istanza di IDelegateCommand per la delega delle operazioni
/// </summary>
public IDelegateCommand QdcListCommand { get; set; }

public IDelegateCommand SelezionaListCommand { get; set; }
```

Nel costruttore vengono inizializzare queste variabili. Quando uno dei due bottoni viene premuto viene mandato il messaggio a MainView che si occupa di settare il ViewModel corretto.

```
public MenuViewModel()
{
    QdcListCommand = new DelegateCommand(OnQdcList, CanQdcList);
    SelezionaListCommand = new DelegateCommand(OnSelezionaList, CanSelezionaList);
}

private void OnQdcList(object obj)
{
    Messenger.Default.Send<BindableBase>(new QdcViewModel());
}
```

#### 3.3.3.3 QdcViewModel

QdcViewModel è il ViewModel relativo alla classe Qdc. Implementa BindableBase e definisce un insieme (ObservableCollection) di Qdc. Vengono definite anche tutte le variabili IDelegateCommand riferite ai vari pulsanti e alle loro operazioni di QdcView e anche di DescrizioneView (infatti DescrizioneView come detto prima viene gestita da questo ViewModel).

```
/// <summary>
/// Insieme che contiene i campi di un Qdc.
/// </summary>
public ObservableCollection<Qdc> Qdcs { get; set; }

public IDelegateCommand DescrizioneCommand { get; set; }

public IDelegateCommand SaveInfo { get; set; }

public IDelegateCommand RequisitoCommand { get; set; }

public IDelegateCommand SaveDescrizione { get; set; }
```

Oltre a questo vengono anche definite tutte le variabili relative ai vari TextBox delle due Views, che vengono gestiti tramite il Binding, quindi è fontamentale che abbiano lo stesso nome della proprietà Binding nelle Views.

```
public string Titolo { get; set; }
```

```
public string NomeFormatore { get; set; }
```
Nel costruttore vengono instanziate tutte le variabili IDelegateCommand.

```
public QdcViewModel()
{
    DescrizioneCommand = new DelegateCommand(OnDescrizioneList, CanDescrizioneList);
    SaveInfo = new DelegateCommand(OnSaveInfo, CanSaveInfo);
    RequisitoCommand = new DelegateCommand(OnRequisitoList, CanRequisitoList);
    SaveDescrizione = new DelegateCommand(OnSaveDescrizione, CanSaveDescrizione);
}
```

Il metodo OnSaveInfo viene generato quando viene premuto il bottone di salvataggio in QdcView. In questo metodo viene eseguito il salvataggio dei dati sul database e la scrittura sul file Word. Inizialmente viene definito il contesto di dati Qdc:

```
AppDbContext ctx = new AppDbContext();
QdcDbDataRepository repoQdc = new QdcDbDataRepository(ctx);
```
Viene quindi eseguito l'Insert dei dati, tramite il metodo Insert:

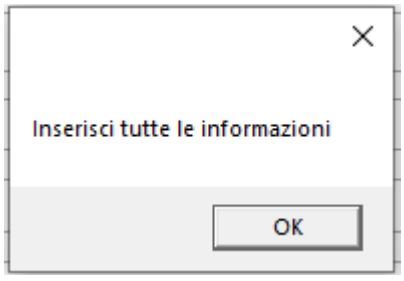
```
repoQdc.Insert(new Qdc(Titolo, NomeFormatore, CognomeFormatore,
    MailFormatore, NomePerito, CognomePerito, MailPerito,
    DataInizio, DataConsegna, OraInizio, OraFine, NumeroOre, Descrizione,
    NomePerito2, CognomePerito2, MailPerito2, PathTemplate, PathSave));
```

Prima di eseguire l'insert vengono effettuati dei controlli sui dati inseriti. L'insert viene eseguito solo se i TextBox non sono vuoti oppure contengono una stringa vuota (ad esempio una serie di spazi). Questo viene eseguito grazie al seguente metodo:

```
if (!string.IsNullOrEmpty(Titolo) && !string.IsNullOrEmpty(NomeFormatore) && . . .)
```

Altrimenti viene stampato un messaggio che indica di inserire tutti i campi:

```
else
{
    MessageBox.Show("Inserisci tutte le informazioni");
}
```




Sono presenti anche degli altri controlli riferiti alla scrittura sul file Word, ma questi li spiego nel capitolo dedicato. Alla fine viene richiamato il metodo per passare a DescrizioneView:

```
OnDescrizioneList(obj);

private void OnDescrizioneList(object obj)
{
    Messenger.Default.Send<BindableBase>(new DescrizioneViewModel());
}
```

Una volta passati a DescrizioneView il metodo di riferimento diventa OnSaveDescrizione. In questo metodo viene nuovamente definito il contesto di dati Qdc. Viene poi eseguita una query per ottenere l'ultimo Qdc inserito (quello appena creato in QdcViewModel, con l'Id più alto):

```
int i = repoQdc.Get().Where(q1 => q1.Id == q1.Id).Max(q1 => q1.Id);
Qdcs = new ObservableCollection<Qdc>(repoQdc.Get().Where(q1 => q1.Id == i));
Qdc q = Qdcs[0];
```

Il primo risultato (che è anche l'unico) trovato viene inserito all'interno di una variabile Qdc, che servirà per eseguire l'Update di quella riga, inserendoci la nuova descrizione:

```
q.Descrizione = Descrizione;
repoQdc.Update(q);
```

Questo viene eseguito solo se il campo inserito non è vuoto:

```
if (!string.IsNullOrEmpty(Descrizione))
```

Alla fine viene richiamato il metodo per passare alla View successiva, ovvero RequisitoView:

```
private void OnRequisitoList(object obj) {
     Messenger.Default.Send<BindableBase>(new RequisitoViewModel());
}
```

#### 3.3.3.4 DescrizioneViewModel

DescrizioneViewModel funziona in modo leggermente diverso, in quanto non esiste una classe Descrizione nella libreria di classi. Descrizione infatti è un campo della classe Qdc, quindi questo ViewModel è vuoto. Le operazioni riferite a DescrizioneView vengono svolte in QcdViewModel.

```
public class DescrizioneViewModel : BindableBase
{
    public DescrizioneViewModel()
    {
    }
}
```

#### 3.3.3.5 RequisitoViewModel

RequisitoViewModel è il ViewModel relativo alla classe Requisito (a alla view RequisitoView). Implementa BindableBase e definisce un insieme (ObservableCollection) di Requisiti. Vengono definite anche tutte le variabili IDelegateCommand ch si riferiscono ai bottoni dela View. Sono presenti sette variabili per i sette bottoni di ricerca del requisito e una per il bottone "Salva e Continua".

```
public class RequisitoViewModel : BindableBase
{
     public ObservableCollection<Requisito> Requisiti { get; set; }
     public IDelegateCommand ContinuaCommand { get; set; }
     public IDelegateCommand CercaA14 { get; set; }
     public IDelegateCommand CercaA15 { get; set; }
     public IDelegateCommand CercaA16 { get; set; }
     public IDelegateCommand CercaA17 { get; set; }
     public IDelegateCommand CercaA18 { get; set; }
     public IDelegateCommand CercaA19 { get; set; }
     public IDelegateCommand CercaA20 { get; set; }
```

sono presenti anche delle variabili di tipo string che contengono il valore dei sette TextBox dei requisiti, tramite il metodo Binding.

```
public string Cod14 { get; set; }
public string Cod15 { get; set; }
public string Cod16 { get; set; }
public string Cod17 { get; set; }
public string Cod18 { get; set; }
public string Cod19 { get; set; }
public string Cod20 { get; set; }
```

Infine ho definito un altra serie di variabili che mi servono per eseguire le prossime operazioni:

- 7 variabili che conterranno gli Id dei requisiti.
- 7 variabili che conterranno il titolo dei requisiti.
- Delle variabili per eseguire i controlli.
- Un array contenente tutti i codici dei requisiti statici (quelli che vanno assegnati a tutti i qdci)

```
string[] codici = { "a1", "a2", "a3", "a4", "a5", "a6", "a7", "a8", . . . ,"c9", "c10"};
```

Il costruttore è simile a quelli dei precedenti ViewModel, infatti ci istanzio le varie variabili che poi userò nei prossimi metodi. La particolarità di questo ViewModel è che ci sono parecchi metodi molto simili tra di loro. Quando per esempio viene premuto il bottone di ricerca per il requisito A14 viene richiesto il metodo OnA14:

```
private void OnA14(object obj)
{
    generale = false;
    CheckRequisiti = new ObservableCollection<Requisito>(repo.Get());
    for(int i = 0; i < CheckRequisiti.Count; i++)
    {
        if(Cod14 == CheckRequisiti[i].CodiceReq)
        {
            try
            {
                Requisiti = new ObservableCollection<Requisito>(repo.Get().Where(r =>
r.CodiceReq == Cod14));
                Requisito r14 = Requisiti[0];
                //Req14 = r14.Titolo;
                Id14 = r14.Id;
```

```
                Titolo14 = r14.Titolo;
                controllo[0] = 1;
                generale = true;
            }
            catch (ArgumentOutOfRangeException e)
            {
                MessageBox.Show("Inserisci un valore valido");
            }
        }
    }

    if (!generale)
    {
        MessageBox.Show("Inserisci un valore valido");
    }
}
```

In questo metodo istanzio una variabile CheckRequisiti che contiene tutti i requisiti esistenti, per controllare che il requisito inserito sia valido. Se il requisito è valido viene cercato all'interno del database e i relativi Id e Titolo vengono assegnati alle rispettive variabili Id14 e Titolo14. Infine viene settato il valore di una variabile di array di controllo a 1. Tutto il codice è inserito in un blocco try-catch che si occupa di gestire gli errori.
Esiste uno di questi metodi per ogni requisito.
Quando viene premuto il pulsante per salvare i dati e continuare viene richiamato il metodo OnVisualizzaClick:

```
QdcDbDataRepository repoQdc = new QdcDbDataRepository(ctx);
int ind = repoQdc.Get().Where(q1 => q1.Id == q1.Id).Max(q1 => q1.Id);
Qdcs = new ObservableCollection<Qdc>(repoQdc.Get().Where(q1 => q1.Id == ind));
Qdc q = Qdcs[0];

for (int i = 0; i < codici.Length; i++)
{
    repoAss.Insert(new Assegnazione(codici[i], q, i + 1));
}

repoAss.Insert(new Assegnazione(Cod14, q, Id14));
repoAss.Insert(new Assegnazione(Cod15, q, Id15));
. . .
repoAss.Insert(new Assegnazione(Cod20, q, Id20));
```

Questo metodo si occupa di salvare i requisiti scelti nella tabella Assegnazione. Viene trovato l'ultimo Qdc nel database (quello appena creato) e vengono fatti sette insert, uno per ognuno dei requisiti scelti dall'utente.
Vengono eseguiti anche gli insert dei requisiti standard, in un ciclo for lungo quanto l'array codice che contiene appunto staticamente i codici dei requisiti da inserire.
Gli insert vengono eseguiti secondo i parametri definiti nel costruttore di Assegnazione, quindi il codice numerico del requisito, l'id del qdc e l'id del requisito.
Tutti gli insert vengono eseguiti esclusivamente se sono stati scelti tutti e sette i requisiti. Questo avviene controllando se l'array di controllo è settato:

```
if(controllo[0] == 1 && controllo[1] == 1 && . . . controllo[6] == 1)
```

Se qualche requisito non è stato scelto viene stampato un messaggio di notifica:

```
MessageBox.Show("Inserisci tutti i requisiti");
```

Sono presenti poi i metodi per scrivere sul file Word, che vengono spiegati nel capitolo dedicato.

#### 3.3.3.6 VisualizzaViewModel

VisualizzaViewModel è il ViewModel associato a VisualizzaView. Al suo interno ci sono solo due metodi, corrispondenti ai due bottoni della View. Il primo serve per tornare al Menu principale:

```
private void OnMenuClick(object obj)
{
    Messenger.Default.Send<BindableBase>(new MenuViewModel());
}
```

Il funzionamento è come quello di tutti gli altri bottoni per la navigazione tra le Views.
L'altro metodo serve per aprire il file Word appena creato:

```
private void OnVisualizzaClick(object obj)
{
    int ind = repoQdc.Get().Where(q1 => q1.Id == q1.Id).Max(q1 => q1.Id);
    Qdcs = new ObservableCollection<Qdc>(repoQdc.Get().Where(q1 => q1.Id == ind));
    Qdc q = Qdcs[0];
    try
    {
        Application application = new Application();
        application.Visible = true;
        Document file = application.Documents.Open(q.PathSave);
    }
    catch (System.Runtime.InteropServices.COMException e)
    {
        System.Windows.MessageBox.Show("Errore con l'apertura del file");
    }
}
```

Come già visto viene ricavato l'ultimo qdc esistente, che contiene la Path del file Word relativo. Per aprire un file Word utilizzo le classi Application e Document. Il metodo **Documents.Open(file da aprire)** apre il file passato come parametro (in questo caso il percorso impostato dall'utente in QdcView).
Per poter utilizzare queste classi bisogna aggiungere la segunte referenza:

```
using Microsoft.Office.Interop.Word;
```

#### 3.3.3.7 AssegnazioneViewModel

Come per la descrizione questa classe non viene usata, perché non esiste nessuna View corrispondente e la scrittura di questa tabella avviene in RequisitoView. Per questo contiene un semplice costruttore vuoto:

```
public AssegnazioneViewModel()
{

}
```

#### 3.3.3.8 MotivazioneViewModel

MotivazioneViewModel è il ViewModel di MotivazioneView. Quindi questo ViewModel è relativo alla classe Assegnazione. Le operazioni che dovrebbe svolgere questa classe non sono implementate, quindi purtroppo al momento non ha alcuna funzione.
Nel costruttore ricavo l'Id dell'ultimo qdc e lo utilizzo per trovare tutti i requisiti associati a quell'Id nella tabella assegnazione:

```
try
 {
     int ind = repoQdc.Get().Where(q1 => q1.Id == q1.Id).Max(q1 => q1.Id);
     Motivazioni = new ObservableCollection<Assegnazione>(repoAss.Get().Where(a1 =>
a1.QdcId == ind));
     Ass = Motivazioni[0];
 }
 catch(Exception e)
 {
     if(e is ArgumentOutOfRangeException || e is InvalidOperationException)
         MessageBox.Show("Non esistono requisiti assegnati a questo Qdc");
 }
```

#### 3.3.3.9 VisualizzaVotiViewModel

### 3.4 Implementazione Office

Una parte molto importante del mio progetto consiste nell'implementare un modo per interfacciare l'applicazione a Microsoft Office Word. Per fare questo è obbligatoria l'installazione del Kit per lavorare con Office: Sviluppo per Office/SharePoint. Si può installare questo kit durante l'installazione di Visual Studio oppure successivamente aprendo l'installer. Dopo averlo installato ho cominciato a cercare una soluzione fattibile. Ho prima di tutto aggiunto i seguenti riferimenti al progetto:

- Microsoft Office 16.0 Object Library (versione 2.8)
- InteropExtension 1.0 Type Library (versione 1.0)
- Microsoft Word 16.0 Object Library (versione 8.7)

Ho dovuto anche aggiungere alcuni using nei file che usano queste librerie:

```
using System.IO;
using System.Reflection;
using Word = Microsoft.Office.Interop.Word;
```

#### 3.4.1 File di template

Ho creato un file di template per i quaderni dei compiti. Il file è vuoto e contiene solo le tabelle e i vari capitoli e dei pattern che indicano i campi da inserire. I pattern <Testo da modificare> sono appunto le parti del file che vanno modificate in base alle informazioni inserite nella varie View.

| | | |
|---|---|---|
| Superiore professionale | Nome: <NomeFormatore> | Cognome: <CognomeFormatore> |
| | <EmailFormatore> | |
| Perito 1 | Nome: <NomePerito> | Cognome: <CognomePerito> |
| | <EmailPerito> | |
| Perito 2 | Nome: | Cognome: |
| | | |
| Periodo | <DataInizio> , <DataConsegna> | |

Ho creato un file come questo anche per i voti di ogni requisito che funziona nello stesso modo.

#### 3.4.2 Salvataggio sul file

Il sistema che ho deciso di utilizzare funziona nel seguente modo:

- Apertura del file di template
- Copia del file
- Ricerca dei pattern da modificare
- Modifica dei pattern in base ai valori inseriti nelle Views
- Salvataggio del file
- Chiusura del file e del processo

Ho cominciato a inserire queste funzionalità in QdcView. Nel metodo OnClick del pulsante Salva vengono istanziate delle variabili che contengono.i pattern da modificare:

```
var titoloKey = "<Titolo>";
var nomeFormatoreKey = "<NomeFormatore>";
var cognomeFormatoreKey = "<CognomeFormatore>";
var nomePeritoKey = "<NomePerito>";
```

Il file viene copiato (la path del file e la path del file da copiare vengono definite nella View dall'utente) e viene creato l'oggetto che rappresenta il file word in C# (tramite le classi Application e Document).

```
// copia del file
File.Copy(path, pathCopia, true);
// valori predefiniti per l'apertura del file
object missing = Missing.Value;
// crea l'oggetto che contiene l'istanza di Word
Word.Application wordApp = new Word.Application();
//  crea l'oggetto che contiene il documento
Word.Document aDoc = null;
// oggetto che definisce il file copiato (e da modificare)
object filename = pathCopia;
```

Se il file esiste viene aperto. Tramite la funzione FindAndReplace vengono cercati i pattern da modificare e vengono modificati con il contenuto dei TextBox. Infine il file viene salvato e chiuso.

```
if (File.Exists((string)filename))
{
    object readOnly = false;
    object isVisible = false;
    wordApp.Visible = false;
    //  apertura del file copiato
    aDoc = wordApp.Documents.Open(ref filename, ref missing,
        ref readOnly, ref missing, ref missing, ref missing,
        ref missing, ref missing, ref missing, ref missing,
        ref missing, ref isVisible, ref missing, ref missing,
        ref missing, ref missing);
    aDoc.Activate();
    // richiama la funzione FindAndReplace passandogli due parametri,
    // il testo da sostituire e con cosa sostituirlo
    this.FindAndReplace(wordApp, titoloKey, TitoloTextBox.Text);
    this.FindAndReplace(wordApp, nomeFormatoreKey, NomeFormatoreTextBox.Text);
    //  salva il file
    aDoc.Save();
    // chiude il processo
    wordApp.Quit();
}
```

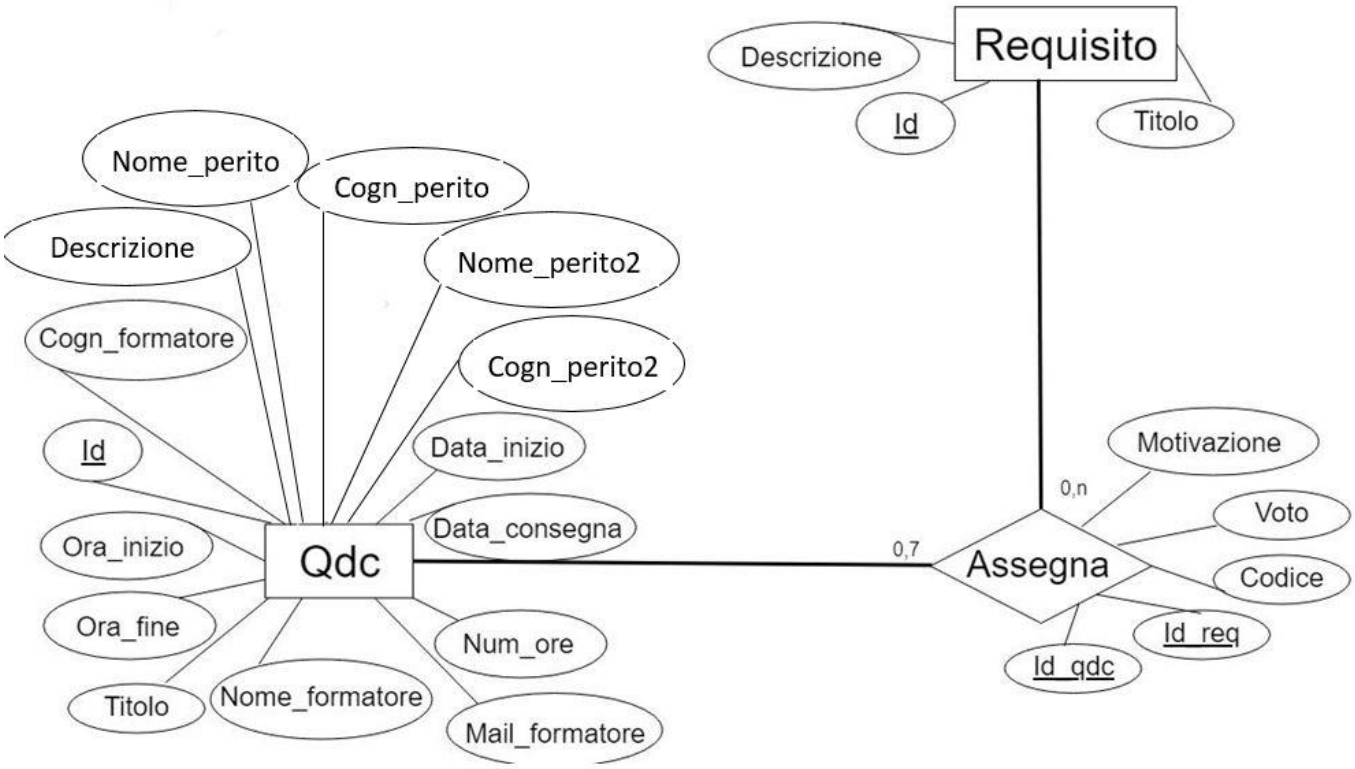
Se il file non esiste viene mostrato un messaggio di errore.
Nel metodo FindAndReplace viene eseguita la ricerca e la sostituzione dei pattern con il metodo Selection.Find.Execute:

```
wordApp.Selection.Find.Execute(ref findText, ref matchCase,
    ref matchWholeWord, ref matchWildCards, ref matchSoundsLike,
    ref matchAllWordForms, ref forward, ref wrap, ref format,
    ref replaceText, ref replace, ref matchKashida,
            ref matchDiacritics,
    ref matchAlefHamza, ref matchControl);
```

### 3.5 Situazione finale

In breve…

Lo schema ER è leggermente diverso da quello sviluppato in fase di progettazione:




Ho eliminato la tabella descrizione, aggiungedola come un campo in Qdc. Inoltre ho aggiunto altri campi come quelli per i periti.

# 4 Test

## 4.1 Protocollo di test

Definire in modo accurato tutti i test che devono essere realizzati per garantire l'adempimento delle richieste formulate nei requisiti. I test fungono da garanzia di qualità del prodotto. Ogni test deve essere ripetibile alle stesse condizioni.

| **Test Case:** | TC-001 | **Nome:** | Funzionamento ambiente di lavoro |
|---|---|---|---|
| **Riferimento**: | RQ_001 / RQ_010 | | |
| **Descrizione:** | | | |
| **Prerequisiti:** | Visual Studio installato con tutti i plugins necessari | | |
| **Procedura:** | 1. Aprire Visual Studio e creare un nuovo progetto<br>2. Eseguire il programma MenuSAMT.vsix<br>3. Verificare il funzionamento del modello MVVM<br>4. Provare a salvare un programma<br>5. Chiudere Visual Studio, riaprirlo e verificare che il modello MVVM funzioni ancora | | |
| **Risultati attesi:** | Dopo aver fatto partire il programma MenuSAMT il template MVVM viene creato. Se si chiude VS e si riapre in seguito il template rimane funzionante. | | |

| **Test Case:** | TC-002 | **Nome:** | Funzionamento database (SqLite) |
|---|---|---|---|
| **Riferimento**: | RQ_009 | | |
| **Descrizione:** | | | |
| **Prerequisiti:** | Database creato correttamente e accessibile | | |
| **Procedura:** | 1. Creare le tabelle del database partendo dallo schema ER<br>2. Inserire manualmente qualche dato per il test<br>3. Provare a eseguire qualche semplice query di select, insert e update per verificare l'effettivo funzionamento del database | | |
| **Risultati attesi:** | Gli insert e gli update e tutti gli altri comandi principali devono funzionare correttamente. Il database si aggiorna ed è accessibile dal codice. | | |

| **Test Case:** | TC-003 | **Nome:** | Inserimento dei dati generici |
|---|---|---|---|
| **Riferimento**: | RQ_002 | | |
| **Descrizione:** | | | |
| **Prerequisiti:** | View per l'inserimento dei dati generici già creata. Interfaccia con il database funzionante (TS_002) | | |
| **Procedura:** | 1. Inserire più volte dei dati nei vari campi della view per i dati generici<br>2. Controllare che tali dati vengano salvati sul database | | |
| **Risultati attesi:** | Quando vengono inseriti dei dati nei vari campi, dopo che viene premuto il pulsante "Salva e continua", questi vengono salvati nel database nelle giuste colonne. | | |

| **Test Case:** | TC-004 | **Nome:** | Inserimento di descrizioni |
|---|---|---|---|
| **Riferimento**: | RQ_003 | | |
| **Descrizione:** | | | |
| **Prerequisiti:** | View per l'inserimento delle descrizioni già creata. Interfaccia con il database funzionante (TS_002) | | |
| **Procedura:** | 1. Inserire più volte dei dati nei vari campi della view per le descrizioni<br>2. Controllare che tali dati vengano salvati sul database | | |
| **Risultati attesi:** | Quando viene inserita una descrizione dopo che viene premuto il pulsante "Salva e continua", questi vengono salvati nel database nella colonna giusta. | | |

| **Test Case:** | TC-005 | **Nome:** | Scelta dei requisiti |
|---|---|---|---|
| **Riferimento**: | RQ_004 / RQ_005 | | |
| **Descrizione:** | | | |
| **Prerequisiti:** | View per la scelta dei requisiti già creata. Interfaccia con il database funzionante (TS_002). | | |
| **Procedura:** | 1. Provare a scegliere sette requisiti dalla select<br>2. Salvare il qdc e verificare che i requisiti vengano salvati correttamente sul database, compresa l'assegnazione (tabella assegna).<br>3. Provare a scegliere i sette requisiti inserendo solo l'id<br>4. Verificare che venga scelto il requisito corrispettivo all'id partendo dal file word contenente tutti i requisiti per esteso. | | |
| **Risultati attesi:** | Gli insert e gli update e tutti gli altri comandi principali devono funzionare correttamente. Il database si aggiorna ed è accessibile dal codice. | | |

| **Test Case:** | TC-006 | **Nome:** | Assegnazione di punteggi |
|---|---|---|---|
| **Riferimento**: | RQ_006 | | |
| **Descrizione:** | | | |
| **Prerequisiti:** | View per assegnazione dei punteggi già creata. Interfaccia con il database funzionante (TS_002) | | |
| **Procedura:** | 1. Inserire più volte dei punteggi per ogni requisito A14-A20<br>2. Controllare che i punteggi vengano scritti correttamente sul database | | |
| **Risultati attesi:** | Quando vengono inseriti dei punteggi, dopo che viene premuto il pulsante "Salva e continua", questi vengono salvati nel database nelle giuste colonne. | | |

| **Test Case:** | TC-007 | **Nome:** | Copia-incolla motivazioni) |
|---|---|---|---|
| **Riferimento**: | RQ_007 | | |
| **Descrizione:** | | | |
| **Prerequisiti:** | View per assegnazione dei punteggi già creata. Interfaccia con il database funzionante (TS_002) | | |
| **Procedura:** | 1. Dopo aver inserito dei punteggi, provare a cliccare su "copia" e verificare | | |

| | |
|---|---|
| | che vengano copiate le motivazioni direttamente dal file word con i requisiti estesi |
| **Risultati attesi:** | Quando viene premuto “copia”, vengono copiate le motivazioni per quel requisito direttamente dal file apposito. |

| | | | |
|---|---|---|---|
| **Test Case:**<br>**Riferimento**: | TC-008<br>- | **Nome:** | Controllo dati |
| **Descrizione:** | | | |
| **Prerequisiti:** | Varie view già create e funzionanti. | | |
| **Procedura:** | 1. Inserire in tutte le view create dei dati validi<br>2. Controllare che questi dati vengono salvati correttamente sul database<br>3. Inserire nelle varie view qualche dato non valido<br>4. Controllare che nel database non venga scritto nulla e che venga richiesto di inseire dei dati validi | | |
| **Risultati attesi:** | Se vengono inseriti dei dati validi, questi vengono salvati sul database. Se in una view viene inserito un dato non valido, nessun dato di quella view viene salvato sul database e viene richiesto di inseire nuovamente i campi contenenti i dati non validi. | | |

### 4.2 Risultati test

Tabella riassuntiva in cui si inseriscono i test riusciti e non del prodotto finale. Se un test non riesce e viene corretto l’errore, questo dovrà risultare nel documento finale come riuscito (la procedura della correzione apparirà nel diario), altrimenti dovrà essere descritto l’errore con eventuali ipotesi di correzione.

### 4.3 Mancanze/limitazioni conosciute

Descrizione con motivazione di eventuali elementi mancanti o non completamente implementati, al di fuori dei test case. Non devono essere riportati gli errori e i problemi riscontrati e poi risolti durante il progetto.

## 5 Consuntivo

Consuntivo del tempo di lavoro effettivo e considerazioni riguardo le differenze rispetto alla pianificazione (cap 1.7) (ad esempio Gannt consuntivo).

## 6 Conclusioni

Quali sono le implicazioni della mia soluzione? Che impatto avrà? Cambierà il mondo? È un successo importante? È solo un’aggiunta marginale o è semplicemente servita per scoprire che questo percorso è stato una perdita di tempo? I risultati ottenuti sono generali, facilmente generalizzabili o sono specifici di un caso particolare? ecc

### 6.1 Sviluppi futuri

Migliorie o estensioni che possono essere sviluppate sul prodotto.

### 6.2 Considerazioni personali

Cosa ho imparato in questo progetto? ecc

# 7 Bibliografia

## 7.1 Sitografia


- https://app.moqups.com/unsaved/11c137b2/edit/page/ad64222d5
- https://creately.com/app/
- https://www.c-sharpcorner.com/Blogs/crud-operation-in-c-sharp-application
- https://docs.microsoft.com/en-us/visualstudio/data-tools/create-a-simple-data-application-with-wpf-and-entity-framework-6?view=vs-2019
- https://stackoverflow.com/questions/17012839/,
- https://stackoverflow.com/questions/33241443/enable-migrations-object-reference-not-set-to-an-instance-of-an-object
- https://stackoverflow.com/questions/13031965/no-context-type-was-found-in-the-assembly
- https://docs.microsoft.com/it-it/dotnet/api/system.notimplementedexception?view=netframework-4.8
https://stackoverflow.com/questions/7660547/how-to-create-bindable-commands-in-custom-control
- https://blogs.windows.com/windowsdeveloper/2017/02/06/using-sqlite-databases-uwp-apps/
- https://www.c-sharpcorner.com/article/reading-a-word-document-using-C-Sharp/
- https://stackoverflow.com/questions/4744293/unable-to-load-dll-sqlite3-the-specified-module-could-not-be-found-exceptio
- https://stackoverflow.com/questions/11011759/how-to-read-ms-word-paragraph-and-table-content-line-by-line
- https://answers.microsoft.com/en-us/office/forum/office_2007-word/make-text-go-to-next-row-in-word-tables/9b6a4b5b-bd6c-445e-9465-02f2530bb45f
- https://docs.microsoft.com/en-us/dotnet/api/microsoft.office.interop.word.range.moveenduntil?view=word-pia
- https://docs.microsoft.com/en-us/dotnet/api/microsoft.office.interop.word.find.execute?view=word-pia
- https://stackoverflow.com/questions/1478527/using-vba-for-word-how-do-i-create-a-range-of-table-cells
- http://vba.relief.jp/word-vba-select-first-column-selected-table/
- https://docs.microsoft.com/en-us/office/vba/api/Word.Column
- http://dotnetpattern.com/mvvm-light-messenger
- https://stackoverflow.com/questions/8224700/how-can-i-check-a-c-sharp-variable-is-an-empty-string-or-null


# 8 Glossario

QDC ..... Quaderno dei compiti, definizione delle informazioni e dei requisiti di un progetto
FK ..... Foreign Key, Chiave esterna di una tabella
PK ..... Primary Key, Chiave primaria di una tabella
MVVM ..... Model,View,ViewModel, template usato per l'implementazione del codice
Template……...Struttura generica o standard
Model……Fornisce il collegamento diretto al database e i metodi per interagirci
View……...Permette all'utente di interagire con i dati del Model
ViewModel……Interfaccia la View al Model
WPF…….....Windows Presentation Foundation, libreria di classi per sviluppo di interfacce
…….grafiche.
GUI……..Graphical user interface, interfaccia grafica
.NET……Piattaforma di sviluppo con moltissime funzionalità
Tool…….Software che offre delle funzionalità utili.
Repository……..Contenitore di dati (che permette anche la manipolazione)

# 9 Allegati

Elenco degli allegati, esempio:

- Diari di lavoro
- Codici sorgente/documentazione macchine virtuali
- Istruzioni di installazione del prodotto (con credenziali di accesso) e/o di eventuali prodotti terzi
- Documentazione di prodotti di terzi
- Eventuali guide utente / Manuali di utilizzo
- Mandato e/o Qdc
- Prodotto
- …