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 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).

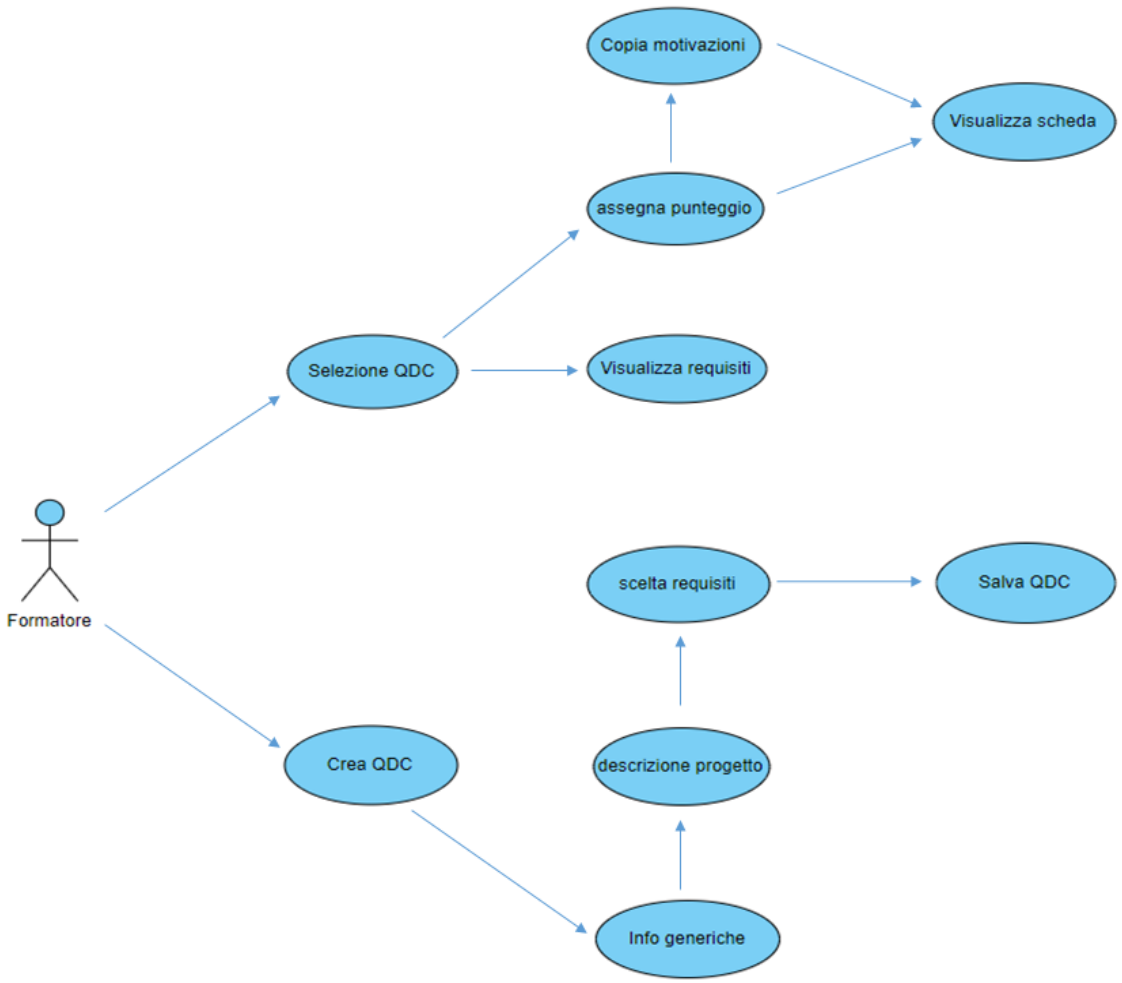
ID: REQ-009	
Nome	Sviluppo della banca dati
Priorità	1
Versione	1.0
Note	-

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 MVC
Priorità	1
Versione	1.0
Note	-
Sotto requisiti	
001	Sviluppare l'intero applicativo usando il modello MVC.

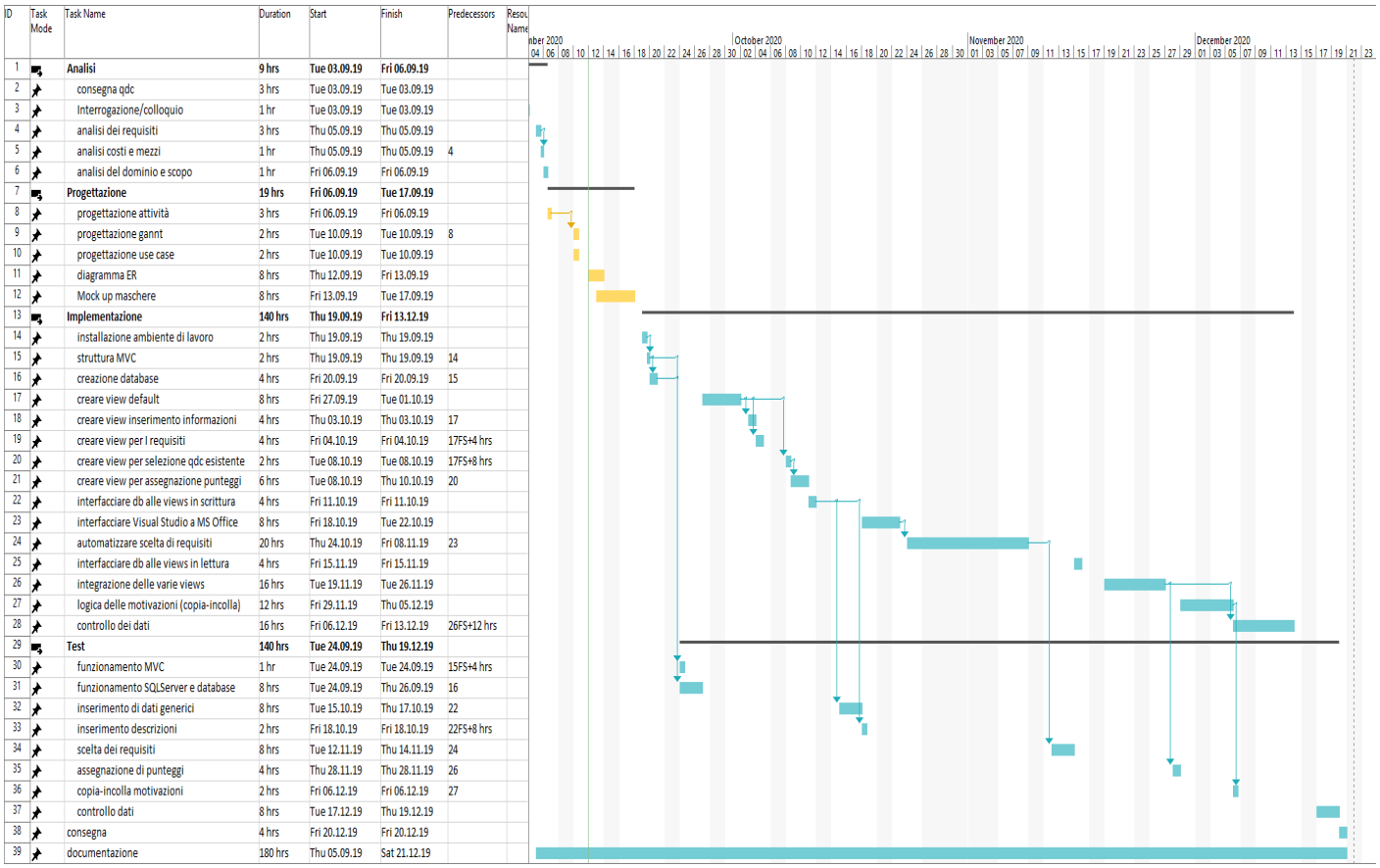
1.6 Use case

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	Resource Name
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

1.8.3 Analisi dei costi

Considero come costo per persona 60 franchi all'ora.

Costo per ora	Ore	Impiegati	Totale
60 CHF	174	1	10440 CHF

2 Progettazione

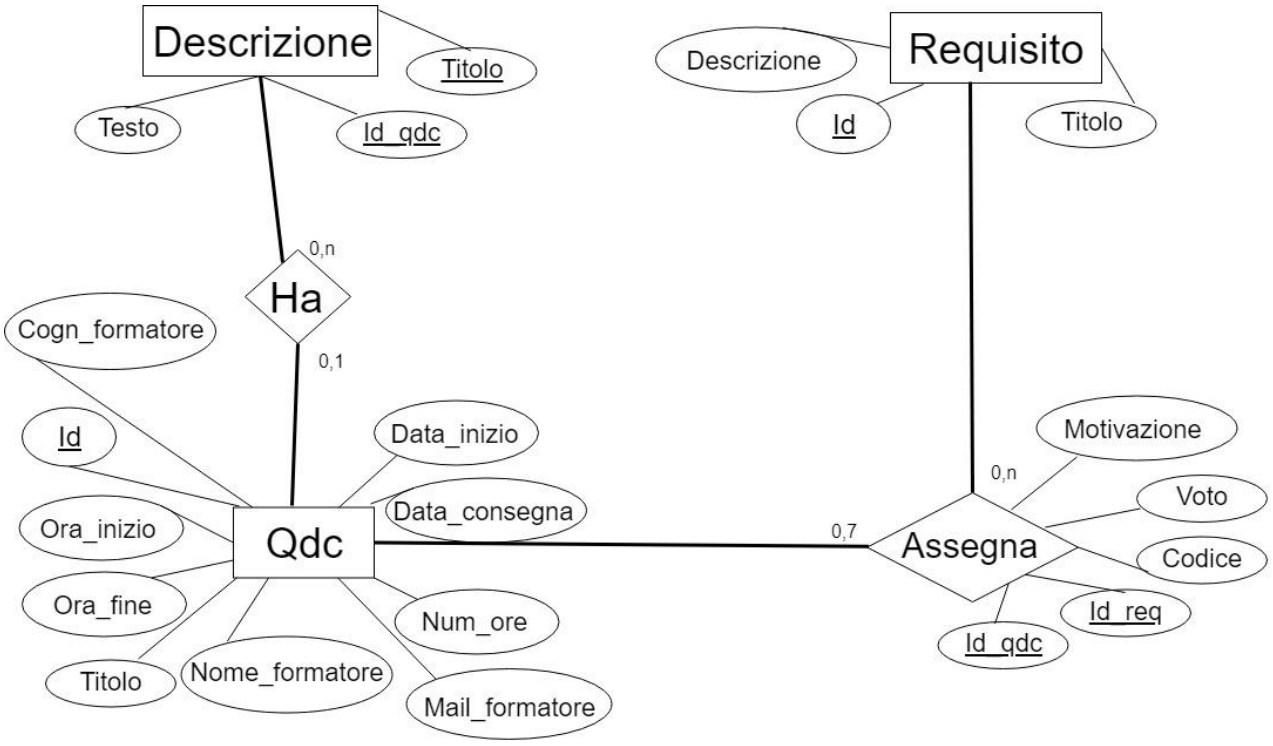
Questo capitolo descrive esaustivamente come deve essere realizzato il prodotto fin nei suoi dettagli. Una buona progettazione permette all'esecutore di evitare fraintendimenti e imprecisioni nell'implementazione del prodotto.

2.1 Design dell'architettura del sistema

Descrive:

- La struttura del programma/sistema lo schema di rete...
- Gli oggetti/moduli/componenti che lo compongono.
- I flussi di informazione in ingresso ed in uscita e le relative elaborazioni. Può utilizzare *diagrammi di flusso dei dati* (DFD).
- Eventuale sitemap

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
Descrizione	varchar

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. Infine è presente la descrizione del progetto, che è composta da un testo molto lungo ma non necessita più di un campo quindi non serve una tabella dedicata.

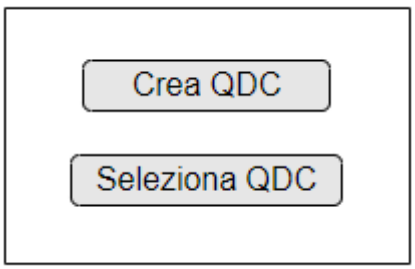
Requisito	
Id	PK integer
Titolo	PK Varchar
Descrizione	Varchar

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

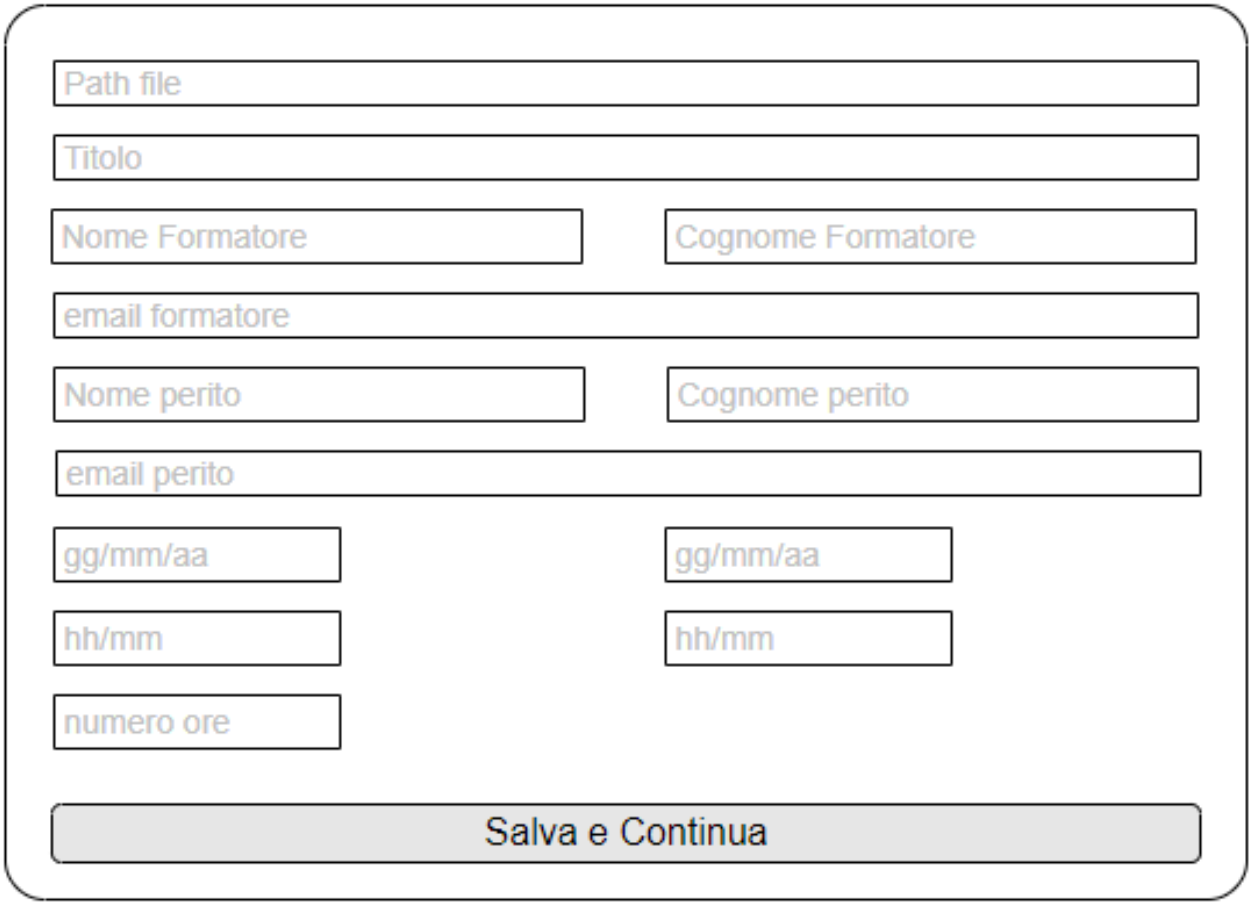
La tabella fondamentale per il progetto è la tabella Assegna, creata dalla relazione molti a molti tra un qdc e un requisito (infatti un qdc contiene 7 requisiti a scelta e 23 requisiti standard che sono uguali per ogni qdc che viene creato; un requisito può essere associato a diversi qdc). In questa 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

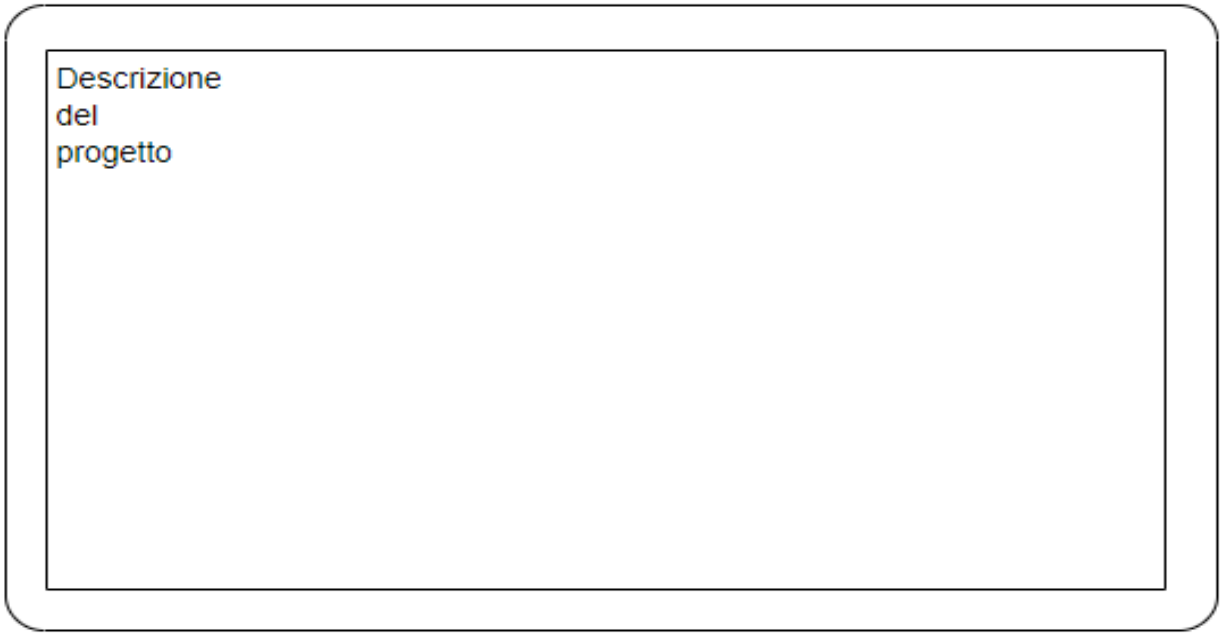


Tutte le maschere che compongono l'applicativo sono sviluppate con l'UI WPF (C#).
La maschera di deafult (compare quando si fa partire l'applicazione) permette di scegliere quale operazione eseguire. A Dipendenza di quale tasto viene premuto si aprono altre maschere che permettono di svolgere il lavoro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

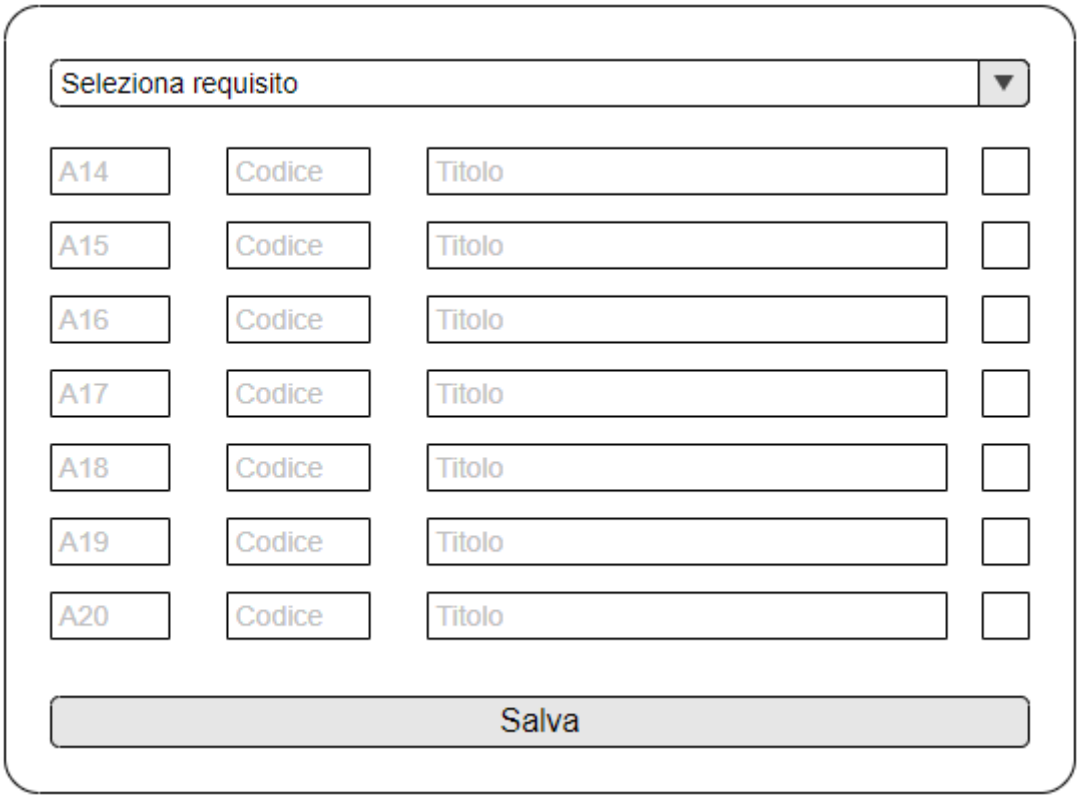
La maschera di default permette di decidere se creare un nuovo quaderno dei compiti oppure di selezionarne uno esistente per assegnare i voti.



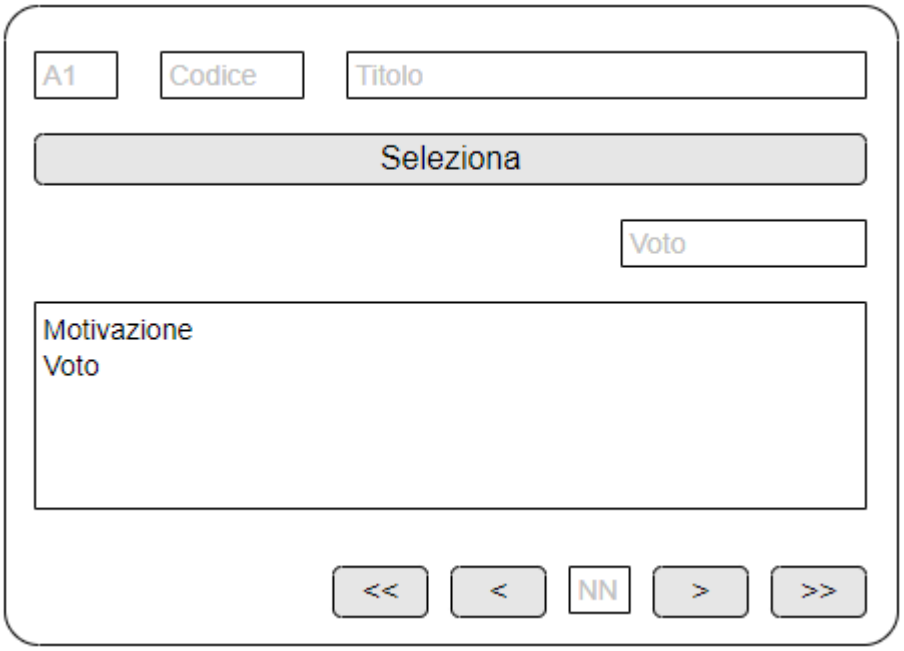
Se viene premuto su Crea QDC nella maschera di default appare questa maschera che serve per inserire le informazioni principali su quaderno dei compiti da creare. Le informazioni che si possono inserire sono il percorso dal quale prendere il file contenente i requisiti estesi, le informazioni sul formatore, 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.

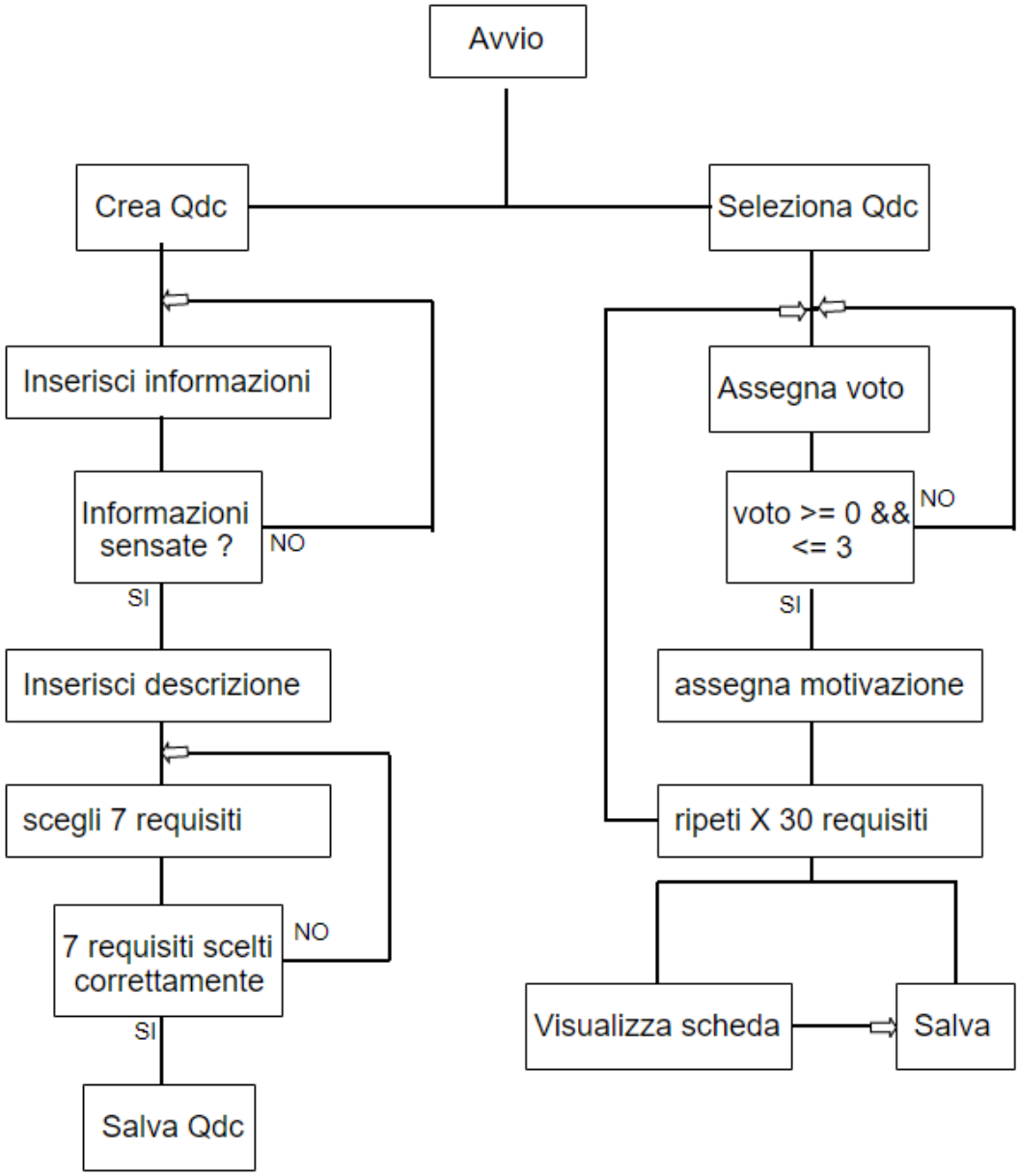


Infine bisogna scegliere i requisiti da assegnare al progetto (7 requisiti da A14 a A20). Per scegliere i requisiti esistono due possibilità. La prima è quella di selezionarli tramite la combo box scegliendo tra tutti i requisiti esistenti. Il requisito scelto viene assegnato al primo campo libero (per 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.

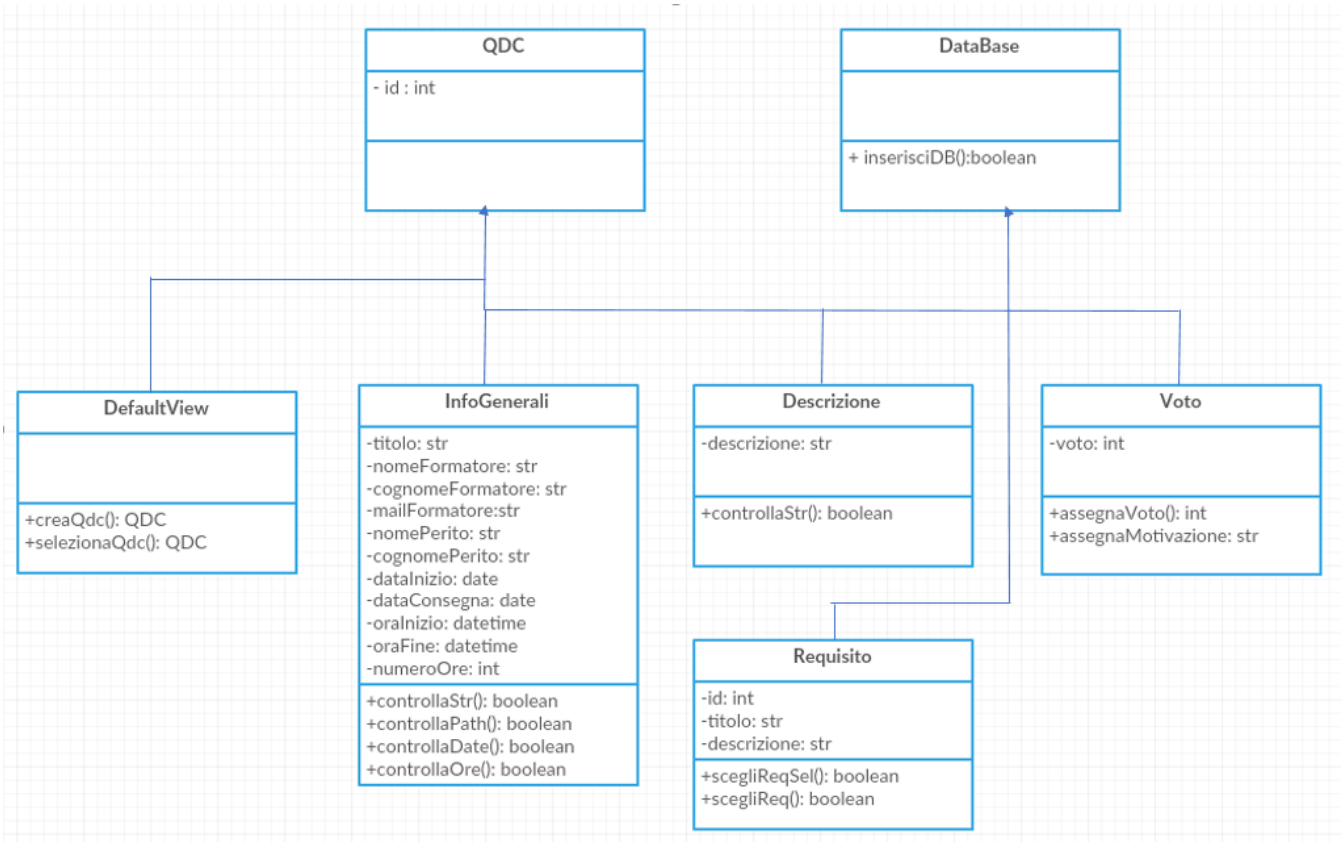


Se viene premuto Seleziona QDC nella maschera iniziale, dopo aver scelto il qdc, appare questa maschera. Questa è la maschera che permette di asseganare i punteggi e le motivazioni per ognuno dei 30 requisiti. Nella prima riga viene mostrato il requisito attuale. Se viene premuto il pulsante seleziona 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.

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. La classe contiene quelli che saranno poi i campi 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.

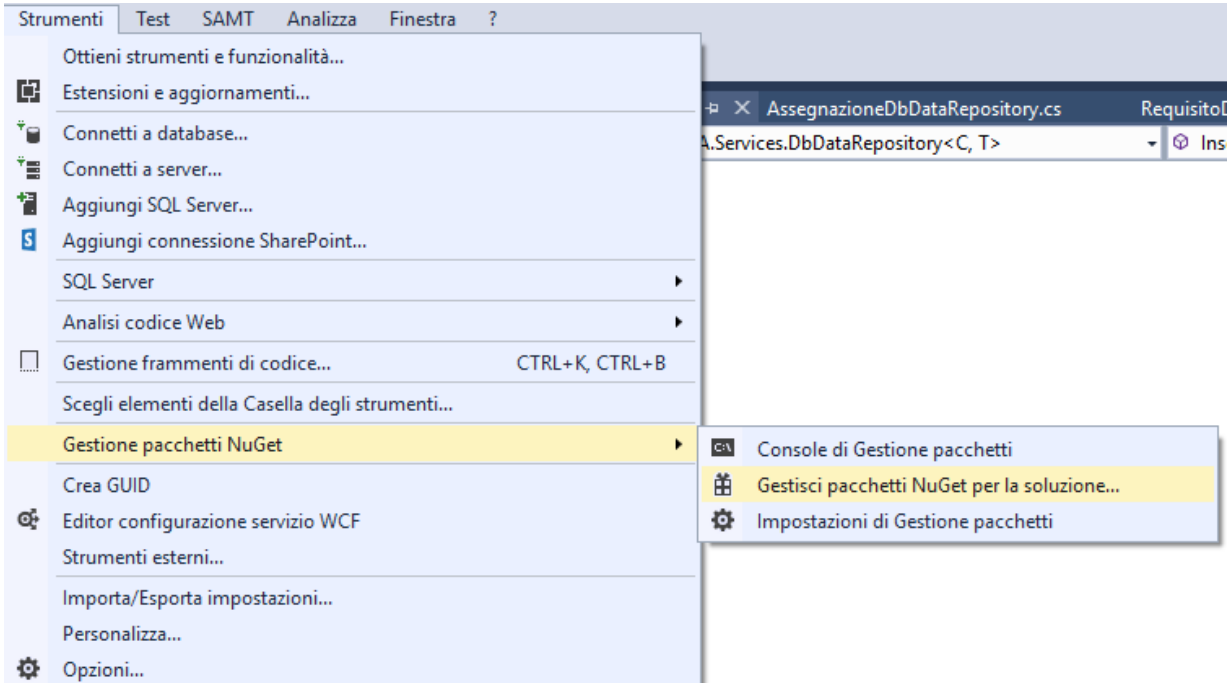
3.1.2 Pacchetti NuGet

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:

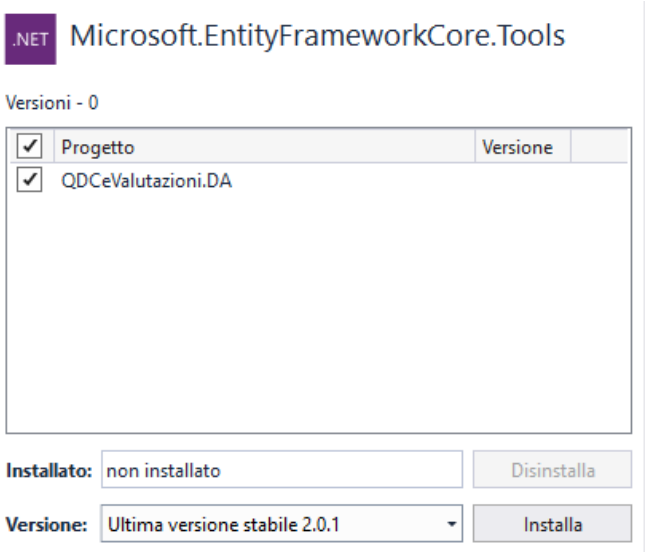
- EntityFramework
- EntityFrameworkCore
- EntityFrameworkCore.Sqlite
- EntityFrameworkCore.Tools

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.

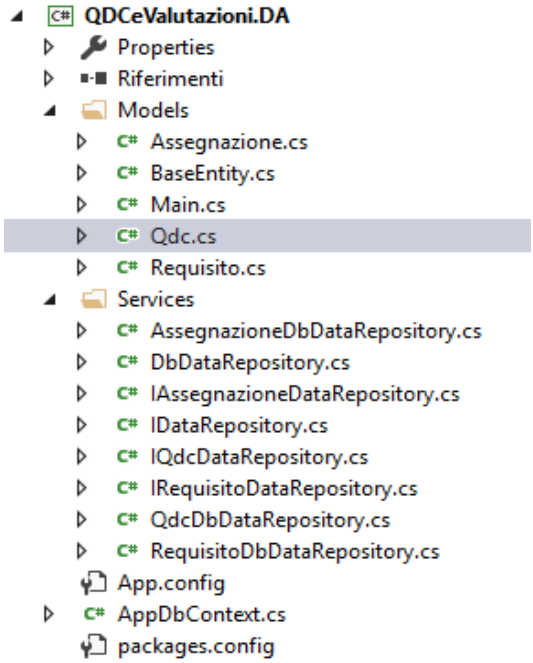


A questo punto si apre un menu per la scelta dei pacchetti da installare. È possibile anche eseguire una ricerca per trovare il pacchetto giusto.Dopo aver scelto il pacchetto desiderato ho premuto semplicemente Installa.



3.2 Libreria di classi:

Dopo che l'ambiente di sviluppo è pronto e la struttura MVC 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.

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>
5 riferimenti
public class BaseEntity
{
    /// <summary>
    /// Numero identificativo che serve a tutte le classi
    /// </summary>
    2 riferimenti
    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>
5 riferimenti
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>
0 riferimenti
public string Titolo { get; set; }
/// <summary>
/// nome, cognome e mail del formatore che si occupa del progetto.
/// </summary>
0 riferimenti
public string NomeFormatore { get; set; }
0 riferimenti
public string CognomeFormatore { get; set; }
0 riferimenti
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>
0 riferimenti
public DateTime? DataInizio { get; set; }
0 riferimenti
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>
0 riferimenti
public string Descrizione { get; set; }
```

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>
    /// id identificativo di un requisito.
    /// </summary>
    //[Key]
    //public int Id { get; set; }

    /// <summary>
    /// titolo del requisito.
    /// </summary>
    1 riferimento
    public string Titolo { get; set; }

    /// <summary>
    /// descrizione del requisito.
    /// </summary>
    0 riferimenti
    public string Descrizione { 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>
    0 riferimenti
    public Requisito Requisito { get; set; }
    /// <summary>
    /// qdc al quale vanno assegnati i requisiti.
    /// </summary>
    0 riferimenti
    public Qdc Qdc { get; set; }
    /// <summary>
    /// voto (da 0 a 3) per un requisito di un qdc.
    /// </summary>
    0 riferimenti
    public int? Voto { get; set; }
    /// <summary>
    /// codice identificativo di un requisito (A01 - A30).
    /// </summary>
    0 riferimenti
    public int Codice { get; set; }
    /// <summary>
    /// Motivazione per il voto.
    /// </summary>
    0 riferimenti
    public string Motivazione { get; set; }
}
```

3.2.1.5 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>
4 riferimenti
public interface IDataRepository<T> where T : BaseEntity
{
    /// <summary>
    /// Ritorna un'entità in base all'id passato.
    /// funziona come una select.
    /// </summary>
    /// <param name="id">id passato per ricavare l'entità</param>
    /// <returns>Entità in base all'id</returns>
    1 riferimento
    T Get(int id);

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

    /// <summary>
    /// Inserisce una nuova entità.
    /// </summary>
    /// <param name="entity">Entità da inserire.</param>
    /// <returns>l'entità inserita</returns>
    1 riferimento
    T Insert(T entity);

    /// <summary>
    /// Update di un'entità.
    /// </summary>
    /// <param name="entity">Entità da modificare.</param>
    1 riferimento
    void Update(T entity);

    /// <summary>
    /// Elimina un'entità.
    /// </summary>
    /// <param name="entity">Entità da eliminare.</param>
    1 riferimento
    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>
4 riferimenti
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>
    /// <param name="ctx">Contesto di dati del database.</param>
    3 riferimenti
    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>
1 riferimento
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>
1 riferimento
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>
1 riferimento
public class RequisitoDbDataRepository : DbDataRepository<AppDbContext, Requisito>, IRequisitoDataRepository
{
    /// <summary>
    /// Definizione del contesto di dati.
    /// </summary>
    /// <param name="ctx">Contesto di dati del database.</param>
    0 riferimenti
    public RequisitoDbDataRepository(AppDbContext ctx) : base(ctx)
    {
        throw new NotImplementedException();
    }

    /// <summary>
    /// Ritorna tutti requisiti ordinati per titolo.
    /// </summary>
    /// <returns>i requisiti ordinati per titolo.</returns>
    8 riferimenti
    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
```

3.2.3 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>
9 riferimenti
public class AppDbContext : DbContext
{
    /// <summary>
    /// Raccolta di Qdc.
    /// </summary>
    0 riferimenti
    public DbSet<Qdc> Qdcs { get; set; }

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

    /// <summary>
    /// Raccolta di Assegnazioni.
    /// </summary>
    0 riferimenti
    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).

```
/// <summary>
/// Metodo costruttore vuoto (ereditato).
/// Ev. potrei mettere come parametro il nome (es: DefaultConnection)
/// </summary>
0 riferimenti
public AppDbContext() : base()
{
}

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

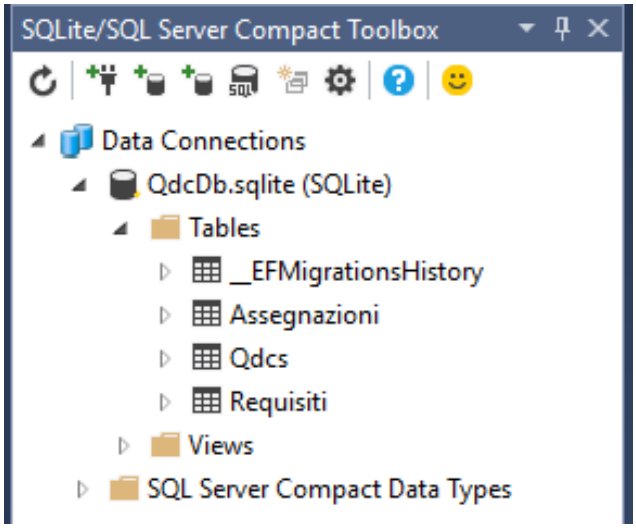
Questo metodo si occupa di configuarare il percorso nel quale salvare il database. Ho usato Sqlite (che funziona solo importando il pacchetto NuGet EntityFrameworkCore.Sqlite).

```
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.



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>
9 riferimenti
public abstract class BindableBase : INotifyPropertyChanged
{
    /// <summary>
    /// Raised when a property on this object has a new value.
    /// </summary>
    public event PropertyChangedEventHandler PropertyChanged = delegate { };

    /// <summary>
    /// Raises this object's PropertyChanged event.
    /// </summary>
    /// <param name="propertyName">The name of the property that has a new value.</param>
    1 riferimento
    protected void OnPropertyChanged(string propertyName)
    {
        PropertyChangedEventHandler handler = PropertyChanged;
        if (handler != null)
        {
            handler(this, new PropertyChangedEventArgs(propertyName));
        }
    }
```

3.3.1.2 IDelegateCommand

IdelegateCommand è una semplice interfaccia …

```
2 riferimenti
public interface IDelegateCommand : ICommand
{
    1 riferimento
    void RaiseCanExecuteChanged();
}
```

3.3.1.3 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>
2 riferimenti
public class DelegateCommand : IDelegateCommand
{
    Action<object> execute;
    Func<object, bool> canExecute;
    // Evento richiesto da ICommand
    public event EventHandler CanExecuteChanged;
    // Construttore
    1 riferimento
    public DelegateCommand(Action<object> onExecute, Func<object, bool> canExecute = null)
    {
        this.execute = onExecute;
        this.canExecute = canExecute;
    }

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

3.3.1.4 Utility

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

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

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

3.3.2 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 nelle View in modo che possano essere fatte delle operazioni. Nel mio caso praticamente ogni View ha un ViewModel corrispondente.

3.3.2.1 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 RequisitoViewModel RequisitoVM;
private DescrizioneViewModel DescrizioneVM;
private MotivazioneViewModel MotivazioneVM;
private MenuViewModel MenuVM;

/// <summary>
/// istanza di IDelegateCommand per la delega delle operazioni
/// </summary>
1 riferimento
public IDelegateCommand QdcListCommand { get; set; }
1 riferimento
public IDelegateCommand RequisitoListCommand { get; set; }
1 riferimento
public IDelegateCommand DescizioneListCommand { get; set; }
1 riferimento
public IDelegateCommand MotivazioneListCommand { get; set; }
1 riferimento
public IDelegateCommand MenuListCommand { get; set; }

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

6 riferimenti
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>
/// Costruttore che richiama RegisterCommands()
/// </summary>
0 riferimenti
public MainViewModel()
{
    RegisterCommands();
}

/// <summary>
/// Delega delle operazioni a DelegateCommand
/// </summary>
1 riferimento
private void RegisterCommands()
{
    MenuVM = new MenuViewModel();
    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>
1 riferimento
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>
1 riferimento
public void OnViewModelReceived(BindableBase viewmodel)
{
    CurrentViewModel = viewmodel;
}
```

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

```
private void OnMenuList(object obj)
{
    CurrentViewModel = MenuVM;
}

1 riferimento
private bool CanMenuList(object arg)
{
    return true;
}
```

3.3.2.2 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).
Viene quindi creata la variabile che rappresenta QdcViewModel e il comando associato al pulsante

```
private QdcViewModel CreaQdcVM;
1 riferimento
public IDelegateCommand QdcListCommand { get; set; }
```

Il metodo usato per passare a QdcView e quindi a QdcViewModel è lo stesso usato nel MainViewModel.

3.3.2.3 QdcViewModel

QdcViewModel è il ViewModel relativo alla classe Qdc. Implementa BindableBase e definisce un insieme (ObservableCollection) di Qdc. Nel costruttore viene generata una variabile di tipo QdcDbDataRepository che serve a definire il contesto dei dati sul quale lavorare e viene istanziato l'insieme di Qdc nel quale vengono inseriti tutti i Qdc esistenti tramite il metodo Get(). Anche questo ViewModel permette coem tutti gli altri di passare a quello successivo, con lo stesso procedimento indicato in MainViewModel.

```
2 riferimenti
public class QdcViewModel : BindableBase
{
    /// <summary>
    /// Insieme che contiene i campi di un Qdc.
    /// </summary>
    1 riferimento
    public ObservableCollection<Qdc> Qdcs { get; set; }

    /// <summary>
    /// Metodo costruttore del ViewModel del Qdc.
    /// </summary>
    0 riferimenti
    public QdcViewModel()
    {
        QdcDbDataRepository repo = new QdcDbDataRepository(new AppDbContext());
        Qdcs = new ObservableCollection<Qdc>(repo.Get());
    }
}
```

3.3.2.4 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 relativo alla classe Qdc (ed è praticamente identico a QdcViewModel).

```
1 riferimento
public class DescrizioneViewModel : BindableBase
{
    1 riferimento
    public ObservableCollection<Qdc> Descrizioni { get; set; }

    0 riferimenti
    public DescrizioneViewModel()
    {
        QdcDbDataRepository repo = new QdcDbDataRepository(new AppDbContext());
        Descrizioni = new ObservableCollection<Qdc>(repo.Get());
    }
}
```

3.3.2.5 RequisitoViewModel

La classe RequisitoViewModel funziona esattamente come la classe QdcViewModel, tranne per il fatto che ovviamente si rifà alla classe Requisito.

```
1 riferimento
public class RequisitoViewModel : BindableBase
{
    1 riferimento
    public ObservableCollection<Requisito> Requisiti { get; set; }

    0 riferimenti
    public RequisitoViewModel()
    {
        RequisitoDbDataRepository repo = new RequisitoDbDataRepository(new AppDbContext());
        Requisiti = new ObservableCollection<Requisito>(repo.Get());
    }
}
```

3.3.2.6 AssegnazioneViewModel

Questa classe funziona esattamente come le due sopra.

3.3.2.7 MotivazioneViewModel

MotivazioneViewModel funziona come DescrizioneViewModel, infatti non esiste una classe Motivazione (o voto) nella libreria di classi, ma solo dei campi presenti nella classe Assegnazione. Quindi questo ViewModel è relativo alla classe Assegnazione (ed è praticamente identico a AssegnazioneViewModel).

```
1 riferimento
public class MotivazioneViewModel : BindableBase
{
    1 riferimento
    public ObservableCollection<Assegnazione> Motivazioni { get; set; }

    0 riferimenti
    public MotivazioneViewModel()
    {
        AssegnazioneDbDataRepository repo = new AssegnazioneDbDataRepository(new AppDbContext());
        Motivazioni = new ObservableCollection<Assegnazione>(repo.Get());
    }
}
```

3.3.3 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.

3.3.3.1 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>
```

3.3.3.2 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).

```
<Grid>
    <DockPanel>
        <Menu DockPanel.Dock="Top">
            <MenuItem Header="_File">
                <MenuItem Header="_Crea QDC" Command="{Binding Path=QdcListCommand}"/>
                <MenuItem x:Name="esciMI" Header="_Esci" Click="EsciMI_Click"/>
            </MenuItem>
        </Menu>
        <StackPanel></StackPanel>
    </DockPanel>
    <ContentControl Content="{Binding Path=CurrentViewModel}"/>
</Grid>
```

3.3.3.3 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=MotivazioneListCommand}">
</Button>
```

3.3.3.4 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.

```
<TextBox Grid.Row="1" Grid.Column="0" Grid.ColumnSpan="2"
         Margin="20,8,20,8" x:Name="TitoloTextBox"
         AutomationProperties.HelpText="Titolo"
         Text="Titolo" Foreground="Gray"></TextBox>

<TextBox Grid.Row="2" Grid.Column="0" Grid.ColumnSpan="1"
         Margin="20,8,20,8" x:Name="NomeFormatoreTextBox"
         AutomationProperties.HelpText="Nome formatore"
         Text="Nome formatore" Foreground="Gray"></TextBox>
```

Il testo iniziale funziona come un placeholder, che ho dovuto gestire "manualmente" perché non c'è una proprietà specifica. Per farlo ho aggiunto due metodi che mi controllano quando il cursore entra nei vari TextBox:

```
MouseEnter="PathTextBox_MouseEnter" MouseLeave="PathTextBox_MouseLeave"/>
```

Nei due metodi gestisco il funzionamento dei placeholder.
Alla fine della View sono presenti due pulsanti, uno che permette di salvare i dati inserendoli nel database e l'altro che permette di passare alla View successiva (in questo caso DescrizioneView).

```
<Button Grid.Row="9" Grid.Column="0" Grid.ColumnSpan="1"
        Margin="40,10,40,10" Content="Salva"
        Command="{Binding SaveInfo}"
        FontSize="14"
        x:Name="SaveInfoButton" Click="SaveInfoButton_Click"/>

<Button Grid.Row="9" Grid.Column="1" Grid.ColumnSpan="1"
        Margin="40,10,40,10" Content="Continua"
        Command="{Binding DescrizioneCommand}"
        FontSize="14"
        x:Name="GoDescrizioneButton"/>
```

Anche in questo caso le operazioni sono gestite tramite dei bind.

3.3.3.5 DescrizioneView

DescrizioneView è la view che permette di inserire una descrizione al quaderno dei compiti. È associata a DescrizioneViewModel. 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="1" Grid.Column="0" Grid.ColumnSpan="2"
    Margin="20,8,20,8"
    x:Name="DescrizionePeritoTextBox" />
```

Anche in questo caso sono presenti i due pulsanti che permettono di salvare e di passare all View successiva (in questo caso RequisitoView).

3.3.3.6 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 settato a false).

```
<TextBox
    Grid.Row="2" Grid.Column="2" Grid.ColumnSpan="1"
    Margin="20,8,20,8" x:Name="req14" IsEnabled="False">
</TextBox>

<TextBox
    Grid.Row="2" Grid.Column="2" Grid.ColumnSpan="1"
    Margin="20,8,20,8" x:Name="req14" IsEnabled="False">
</TextBox>
```

Come per le altre View sono presenti i due pulsanti che permettono di salvare i dati nel Database oppure di passare alla view successiva.

3.3.3.7 MotivazioneView

MotivazioneView è la view che permette di inserire un voto e una motivazione per ogni requisito (sia i sette specifici scelti in RequisitoView che gli altri 33 fissi per ogni progetto). La View è formata da 3 TextBox disabilitati che stampano 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. Sono presenti anche dei pulsanti che permettono di navigare nella View stessa di requisito in requisito.

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);
```

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 2. Eseguire il programma MenuSAMT.vsix 3. Verificare il funzionamento del modello MVC 4. Provare a salvare un programma 5. Chiudere Visual Studio, riaprirlo e verificare che il modello MVC funzioni ancora		
Risultati attesi:	Dopo aver fatto partire il programma MenuSAMT il template MVC viene creato. Se si chiude VS e si riapre in seguito il template rimane funzionante.		

Test Case:	TC-002	**Nome:**	Funzionamento database (SQLServer)
Riferimento:	RQ_009		
Descrizione:			
Prerequisiti:	Database creato correttamente e accessibile		
Procedura:	1. Creare le tabelle del database partendo dallo schema ER 2. Inserire manualmente qualche dato per il test 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 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 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 2. Salvare il qdc e verificare che i requisiti vengano salvati correttamente sul database, compresa l'assegnazione (tabella assegna). 3. Provare a scegliere i sette requisiti inserendo solo l'id 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 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: **Riferimento**:	TC-008 -	**Nome:**	Controllo dati
Descrizione:			
Prerequisiti:	Varie view già create e funzionanti.		
Procedura:	1. Inserire in tutte le view create dei dati validi 2. Controllare che questi dati vengono salvati correttamente sul database 3. Inserire nelle varie view qualche dato non valido 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 Bibliografia per articoli di riviste:

1. Cognome e nome (o iniziali) dell'autore o degli autori, o nome dell'organizzazione,
2. Titolo dell'articolo (tra virgolette),
3. Titolo della rivista (in italico),
4. Anno e numero
5. Pagina iniziale dell'articolo,

7.2 Bibliografia per libri

1. Cognome e nome (o iniziali) dell'autore o degli autori, o nome dell'organizzazione,
2. Titolo del libro (in italico),
3. ev. Numero di edizione,
4. Nome dell'editore,
5. Anno di pubblicazione,
6. ISBN.

7.3 Sitografia

1. URL del sito (se troppo lungo solo dominio, evt completo nel diario),
2. Eventuale titolo della pagina (in italico),
3. Data di consultazione (GG-MM-AAAA).

Esempio:

- http://standards.ieee.org/guides/style/section7.html, *IEEE Standards Style Manual*, 07-06-2008.

8 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
- …