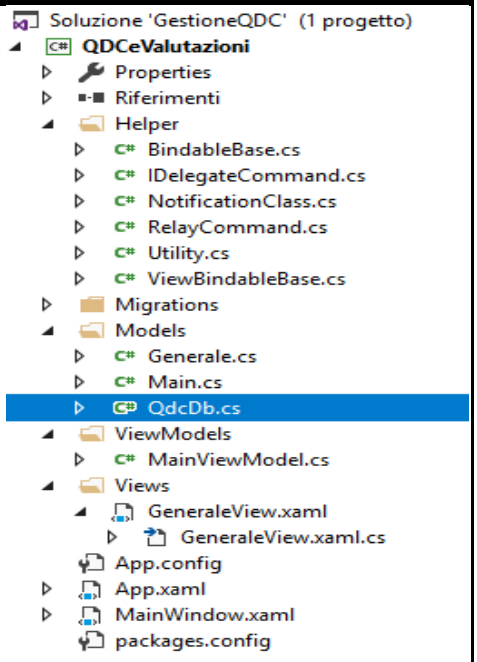
Diario di lavoro

Luogo	SAM Trevano
Data	01.10.2019

Lavori svolti

Oggi ho provato a cambiare approccio per la gestione del database: ho creato velocemente una nuova solution di test con all'interno un progetto WPF ma questa volta senza la libreria di classe. Nel progetto ho creato la struttura MVVM tramite il tool Menu Samt. Negli helper ho definito RelayCommand.cs (uguale a DelegateCommand), NotificationClass.cs (uguale a BindableBase, contiene il metodo onPropertyChanged), Utility.cs che contiene il metodo per gestire gli errori. Nel Model ho creato la classe Generale.cs, che contiene i vari campi che formeranno il database e la classe Main.cs, che contiene dei metodi di supporto per la gestione del database (get, gestione degli update e dei delete). Nel ViewModel ho creato la classe MainViewModel.cs che si occupa di interfacciare la view al model Main. Nella View ho creato una view semplicistica che permette di inserire dei campi di un qdc e di fare anche update o delete. Ho poi creato il context QdcDb che contiene il costruttore per interfacciarsi con il database e un istanza dello stesso (Qdcs). Ho infine creato la stringa di connessione nel file app.config.



Come si vede nell'immagine è presente un ulteriore cartella Migrations. Questa cartella è stata creata automaticamente dopo che ho eseguito appunto le migrazioni. I passaggi che ho svolto sono i seguenti:
Strumenti → **Gestione pacchetti NuGet** → Console di gestione pacchetti.
>Enable-Migrations
>Add-Migration InitialCreate
> Update-Database
Queste migrazioni servono per permettere di avere I dati sempre aggiornati, sincronizzati e coerenti in modo automatico. Se per esempio alla chiusura dell'applicazione dovvesse esserci qualche tipo di problema i dati non verrebbero eliminati (lo stesso vale anche se all'avvio dell'applicazione è previsto un Drop If Exists del database o di una tabella). Poi ho provato l'applicazione e l'inserimento, l'aggiornamento e l'eliminazione di qualche dato. Queste semplici operazioni funzionano.
Le informazioni per creare questo test le ho prese da questo link: **https://www.dotnetforall.com/wpf-mvvm-practical-data-application-example/**, che spiega molto bene tutti i passaggi per creare un'applicazione MVVM con la connessione a un database. Per quanto riguarda le migrazioni invece mi sono rifatto al sito ufficiale delle Microsoft: **https://docs.microsoft.com/en-us/ef/core/managing-schemas/migrations/**.
Quindi sono tornato sul progetto originale ma siccome dava qualche errore di dipendenza e nel corso del progetto ho sbagliato a inserire i riferimenti ho deciso di continuare dal progetto creato oggi. Ho quindi aggiunto una Libreria di classi QDCeValutazioni.DA e ho creato la struttura corretta sempre usando il programma Menu Samt. Ho Eliminato le classi dal Model creato in precedenza e ho creato la classe Qdc nel Model corretto (la classe contiene gli stessi campi che conteneva la classe Generale). Ho anche spostato il context dal progetto WPF al progetto DA. Così facendo la creazione del database è funzionante

anche con questo modello. Dopo ho fatto ancora qualche breve test ho iniziato a modificare la view per renderla più simile a quella che dovrà essere quella finale (con tutti i campi necessari, visto che per i test ne avevo inseriti solo tre). Infine ho aggiornato la documentazione, togliendo qualche passaggio inserito l'ultima lezione (principalmente parti del progetto svolte in modo sbagliato) e inserendo quanto svolto oggi.

Stringa di connessione:

```
connectionStrings>
<add name="DefaultConnection" connectionString="Data Source=(LocalDB)\MSSQLLocalDB;
Initial Catalog=QdcDb; Integrated Security=SSPI" providerName="System.Data.SqlClient"/>
</connectionStrings>
```

Non ho inserito il parametro AttachName perché generava un errore e ho notato che il nome viene asseganto in modo corretto (prendendolo la costruttore nel context).

Nella classe Main.cs viene istanziato il database in una variabile:

```
QdcDb _dbContext = null;
public Main()
{
    AppDomain.CurrentDomain.SetData("DataDirectory",
    Environment.GetFolderPath(Environment.SpecialFolder.CommonApplicationData));
    _dbContext = new QdcDb();
}
```

Nel ViewModel viene istanziato un campo che si occupa di fornire notifiche quando viene modificato un elenco (in questo caso il database):

```
private ObservableCollection<Generale> personCollection;
public ObservableCollection<Generale> PersonCollection
{
    get { return personCollection; }
    set
    {
        personCollection = value;
        OnPropertyChanged();
    }
}
```

Problemi riscontrati e soluzioni adottate

Il primo problema che ho riscontrato è riferito alla scorsa lezione, infatti mi ero dimenticato di eseguire le migrazioni. Comunque anche oggi dopo averle fatte il problema nel progetto su cui lavoravo è persistito, da qui la scelta di cambiare approccio. Un altro problema è stato che quando ho creato la stringa di connessione con il campo: AttachDBFilename=|DataDirectory|\PersonDb.mdf, il comando Update-Database non funzionava, generando l'errore Cannot attach the file *.mdf as database. Ho cercato su internet una possibile soluzione e ho provato a togliere del tutto il campo AttachName. Così facendo ha funzionato (**https://stackoverflow.com/questions/17012839/cannot-attach-the-file-mdf-as-database**). L'ultimo errore era generato dopo la compilazione. L'errore era cs0111: Il tipo 'class' definisce già un membro denominato 'member' con gli stessi tipi di parametro. Questo perché nel codice della view avevo implementato (in modo automatico credo perché effettivamente non l'ho mai scritto io) il metodo InitializeComponent(), che però era già istanziato in modo automatico in un file di debug (obj -> debug -> ViewGenerale.g.xaml.cs).

Punto della situazione rispetto alla pianificazione

Sono leggermente in ritardo anche a causa di un assenza la scorsa lezione (27.09.2019). Probabilmente riuscirò a recuperare senza dover recuperare ore extra-scolastiche, nel caso fosse necessario comunque il martedì mattina ho due ore liberi a disposizione.

Programma di massima per la prossima giornata di lavoro
Rendere la view creata oggi più simile a quella finale. Ricreare la view di partenza (ho già il codice). Creare tutte le altre view e provare a interfacciarle tra di loro (tramite dei pulsanti) in modo da avere la struttura di base del progetto.