Diario di lavoro

Luogo	SAM Trevano
Data	15.10.2019

Lavori svolti

La scorsa lezione ho finito di documentare il progetto DA, con tutte le classi create. Oggi i primi 20 minuti gli ho usati per creare una nuova soluzione vuota dove ho ricreato velocemente tutta la struttura (ho fatto copia e incolla dei vari files). Ho deciso di farlo perché in questo modo il codice è molto più pulito così come le varie dipendenza. Dopo aver finito ho ricreato anche il progetto WPF e non ho aggiunto le views che ho già creato, in questo modo posso concentrarmi solo sui ViewModel e una volta pronti farò copia e incolla delle Views che comunque sono già pronte all'utilizzo. Ho anche deciso che per questo caso documenterò ogni classe passo per passo, per non dover fare tutto alla fine.
Per prima cosa ho ricopiato gli Helper fatti in precendenza. Questi al momento non necessitano di nessuna modifica. Ho copiato BindableBase, DelegateCommand, Utility e ViewBindableBase, invece RelayCommand e NotificationClass non le ho copiate perché erano sostanzialmente una copia di DelegateCommand e BindableBase che ho scritto per fare dei test. In questo modo il codice è più pulito.

Nei ViewModel Ho creato la classe QdcViewModel che implementa BindableBase. Ho definito un insieme (ObservableCollection) di qdc che contiene tutti gli attributi di un qdc e ho definito il costruttore nel quale istanzio un nuovo contesto di dati per il qdc ed eseguo un Get() (serivirà per capire se funziona il tutto):

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

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

Ho creato dopodiché le classi RequisitoViewModel e AssegnazioneViewModel che funzionano allo stesso modo ma si riferiscono ovviamente alle rispettive classi. Infine ho creato il MainViewModel. Questa particolare classe si occupa di gestire tutti gli altri ViewModel e di capire quali mostrare. Per ora ho creato una variabile per il QdcViewModel, perché il primo che mi serve:

```
private QdcViewModel QdcVM;
```

Ho implementato anche un istanza di IdelegateCommand:

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

e di BindableBase:

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

QdcListCommand serve per delegare il comando da effettuare (a DelegateCommand), currentViewModel invece serve per settare il ViewModel da utilizzare.

```
private void RegisterCommands()
{
   QdcListCommand = new DelegateCommand(OnQdcList, CanQdcList);
}
```

Il metodo RegisterCommands viene richiamato dal costruttore. Vengono richiamati i metodi OnQdcList() e CanQdcLis()t (gestiti da DelegateCommand):

```
private void OnQdcList(object obj)
{
    CurrentViewModel = QdcVM;
}

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

OnQdcList() viene usato per definire QdcViewModel come viewmodel da utilizzare, CanQdcList() invece ritorna true se è possibile listare i qdc.

A Questo punto i ViewModel sono creati e dovrebbero permettere di eseguire già qualche operazione. Prima di copiare le View ho eliminato il Database che avevo utilizzato in precedenza e ho eseguito nuovamente le migrazioni. Poi ho copiato le view GeneraleView (l'ho rinominata QdcView), RequisitoView, DescrizioneView e MotivazioneView. AssegnazioneView non esiste perché non serve, l'assegnazione avverrà tramite un bottone. Dopo aver copiato queste view mi sono ricordato che esistono anche le descrizioni e le motivazioni, quindi ho creato anche i ViewModel DescrizioneViewModel e AssegnazioneViewModel. La descrizione contiene un ObservableCollection di QDC, perché descrizione è un campo che fa parte della classe Qdc.

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

Allo stesso modo AssegnazioneViewModel contiene un ObservableCollection di Assegnazione, perché la motivazione e il voto sono dei campi della classe Assegnazione.

Dopo aver finito questa parte ho cominciato la documentazione del progetto WPF.

Problemi riscontrati e soluzioni adottate

Il primo problema, facilmente risolvibile, è stato il fatto di aver creato i ViewModel riferiti ai Model, quindi Qdc, Requisito e Assegnazione. Mi ero dimenticato però che ci sono delle Views dedicate anche alla descrizione e al voto e alla motivazione. Quando ho copiato queste View dal progetto vecchio, me ne sono ricordato, ho creato i ViewModel Descrizione e Motivazione e ho aggiunto i riferimenti. I due ViewModel sono però uguali a quelli del Qdc e Assegnazione, quindi non so se sono necessari o se si possono assegnare più Views allo stesso ViewModel senza avere problemi di qualche tipo.

Un altro problema è nei riferimenti tra la view e il viewmodel. Per esempio in QdcView ho messo il seguente riferimento:

```
<UserControl.DataContext>
    <viewmodel:QdcViewModel/>
</UserControl.DataContext>
```

Durante la compilazione però viene generato l'errore:

Operazione o metodo non implementato.

Non vengono date ulteriori informazioni, quindi non so a cosa possa essere riferito. Sicuramente il problema non è che non riesce a trovare il ViewModel, perché l'autocompilazione funziona. Se sostituisco QdcViewModel con MainViewModel l'errore non viene generato e la compilazione funziona.

Punto della situazione rispetto alla pianificazione
Sono in ritardo di circa 4 ore. Riguardando bene il gantt preventivo comunque mi sono accorto che ho previsto relativamente poco tempo per l'interfacciamento tra Visual Studio e MS Office e molto tempo per il controllo dei dati. Ovviamente può essere aumentato il tempo da dedicare alle operazioni sul file Word e diminuito quello sul controllo dei dati. Questo perché se il sistema funziona in modo corretto i controlli potranno essere eseguiti in un secondo momento.

Programma di massima per la prossima giornata di lavoro
La prossima lezione finirò di documentare le classi create oggi e farò la MainView (che permetterà di navigare tra le view). Farò degli insert nel database direttamente (statici) e proverò il funzionamento della funzione Get(). Venerdì proverò invece a fare gli insert direttamente dalla View. Settimana prossima comincerò a cercare il metodo per interfacciare Visual Studio a MS Office.