



**Scuola d'Arti e Mestieri di Trevano**

**Sezione informatica**

## **Gestione degli esercizi delle prove svolte durante la professione di docente**

**Titolo del progetto:** Gestione degli esercizi delle prove svolte durante la professione di docente  
**Alunno:** Gabriele Alessi  
**Classe:** Info 4  
**Anno scolastico:** 2019/2020  
**Docente responsabile:** Ugo Bernasconi

## Sommario

1	Introduzione .....	4
1.1	Informazioni sul progetto .....	4
1.2	Abstract .....	4
1.3	Scopo .....	4
	Analisi .....	5
1.4	Analisi del dominio .....	5
1.5	Analisi e specifica dei requisiti .....	5
1.6	Use case .....	7
1.7	Pianificazione .....	8
1.8	Analisi dei mezzi .....	9
1.8.1	Software .....	9
1.8.2	Hardware .....	9
2	Progettazione .....	10
2.1	Design dell'architettura del sistema .....	10
2.2	Design dei dati e database .....	12
2.3	Design delle interfacce .....	13
2.3.1	Schermata principale .....	13
2.3.2	Impostazioni di base .....	14
2.3.3	Esercizi .....	15
2.3.4	Creazione prova .....	16
2.4	Design procedurale .....	17
3	Implementazione .....	18
3.1	Creazione progetto .....	18
3.2	Libreria di Classi .....	19
3.2.1	Models .....	19
3.2.1.1	BaseEntity .....	19
3.2.1.2	Classe .....	19
3.2.1.3	Modulo .....	19
3.2.1.4	Tematica .....	20
3.2.1.5	Anno .....	20
3.2.1.6	Esercizio .....	20
3.2.1.7	EsercizioProva .....	20
3.2.1.8	Prova .....	20
3.2.2	Services .....	21
3.2.2.1	IDataRepository .....	21
3.2.2.2	DbDataRepository .....	21
3.2.2.3	IClasseRepository .....	22
3.2.2.4	ClasseDbRepository .....	22
3.2.2.5	IModuloRepository .....	22
3.2.2.6	ModuloDbRepository .....	22
3.2.2.7	ITematicaRepository .....	22
3.2.2.8	TematicaDbRepository .....	22
3.2.2.9	IAnnoRepository .....	23
3.2.2.10	AnnoDbRepository .....	23
3.2.2.11	IEsercizioRepository .....	23
3.2.2.12	EsercizioDbRepository .....	23
3.2.2.13	IEsercizioProvaDbRepository .....	23
3.2.2.14	EsercizioProvaDbRepository .....	23
3.2.2.15	IProvaRepository .....	23
3.2.2.16	ProvaDbRepository .....	24
3.2.3	AppDbContext .....	24
3.2.4	Database .....	25
3.2.5	Diagramma delle classi .....	26

3.3	App WPF .....	27
3.3.1	ViewModels .....	27
3.3.1.1	MainViewModel .....	28
3.3.1.2	BenvenutoViewModel .....	29
3.3.1.3	AboutViewModel .....	29
3.3.1.4	GuidaViewModel .....	29
3.3.1.5	EsercizioListViewModel .....	29
3.3.1.6	ProvaListViewModel .....	30
3.3.1.7	ImpostazioniBaseViewModel .....	30
3.3.1.8	ClasseViewModel .....	30
3.3.1.9	ModuloViewModel .....	31
3.3.1.10	TematicaViewModel .....	31
3.3.1.11	EsercizioViewModel .....	31
3.3.1.12	ProvaViewModel .....	32
3.3.2	Diagramma delle classi .....	33
3.3.3	Views .....	34
3.3.3.1	MainView .....	34
3.3.3.2	BenvenutoView .....	35
3.3.3.3	AboutView .....	35
3.3.3.4	GuidaView .....	36
3.3.3.5	EsercizioListView .....	36
3.3.3.6	ProvaListView .....	37
3.3.3.7	ImpostazioniBaseView .....	37
3.3.3.8	ClasseView .....	38
3.3.3.9	ModuloView .....	38
3.3.3.10	TematicaView .....	38
3.3.3.11	EsercizioView .....	39
3.3.3.12	ProvaView .....	41
4	Test .....	42
4.1	Protocollo di test .....	42
4.2	Risultati test .....	45
4.3	Mancanze/limitazioni conosciute .....	45
4.3.1	Impostazioni di base .....	45
4.3.2	Esercizio .....	45
4.3.3	Creazione Prova .....	45
5	Consuntivo .....	46
6	Conclusioni .....	47
6.1	Sviluppi futuri .....	47
6.2	Considerazioni personali .....	47
7	Sitografia .....	48
8	Allegati .....	49

## **1 Introduzione**

---

### **1.1 Informazioni sul progetto**

- Allievo: Gabriele Alessi  
Superiore professionale: Ugo Bernasconi
- Scuola d'Arti e Mestieri di Trevano, Sezione informatica, Classe 4, Progetti individuali
- Data inizio: 03.09.2019  
Data fine: 20.12.2019

### **1.2 Abstract**

The teachers of the Scuola d'Arti e Mestieri di Trevano use to create the exercises and the tests without any type of help from a tool or system. This project consists of creating a computer program that allows you to create and manage the exercises and the tests quickly and easily. In detail, the program works so you can set up the basic info (subjects, classes and themes) to then use them to create the exercises.

When you create an exercise, you have a global vision of the basic settings and then you can define the text and eventually insert an image. Finally, you can create the document by entering the fundamental fields (title, date, class and subject) and selecting the related exercises.

The application is entirely developed in C# MVVM with Visual Studio 2019 and for the data storage, SQLite is used.

### **1.3 Scopo**

Il progetto consiste nel sviluppare una piccola applicazione che gestisca e crei gli esercizi delle prove per poi prepararne il documento. Il programma deve funzionare in modo che si possano inserire delle informazioni di base: definizione di moduli, tematiche e classi.

In seguito si gestiscono i veri e propri esercizi, in cui si ha una visione generale delle impostazioni di base al fine di iniziare a inserire i dettagli dell'esercizio (titolo, testo, immagine, ...).

Infine si passa alla creazione del documento, dove vanno definiti i campi fondamentali e si selezionano gli esercizi da inserire nella prova.

Il progetto ha anche uno scopo scolastico, cioè la preparazione al lavoro finale LPI che si svolge alla fine dell'anno per l'ottenimento dell'AFC.

## Analisi

### 1.4 Analisi del dominio

Fino a ora i professori gestivano e creavano gli esercizi e le prove a mano, quindi l'obiettivo è quello di sviluppare un programma che agevoli appunto la gestione degli esercizi. Il prodotto è leggero e semplice e veloce da usare, quindi può funzionare senza problemi sui computer dei docenti della SAMT. Attualmente non sembra esistere una soluzione simile, dunque il sistema funzionerà in modo che sia user-friendly e che più docenti potranno usarlo.

### 1.5 Analisi e specifica dei requisiti

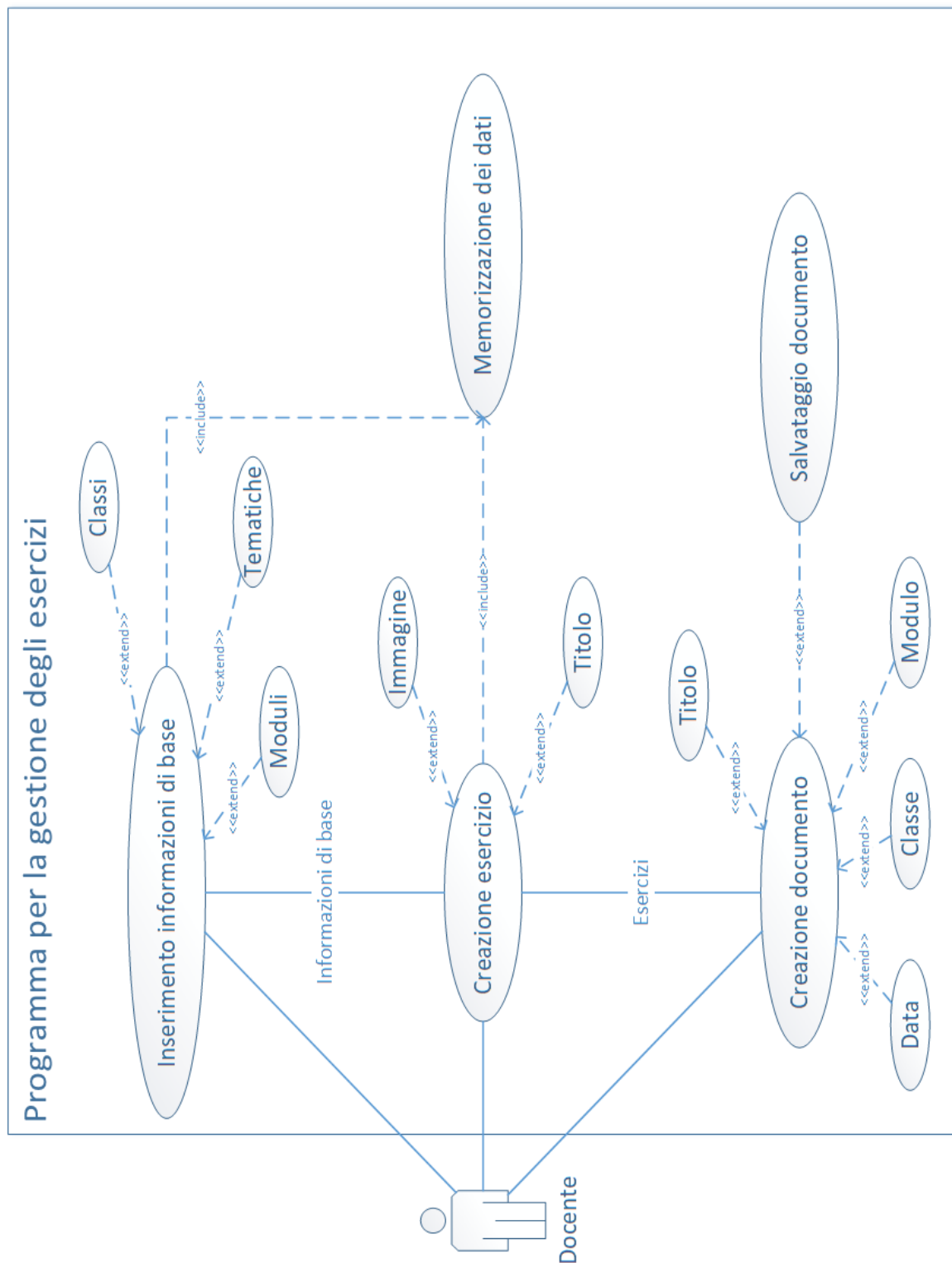
ID: REQ-001	
<b>Nome</b>	Realizzare un programma che gestisca gli esercizi
<b>Priorità</b>	1
<b>Versione</b>	1.0
<b>Note</b>	-
Sotto requisiti	
<b>001</b>	È necessario poter inserire delle informazioni di base (REQ-002)
<b>002</b>	Maschera di inserimento delle informazioni dell'esercizio (REQ-003)
<b>003</b>	Creazione delle prove (REQ-004)

ID: REQ-002	
<b>Nome</b>	Impostazioni di base
<b>Priorità</b>	1
<b>Versione</b>	1.0
<b>Note</b>	Il programma deve essere strutturato in modo che si possano definire delle impostazioni di base
Sotto requisiti	
<b>001</b>	Definizione dei moduli
<b>002</b>	Definizione delle tematiche (con una sequenza cronologica)
<b>003</b>	Definizione delle classi

ID: REQ-003	
<b>Nome</b>	Definizione degli esercizi
<b>Priorità</b>	1
<b>Versione</b>	1.1
<b>Note</b>	Si devono poter vedere le informazioni principali per poi creare gli esercizi
Sotto requisiti	
<b>001</b>	Impostazione informazioni di base
<b>002</b>	Testo esercizio
<b>003</b>	Immagine esercizio (Drag & Drop per semplificare l'UI)

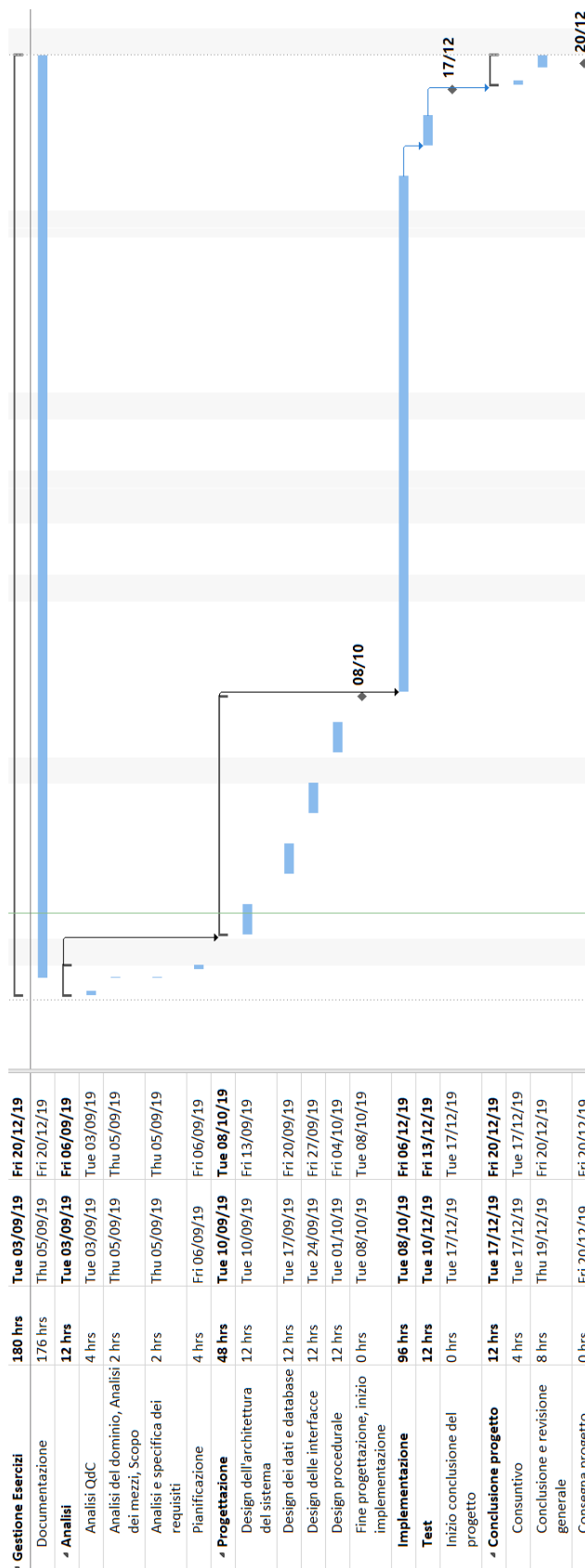
ID: REQ-004	
<b>Nome</b>	Creazione prove
<b>Priorità</b>	1
<b>Versione</b>	1.1
<b>Note</b>	Deve essere preparato un documento con i vari esercizi
Sotto requisiti	
<b>001</b>	Definizione campi fondamentali
<b>002</b>	Definizione esercizi (Drag & Drop per semplificare l'UI)
<b>003</b>	Preparazione documento

## 1.6 Use case



## 1.7 Pianificazione

La pianificazione del progetto è stata effettuata mediante la realizzazione di un diagramma di Gantt.





## **1.8 Analisi dei mezzi**

Qui vengono elencati i software usati e viene descritto il personal computer utilizzato per lo sviluppo del progetto.

### **1.8.1 Software**

I software utilizzati per la realizzazione di questo progetto sono:

- Microsoft Word 2016  
Realizzazione della documentazione.
- GitHub Desktop 2.2.4  
Gestione del sistema di versioning del progetto.
- Visual Studio Code 1.40  
Realizzazione dei diari giornalieri e gestione dei vari documenti di testo.
- Microsoft Visual Studio Community 2019  
Sviluppo del programma.
- Microsoft Project 2016  
Realizzazione del diagramma di Gantt.
- Microsoft Visio 2016  
Realizzazione di vari schemi e diagrammi.

### **1.8.2 Hardware**

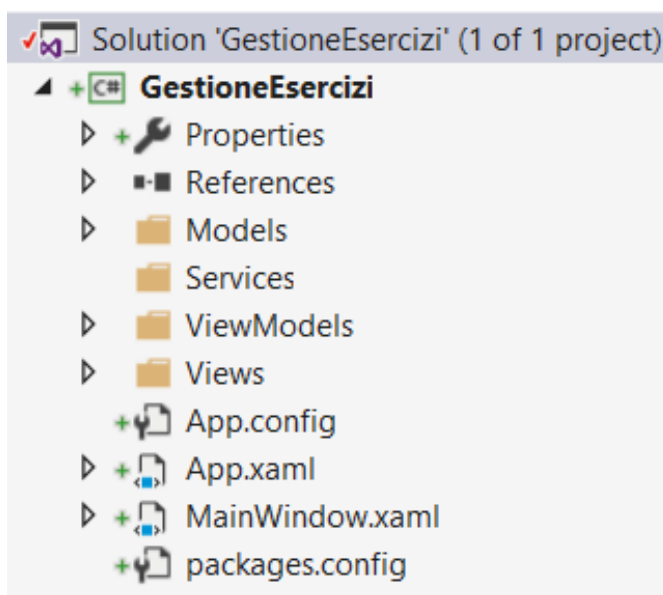
Personal Computer:

- HP Envy Notebook
- Intel® Core™ i7-6500U @ 2.50GHz
- 16GB
- Windows 10 Home 64bit
- Intel® HD Graphics 520

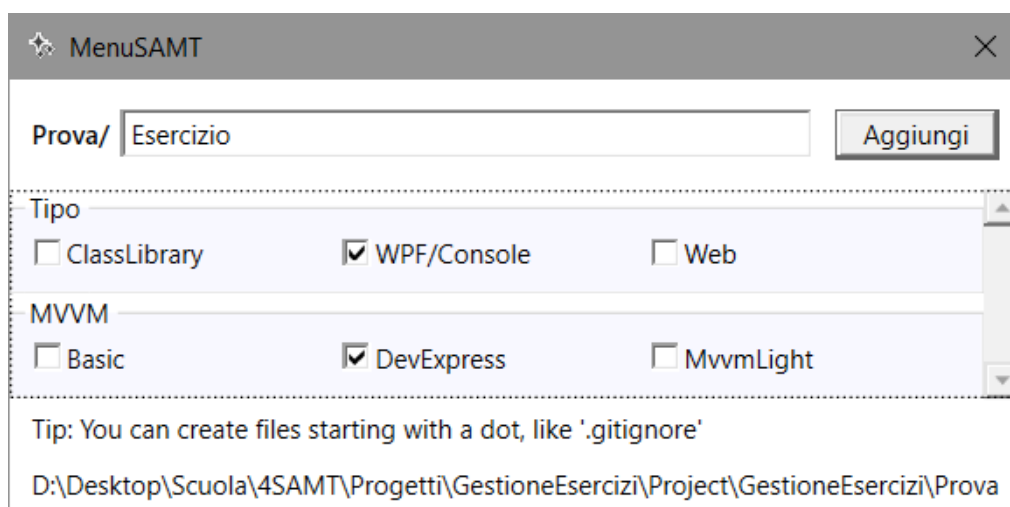
## 2 Progettazione

### 2.1 Design dell'architettura del sistema

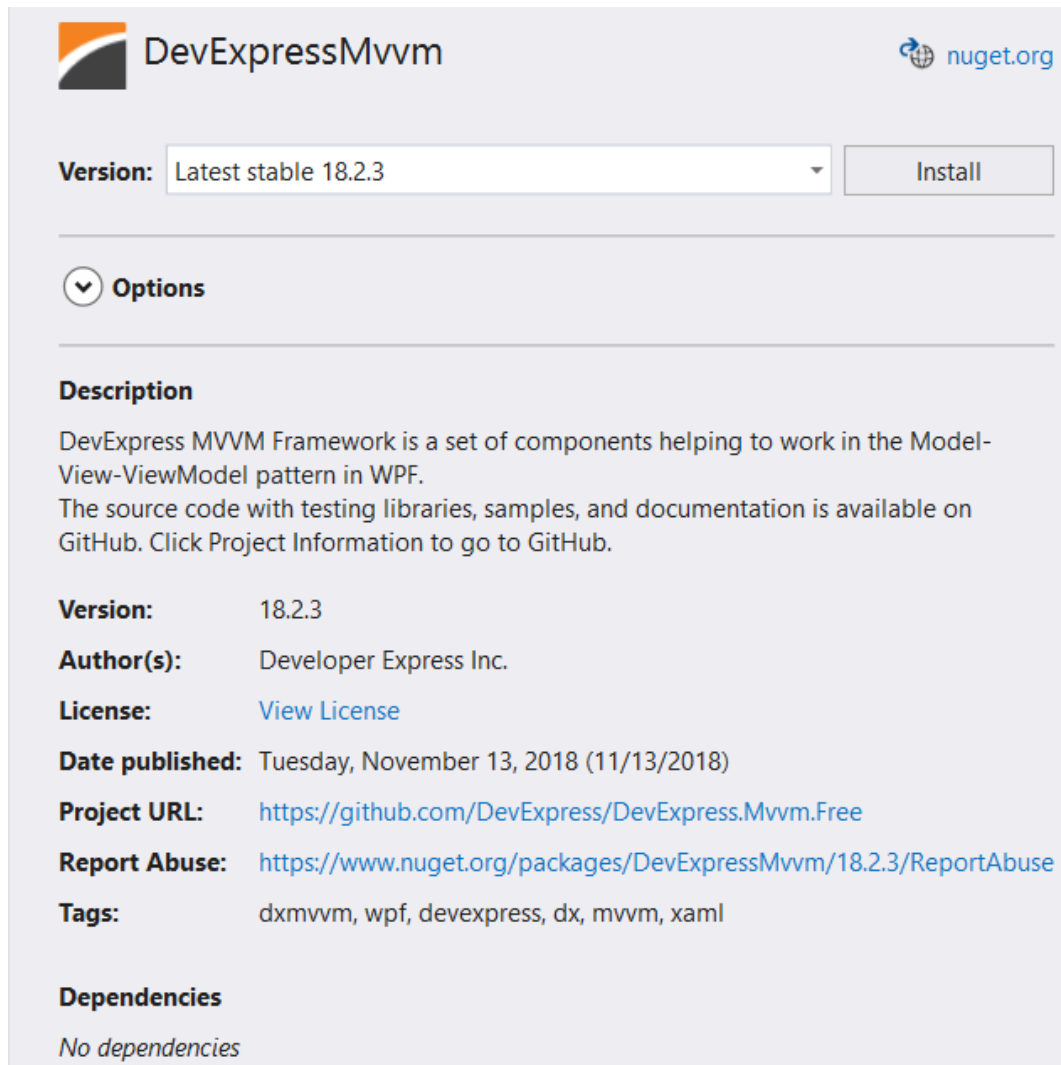
In questo capitolo viene spiegata la struttura generale del sistema e in particolare l'architettura del progetto. Il programma è sviluppato in C# utilizzando il pattern MVVM, ma ci sono altri componenti utili per semplificare il lavoro che sono spiegati anche nel capitolo di implementazione. La struttura del progetto parte da una Soluzione Vuota che contiene un progetto App WPF il quale gestisce l'intero sistema di dati e interfacce.



Per creare questa struttura mi sono valso dell'aiuto del superiore professionale che mi ha consegnato un'estensione che crea automaticamente l'oggetto Model, View, e ViewModel definito.



Per fare ciò ho anche usato il pacchetto DevExpressMvvm che è un insieme di strumenti che agevola il lavoro se si utilizza appunto il pattern MVVM. Inoltre esso lavora insieme all'estensione SAMT, quindi è molto semplice creare la struttura del progetto e gestire il sistema.



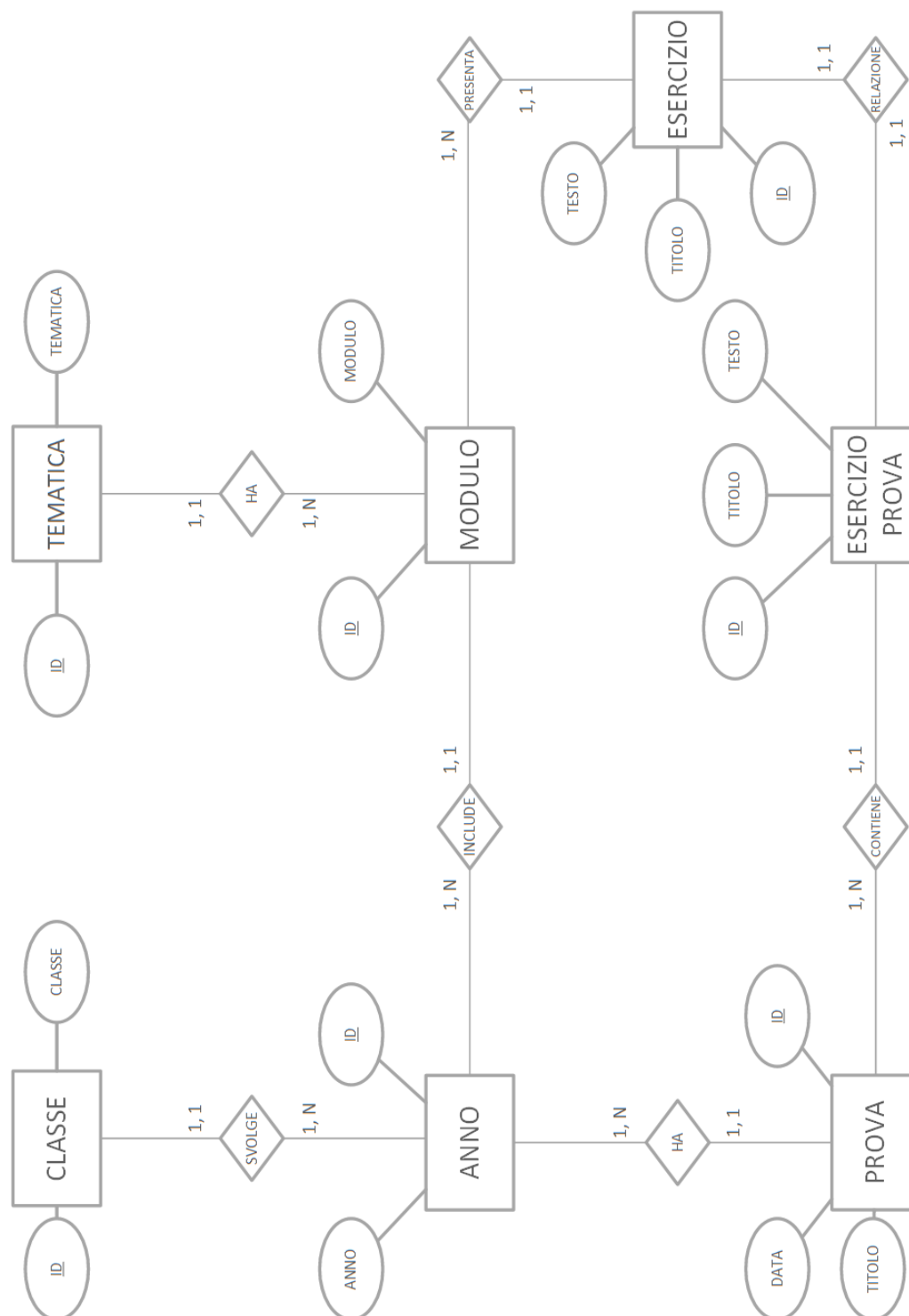
Infine nella cartella Services vengono gestiti i DataRepository, cioè tutto ciò che riguarda la struttura dei dati. In concreto ci saranno delle interfacce e delle classi che impostano le azioni che verranno sui dati (insert, delete, update).

```
public interface IRepository<T> where T : BaseEntity
{
    T Get(int id);
    IQueryable<T> Get();
    T Insert(T entity);
    void Update(T entity);
    void Delete(T entity);
}
```

Per il database verrà usato SQLite visto che funziona in modo facile e utilizza semplicemente un file per la memorizzazione dei dati.

## 2.2 Design dei dati e database

L'immagine seguente rappresenta la progettazione del database del sistema. In generale non è un diagramma molto articolato quindi da esso si può più o meno capire anche la struttura del sistema.



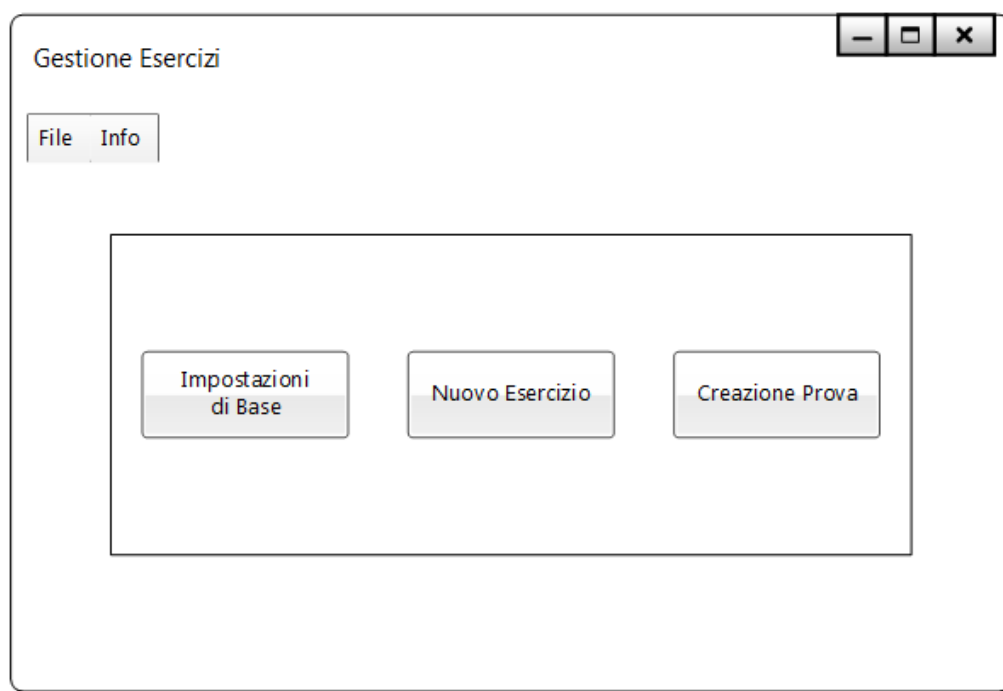
Le entità come *tematica*, *classe*, *anno* e *modulo* presenteranno bene o male sempre gli stessi valori in quanto rappresentano le impostazioni di base. Dopodiché ci sono le entità che simboleggiano gli esercizi e le prove, le quali sono relazionate tramite l'*anno* e il *modulo*. L'oggetto *esercizio prova* funge da collegamento tra *prova* e *esercizio* in modo che sia più facile gestire gli esercizi che si trovano nelle prove.

## 2.3 Design delle interfacce

In questa sezione vengono mostrate e descritte le interfacce dell'applicazione con cui l'utente interagisce.

### 2.3.1 Schermata principale

La prima schermata è molto semplice e presenta solamente tre pulsanti che rappresentano le tre azioni principali del programma.



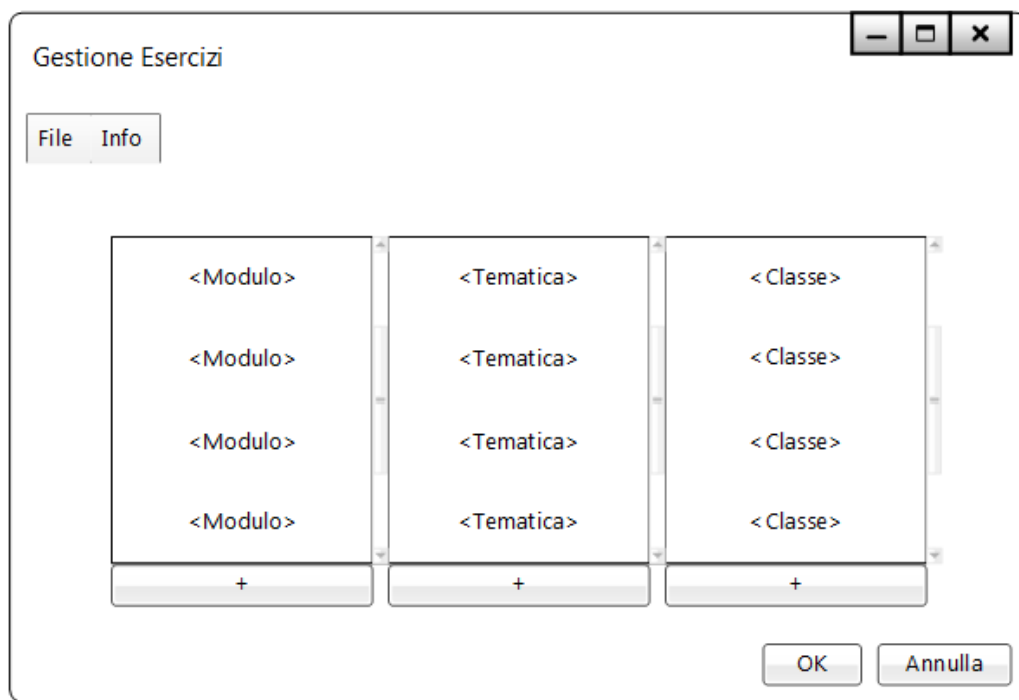
Il menu ha due opzioni e si estendono in questo modo:

- File
  - Esercizi  
Mostra una schermata con lista degli esercizi salvati nel database.
  - Prove  
Mostra una schermata con lista delle prove salvati nel database.
  - Opzioni  
Permette di configurare diverse opzioni come la posizione dei salvataggi.
  - Esci  
Chiusura dell'applicazione.
- Info
  - Guida  
Mostra una schermata con una breve guida sul prodotto.
  - About  
Mostra i diritti d'autore e altre informazioni riguardanti il prodotto (versione, licenza, ...).

I tre pulsanti apriranno una nuova schermata in base alla relativa scelta e non sarà possibile uscire da quella schermata senza l'uso dei pulsanti "OK" o "Annulla".

### 2.3.2 Impostazioni di base

Le impostazioni di base sono una parte molto importante per il corretto funzionamento e uso del sistema. Infatti definendole sarà possibile creare gli esercizi per le specifiche classi che svolgono i moduli e seguono delle precise tematiche.



Questa schermata è composta da tre menu in cui si può scorrere per vedere i dati inseriti (moduli, tematiche e classi) e i relativi pulsanti per aggiungere un nuovo elemento.

L'interfaccia sembra molto semplice, ma dietro ci sono anche alcune funzionalità che permettono di collegare queste tre entità tra loro e gestirle:

- Doppio click su una classe  
Mostra i moduli che essa svolge e permette di gestirli.
- Doppio click su un modulo  
Mostra le tematiche che esso comprende e permette di gestirli.
- Click + delete su un elemento lo elimina.

Infine ci sono i soliti pulsanti che consentono di annullare o applicare le eventuali modifiche apportate.

### 2.3.3 Esercizi

L'interfaccia che gestisce gli esercizi permette innanzitutto di scegliere tra le informazioni di base, per poi definire i dettagli dell'esercizio, cioè il testo e un eventuale immagine. Gli esercizi salvati verranno memorizzati per poi essere usati durante la realizzazione di una prova.

The screenshot shows the 'Gestione Esercizi' window with the 'Base' tab selected. The window has a title bar with standard minimize, maximize, and close buttons. Below the title bar are two tabs: 'File' and 'Info'. The 'Base' tab is active, showing a form with the following fields: 'Titolo esercizio...' (text input), 'Modulo' (dropdown menu), 'Tematica' (dropdown menu), and 'Classe' (dropdown menu). At the bottom right of the window are 'OK' and 'Annulla' buttons.

The screenshot shows the 'Gestione Esercizi' window with the 'Dettaglio' tab selected. The window has the same title bar and tabs as the previous screenshot. The 'Dettaglio' tab is active, showing a form with two main sections: 'Testo esercizio...' (a large text area) and 'Immagine' (a section containing an 'Sfogli...' button). At the bottom right of the window are 'OK' and 'Annulla' buttons.

La prima maschera consente di attribuire un titolo all'esercizio e di scegliere tramite un menu a tendina le informazioni di base stabilite nell'applicazione.

Nell'altra maschera è possibile comporre il testo dell'esercizio e inserire un'immagine attraverso Drag & Drop oppure sfogliando tra i file. Se si clicca "OK" il programma genererà un file RFT con il contenuto del testo e dell'immagine.

### 2.3.4 Creazione prova

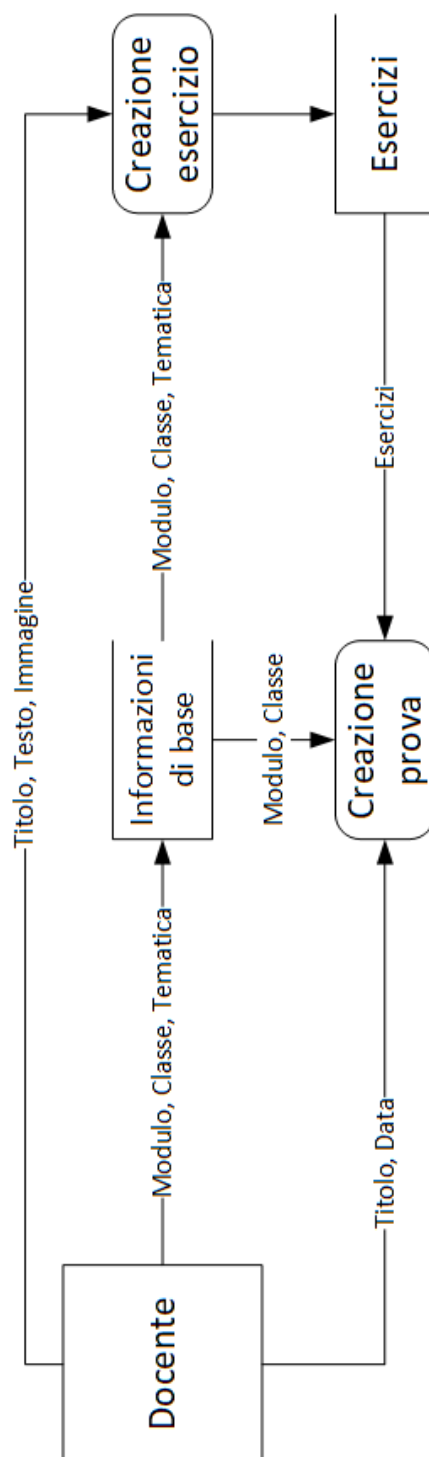
L'opzione finale dell'applicazione è la creazione delle prove. In questa schermata vengono dapprima definiti i campi fondamentali (più o meno come negli esercizi) e infine vengono selezionati gli esercizi da inserire.

Nel primo campo si scrive il titolo della prova, nel secondo si inserisce la data (anche con l'aiuto del calendario), poi si inseriscono classe e modulo, che vengono estrapolati sempre dalle impostazioni di base. Finalmente si aggiungono gli esercizi tramite Drag & Drop o sfogliando tra i propri file. Per avere un'anteprima della prova esiste un apposito pulsante che mostrerà un'altra finestra con ciò che produrrà il programma se si dovesse generare il file.



## 2.4 Design procedurale

In questo capitolo viene spiegato il ciclo di vita del prodotto e il suo comportamento in conseguenza di determinate azioni. Ciò è stato schematizzato in un diagramma di flusso dei dati.



In questo caso il docente è l'unico oggetto che effettua delle azioni di inserimento dati. Le informazioni che quest'ultimo inserisce vengono memorizzate nelle informazioni di base o possono servire per creare un esercizio o una prova.

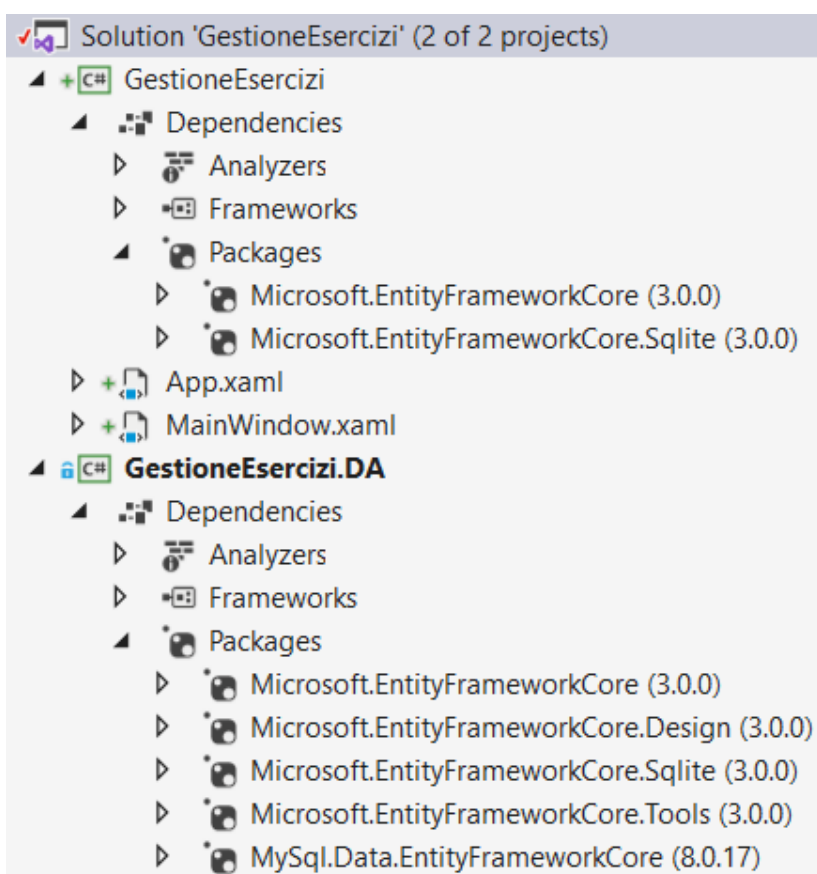
### 3 Implementazione

Nel capitolo di implementazione viene spiegato in dettaglio come il prodotto finale è stato sviluppato.



#### 3.1 Creazione progetto

Per iniziare si crea una Soluzione Vuota che fungerà da contenitore generale, quindi si inseriscono i progetti del sistema .NET Core, cioè una App WPF e una Libreria di Classi. Il prossimo passo è l'implementazione dei pacchetti necessari per l'amministrazione dei dati tramite il package manager NuGet.



Dall'immagine si può vedere che i pacchetti utilizzati sono quelli inerenti a EntityFrameworkCore, infatti questi sono utili per l'accesso ai dati e la gestione del database (in questo caso SQLite).

## 3.2 Libreria di Classi

La Libreria di Classi (.NET Core) è uno dei progetti della soluzione Visual Studio e contiene tutto ciò che ha a che fare con la gestione dei dati e la definizione del database.

### 3.2.1 Models

La cartella Models contiene le entità di base del sistema che rappresentano i dati, quindi per fare ciò si fa riferimento al diagramma del database. Inoltre le classi modello contengono il metodo ToString() che permette di stampare l'oggetto sotto forma di stringa mostrando il contenuto desiderato (di solito il nome). Infine si può dire che ci sono dei metodi costruttori (di cui uno vuoto per far funzionare i repository) usati per la creazione di nuove entità da inserire nel database. Sono anche presenti le proprietà che rappresentano gli ID delle entità relative in modo da gestire le foreign key durante gli inserimenti di dati nel database.

#### 3.2.1.1 BaseEntity

BaseEntity è la classe modello principale ed è la base di ogni altra entità, infatti essa contiene solo l'Id.

```
public class BaseEntity
{
    public int Id { get; set; }
}
```

#### 3.2.1.2 Classe

L'entità Classe contiene il nome della stessa (ad esempio "I4AA"), l'anno che sta svolgendo (ad esempio "2019/2020") e i relativi moduli e prove.

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

public virtual Anno Anno { get; set; }

public int AnnoId { get; set; }

public virtual ICollection<Modulo> Moduli { get; set; }

public virtual ICollection<Prova> Prove { get; set; }

public Classe(string nome, Anno anno)
{
    Nome = nome;
    AnnoId = anno.Id;
}
```

#### 3.2.1.3 Modulo

La classe Modulo presenta i campi che descrivono il nome ("Modulo <numero>"), la classe, le tematiche e gli esercizi. Di quest'ultimi ci sono i campi non mappati che servono per ottenere le liste sotto forma di stringa.

```
public virtual ICollection<Tematica> Tematiche { get; set; }
public virtual ICollection<Esercizio> Esercizi { get; set; }

[NotMapped]
public string TematicheList => string.Join(", ", Tematiche);

[NotMapped]
public string EserciziList => string.Join(", ", Esercizi);
```

### 3.2.1.4 Tematica

Il modello della Tematica contiene il nome (ad esempio “MVC”) e il relativo Modulo.

```
public Tematica(string nome, Modulo modulo)
{
    Nome = nome;
    ModuloId = modulo.Id;
}
```

### 3.2.1.5 Anno

La classe Anno definisce l'annata (ad esempio “2019/2020”) e la lista delle relative classi.

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

public virtual ICollection<Classe> Classi { get; set; }

public Anno(string annata) => Annata = annata;
```

### 3.2.1.6 Esercizio

L'entità Esercizio rappresenta il titolo, il testo (che conterrà anche l'immagine in quanto è tutto salvato in byte) e i relativi modulo e esercizi della prova.

```
public virtual ICollection<EsercizioProva> EserciziProva { get; set; }

public Esercizio(string titolo, string testo, Modulo modulo)
{
    Titolo = titolo;
    Testo = testo;
    ModuloId = modulo.Id;
}

public override string ToString() => Titolo;
```

### 3.2.1.7 EsercizioProva

La classe EsercizioProva è un'estensione di Esercizio ma rappresenta essenzialmente gli esercizi che vengono inseriti nelle prove, quindi si definisce il relativo esercizio e la prova.

```
public EsercizioProva(Esercizio esercizio, Prova prova)
{
    EsercizioId = esercizio.Id;
    ProvaId = prova.Id;
}

public override string ToString() => Esercizio.ToString();
```

### 3.2.1.8 Prova

Infine c'è il modello della Prova, il quale contiene il titolo, la data, la classe e gli esercizi (EsercizioProva).

```
public Prova(string titolo, DateTime data, Classe classe)
{
    Titolo = titolo;
    Data = data;
    ClasseId = classe.Id;
}
```

### 3.2.2 Services

La cartella Services contiene le interfacce e le classi che sono utili per interagire con il database. Infatti in queste classi sono presenti i metodi che permettono di inserire, modificare o eliminare i dati.

#### 3.2.2.1 IDataRepository

IDataRepository è l'interfaccia di base che implementa i metodi inerenti alle operazioni sul database. Bisogna definire il modello di dati su cui operare (T) e di conseguenza i metodi lavoreranno in base a quello.

```
public interface IDataRepository<T> where T : BaseEntity
{
    T Get(int id);

    IQueryable<T> Get();

    T Insert(T entity);

    void Update(T entity);

    void Delete(T entity);
}
```

#### 3.2.2.2 DbDataRepository

Questa è la classe che implementa l'interfaccia IDataRepository e viene usata principalmente come base per i repository degli altri modelli di dati implementando i metodi relativi al database. Essenzialmente si definisce il contesto del database e il modello di dati e in base a ciò vengono implementati i metodi per ottenere, inserire, modificare e eliminare i dati attraverso LINQ.

```
public abstract class DbDataRepository<C, T> : IDataRepository<T> where T : BaseEntity
                                                                    where C : DbContext, new()
{
    protected C context;
    protected DbDataRepository(C ctx) => context = ctx;

    public T Get(int id) => Get().SingleOrDefault(be => be.Id == id);

    public virtual IQueryable<T> Get() => context.Set<T>();

    public virtual T Insert(T entity)
    {
        context.Set<T>().Add(entity);
        context.SaveChanges();
        return entity;
    }

    public virtual void Update(T entity)
    {
        context.Entry(entity).State = EntityState.Modified;
        context.SaveChanges();
    }

    public virtual void Delete(T entity)
    {
        context.Set<T>().Remove(entity);
        context.SaveChanges();
    }
}
```

### 3.2.2.3 IClasseRepository

Interfaccia figlia di IDataRepository relativa al modello di dati della classe.

```
public interface IClasseRepository : IDataRepository<Classe> { }
```

### 3.2.2.4 ClasseDbRepository

Classe figlia di DbDataRepository relativa al modello di dati della classe. È presente il metodo che ritorna tutte le entità ordinate per l'anno.

```
public class ClasseDbRepository : DbDataRepository<AppDbContext, Classe>, IClasseRepository
{
    public ClasseDbRepository(AppDbContext ctx) : base(ctx) { }

    public override IQueryable<Classe> Get() => base.Get().OrderByDescending(s => s.Anno);
}
```

### 3.2.2.5 IModuloRepository

Interfaccia figlia di IDataRepository relativa al modello di dati del modulo.

```
public interface IModuloRepository : IDataRepository<Modulo> { }
```

### 3.2.2.6 ModuloDbRepository

Classe figlia di DbDataRepository relativa al modello di dati della classe. Viene implementato il metodo che ritorna tutti i moduli ordinati per nome.

```
public class ModuloDbRepository : DbDataRepository<AppDbContext, Modulo>, IModuloRepository
{
    public ModuloDbRepository(AppDbContext ctx) : base(ctx) { }

    public override IQueryable<Modulo> Get() => base.Get().OrderBy(s => s.Nome);
}
```

### 3.2.2.7 ITematicaRepository

Interfaccia figlia di IDataRepository relativa al modello di dati della tematica.

```
public interface ITematicaRepository : IDataRepository<Tematica> { }
```

### 3.2.2.8 TematicaDbRepository

Classe figlia di DbDataRepository relativa al modello di dati della tematica. Viene ereditato il metodo Get() in modo da ottenere tutte le tematiche ordinate per il nome del relativo modulo.

```
public class TematicaDbRepository : DbDataRepository<AppDbContext, Tematica>, ITematicaRepository
{
    public TematicaDbRepository(AppDbContext ctx) : base(ctx) { }

    public override IQueryable<Tematica> Get() => base.Get().OrderBy(s => s.Modulo.Nome);
}
```

### 3.2.2.9 IAnnoRepository

Interfaccia figlia di IDataRepository relativa al modello di dati dell'anno.

```
public interface IAnnoRepository : IDataRepository<Anno> { }
```

### 3.2.2.10 AnnoDbRepository

Classe figlia di DbDataRepository per il modello di dati dell'anno. C'è il metodo che ritorna tutte le entità ordinate per l'annata.

```
public class AnnoDbRepository : DbDataRepository<AppDbContext, Anno>, IAnnoRepository
{
    public AnnoDbRepository(AppDbContext ctx) : base(ctx) { }

    public override IQueryable<Anno> Get() => base.Get().OrderByDescending(s => s.Annata);
}
```

### 3.2.2.11 IEsercizioRepository

Interfaccia figlia di IDataRepository relativa al modello di dati dell'esercizio.

```
public interface IEsercizioRepository : IDataRepository<Esercizio> { }
```

### 3.2.2.12 EsercizioDbRepository

Classe figlia di DbDataRepository relativa al modello di dati dell'esercizio. È presente il metodo che ritorna gli esercizi ordinati per titolo.

```
public class EsercizioDbRepository : DbDataRepository<AppDbContext, Esercizio>, IEsercizioRepository
{
    public EsercizioDbRepository(AppDbContext ctx) : base(ctx) { }

    public override IQueryable<Esercizio> Get() => base.Get().OrderBy(s => s.Titolo);
}
```

### 3.2.2.13 IEsercizioProvaDbRepository

Interfaccia figlia di IDataRepository che implementa i metodi relativi al modello degli esercizi della prova.

```
public interface IEsercizioProvaRepository : IDataRepository<EsercizioProva> { }
```

### 3.2.2.14 EsercizioProvaDbRepository

DbDataRepository relativo al modello di dati dell'esercizio della prova.

```
public class EsercizioProvaDbRepository : DbDataRepository<AppDbContext, EsercizioProva>,
IEsercizioProvaRepository
{
    public EsercizioProvaDbRepository(AppDbContext ctx) : base(ctx) { }
}
```

### 3.2.2.15 IProvaRepository

Interfaccia figlia di IDataRepository relativa al modello di dati della prova.

```
public interface IProvaRepository : IDataRepository<Prova> { }
```

### 3.2.2.16 ProvaDbRepository

Classe figlia di DbDataRepository relativa al modello di dati della prova.

```
public class ProvaDbRepository : DbDataRepository<AppDbContext, Prova>, IProvaRepository
{
    public ProvaDbRepository(AppDbContext ctx) : base(ctx) { }

    public override IQueryable<Prova> Get() => base.Get().OrderBy(s => s.Data);
}
```

### 3.2.3 AppDbContext

AppDbContext è la classe utilizzata dai repository (Services) e che praticamente mette insieme i modelli di dati (Models) per impostare la base di dati configurando le raccolte delle entità e il percorso di memorizzazione SQLite.

```
public class AppDbContext : DbContext
{
    public DbSet<Classe> Classi { get; set; }

    public DbSet<Modulo> Moduli { get; set; }

    public DbSet<Tematica> Tematiche { get; set; }

    public DbSet<Anno> Anni { get; set; }

    public DbSet<Esercizio> Esercizi { get; set; }

    public DbSet<EsercizioProva> EserciziProva { get; set; }

    public DbSet<Prova> Prove { get; set; }

    public AppDbContext() : base() { }

    public AppDbContext(DbContextOptions<AppDbContext> options) : base(options) { }

    protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
    {
        if (!optionsBuilder.IsConfigured)
        {
            string dbPath =
                Environment.GetFolderPath(Environment.SpecialFolder.ApplicationData)
                + "\\GestioneEsercizi";
            Directory.CreateDirectory(dbPath);
            optionsBuilder.UseLazyLoadingProxies().UseSqlite(
                "Data Source=" + dbPath + "\\GestioneEsercizi.sqlite");
        }
    }
}
```

Le proprietà della classe (DbSet) rappresentano le tabelle della base di dati, mentre il metodo OnConfiguring imposta il percorso di memorizzazione prendendo come cartella di base quella di AppData.

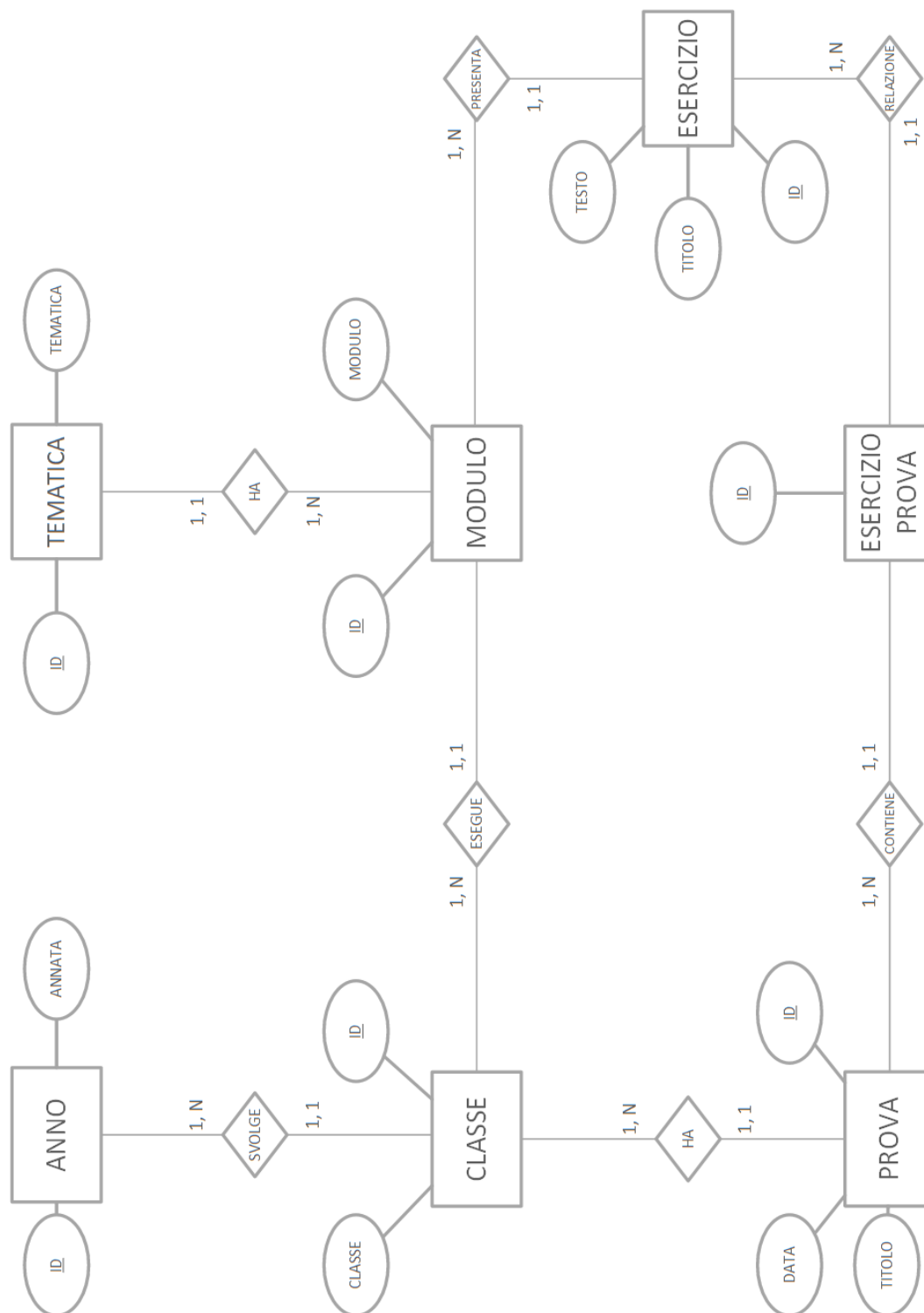
Successivamente è necessario creare il database tramite la console di gestione dei pacchetti. La prima cosa da fare è la creazione delle migrazioni, le quali definiscono i parametri del database tramite il DbContext, per poi aggiornare la base di dati. I comandi da eseguire sono i seguenti (Microsoft.EntityFrameworkCore.Tools):

- Add-Migration Initial
- Update-Database



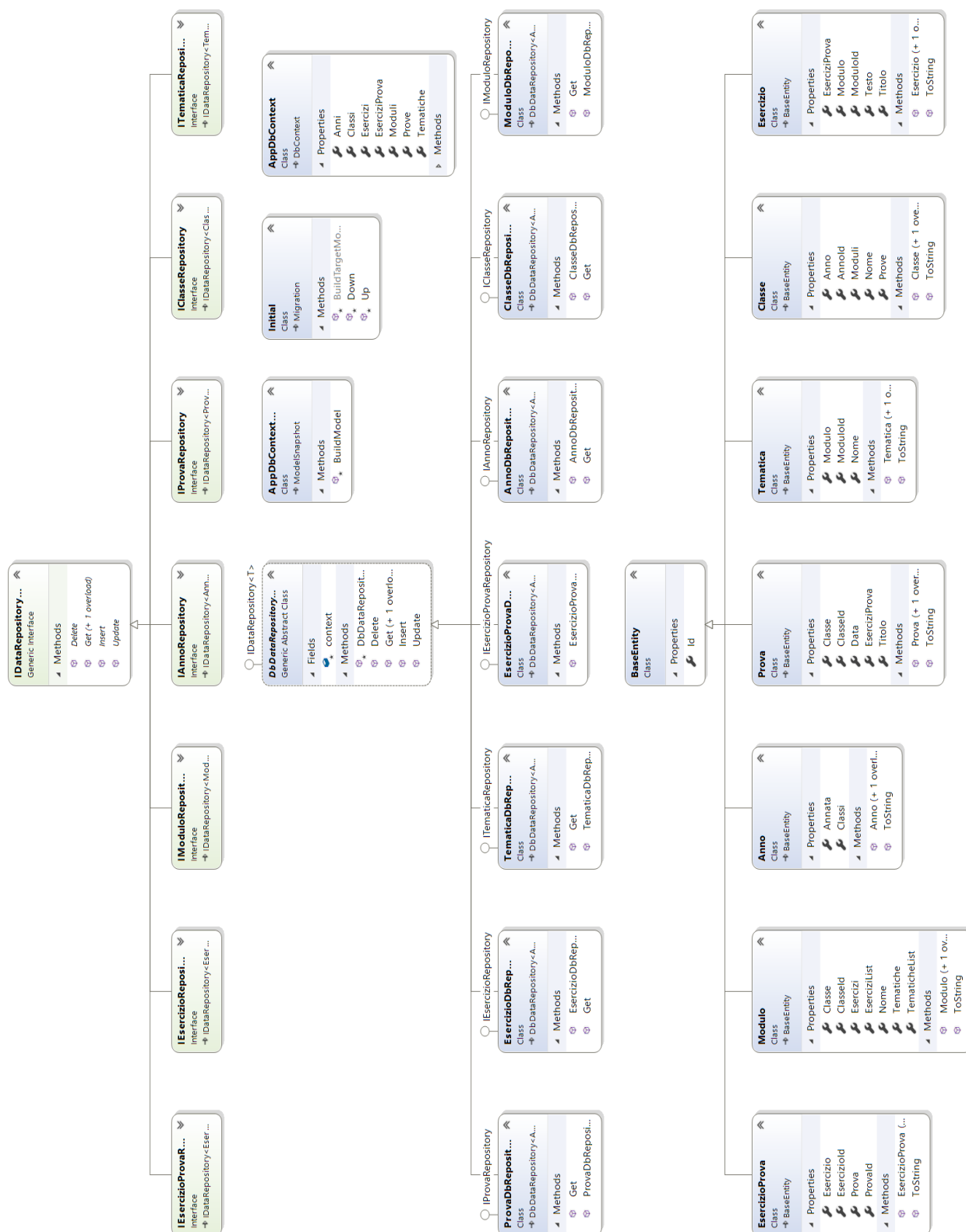
### 3.2.4 Database

Il database definitivo ha subito qualche modifica rispetto a quello inizialmente progettato.



Ciò che è cambiato è il fatto che essenzialmente le entità della classe e quella dell'anno sono state scambiate tra di loro, rendendo più chiara la struttura visto che i collegamenti che l'anno creava non erano ben definiti. Inoltre un'altra piccola modifica è la relazione tra Esercizio e EsercizioProva che è uno a molti.

### 3.2.5 Diagramma delle classi



### 3.3 App WPF

L'App WPF (.NET Core) è il progetto della soluzione Visual Studio che gestisce i dati della libreria di classi mostrandoli su un'interfaccia grafica user-friendly e permettendo di operare appunto sui dati.

#### 3.3.1 ViewModels

La cartella ViewModels contiene le classi che collegano i modelli di dati della libreria di classi con il progetto WPF permettendo di mostrarli nell'interfaccia grafica. Per fare ciò si usa la classe base BindableBase, la quale presenta i metodi principali per l'implementazione di un ViewModel.

```
/// Base class for all ViewModel classes in the application. Provides support for
/// property changes notification. Original implementation by Josh Smith.

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));
    }

    protected virtual void SetProperty<T>(ref T member,
                                           T value,
                                           [CallerMemberName] string propertyName = null)
    {
        if (object.Equals(member, value))
            return;
        member = value;
        OnPropertyChanged(propertyName);
    }

    protected void OnPropertyChanged<T>(Expression<Func<T>> propertyExpression)
    {
        var body = propertyExpression.Body as MemberExpression;
        var expression = body.Expression as ConstantExpression;
        PropertyChanged(expression.Value, new PropertyChangedEventArgs(body.Member.Name))
    }
}
```

Riassumendo qui vengono implementate le funzioni base per far funzionare un ViewModel, quindi il binding dei dati e l'impostazione dei dati delle proprietà dei modelli di dati.

### 3.3.1.1 MainViewModel

Il ViewModel principale ha il compito di impostare gli altri ViewModels e gestirli tramite i Commands e la proprietà CurrentViewModel. Qui è documentata la soluzione simbolica, visto che in realtà nel progetto ci sono molti ViewModels ma comunque il sistema funziona istanziando tutti i ViewModels e i Commands relativi alla schermata corrente (ad esempio se una schermata può portare ad altre due schermate allora conterrà due Commands).

```
public class MainViewModel : BindableBase
{
    private ViewModel vm;

    private AltroViewModel avm;

    public IDelegateCommand AltroCommand { get; set; }

    private BindableBase currentViewModel;

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

    public MainViewModel()
    {
        ViewModel = new ViewModel();
        AltroViewModel = new AltroViewModel();

        CurrentViewModel = ViewModel;

        AltroCommand = new DelegateCommand(OnAltro, CanAltro);

        // Ricezione dei messaggi da parte degli altri ViewModels.
        Messenger.Default.Register<BindableBase>(this, OnViewModelReceived);
    }

    public void OnViewModelReceived(BindableBase viewmodel) => CurrentViewModel = viewmodel;

    private void OnAltro(object obj) => CurrentViewModel = AltroViewModel;

    private bool CanAltro(object arg) => true;
}
```

In questo caso viene utilizzata la classe Messenger che permette l'invio e la ricezione di dati tra ViewModels. Viene fatto ciò perché è necessario basare tutto il sistema sul MainViewModel e la proprietà CurrentViewModel, in quanto le altre classi mandano il messaggio contenente il ViewModel da utilizzare.

### 3.3.1.2 BenvenutoViewModel

ViewModel relativo alla schermata di benvenuto, si impostano i comandi che portano ai ViewModels delle funzioni principali del programma (configurazione impostazioni di base, nuovo esercizio, creazione prova).

```
public BenvenutoViewModel()
{
    ImpostazioniBaseCommand = new DelegateCommand(OnImpostazioniBase, CanImpostazioniBase);
    EsercizioCommand = new DelegateCommand(OnEsercizio, CanEsercizio);
    ProvaCommand = new DelegateCommand(OnProva, CanProva);
}
```

Inoltre viene usato il Messenger per selezionare il ViewModel da far mostrare al MainViewModel.

```
private void OnProva(object obj)
=> Messenger.Default.Send<BindableBase>(new ProvaViewModel());
```

### 3.3.1.3 AboutViewModel

ViewModel relativo alla schermata di informazioni sul prodotto, si imposta il comando che porta alla schermata di benvenuto.

```
public IDelagateCommand BenvenutoCommand { get; set; }

public AboutViewModel()
=> BenvenutoCommand = new DelegateCommand(OnBenvenuto, CanBenvenuto);

private void OnBenvenuto(object obj)
=> Messenger.Default.Send<BindableBase>(new BenvenutoViewModel());

private bool CanBenvenuto(object arg) => true;
```

### 3.3.1.4 GuidaViewModel

Rappresenta la schermata con la guida del prodotto, viene solo impostato il ViewModel di benvenuto in quanto non servono altre informazioni.

```
public GuidaViewModel()
=> BenvenutoCommand = new DelegateCommand(OnBenvenuto, CanBenvenuto);
```

### 3.3.1.5 EsercizioListViewModel

ViewModel utile per mostrare la lista degli esercizi memorizzati.

```
public ObservableCollection<Esercizio> Esercizi { get; set; }

public EsercizioListViewModel()
{
    BenvenutoCommand = new DelegateCommand(OnBenvenuto, CanBenvenuto);
    EsercizioDbRepository repo = new EsercizioDbRepository(new AppDbContext());
    Esercizi = new ObservableCollection<Esercizio>(repo.Get());
}
```

### 3.3.1.6 ProvaListViewModel

Simile a EsercizioListViewModel però vengono mostrate le prove memorizzate anziché gli esercizi.

```
public ProvaListViewModel()
{
    BenvenutoCommand = new DelegateCommand(OnBenvenuto, CanBenvenuto);
    ProvaDbRepository repo = new ProvaDbRepository(new AppDbContext());
    Prove = new ObservableCollection<Prova>(repo.Get());
}
```

### 3.3.1.7 ImpostazioniBaseViewModel

ImpostazioniBaseViewModel ha il principale scopo di impostare le liste delle tre impostazioni di base. Inoltre sono presenti tre pulsanti che portano ai rispettivi ViewModels dell'oggetto che si vuole inserire.

```
public ImpostazioniBaseViewModel()
{
    BenvenutoCommand = new DelegateCommand(OnBenvenuto, CanBenvenuto);
    ClasseCommand = new DelegateCommand(OnClasse, CanClasse);
    ModuloCommand = new DelegateCommand(OnModulo, CanModulo);
    TematicaCommand = new DelegateCommand(OnTematica, CanTematica);

    AppDbContext ctx = new AppDbContext();
    ClasseDbRepository repoClasse = new ClasseDbRepository(ctx);
    ModuloDbRepository repoModulo = new ModuloDbRepository(ctx);
    TematicaDbRepository repoTematica = new TematicaDbRepository(ctx);

    Classi = new ObservableCollection<Classe>(repoClasse.Get());
    Moduli = new ObservableCollection<Modulo>(repoModulo.Get());
    Tematiche = new ObservableCollection<Tematica>(repoTematica.Get());
}
```

### 3.3.1.8 ClasseViewModel

Questo ViewModel è utile per implementare l'inserimento di una nuova classe.

```
public ClasseViewModel()
{
    SalvaCommand = new DelegateCommand(OnSalva, CanSalva);

    AnnoDbRepository repo = new AnnoDbRepository(new AppDbContext());
    Anni = new ObservableCollection<Anno>(repo.Get());
}
```

Prima di inserire i dati nel database si controlla che il campo non sia vuoto.

```
private void OnSalva(object obj)
{
    if (!string.IsNullOrEmpty(Nome) && Anno != null)
    {
        ClasseDbRepository repo = new ClasseDbRepository(new AppDbContext());
        // Aggiungo la classe
        repo.Insert(new Classe(Nome, Anno));
        OnBenvenuto(obj);
    }
}
```

### 3.3.1.9 ModuloViewModel

Funziona come ClasseViewModel ma per l'inserimento di un nuovo modulo.

```
public ModuloViewModel()
{
    ClasseDbRepository repo = new ClasseDbRepository(new AppDbContext());
    Classi = new ObservableCollection<Classe>(repo.Get());
}

private void OnSalva(object obj)
{
    if (!string.IsNullOrEmpty(Nome) && Classe != null)
    {
        ModuloDbRepository repo = new ModuloDbRepository(new AppDbContext());
        // Aggiungo il modulo
        repo.Insert(new Modulo(Nome, Classe));
        OnBenvenuto(obj);
    }
}
```

### 3.3.1.10 TematicaViewModel

ViewModel simile a ClasseViewModel ma viene inserita una tematica anziché una classe.

```
private void OnSalva(object obj)
{
    if (!string.IsNullOrEmpty(Nome) && Modulo != null)
    {
        TematicaDbRepository repo = new TematicaDbRepository(new AppDbContext());
        // Aggiungo la tematica
        repo.Insert(new Tematica(Nome, Modulo));
        OnBenvenuto(obj);
    }
}
```

### 3.3.1.11 EsercizioViewModel

Questo è il ViewModel che gestisce la creazione di un nuovo esercizio. In questo caso vengono gestiti i due comandi per l'annullamento e il salvataggio dell'operazione, viene creata la lista di moduli e vengono reperiti dalla View i dati utili per creare un nuovo esercizio (titolo, testo, immagine e modulo).

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

public string Testo { get; set; }

public BitmapImage Immagine { get; set; }

public Modulo Modulo { get; set; }

public EsercizioViewModel()
{
    BenvenutoCommand = new DelegateCommand(OnBenvenuto, CanBenvenuto);
    SalvaCommand = new DelegateCommand(OnSalva, CanSalva);
    ModuloDbRepository repo = new ModuloDbRepository(new AppDbContext());
    Moduli = new ObservableCollection<Modulo>(repo.Get());
}
```

Infine viene implementato il salvataggio dell'esercizio nel database.

```
private void OnSalva(object obj)
{
    if (!string.IsNullOrEmpty(Titolo)
        && !string.IsNullOrEmpty(Testo)
        && Modulo != null)
    {
        EsercizioDbRepository repo = new EsercizioDbRepository(new AppDbContext());
        // Aggiungo l'esercizio
        repo.Insert(new Esercizio(Titolo, Testo + Immagine, Modulo));
        OnBenvenuto(obj);
    }
}
```

### 3.3.1.12 ProvaViewModel

ViewModel che gestisce la creazione di una prova. In generale questa classe è simile al ViewModel dell'esercizio in quanto le operazioni da fare sono più o meno le stesse, però in questo caso bisogna anche gestire il salvataggio degli esercizi della prova (EsercizioProva).

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

public DateTime Data { get; set; }

public Classe Classe { get; set; }

public ProvaViewModel()
{
    BenvenutoCommand = new DelegateCommand(OnBenvenuto, CanBenvenuto);
    SalvaCommand = new DelegateCommand(OnSalva, CanSalva);

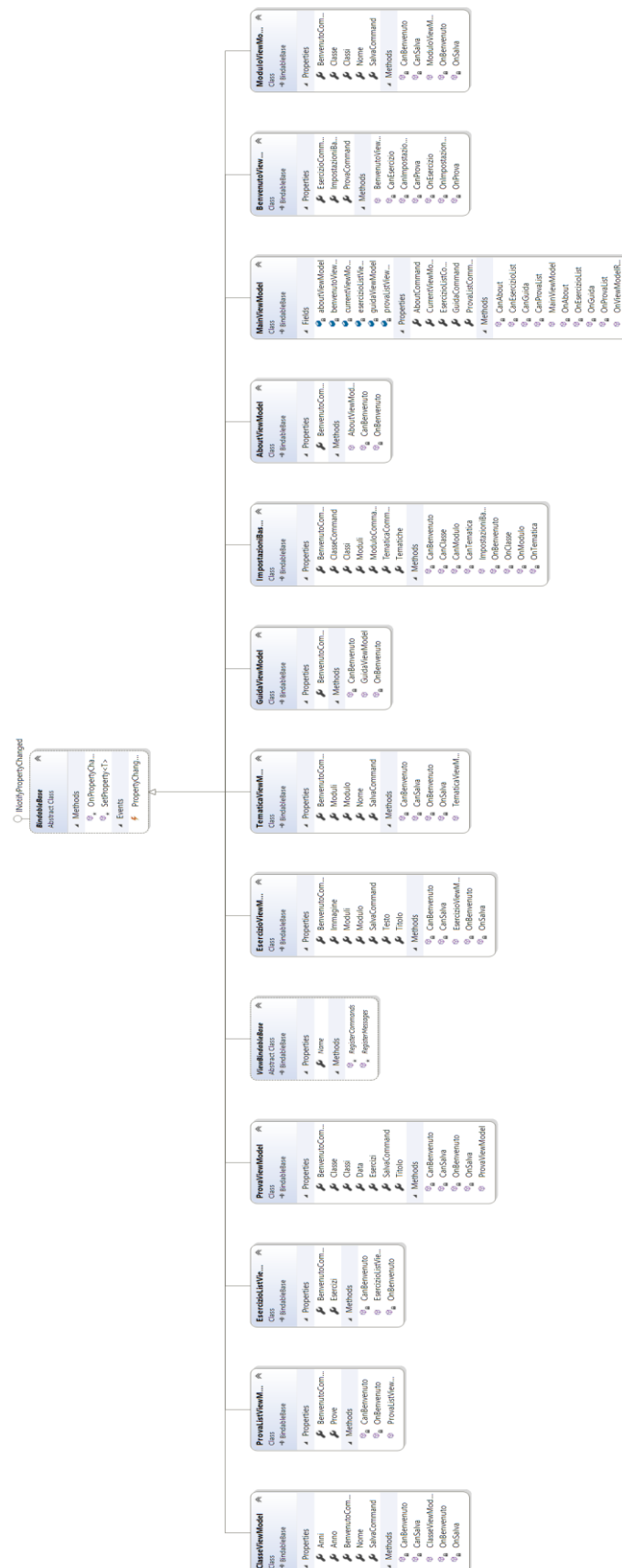
    ClasseDbRepository repoClasse = new ClasseDbRepository(new AppDbContext());
    EsercizioDbRepository repoEsercizio = new EsercizioDbRepository(new AppDbContext());
    Classi = new ObservableCollection<Classe>(repoClasse.Get());
    Esercizi = new ObservableCollection<Esercizio>(repoEsercizio.Get());
}
```

Dopodiché c'è il metodo che implementa il salvataggio della prova aggiungendo anche gli esercizi prova.

```
private void OnSalva(object obj)
{
    if (!string.IsNullOrEmpty(Titolo)
        && !string.IsNullOrEmpty(Data.ToString())
        && Classe != null)
    {
        ProvaDbRepository repoProva = new ProvaDbRepository(new AppDbContext());
        EsercizioProvaDbRepository rep = new EsercizioProvaDbRepository(new AppDbContext());
        List<Esercizio> esercizi = new List<Esercizio>(Esercizi);
        // Aggiungo la prova
        Prova prova = new Prova(Titolo, Data, Classe);
        repoProva.Insert(prova);
        // Aggiungo gli esercizi della prova
        foreach (Esercizio esercizio in esercizi)
            repoEseProva.Insert(new EsercizioProva(esercizio, prova));
        OnBenvenuto(obj);
    }
}
```



### 3.3.2 Diagramma delle classi



### 3.3.3 Views

La cartella Views rappresenta gli elementi dell'interfaccia utente dell'applicazione. La struttura di base funziona in modo che ogni View utilizzi il relativo ViewModel come base per i dati passandosi le informazioni tramite il Messenger.

#### 3.3.3.1 MainView

La MainView è l'interfaccia principale e ha il compito di rappresentare il MainViewModel, quindi viene selezionata la View da mostrare attraverso i Commands e il campo CurrentViewModel.

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

Il contenuto principale del controllo utente è il menu e lo spazio che mostra il ViewModel.

```
<Menu>
    <MenuItem Header="_File">
        <MenuItem Header="_Esercizi" Command="{Binding Path=EsercizioListCommand}"/>
        <MenuItem Header="_Prove" Command="{Binding Path=ProvaListCommand}"/>
        <MenuItem Header="_Esci" Click="Exit"/>
    </MenuItem>

    <MenuItem Header="_Info">
        <MenuItem Header="_Guida" Command="{Binding Path=GuidaCommand}"/>
        <MenuItem Header="_About" Command="{Binding Path=AboutCommand}"/>
    </MenuItem>
</Menu>

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

Sono inoltre presenti i riferimenti di tutti gli altri ViewModels e le rispettive Views nella sezione delle risorse.

```
<UserControl.Resources>
    <DataTemplate DataType="{x:Type viewmodel:AltroViewModel}">
        <local:BenvenutoView/>
    </DataTemplate>
```

Nel code-behind si implementa la funzione che chiude l'applicazione quando si clicca l'apposito pulsante dal menu.

```
private void Exit(object sender, RoutedEventArgs e)
=> System.Windows.Application.Current.Shutdown();
```

Questa View sarà l'unico contenuto presente nella finestra principale del progetto, cioè la MainWindow.

```
<Window x:Class="GestioneEsercizi.MainWindow"
        xmlns:view="clr-namespace:GestioneEsercizi.Views"
        Title="Gestione Esercizi"
        Height="500" Width="800" MinHeight="200" MinWidth="400">
    <Grid>
        <view:MainView/>
    </Grid>
</Window>
```

### 3.3.3.2 BenvenutoView

View che rappresenta la schermata principale di benvenuto.

```
<Button Content="Nuovo Esercizio" Command="{Binding Path=EsercizioCommand}"/>
```

Sono presenti i tre pulsanti che permettono di aprire le schermate relative alle operazioni principali, quindi la gestione delle impostazioni di base, la creazione di un nuovo esercizio e la creazione di una nuova prova. Questo elemento è uguale a come è stato progettato anche perché è molto semplice e non presenta particolari implementazioni.

### 3.3.3.3 AboutView

View relativa alla schermata delle informazioni del prodotto.

```
<StackPanel>

    <Image Source="/Images/GA.png"/>

    <TextBlock>
        <Hyperlink RequestNavigate="NavigateWebsite"
            NavigateUri="https://github.com/gabrialessi/GestioneEsercizi">
            GitHub - GestioneEsercizi
        </Hyperlink>
    </TextBlock>

    <TextBlock>
        <Hyperlink RequestNavigate="NavigateWebsite"
            NavigateUri="http://www.samtinfo.ch/i16alegab/index.html">
            Author - Gabriele Alessi
        </Hyperlink>
    </TextBlock>

    <Label Content="Version - 1.0" HorizontalAlignment="Center"/>

</StackPanel>
```

In questo caso viene mostrata un'immagine, il repository del progetto su GitHub, l'autore e la versione.



[GitHub - GestioneEsercizi](#)

[Author - Gabriele Alessi](#)

Version - 1.0

Indietro

### 3.3.3.4 GuidaView

View che rappresenta la schermata della guida del prodotto.

```
<Label FontSize="25" FontWeight="Bold" Content="Guida - Gestione Esercizi"/>
```

```
<Label FontSize="20" Content="Introduzione"/>
```

```
<Label FontSize="20" Content="Impostazioni di Base"/>
```

```
<Label FontSize="20" Content="Nuovo Esercizio"/>
```

```
<Label FontSize="20" Content="Creazione Prova"/>
```

In questo caso viene solo impostato del testo a scopo informativo (redatto e impostato nel code-behind).

## Guida - Gestione Esercizi

### Introduzione

Lo scopo di questo prodotto è quello di facilitare il lavoro dei docenti permettendo di creare degli esercizi e delle prove in un sistema automatizzato. L'applicazione presenta un menu dove è possibile vedere gli esercizi e le prove salvati nel database e altre informazioni. Questo programma è stato sviluppato in modo da essere semplice e veloce, tuttavia è necessario utilizzarlo nel modo giusto seguendo appunto il resto di questa guida.

### Impostazioni di Base

Le informazioni presenti nelle impostazioni di base sono un elemento fondamentale per il corretto funzionamento del prodotto. Infatti questi dati sono necessari per la creazione di un nuovo esercizio e di conseguenza per la generazione di una prova.

Indietro

### 3.3.3.5 EsercizioListView

Questa View ha il semplice scopo di mostrare la lista di esercizi presenti nel database.

```
<DataGrid AutoGenerateColumns="False" ItemsSource="{Binding Path=Esercizi}">
  <DataGrid.Columns>
    <DataGridTextColumn Header="Titolo" Binding="{Binding Path=Titolo}" />
    <DataGridTextColumn Header="Modulo" Binding="{Binding Path=Modulo}" />
  </DataGrid.Columns>
</DataGrid>
```

File Info	
Titolo	Modulo
Cantanti	Modulo 151
SalvataggioDati	Modulo 335
Indietro	

### 3.3.3.6 ProvaListView

ProvaListView funziona come EsercizioListView però mostra le prove anziché gli esercizi.

```
<DataGrid ItemsSource="{Binding Path=Prove}" AutoGenerateColumns="False">
  <DataGrid.Columns>
    <DataGridTextColumn Header="Titolo" Binding="{Binding Path=Titolo}" />
    <DataGridTextColumn Header="Data" Binding="{Binding Path=Data}" />
    <DataGridTextColumn Header="Classe" Binding="{Binding Path=Classe}" />
  </DataGrid.Columns>
</DataGrid>
```

### 3.3.3.7 ImpostazioniBaseView

View che mostra le impostazioni di base presenti nel sistema e permette di gestirle inserendo nuove entità. Le tabelle vengono istanziate in base ai singoli campi presenti nel ViewModel, così facendo le informazioni vengono mostrate correttamente.

```
<DataGrid ItemsSource="{Binding Path=Classi}" AutoGenerateColumns="False">
  <DataGrid.Columns>
    <DataGridTextColumn Header="Classe" Binding="{Binding Path=Nome}" />
    <DataGridTextColumn Header="Anno" Binding="{Binding Path=Anno}" />
  </DataGrid.Columns>
</DataGrid>
```

Quando si vuole aggiungere un nuovo elemento viene chiamata la rispettiva View che permette di effettuare l'inserimento basandosi sul sistema dei dati.

```
<Button Content="Nuova classe" Command="{Binding Path=ClasseCommand}"/>
```

Infine c'è il pulsante che porta alla schermata di benvenuto.

```
<Button Content="Indietro" Command="{Binding Path=BenvenutoCommand}" />
```

File Info

Classi			Moduli				Tematiche		
Classe	Anno		Nome	Classe	Tematiche	Ese	Nome	Modulo	
I4AA	2019/2020		Modulo 151	I4AA	PDO		PDO	Modulo 151	
I3AA	2019/2020		Modulo 223	I3AA	MVVM, SQLite, MVC		MVVM	Modulo 223	
I4AC	2019/2020		Modulo 300	I4AC	Linux NAS, Client MacOS		SQLite	Modulo 223	
							MVC	Modulo 223	
							Linux NAS	Modulo 300	
							Client MacOS	Modulo 300	
Nuova classe			Nuovo modulo				Nuova tematica		

Indietro

### 3.3.3.8 ClasseView

Questa View serve ad aggiungere una nuova classe nel sistema definendo il nome della classe e l'anno.

```
<TextBox Text="{Binding Path=Nome}"/>
```

```
<ComboBox ItemsSource="{Binding Path=Anni}" SelectedItem="{Binding Path=Anno}" />
```

L'entità Anno è l'unica che non necessita la definizione di altri elementi (FK) se si vuole aggiungerne una, quindi è stato implementato un algoritmo che aggiunge l'anno corrente nel database quando si avvia il programma (code-behind della MainWindow).

```
AnnoDbRepository repo = new AnnoDbRepository(new AppDbContext());
// Anno corrente
string anno = DateTime.Today.Year.ToString() + "/" + (DateTime.Today.Year + 1).ToString();
// Inserisco l'anno se non è presente
try
{
    if (repo.Get().FirstOrDefault().Annata != anno) repo.Insert(new Anno(anno));
}
catch (NullReferenceException)
{
    repo.Insert(new Anno(anno));
}
```

### 3.3.3.9 ModuloView

ModuloView funziona come ClasseView ma serve per aggiungere un modulo indicando nome e classe.

```
<TextBox Text="{Binding Path=Nome}"/>
```

```
<ComboBox ItemsSource="{Binding Path=Classi}" SelectedItem="{Binding Path=Classe}" />
```

### 3.3.3.10 TematicaView

Anche questa View è simile a ClasseView ed è utile per aggiungere una tematica.

```
<TextBox Text="{Binding Path=Nome}"/>
```

```
<ComboBox ItemsSource="{Binding Path=Moduli}" SelectedItem="{Binding Path=Modulo}" />
```

### 3.3.3.11 EsercizioView

Gli elementi presenti in questa View permettono di creare un nuovo esercizio. Vengono definite le impostazioni di base, il titolo, il testo e un'eventuale immagine. La seguente parte di XML è utile per creare del testo di suggerimento che viene mostrato quando il campo è vuoto e viene usato per il titolo e per il testo.

```
<TextBox x:Name="titolo" Text="{Binding Path=Titolo}" />
<TextBlock IsHitTestVisible="False" Text="Titolo esercizio...">
    <TextBlock.Style>
        <Style TargetType="{x:Type TextBlock}">
            <Setter Property="Visibility" Value="Collapsed"/>
            <Style.Triggers>
                <DataTrigger Binding="{Binding Text, ElementName=titolo}" Value="">
                    <Setter Property="Visibility" Value="Visible"/>
                </DataTrigger>
            </Style.Triggers>
        </Style>
    </TextBlock.Style>
</TextBlock>
```

Dopodiché c'è il menu a tendina che permette di selezionare il modulo relativo all'esercizio.

```
<ComboBox ItemsSource="{Binding Path=Moduli}" SelectedItem="{Binding Path=Modulo}"/>
```

Infine ci sono degli elementi che consentono di trascinare l'immagine o selezionarla tramite il file explorer.

```
<Image x:Name="immagine" Source="{Binding Path=Immagine}" />
<Button Content="Sfoglia..." Click="Sfoglia" />
<Label Content="Trascina l'immagine qui" AllowDrop="True" Drop="ImageDrop"/>
```

Poi ovviamente sono presenti i pulsanti “Annulla” e “Salva”.

```
<DockPanel>
    <Button Content="Annulla" Command="{Binding Path=BenvenutoCommand}" />
    <Button Content="Salva" Command="{Binding Path=SalvaCommand}"/>
</DockPanel>
```

The screenshot displays the EsercizioView user interface. At the top, there is a title input field labeled 'Titolo esercizio...' and a dropdown menu currently showing 'Modulo 151'. Below these, on the left, is a large text area labeled 'Testo esercizio...'. To the right of the text area is a rectangular placeholder for an image. Below the image placeholder are two buttons: 'Sfoglia...' (Browse...) and 'Trascina l'immagine' (Drag the image). At the bottom right of the window are two buttons: 'Annulla' (Cancel) and 'Salva' (Save).

Nella parte di C# della View vengono implementati i metodi relativi all'inserimento e al trascinamento dell'immagine. Nel primo caso viene usato un oggetto che rappresenta il file explorer al quale viene impostato un filtro in modo da mostrare solo le immagini.

```
private void Sfogliare(object sender, RoutedEventArgs e)
{
    // Creazione File Explorer
    Microsoft.Win32.OpenFileDialog dlg = new Microsoft.Win32.OpenFileDialog();
    // Impostazione estensioni files
    dlg.Filter = "Image Files | *.jpg; *.jpeg; *.png; *.gif";
    // Mostrare il file selezionato
    if (dlg.ShowDialog() == true) SetImage(dlg.FileName);
}
```

Invece per quanto riguarda il rilascio di un'immagine vengono utilizzato i parametri relativo metodo in modo da ottenere l'ultimo file trascinato.

```
private void ImageDrop(object sender, DragEventArgs e)
{
    if (e.Data.GetDataPresent(DataFormats.FileDrop))
    {
        string[] files = (string[])e.Data.GetData(DataFormats.FileDrop);
        SetImage(files[0]);
    }
}
```

Il metodo che imposta l'anteprima dell'immagine si presenta in questo modo.

```
private void SetImage(string filename)
=> immagine.Source = new BitmapImage(new Uri(filename, UriKind.Absolute));
```



### 3.3.3.12 ProvaView

La ProvaView è simile a EsercizioView visto che lo scopo è un più o meno lo stesso, cioè creare un nuovo elemento in base ai parametri presenti nel database. Anche in questo caso c'è la parte di inserimento del titolo dove è presente l'implementazione del placeholder.

```
<TextBox x:Name="titolo" Text="{Binding Path=Titolo}"/>
```

Poi c'è un campo dove si può selezionare la data esatta della prova.



```
<DatePicker SelectedDate="{Binding Path=Data}" />
```

Per definire la classe della nuova prova si utilizzano le entità presenti nel database con un menu a tendina.

```
<ComboBox ItemsSource="{Binding Path=Classi}" SelectedItem="{Binding Path=Classe}" />
```

Infine per scegliere gli esercizi da inserire viene usata una tabella in cui si selezionano quelli da inserire.

```
<DataGrid ItemsSource="{Binding Path=Esercizi}" AutoGenerateColumns="False">
    <DataGrid.Columns>
        <DataGridTextColumn Header="Titolo" Binding="{Binding Path=Titolo}" />
        <DataGridTextColumn Header="Modulo" Binding="{Binding Path=Modulo}" />
        <DataGridCheckBoxColumn Header="Inserire nella prova" />
    </DataGrid.Columns>
</DataGrid>
```

<input type="text" value="Titolo prova..."/>  <input type="text" value="01/01/0001"/>   <input type="text" value="I4AA"/> 	<table border="1"> <thead> <tr> <th>Titolo</th> <th>Modulo</th> <th>Inserire nella prova</th> </tr> </thead> <tbody> <tr> <td>Cantanti</td> <td>Modulo 151</td> <td><input type="checkbox"/></td> </tr> <tr> <td>SalvataggioDati</td> <td>Modulo 335</td> <td><input type="checkbox"/></td> </tr> </tbody> </table>	Titolo	Modulo	Inserire nella prova	Cantanti	Modulo 151	<input type="checkbox"/>	SalvataggioDati	Modulo 335	<input type="checkbox"/>
Titolo	Modulo	Inserire nella prova								
Cantanti	Modulo 151	<input type="checkbox"/>								
SalvataggioDati	Modulo 335	<input type="checkbox"/>								

## 4 Test

### 4.1 Protocollo di test

<b>Test Case:</b>	TC-001	<b>Nome:</b>	Impostazioni di base
<b>Riferimento:</b>	REQ-001		
<b>Descrizione:</b>	Il programma deve essere strutturato in modo che si possano definire delle impostazioni di base.		
<b>Prerequisiti:</b>	Implementazione di una base di dati che rispetti la struttura del progetto.		
<b>Procedura:</b>	Aprire l'applicazione e scegliere l'opzione "Impostazioni di Base".		
<b>Risultati attesi:</b>	Le impostazioni di base salvate vengono mostrate correttamente nelle rispettive tabelle e c'è l'opzione di inserirne delle nuove.		

<b>Test Case:</b>	TC-002	<b>Nome:</b>	Esercizi
<b>Riferimento:</b>	REQ-001		
<b>Descrizione:</b>	Si devono poter vedere le informazioni principali per poi creare gli esercizi.		
<b>Prerequisiti:</b>	Devono essere presenti delle impostazioni di base nel database per avere un sistema coerente.		
<b>Procedura:</b>	Aprire l'applicazione e scegliere l'opzione "Nuovo Esercizio".		
<b>Risultati attesi:</b>	Deve esserci il necessario per definire un nuovo esercizio. Gli esercizi saranno visibili nella lista raggiungibile dal menu "Esercizi".		

<b>Test Case:</b>	TC-003	<b>Nome:</b>	Prove
<b>Riferimento:</b>	REQ-001		
<b>Descrizione:</b>	Deve essere preparato un documento con gli esercizi scelti.		
<b>Prerequisiti:</b>	Devono essere presenti delle impostazioni di base e degli esercizi nel database per avere un sistema coerente.		
<b>Procedura:</b>	Aprire l'applicazione e scegliere l'opzione "Creazione Prova".		
<b>Risultati attesi:</b>	Deve esserci il necessario per creare una nuova prova. Le prove saranno visibili nella lista raggiungibile dal menu "Prove".		

<b>Test Case:</b>	TC-004	<b>Nome:</b>	Inserimento classe
<b>Riferimento:</b>	REQ-002		
<b>Descrizione:</b>	Si deve poter inserire una nuova classe nel sistema.		
<b>Prerequisiti:</b>	Il sistema deve impostare automaticamente l'entità che definisce l'anno corrente nella base di dati per poter inserire una nuova classe.		
<b>Procedura:</b>	<ul style="list-style-type: none"> <li>• Aprire l'applicazione e scegliere l'opzione "Impostazioni di Base".</li> <li>• Cliccare sul pulsante "Nuova classe".</li> <li>• Digitare il nome della classe e scegliere l'anno dal menu.</li> <li>• Cliccare "Salva". Se i campi sono vuoti il salvataggio non viene effettuato.</li> </ul>		
<b>Risultati attesi:</b>	La classe appena inserita sarà visibile nella griglia.		

<b>Test Case:</b>	TC-005	<b>Nome:</b>	Inserimento modulo
<b>Riferimento:</b>	REQ-002		
<b>Descrizione:</b>	Si deve poter inserire un nuovo modulo nel sistema.		
<b>Prerequisiti:</b>	Nel sistema devono essere presenti delle classi per poter definire un nuovo modulo.		
<b>Procedura:</b>	<ul style="list-style-type: none"> <li>• Aprire l'applicazione e scegliere l'opzione "Impostazioni di Base".</li> <li>• Cliccare sul pulsante "Nuovo modulo".</li> <li>• Digitare il nome del modulo e scegliere la classe dal menu.</li> <li>• Cliccare "Salva". Se i campi sono vuoti il salvataggio non viene effettuato.</li> </ul>		
<b>Risultati attesi:</b>	Il modulo appena inserito sarà visibile nella griglia.		

<b>Test Case:</b>	TC-006	<b>Nome:</b>	Inserimento tematica
<b>Riferimento:</b>	REQ-002		
<b>Descrizione:</b>	Si deve poter inserire una nuova tematica nel sistema.		
<b>Prerequisiti:</b>	Nel sistema devono essere presenti dei moduli per poter definire una nuova tematica.		
<b>Procedura:</b>	<ul style="list-style-type: none"> <li>• Aprire l'applicazione e scegliere l'opzione "Impostazioni di Base".</li> <li>• Cliccare sul pulsante "Nuova tematica".</li> <li>• Digitare il nome della tematica e scegliere il modulo dal menu.</li> <li>• Cliccare "Salva". Se i campi sono vuoti il salvataggio non viene effettuato.</li> </ul>		
<b>Risultati attesi:</b>	La tematica appena inserita sarà visibile nella griglia.		

<b>Test Case:</b>	TC-007	<b>Nome:</b>	Nuovo esercizio
<b>Riferimento:</b>	REQ-003		
<b>Descrizione:</b>	Si deve poter inserire un nuovo esercizio nel sistema.		
<b>Prerequisiti:</b>	Nella base di dati devono essere presenti delle impostazioni di base per avere un sistema coerente.		
<b>Procedura:</b>	<ul style="list-style-type: none"> <li>• Aprire l'applicazione e scegliere l'opzione "Nuovo Esercizio".</li> <li>• Digitare il titolo dell'esercizio.</li> <li>• Scegliere il modulo relativo all'esercizio dal menu a tendina.</li> <li>• Redigere il testo che descrive l'esercizio.</li> <li>• Scegliere o trascinare nell'apposita area un'eventuale immagine.</li> <li>• Cliccare "Salva". Se i campi sono vuoti il salvataggio non viene effettuato.</li> </ul>		
<b>Risultati attesi:</b>	L'esercizio appena inserito sarà visualizzabile dal menu "File" → "Esercizi".		

<b>Test Case:</b>	TC-008	<b>Nome:</b>	Creazione prova
<b>Riferimento:</b>	REQ-003		
<b>Descrizione:</b>	Si deve poter creare una nuova prova inserendola nel sistema per poi generare un documento che la rappresenta.		
<b>Prerequisiti:</b>	Nella base di dati devono essere presenti delle impostazioni di base e degli esercizi per avere un sistema coerente.		
<b>Procedura:</b>	<ul style="list-style-type: none"> <li>• Aprire l'applicazione e scegliere l'opzione "Creazione Prova".</li> <li>• Digitare il titolo della prova.</li> <li>• Scegliere o inserire la data della prova.</li> <li>• Definire la classe relativa alla prova dal menu a tendina.</li> <li>• Scegliere gli esercizi da inserire nella prova dalla tabella.</li> <li>• Cliccare "Salva". Se i campi sono vuoti il salvataggio non viene effettuato.</li> <li>• Dalla lista delle prove generare il documento che la rappresenta.</li> </ul>		
<b>Risultati attesi:</b>	La prova appena inserita sarà visualizzabile dal menu "File" → "Prove" e si potrà generare il documento.		

## 4.2 Risultati test

Test	Risultato	Note
TC-001	OK	-
TC-002	OK	-
TC-003	OK	-
TC-004	OK	-
TC-005	OK	-
TC-006	OK	-
TC-007	FALLITO	L'immagine non viene inserita
TC-008	FALLITO	Gli esercizi non vengono inseriti e non è possibile generare il file

## 4.3 Mancanze/limitazioni conosciute

I sottocapitoli di questa sezione spiegano le eventuali mancanze e le limitazioni riscontrate nelle parti principali del prodotto durante il protocollo di test.

### 4.3.1 Impostazioni di base

Per quanto riguarda le impostazioni di base ritengo non ci siano particolari implementazioni incomplete secondo i requisiti. Tuttavia nella schermata in cui vengono mostrate le impostazioni di base presenti nel database non è implementata la gestione, quindi non è possibile eliminare o modificare i dati.

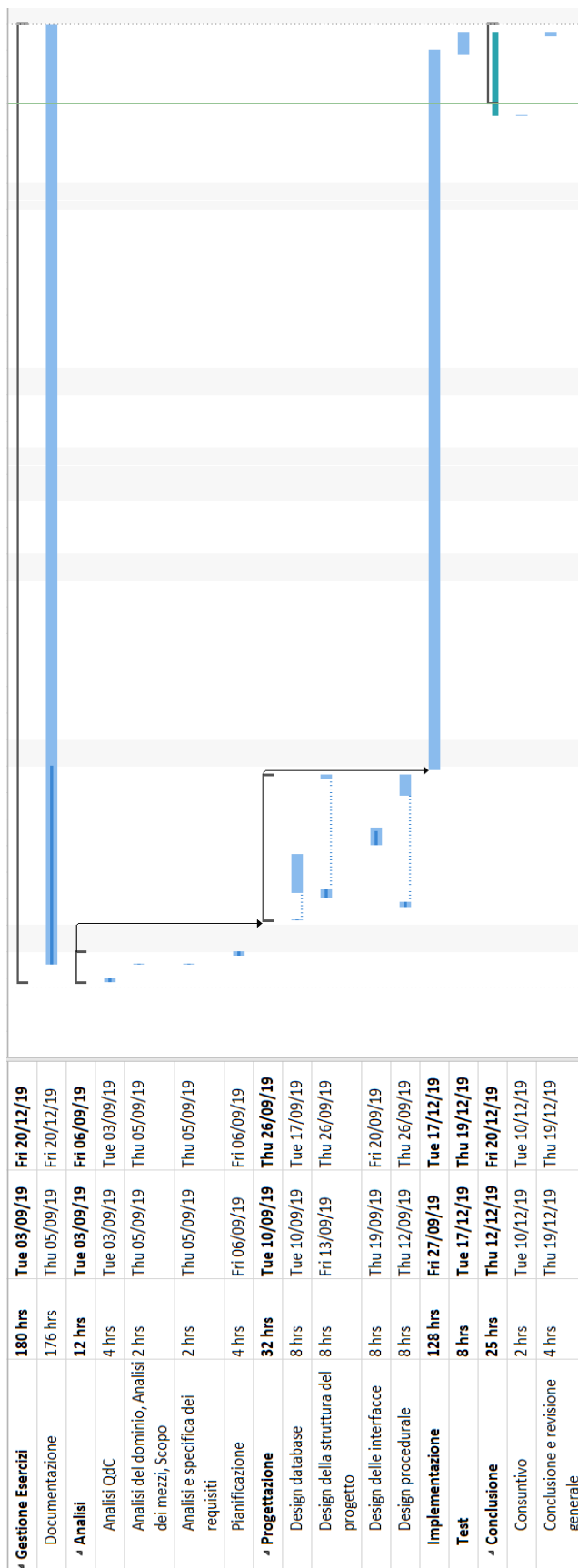
### 4.3.2 Esercizio

L'unica limitazione presente nella parte di definizione di un esercizio è il fatto che l'immagine inserita non viene salvata nella base di dati del sistema. Un'altra possibile limitazione può essere il fatto che le classi con lo stesso nome possono essere non riconoscibili. Per questo si potrebbe scrivere accanto ad ogni istanza di una classe mostrata nelle Views l'anno che la rappresenta.

### 4.3.3 Creazione Prova

Quando si vuole creare una prova viene mostrata la lista degli esercizi, però non è possibile scegliere quali inserire nel documento e alla fine non è implementata l'opzione che permette di generare quest'ultimo.

**5 Consuntivo**



## 6 Conclusioni

L'obiettivo del progetto è stato quello di sviluppare un prodotto che avrebbe aiutato soprattutto i docenti della sezione informatica del CPT a gestire gli esercizi per gli allievi e eventualmente generare delle prove. Purtroppo ritengo che l'applicazione sia ora come ora poco utile a causa di alcune implementazioni incomplete e/o non sviluppate in maniera ottimale. Tuttavia la mia soluzione può essere ripresa senza problemi per poter migliorare i moduli e quindi avere un buon prodotto da offrire ai docenti. Comunque questo lavoro è stato in generale utile per la mia formazione aiutandomi a capire meglio alcuni tipi di implementazione (Models, Views e ViewModels) e sono anche contento di come ho costruito gli elementi non inerenti allo sviluppo (documentazione, diari, diagrammi).

### 6.1 Sviluppi futuri

Oltre a quelli elencati nel capitolo di mancanze e limitazioni, gli sviluppi futuri del prodotto possono essere molti, tra cui:

- Implementazione di impostazioni personalizzate per un'esperienza utente più ricca.
- Miglioramento dell'interfaccia grafica sviluppando degli stili e portando più contenuto soprattutto nelle parti di creazione esercizi e prove.  
Ad esempio la visualizzazione grafica degli esercizi e l'anteprima delle prove.
- Connessione dell'applicazione al sistema del CPT permettendo ai docenti di identificarsi e condividere i contenuti.
- Connessione del programma con gli allievi per semplificare la condivisione di risorse e documenti con i docenti.

### 6.2 Considerazioni personali

Il motivo principale per cui questo progetto mi è stato utile è per il fatto che ho finalmente capito bene come funziona il pattern MVVM e come utilizzarlo al meglio. In generale non è stato un lavoro pesante anche se durante lo sviluppo di alcuni moduli ho riscontrato vari problemi su cui ho dovuto lavorare molto.

Avrei preferito che il docente responsabile fosse stato più disponibile (soprattutto all'inizio e alla fine del progetto), tuttavia quando sono andato incontro a dei problemi di implementazione è quasi sempre riuscito a risolverli spiegando i procedimenti da adottare.

Personalmente sono soddisfatto del lavoro svolto per quanto riguarda la gestione del progetto e della documentazione. Purtroppo il prodotto finale non è il risultato aspettato ma ciò mi aiuterà portandomi a migliorare su aspetti come la metodologia di lavoro e la gestione del tempo.

## 7 Sitografia

- <https://fmoralesdev.com/2019/05/16/generate-class-diagram-vs2019-net-core/>, Generate a class diagram in VS2019, 10.09.2019
- <https://www.c-sharpcorner.com/article/difference-between-net-framework-and-net-core/>, Difference Between .NET Framework and .NET Core, 27.09.2019
- <https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/language-specification/documentation-comments>, Documentation comments, 03.10.2019
- <https://stackoverflow.com/questions/9936796/create-a-menu-bar-in-wpf>, Create a menu Bar in WPF, 15.10.2019
- <https://stackoverflow.com/questions/2820357/how-do-i-exit-a-wpf-application-programmatically>, Exit a WPF application programmatically, 15.10.2019
- <https://stackoverflow.com/questions/11133947/how-do-i-open-a-second-window-from-the-first-window-in-wpf>, Open a second window from the first window in WPF, 15.10.2019
- <https://stackoverflow.com/questions/3419909/how-do-i-lock-a-wpf-window-so-it-can-not-be-moved-resized-minimized-maximized?rq=1>, Lock WPF Window, 15.10.2019
- <https://stackoverflow.com/questions/24485197/relative-path-not-working-while-accessing-a-sqlite-database-through-c-sharp>, Relative Path, 25.10.2019
- <https://stackoverflow.com/questions/833943/watermark-hint-text-placeholder-textbox>, Hint text, 07.11.2019
- <https://stackoverflow.com/questions/10315188/open-file-dialog-and-select-a-file-using-wpf-controls-and-c-sharp>, Open file dialog and select a file using WPF controls and C#, 07.11.2019
- <https://stackoverflow.com/questions/50180326/how-to-make-lazy-loading-work-with-ef-core-2-1-0-and-proxies/52432651>, Lazy Loading with Proxies EF Core, 12.11.2019
- <https://docs.microsoft.com/en-us/ef/core/modeling/included-properties>, Including & Excluding Properties, 19.11.2019
- <https://stackoverflow.com/questions/14959824/convert-list-into-comma-separated-string/29575082>, Convert List into Comma-Separated String, 19.11.2019
- <https://www.wpf-tutorial.com/datagrid-control/custom-columns/>, DataGrid columns, 21.11.2019
- <https://stackoverflow.com/questions/3787137/change-image-source-in-code-behind-wpf>, Change image source in code behind – Wpf, 10.12.2019
- <https://www.geeksforgeeks.org/c-sharp-isnullorempty-method/>, C# | IsNullOrEmpty() Method, 13.12.2019



## **8 Allegati**

---

- Diari
- Quaderno dei Compiti