Gestione degli esercizi delle prove svolte durante la professione di docente

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 20

3.2.1.4 Tematica 20

3.2.1.5 Anno 21

3.2.1.6 Esercizio 21

3.2.1.7 EsercizioProva 22

3.2.1.8 Prova 22

3.2.2 Services 23

3.2.2.1 IDataRepository 23

3.2.2.2 DbDataRepository 24

3.2.2.3 IClasseRepository 25

3.2.2.4 ClasseDbRepository 25

3.2.2.5 IModuloRepository 26

3.2.2.6 ModuloDbRepository 26

3.2.2.7 ITematicaRepository 27

3.2.2.8 TematicaDbRepository 27

3.2.2.9 IAnnoRepository 28

3.2.2.10 AnnoDbRepository 28

3.2.2.11 IEsercizioRepository 29

3.2.2.12 EsercizioDbRepository 29

3.2.2.13 IEsercizioProvaRepository 30

3.2.2.14 EsercizioProvaDbRepository 30

3.2.2.15 IProvaRepository 31

3.2.2.16 ProvaDbRepository 31

3.2.3 AppDbContext 32

3.3 App WPF 33

3.3.1 ViewModels 33

3.3.1.1 BenvenutoViewModel 34

3.3.1.2 AboutViewModel 34

3.3.1.3 ImpostazioniBaseViewModel 34

3.3.1.4 AnnoViewModel 35

3.3.1.5 EsercizioViewModel 35

3.3.1.6 EsercizioProvaViewModel 35

3.3.1.7 ProvaViewModel 36

3.3.1.8 MainViewModel 37

3.3.2 Views 38

3.3.2.1 BenvenutoView 38

3.3.2.2 AboutView 38

4 Test 39

4.1 Protocollo di test 39

4.2 Risultati test 39

4.3 Mancanze/limitazioni conosciute 39

5 Consuntivo 40

6 Conclusioni 41

6.1 Sviluppi futuri 41

6.2 Considerazioni personali 41

7 Bibliografia 41

7.1 Sitografia 41

8 Allegati 41

# Introduzione

## 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

## 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.

## 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

## 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.

## Analisi e specifica dei requisiti

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

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

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

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

## Use case



## Pianificazione

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

## Analisi dei mezzi

### Software

I software utilizzati per la realizzazione di questo progetto sono:

* Microsoft Word 2016   
  Realizzazione della documentazione.
* GitHub Desktop 2.1.3  
  Gestione del sistema di versioning del progetto.
* Visual Studio Code 1.37.1  
  Realizzazione dei diari giornalieri e gestione dei vari documenti di testo.
* Visual Studio 2019  
  Sviluppo del programma.
* Microsoft Project 2016  
  Realizzazione del diagramma di Gantt.
* Microsoft Visio 2016   
  Realizzazione di vari schemi e diagrammi.

### Hardware

Personal Computer:

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

# Progettazione

## 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).



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

## 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.

## Design delle interfacce

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

### 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”.

### 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.

### 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.



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.

### 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.

## 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.

# Implementazione

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

## 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).

## 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.

### 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.

#### BaseEntity

BaseEntity è la superclasse principale ed è la base di ogni altra entità, infatti essa contiene solo l’Id, che è un campo comune a ogni oggetto.

/// <summary>

/// Base di ogni altro modello di dati.

/// </summary>

public class BaseEntity

{

/// <summary>

/// Numero identificativo dell'entità.

/// </summary>

public int Id { get; set; }

}

#### Classe

L’entità Classe contiene il nome della stessa (ad esempio I4AA) e l’anno che sta svolgendo (ad esempio 2019/2020).

/// <summary>

/// Modello di dati della classe.

/// </summary>

public class Classe : BaseEntity

{

/// <summary>

/// Nome della classe (esempio: "I4AA").

/// </summary>

public string Nome { get; set; }

/// <summary>

/// Anno della classe.

/// </summary>

public Anno Anno { get; set; }

}

#### Modulo

La classe Modulo presenta i campi che descrivono il nome (Modulo x), l’anno, le tematiche e gli esercizi.

/// <summary>

/// Modello di dati del modulo.

/// </summary>

public class Modulo : BaseEntity

{

/// <summary>

/// Nome del modulo (esempio: "Modulo 151").

/// </summary>

public string Nome { get; set; }

/// <summary>

/// Anno del modulo.

/// </summary>

public Anno Anno { get; set; }

/// <summary>

/// Insieme delle tematiche del modulo.

/// </summary>

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

/// <summary>

/// Insieme degli esercizi del modulo.

/// </summary>

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

}

#### Tematica

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

/// <summary>

/// Modello di dati della tematica.

/// </summary>

public class Tematica : BaseEntity

{

/// <summary>

/// Nome della tematica.

/// </summary>

public string Nome { get; set; }

/// <summary>

/// Modulo relativo alla tematica.

/// </summary>

public Modulo Modulo { get; set; }

}

#### Anno

La classe Anno è principalmente utilizzata per collegare una classe e un modulo con una prova. Fondamentalmente la classe include l’annata e gli insiemi di classi, moduli e prove.

/// <summary>

/// Modello di dati dell'anno.

/// </summary>

public class Anno : BaseEntity

{

/// <summary>

/// Annata (esempio: "2019/2020").

/// </summary>

public string Annata { get; set; }

/// <summary>

/// Insieme delle classi dell'anno.

/// </summary>

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

/// <summary>

/// Insieme dei moduli dell'anno.

/// </summary>

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

/// <summary>

/// Insieme delle prove dell'anno.

/// </summary>

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

}

#### 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 esercizio della prova.

/// <summary>

/// Modello di dati dell'esercizio.

/// </summary>

public class Esercizio : BaseEntity

{

/// <summary>

/// Titolo dell'esercizio.

/// </summary>

public string Titolo { get; set; }

/// <summary>

/// Testo dell'esercizio (include l'immagine).

/// </summary>

public string Testo { get; set; }

/// <summary>

/// Modulo relativo all'esercizio.

/// </summary>

public Modulo Modulo { get; set; }

/// <summary>

/// Esercizi della prova relativi all'esercizio.

/// </summary>

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

}

#### 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.

/// <summary>

/// Modello di dati dell'esercizio della prova.

/// </summary>

public class EsercizioProva : Esercizio

{

/// <summary>

/// Esercizio relativo all'esercizio della prova.

/// </summary>

public Esercizio Esercizio { get; set; }

/// <summary>

/// Prova relativa all'esercizio della prova.

/// </summary>

public Prova Prova { get; set; }

}

#### Prova

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

/// <summary>

/// Modello di dati della prova.

/// </summary>

public class Prova : BaseEntity

{

/// <summary>

/// Titolo della prova.

/// </summary>

public string Titolo { get; set; }

/// <summary>

/// Data della prova.

/// </summary>

public DateTime Data { get; set; }

/// <summary>

/// Anno della prova.

/// </summary>

public Anno Anno { get; set; }

/// <summary>

/// Insieme degli esercizi della prova.

/// </summary>

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

}

### 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.

#### IDataRepository

IDataRepository è l’interfaccia di base che implementa i metodi inerenti alle operazioni sul database.

/// <summary>

/// Interfaccia di base che implementa i metodi relativi alle

/// operazioni sui dati nel database.

/// </summary>

/// <typeparam name="T">Modello di dati di riferimento.</typeparam>

public interface IDataRepository<T> where T : BaseEntity

{

/// <summary>

/// Ritorna un'entità in base all'id passato.

/// </summary>

/// <param name="id">Id dell'entità da ricavare.</param>

/// <returns>Entità in base all'id passato.</returns>

T Get(int id);

/// <summary>

/// Ritorna tutte le entità.

/// </summary>

/// <returns>Tutte le entità del modello di dati in questione.</returns>

IQueryable<T> Get();

/// <summary>

/// Inserisce una nuova entità.

/// </summary>

/// <param name="entity">Entità da inserire.</param>

/// <returns>Entità appena inserita.</returns>

T Insert(T entity);

/// <summary>

/// Modifica un'entità.

/// </summary>

/// <param name="entity">Entità da modificare.</param>

void Update(T entity);

/// <summary>

/// Elimina un'entità.

/// </summary>

/// <param name="entity">Entità da eliminare.</param>

void Delete(T entity);

}

#### 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.

/// <summary>

/// Deposito di dati di base che implementa i metodi relativi al database.

/// </summary>

/// <typeparam name="C">Contesto di dati del database.</typeparam>

/// <typeparam name="T">Modello di dati di riferimento.</typeparam>

public abstract class DbDataRepository<C, T> : IDataRepository<T> where T : BaseEntity

where C : DbContext, new()

{

protected C context;

/// <summary>

/// Metodo costruttore, si definisce il contesto di dati del database.

/// </summary>

/// <param name="ctx">Contesto di dati del database.</param>

protected DbDataRepository(C ctx)

{

context = ctx;

}

public T Get(int id)

{

return Get().SingleOrDefault(be => be.Id == id);

}

public virtual IQueryable<T> Get()

{

return 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();

}

public virtual IQueryable<T> Where(Expression<Func<T, bool>> predicate)

{

return context.Set<T>().Where(predicate);

}

}

#### IClasseRepository

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

/// <summary>

/// Interfaccia che implementa i metodi relativi al database

/// sul modello di dati della classe.

/// </summary>

public interface IClasseRepository : IDataRepository<Classe>

{

}

#### ClasseDbRepository

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

/// <summary>

/// Deposito dati relativo al modello di dati della classe.

/// </summary>

public class ClasseDbRepository : DbDataRepository<AppDbContext, Classe>, IClasseRepository

{

/// <summary>

/// Metodo costruttore, si definisce il contesto di dati.

/// </summary>

/// <param name="ctx">Contesto di dati del database.</param>

public ClasseDbRepository(AppDbContext ctx) : base(ctx)

{

}

/// <summary>

/// Ritorna tutte le entità ordinate per nome.

/// </summary>

/// <returns>Le entità ordinate per nome.</returns>

public override IQueryable<Classe> Get()

{

return base.Get().OrderBy(s => s.Nome);

}

}

#### IModuloRepository

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

/// <summary>

/// Interfaccia che implementa i metodi relativi al database

/// sul modello di dati del modulo.

/// </summary>

public interface IModuloRepository : IDataRepository<Modulo>

{

}

#### ModuloDbRepository

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

/// <summary>

/// Deposito dati relativo al modello di dati del modulo.

/// </summary>

public class ModuloDbRepository : DbDataRepository<AppDbContext, Modulo>, IModuloRepository

{

/// <summary>

/// Metodo costruttore, si definisce il contesto di dati.

/// </summary>

/// <param name="ctx">Contesto di dati del database.</param>

public ModuloDbRepository(AppDbContext ctx) : base(ctx)

{

}

/// <summary>

/// Ritorna tutte le entità ordinate per nome.

/// </summary>

/// <returns>Le entità ordinate per nome.</returns>

public override IQueryable<Modulo> Get()

{

return base.Get().OrderBy(s => s.Nome);

}

}

#### ITematicaRepository

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

/// <summary>

/// Interfaccia che implementa i metodi relativi al database

/// sul modello di dati della tematica.

/// </summary>

public interface ITematicaRepository : IDataRepository<Tematica>

{

}

#### TematicaDbRepository

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

/// <summary>

/// Deposito dati relativo al modello di dati della tematica.

/// </summary>

public class TematicaDbRepository : DbDataRepository<AppDbContext, Tematica>, ITematicaRepository

{

/// <summary>

/// Metodo costruttore, si definisce il contesto di dati.

/// </summary>

/// <param name="ctx">Contesto di dati del database.</param>

public TematicaDbRepository(AppDbContext ctx) : base(ctx)

{

}

/// <summary>

/// Ritorna tutte le entità ordinate per nome.

/// </summary>

/// <returns>Le entità ordinate per nome.</returns>

public override IQueryable<Tematica> Get()

{

return base.Get().OrderBy(s => s.Nome);

}

}

#### IAnnoRepository

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

/// <summary>

/// Interfaccia che implementa i metodi relativi al database

/// sul modello di dati dell'anno.

/// </summary>

public interface IAnnoRepository : IDataRepository<Anno>

{

}

#### AnnoDbRepository

Classe figlia di DbDataRepository relativa al modello di dati dell’anno.

/// <summary>

/// Deposito dati relativo al modello di dati dell'anno.

/// </summary>

public class AnnoDbRepository : DbDataRepository<AppDbContext, Anno>, IAnnoRepository

{

/// <summary>

/// Metodo costruttore, si definisce il contesto di dati.

/// </summary>

/// <param name="ctx">Contesto di dati del database.</param>

public AnnoDbRepository(AppDbContext ctx) : base(ctx)

{

}

/// <summary>

/// Ritorna tutte le entità ordinate per annata.

/// </summary>

/// <returns>Le entità ordinate per annata.</returns>

public override IQueryable<Anno> Get()

{

return base.Get().OrderBy(s => s.Annata);

}

}

#### IEsercizioRepository

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

/// <summary>

/// Interfaccia che implementa i metodi relativi al database

/// sul modello di dati dell'esercizio.

/// </summary>

public interface IEsercizioRepository : IDataRepository<Esercizio>

{

}

#### EsercizioDbRepository

Classe figlia di DbDataRepository relativa al modello di dati dell’esercizio.

/// <summary>

/// Deposito dati relativo al modello di dati dell'esercizio.

/// </summary>

public class EsercizioDbRepository : DbDataRepository<AppDbContext, Esercizio>, IEsercizioRepository

{

/// <summary>

/// Metodo costruttore, si definisce il contesto di dati.

/// </summary>

/// <param name="ctx">Contesto di dati del database.</param>

public EsercizioDbRepository(AppDbContext ctx) : base(ctx)

{

}

/// <summary>

/// Ritorna tutte le entità ordinate per titolo.

/// </summary>

/// <returns>Le entità ordinate per titolo.</returns>

public override IQueryable<Esercizio> Get()

{

return base.Get().OrderBy(s => s.Titolo);

}

}

#### IEsercizioProvaRepository

Interfaccia figlia di IDataRepository relativa al modello di dati dell’esercizio della prova.

/// <summary>

/// Interfaccia che implementa i metodi relativi al database

/// sul modello di dati dell'esercizio della prova.

/// </summary>

public interface IEsercizioProvaRepository : IDataRepository<EsercizioProva>

{

}

#### EsercizioProvaDbRepository

Classe figlia di DbDataRepository relativa al modello di dati dell’esercizio della prova.

/// <summary>

/// Deposito dati relativo al modello di dati dell'esercizio della prova.

/// </summary>

public class EsercizioProvaDbRepository : DbDataRepository<AppDbContext, EsercizioProva>, IEsercizioProvaRepository

{

/// <summary>

/// Metodo costruttore, si definisce il contesto di dati.

/// </summary>

/// <param name="ctx">Contesto di dati del database.</param>

public EsercizioProvaDbRepository(AppDbContext ctx) : base(ctx)

{

}

/// <summary>

/// Ritorna tutte le entità ordinate per titolo.

/// </summary>

/// <returns>Le entità ordinate per titolo.</returns>

public override IQueryable<EsercizioProva> Get()

{

return base.Get().OrderBy(s => s.Titolo);

}

}

#### IProvaRepository

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

/// <summary>

/// Interfaccia che implementa i metodi relativi al database

/// sul modello di dati della prova.

/// </summary>

public interface IProvaRepository : IDataRepository<Prova>

{

}

#### ProvaDbRepository

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

/// <summary>

/// Deposito dati relativo al modello di dati della prova.

/// </summary>

public class ProvaDbRepository : DbDataRepository<AppDbContext, Prova>, IProvaRepository

{

/// <summary>

/// Metodo costruttore, si definisce il contesto di dati.

/// </summary>

/// <param name="ctx">Contesto di dati del database.</param>

public ProvaDbRepository(AppDbContext ctx) : base(ctx)

{

}

/// <summary>

/// Ritorna tutte le entità ordinate per data.

/// </summary>

/// <returns>Le entità ordinate per data.</returns>

public override IQueryable<Prova> Get()

{

return base.Get().OrderBy(s => s.Data);

}

}

### AppDbContext

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

/// <summary>

/// Classe che imposta le entità del database e configura

/// il percorso di memorizzazione della base di dati.

/// </summary>

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> Anno { get; set; }

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

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

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

/// <summary>

/// Metodo costruttore vuoto (ereditato).

/// </summary>

public AppDbContext() : base()

{

}

/// <summary>

/// Metodo costruttore utilizzato per preparare il database.

/// </summary>

/// <param name="options">Configurazione del database.</param>

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

{

}

/// <summary>

/// Configurazione del percorso di memorizzazione del database SQLite.

/// </summary>

protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)

{

if (!optionsBuilder.IsConfigured)

{

optionsBuilder.UseSqlite("Data Source=D:\\Temp\\GestioneEsercizi.db");

}

}

}

## 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.

### 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.

/// <summary>

/// Base class for all ViewModel classes in the application. Provides support for

/// property changes notification. Original implementation by Josh Smith.

/// </summary>

public abstract class BindableBase : INotifyPropertyChanged

{

public event PropertyChangedEventHandler PropertyChanged = delegate { };

protected void OnPropertyChanged(string propertyName)

{

PropertyChangedEventHandler handler = PropertyChanged;

if (handler != null)

{

handler(this, new PropertyChangedEventArgs(propertyName));

}

}

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));

}

}

#### BenvenutoViewModel

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

public class BenvenutoViewModel : BindableBase

{

public BenvenutoViewModel()

{

}

}

#### AboutViewModel

ViewModel relativo alla schermata di informazioni sul prodotto, si imposta il ViewModel della schermata di benvenuto.

public class AboutViewModel : BindableBase

{

private BenvenutoViewModel benvenutoViewModel;

public IDelegateCommand BenvenutoCommand { get; set; }

/// <summary>

/// Metodo costruttore del ViewModel.

/// </summary>

public AboutViewModel() { RegisterCommands(); }

/// <summary>

/// Impostazione dei ViewModels e Commands relativi.

/// </summary>

private void RegisterCommands()

{

benvenutoViewModel = new BenvenutoViewModel();

BenvenutoCommand = new DelegateCommand(OnBenvenuto, CanBenvenuto);

}

private void OnBenvenuto(object obj) { Messenger.Default.Send<BindableBase>(benvenutoViewModel); }

private bool CanBenvenuto(object arg) { return true; }

}

#### ImpostazioniBaseViewModel

ViewModel relativo alla schermata delle impostazioni di base.

public class ImpostazioniBaseViewModel : BindableBase

{

public ObservableCollection<Classe> Classi { get; set; }

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

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

public ImpostazioniBaseViewModel()

{

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());

}

}

#### AnnoViewModel

ViewModel relativo al modello di dati dell’anno.

public class AnnoViewModel : BindableBase

{

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

public AnnoViewModel()

{

AnnoDbRepository repo = new AnnoDbRepository(new AppDbContext());

Anni = new ObservableCollection<Anno>(repo.Get());

}

}

#### EsercizioViewModel

ViewModel relativo al modello di dati dell’esercizio.

public class EsercizioViewModel : BindableBase

{

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

public EsercizioViewModel()

{

EsercizioDbRepository repo = new EsercizioDbRepository(new AppDbContext());

Esercizi = new ObservableCollection<Esercizio>(repo.Get());

}

}

#### EsercizioProvaViewModel

ViewModel relativo al modello di dati dell’esercizio della prova.

public class EsercizioProvaViewModel : BindableBase

{

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

public EsercizioProvaViewModel()

{

EsercizioProvaDbRepository repo = new EsercizioProvaDbRepository(new AppDbContext());

EserciziProva = new ObservableCollection<EsercizioProva>(repo.Get());

}

}

#### ProvaViewModel

ViewModel relativo al modello di dati della prova.

public class ProvaViewModel : BindableBase

{

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

public ProvaViewModel()

{

ProvaDbRepository repo = new ProvaDbRepository(new AppDbContext());

Prove = new ObservableCollection<Prova>(repo.Get());

}

}

#### MainViewModel

Questo è il ViewModel principale e il suo compito è quello di gestire gli altri ViewModel mostrando quello voluto. Per fare ciò si utilizzano le proprietà relative ai ViewModels e i metodi del pacchetto MVVM.

public class MainViewModel : BindableBase

{

private BenvenutoViewModel benvenutovm;

private AboutViewModel aboutvm;

private ImpostazioniBaseViewModel impostazionibasevm;

public IDelegateCommand BenvenutoCommand { get; set; }

public IDelegateCommand AboutCommand { get; set; }

public IDelegateCommand ImpostazioniBaseCommand { get; set; }

private BindableBase currentViewModel;

public BindableBase CurrentViewModel

{

get { return currentViewModel; }

set { SetProperty(ref currentViewModel, value); }

}

public MainViewModel()

{

RegisterCommands();

benvenutovm = new BenvenutoViewModel();

aboutvm = new AboutViewModel();

impostazionibasevm = new ImpostazioniBaseViewModel();

CurrentViewModel = benvenutovm;

}

private void RegisterCommands()

{

AboutCommand = new DelegateCommand(OnAbout, CanAbout);

BenvenutoCommand = new DelegateCommand(OnBenvenuto, CanBenvenuto);

ImpostazioniBaseCommand = new DelegateCommand(OnImpostazioniBase, CanImpostazioniBase);

}

private void OnBenvenuto(object obj) { CurrentViewModel = benvenutovm; }

private bool CanBenvenuto(object arg) { return true; }

private void OnAbout(object obj) { CurrentViewModel = aboutvm; }

private bool CanAbout(object arg) { return true; }

private void OnImpostazioniBase(object obj) { CurrentViewModel = impostazionibasevm; }

private bool CanImpostazioniBase(object arg) { return true; }

}

### Views

La cartella Views rappresenta gli elementi dell’interfaccia utente dell’applicazione.

#### BenvenutoView

View che rappresenta la schermata principale di benvenuto.

<Grid Margin="50">

<Grid.ColumnDefinitions>

<ColumnDefinition Width="\*"/>

<ColumnDefinition Width="\*"/>

<ColumnDefinition Width="\*"/>

</Grid.ColumnDefinitions>

<Button

Grid.Column="0"

VerticalAlignment="Center"

HorizontalAlignment="Center"

Content="Impostazioni di Base"

Padding="10"

Command="{Binding Path=ImpostazioniBaseCommand}">

</Button>

<Button

Grid.Column="1"

VerticalAlignment="Center"

HorizontalAlignment="Center"

Content="Nuovo Esercizio"

Padding="10"

Command="{Binding Path=EsercizioCommand}">

</Button>

<Button

Grid.Column="2"

VerticalAlignment="Center"

HorizontalAlignment="Center"

Content="Creazione Prova"

Padding="10"

Command="{Binding Path=ProvaCommand}">

</Button>

</Grid>

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.

#### AboutView

View relativa alla schermata delle informazioni del prodotto.

<Image Grid.Column="1" Source="/Images/GA.png"/>

<Label Grid.Column="1" Grid.Row="1" VerticalAlignment="Center" HorizontalAlignment="Center">

<Hyperlink x:Name="hlGitHub" Click="hlGitHub\_Click">GestioneEsercizi</Hyperlink>

</Label>

<Button x:Name="bChiudi" Grid.Column="2" Grid.Row="2" VerticalAlignment="Bottom"

Margin="20" Content="Indietro" Command="{Binding Path=BenvenutoCommand}" />

In questo caso viene mostrata un’immagine e il link del repository del progetto su GitHub.

# Test

## 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:**  **Riferimento**: | TC-001  REQ-012 | **Nome:** | Import a card with KIC, KID and KIK keys, but not shown with the GUI |
| **Descrizione:** | Import a card with KIC, KID and KIK keys with no obfuscation, but not shown with the GUI | | |
| **Prerequisiti:** | Store on local PC: Profile\_1.2.001.xml (appendix n\_n) and Cards\_1.2.001.txt (appendix n\_n).  PIN (OTA\_VIEW\_PIN\_PUK\_KEY) and ADM (OTA\_VIEW\_ADM\_KEY) user right not set. | | |
| **Procedura:** | 1. Go to “Cards manager” menu,  in main page click “Import Profiles” link, Select the “1.2.001.xml” file, Import the Profile 2. Go to “Cards manager” menu,  in main page click “Import Cards” link, Select the “1.2.001.txt” file, Delete the cards,  Select the “1.2.001.txt” file, Import the cards 3. Research the “41795924770” Card, Click the imsi card link Check the card details 4. Execute the SQL: SELECT imsi, dir, keyset, cntr, rawtohex(kickey), rawtohex(kidkey), rawtohex(kikkey), rawtohex(chv), rawtohex(dap)FROM otacardkey a where imsi='340041795924770' ORDER BY keyset; | | |
| **Risultati attesi:** | Keys visible in the DB (OtaCardKey) but not visible in the GUI (Card details) | | |

## 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.

## 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.

# Consuntivo

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

# 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

## Sviluppi futuri

Migliorie o estensioni che possono essere sviluppate sul prodotto.

## Considerazioni personali

Cosa ho imparato in questo progetto? ecc

# Bibliografia

## 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

# 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
* …