DOCUMENTAZIONE TECNICA PROGETTO “GESTIONE SCUOLA”

TEAM: MISSION IMPOSSIBLE

1. Introduzione

1.1 Scopo del Documento

Questo è un documento di specifica dei requisiti per un nuovo programma di gestione di una scuola. Il nuovo programma dà la possibilità di visualizzare, aggiungere, modificare e cancellare sia singole classi che studenti, insegnanti, valutazioni degli studenti e materie, con la possibilità di calcolare la media delle valutazioni. Il documento in oggetto, descrive lo scopo, gli obiettivi e le finalità del nuovo programma. Oltre a descrivere i requisiti non funzionali, il medesimo modella i requisiti funzionali con casi d'uso, diagrammi di interazione e modelli di classi. Infine, lo stesso è destinato a guidare la progettazione e l'implementazione del programma target in un linguaggio orientato agli oggetti.

1.2 Background

La richiesta di sviluppare un nuovo programma per la gestione di una scuola è stata motivata dalla necessità di affrontare diverse problematiche che non erano state precedentemente gestite in modo adeguato. Prima dell'implementazione di questo nuovo software, non esisteva un sistema dedicato specificamente alla gestione scolastica, il che ha portato a inefficienze e difficoltà nella gestione di varie attività chiave. I principali problemi riscontrati erano:

- Gestione degli studenti e degli insegnanti: Non esisteva un sistema centralizzato per registrare e mantenere le informazioni degli studenti e degli insegnanti. La gestione dei dettagli personali, delle iscrizioni alle classi e delle informazioni di contatto avveniva manualmente o attraverso strumenti non integrati, rendendo difficile l'accesso rapido e l'aggiornamento accurato delle informazioni.

- Gestione delle valutazioni degli studenti: La registrazione, l'archiviazione e il calcolo delle valutazioni degli studenti erano un processo complesso e suscettibile di errori. Non esisteva un sistema centralizzato per registrare le valutazioni per ogni studente, materia e periodo scolastico, rendendo difficile monitorare il progresso degli studenti e generare report accurati.

- Gestione delle valutazioni: La raccolta, l'organizzazione e l'analisi delle valutazioni degli studenti richiedevano un notevole sforzo manuale. Non era disponibile un sistema automatizzato per calcolare la media delle valutazioni e generare statistiche utili per gli insegnanti e l'amministrazione scolastica.

Il nuovo programma di gestione scolastica è stato sviluppato per affrontare queste problematiche e fornire una soluzione integrata, efficiente e affidabile per la gestione di studenti, insegnanti e valutazioni.

1.3 Stakeholder

1.3.1 Utenti

I principali stakeholder per questo progetto includono:

1. Responsabile scolastico: Il responsabile scolastico è responsabile della supervisione generale del sistema di gestione della scuola. Ha interesse nel garantire che il programma soddisfi le esigenze e i requisiti della scuola.

2. Insegnanti: Gli insegnanti utilizzano il programma per registrare le valutazioni degli studenti e gestire le informazioni delle classi e delle materie. Hanno interesse che il sistema sia facile da usare, efficiente e in grado di gestire correttamente i dati degli studenti.

3. Studenti: Gli studenti sono i principali soggetti delle valutazioni e delle informazioni registrate nel sistema. Hanno interesse che il sistema sia accurato, sicuro e che consenta loro di visualizzare le proprie valutazioni e altre informazioni pertinenti.

1.4 Panoramica del Documento

Il resto del documento fornisce le specifiche dettagliate del nuovo sistema. È organizzato come segue:

Sezione 2: Requisiti funzionali

Sezione 3: Requisiti non funzionali

Sezione 4: Diagrammi UML

Sezione 5: Analisi del codice

Sezione 6: Testing del codice

Sezione 7: Appendice

2. Requisiti Funzionali

Sono stati identificati i seguenti requisiti funzionali per il software.

1. Visualizzazione delle classi:
    - Il software deve consentire la visualizzazione delle classi presenti nella scuola, mostrando l'ID e il nome di ciascuna classe.

2. Gestione delle classi:
    - Il software deve consentire l'aggiunta di nuove classi, richiedendo l'inserimento di un ID univoco e del nome della classe.
    - Il software deve consentire la modifica delle informazioni di una classe esistente, come il nome della classe.
    - Il software deve consentire la cancellazione di una classe esistente, considerando anche l'eventuale riassegnazione degli studenti ad altre classi.
3. Visualizzazione degli studenti:
    - Il software deve consentire la visualizzazione degli studenti presenti nella scuola, mostrando l'ID, il nome, il cognome, la data di nascita, l'anno di corso e la classe di appartenenza di ciascuno studente.

4. Gestione degli studenti:
    - Il software deve consentire l'aggiunta di nuovi studenti, richiedendo l'inserimento di un ID univoco, nome, cognome, data di nascita, anno di corso e la classe di appartenenza.
    - Il software deve consentire la modifica delle informazioni di uno studente esistente, come nome, cognome, data di nascita, anno di corso e classe di appartenenza.
    - Il software deve consentire la cancellazione di uno studente esistente, considerando anche l'eventuale aggiornamento delle valutazioni e delle medie.

5. Visualizzazione degli insegnanti:
    - Il software deve consentire la visualizzazione degli insegnanti presenti nella scuola, mostrando l'ID, il nome, il cognome, la classe assegnata e la materia insegnata.

6. Gestione degli insegnanti:
    - Il software deve consentire l'aggiunta di nuovi insegnanti, richiedendo l'inserimento di un ID univoco, nome, cognome, classe assegnata e materia insegnata.

- Il software deve consentire la modifica delle informazioni di un insegnante esistente, come nome, cognome, classe assegnata e materia insegnata.
- Il software deve consentire la cancellazione di un insegnante esistente, considerando anche l'eventuale riassegnazione degli insegnanti ad altre classi o materie.

7. Gestione delle valutazioni:
    - Il software deve consentire l'aggiunta di nuove valutazioni per gli studenti, specificando l'ID dello studente, l'ID della materia e il voto.
    - Il software deve consentire la modifica di una valutazione esistente, consentendo la modifica del voto.
    - Il software deve consentire la cancellazione di una valutazione esistente.

8. Calcolo delle medie delle valutazioni:
    - Il software deve calcolare la media dei voti di ciascuno studente per ciascuna materia e visualizzarla.
    - Il software deve calcolare la media dei voti di ciascuna materia e visualizzarla.
    - Il software deve aggiornare automaticamente la media delle valutazioni di uno studente o di una materia quando vengono aggiunte, modificate o cancellate le valutazioni corrispondenti.

3. Requisiti Non Funzionali

I requisiti non funzionali identificati per il programma di gestione della scuola includono:

1. Usabilità: Il sistema deve essere intuitivo e facile da usare per gli utenti finali, tra cui insegnanti, studenti e amministrazione scolastica.

2. Affidabilità: Il sistema deve essere affidabile e garantire l'integrità dei dati. Deve essere in grado di gestire carichi di lavoro elevati senza compromettere le prestazioni.

3. Prestazioni: Il sistema deve essere efficiente e reattivo, in grado di gestire un numero elevato di operazioni contemporaneamente senza rallentamenti significativi.

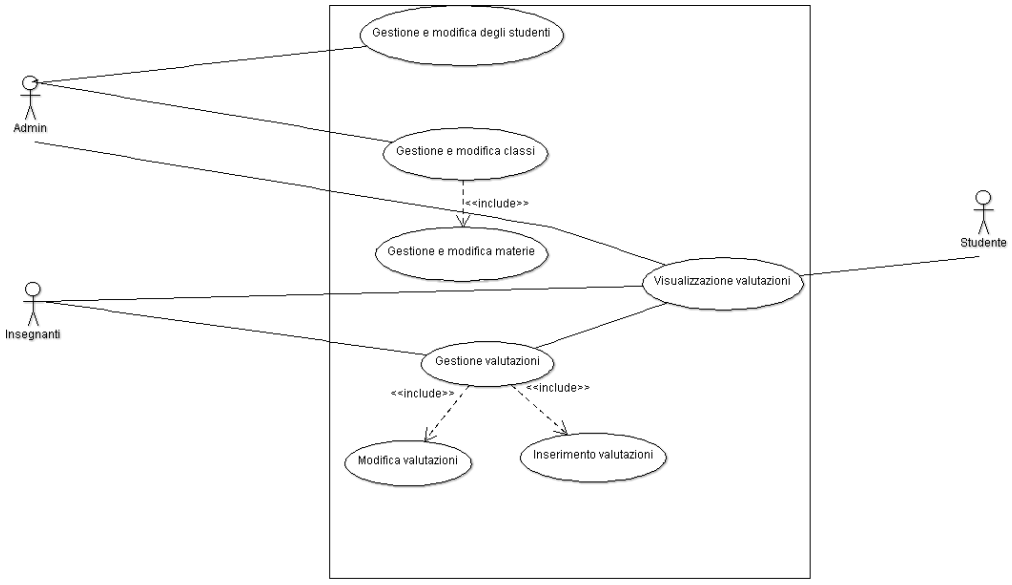
4. Portabilità: Il sistema deve essere compatibile con diverse piattaforme e ambienti operativi, consentendo l'accesso da diversi dispositivi come computer, tablet o smartphone.

5. Manutenibilità: Il sistema deve essere progettato in modo modulare e ben strutturato, facilitando la manutenzione, l'estensibilità e l'aggiornamento del software.

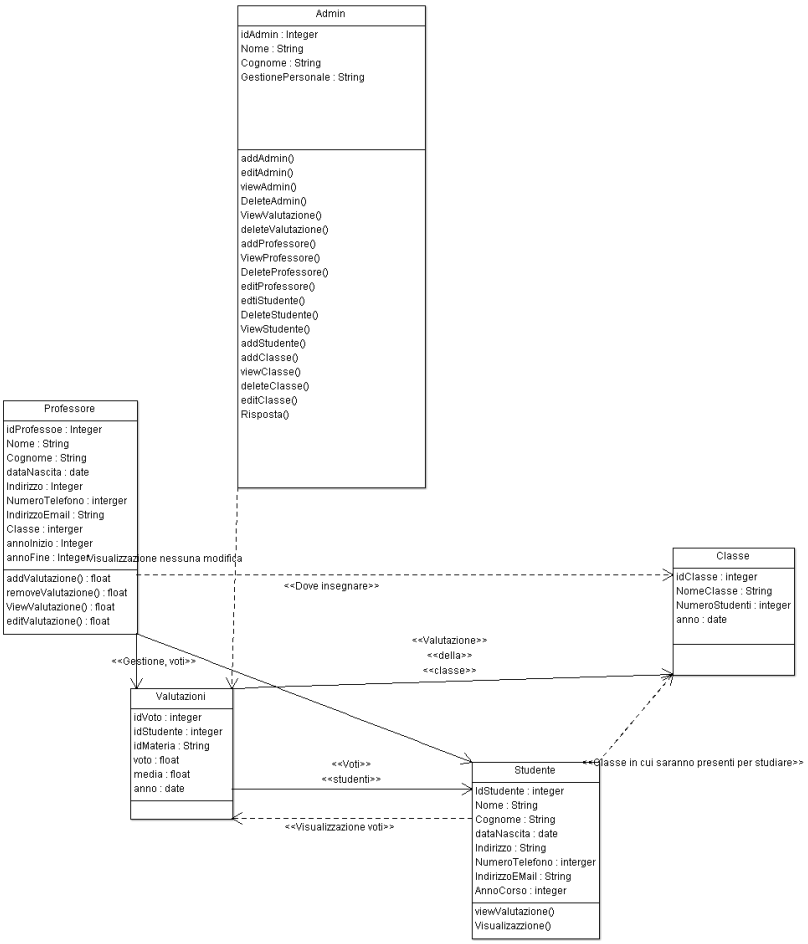
6. Documentazione: Il sistema deve essere corredato da una documentazione completa, chiara e dettagliata che descriva le funzionalità, l'utilizzo e la configurazione del sistema. Fare riferimento al manuale tecnico in appendice

SEZIONE 4: Diagrammi

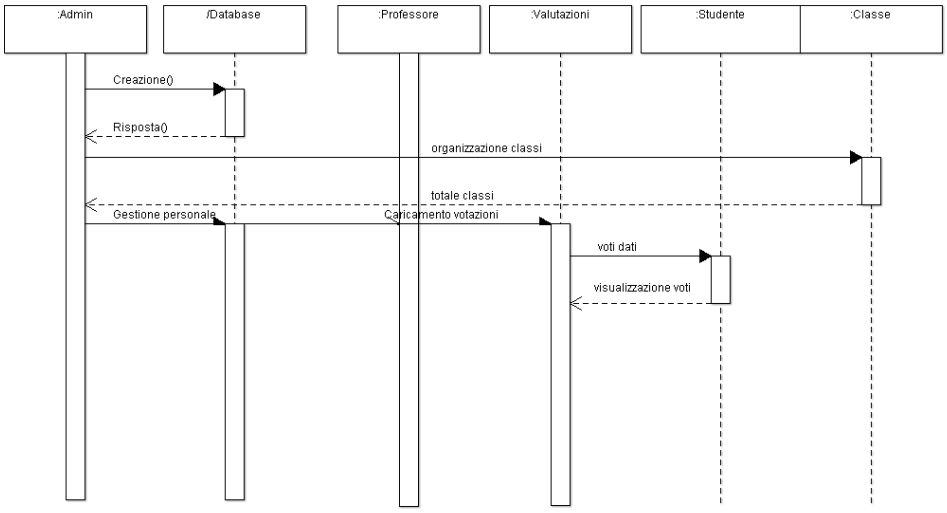
4.1 DIAGRAMMA CASO D'USO



4.2 DIAGRAMMA DI CLASSE



4.3 DIAGRAMMA DI SEQUENZA



SEZIONE 5: Analisi del codice

5.1 Scrittura del codice sorgente

Quest'applicazione è stata sviluppata utilizzando il linguaggio di programmazione C#.

Il codice sorgente è organizzato in diverse classi all'interno del namespace "Gestione". La classe principale è denominata "Program" e contiene il metodo "Main" che rappresenta il punto di ingresso dell'applicazione.

Il codice è strutturato in maniera modulare, suddividendo le funzionalità di gestione in metodi separati. Ogni metodo gestisce un aspetto specifico, come la gestione delle classi, degli studenti, degli insegnanti, delle valutazioni e delle materie. Questa suddivisione consente una migliore leggibilità e manutenibilità del codice.

L'applicazione interagisce con l'utente attraverso un menu a selezione multipla, consentendo di scegliere le diverse opzioni di gestione. Ogni opzione del menu corrisponde a un metodo specifico che implementa la funzionalità desiderata. Inoltre, l'applicazione include anche la possibilità di caricare i dati da un file JSON e di salvarli in modo persistente.

Il codice fa ampio uso di strutture dati come liste per memorizzare le diverse entità, quali classi, studenti, insegnanti, valutazioni e materie. Queste liste vengono aggiornate e manipolate in base alle azioni dell'utente.

Inoltre, il codice include anche controlli di input per garantire che l'utente fornisca valori validi durante l'inserimento dei dati, ad esempio controllando che gli ID siano numeri interi e che i voti siano compresi entro un limite specifico - non è possibile inserire valutazioni superiori al 100 -.

Infine, l'applicazione include anche la funzionalità di calcolo delle medie dei voti sia per gli studenti che per le valutazioni associate a una determinata materia.

5.2 JSON

Durante lo sviluppo del nostro software, il team ha integrato diverse librerie e plugin per ottenere funzionalità avanzate e un'efficiente gestione dei dati. Tra queste, uno degli elementi fondamentali è stato Newtonsoft.Json, comunemente noto come JSON.NET.

Newtonsoft.Json è una libreria ampiamente adottata nell'ecosistema .NET per la manipolazione e la gestione dei dati JSON. Grazie a questa libreria, siamo riusciti a semplificare notevolmente il processo di conversione degli oggetti .NET da e verso il formato JSON.

Una delle principali sfide affrontate durante il progetto era garantire la persistenza dei dati in un formato leggibile e interoperabile. Sfruttando Newtonsoft.Json, siamo riusciti a convertire agevolmente gli oggetti complessi del nostro software in una rappresentazione JSON e viceversa. Questo ci ha consentito di salvare i dati su supporti di archiviazione o scambiarli con altre applicazioni in modo semplice ed efficace.

In conclusione, l'inclusione del plugin Newtonsoft.Json nel nostro software ha svolto un ruolo cruciale nella gestione dei dati JSON. Grazie alle sue potenti funzionalità e alla sua facilità d'uso, siamo riusciti a garantire la persistenza dei dati in modo affidabile e a semplificare la comunicazione con altre applicazioni. L'utilizzo di questa libreria si è dimostrato una scelta strategica che ha contribuito al successo generale
del nostro progetto

Sezione 6: Testing del codice

6.1 Fase di Verifica:

Durante la fase di verifica, sono stati considerati i requisiti utente e i requisiti di sistema per garantire un'analisi accurata per la progettazione dell'applicazione.

Requisiti Utente/Analisi dei Requisiti

Nella fase di analisi dei requisiti utente, sono stati identificati i seguenti requisiti per l'applicazione di gestione della scuola:

Gestione degli studenti:
Inserimento di nuovi studenti nel sistema con dati personali, come nome, cognome, data di nascita, ecc.
Modifica delle informazioni degli studenti esistenti.
Visualizzazione dei dettagli degli studenti.

Gestione delle classi:
Creazione di nuove classi con un nome e un insegnante responsabile.
Aggiunta e rimozione degli studenti da una classe.
Visualizzazione delle classi e dei relativi studenti.

Gestione dei voti delle materie:
Assegnazione dei voti agli studenti per ciascuna materia.
Modifica dei voti assegnati in precedenza.
Visualizzazione dei voti degli studenti.

6.2 Requisiti di Sistema:

I requisiti di sistema per l'applicazione di gestione della scuola sono stati identificati come segue:

Archiviazione dei dati:

I dati degli studenti, delle classi e dei voti devono essere salvati in modo persistente per poter essere utilizzati anche dopo un riavvio dell'applicazione.
È stata preferita l'implementazione di un database per garantire la conservazione e l'accesso efficiente ai dati.

6.3 Fase di Validazione:

Durante la fase di validazione, sono stati eseguiti test per garantire che l'applicazione soddisfi i requisiti e funzioni correttamente.

Esecuzione del Test dei Componenti/Test delle Unità

Per la verifica dei componenti dell'applicazione, sono stati sviluppati test delle unità specifici per ogni componente. I test delle unità mirano a verificare il corretto funzionamento di singoli moduli o parti dell'applicazione. Durante l'esecuzione dei test delle unità, sono stati verificati i seguenti aspetti:

- Verifica dell'inserimento, modifica e visualizzazione degli studenti.
- Verifica della creazione, aggiunta e visualizzazione delle classi.
- Verifica dell'assegnazione, modifica e visualizzazione dei voti delle materie.

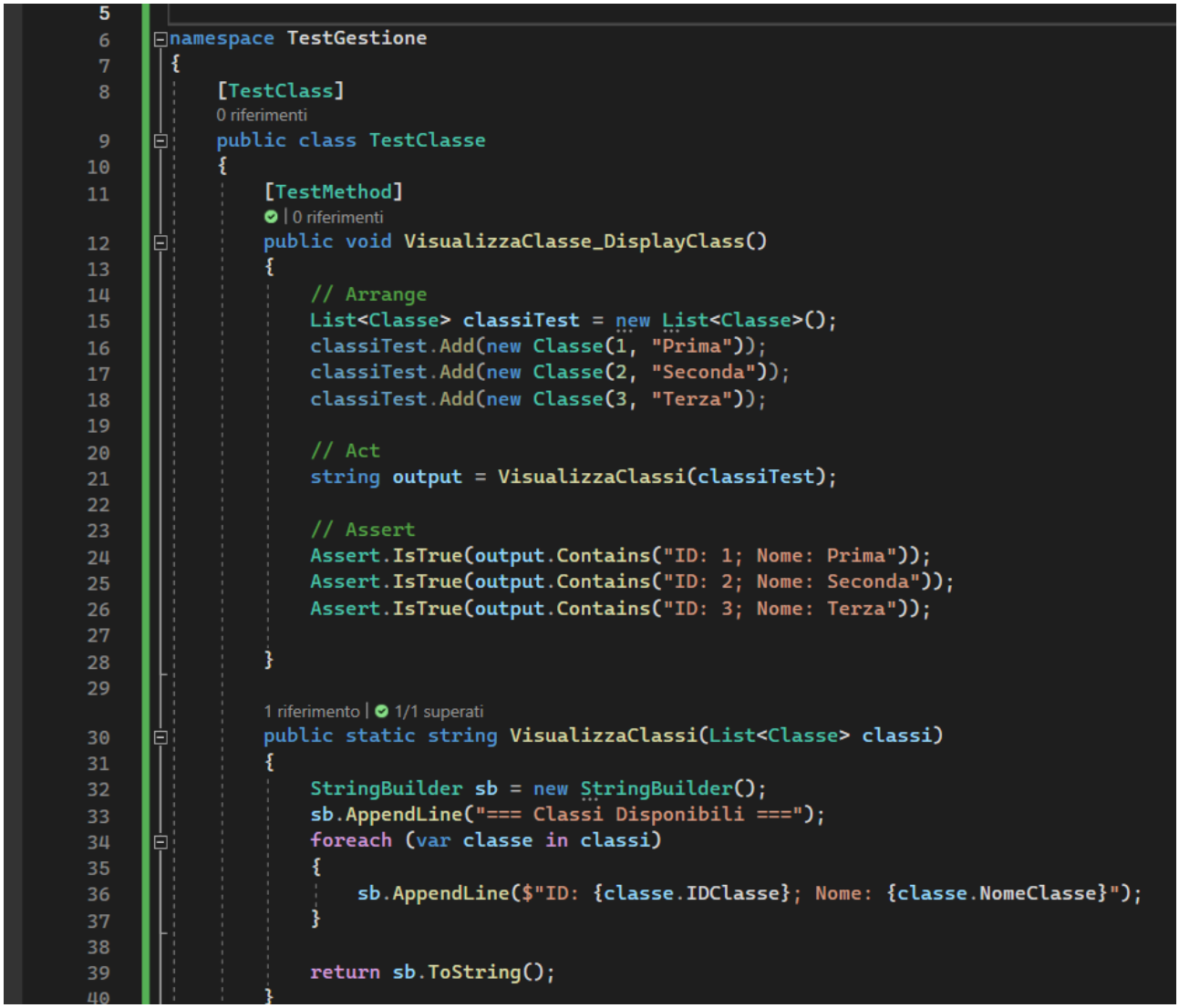
Esecuzione del Test di Accettazione/Accettazione dell'Utente
Durante il test di accettazione si è valutato se si soddisfano le aspettative e i requisiti definiti. Durante questo test, sono stati verificati i seguenti aspetti:
Verifica che l'applicazione consenta l'inserimento, la modifica e la visualizzazione corretta degli studenti, delle classi e dei voti.
Verifica che l'applicazione offra un'interfaccia utente intuitiva e di facile utilizzo.
Verifica che l'applicazione risponda in modo adeguato alle esigenze dell'utente e soddisfi i requisiti specifici della scuola.

6.4 Esecuzione dei test

I seguenti test sono stati eseguiti sulla classe "TestClasse" nel namespace "TestGestione". La classe contiene metodi per la gestione delle classi, inclusi i metodi per visualizzare, aggiungere, modificare e cancellare classi.

Test del metodo "VisualizzaClasse_DisplayClass":

1) Scopo: Verificare se il metodo "VisualizzaClassi" visualizza correttamente le informazioni delle classi.
2) Descrizione:
   Si crea una lista di oggetti "Classe" con tre elementi di test. Si richiama il metodo "VisualizzaClassi" passando la lista di classi di test. Si verifica che l'output contenga le informazioni corrette per ogni classe nella lista di test.
3) Esito atteso: Il test è considerato superato se l'output contiene le informazioni corrette per tutte le classi nella lista di test.



```
namespace TestGestione
{
    [TestClass]
    0 riferimenti
    public class TestClasse
    {
        [TestMethod]
        | 0 riferimenti
        public void VisualizzaClasse_DisplayClass()
        {
            // Arrange
            List<Classe> classiTest = new List<Classe>();
            classiTest.Add(new Classe(1, "Prima"));
            classiTest.Add(new Classe(2, "Seconda"));
            classiTest.Add(new Classe(3, "Terza"));

            // Act
            string output = VisualizzaClassi(classiTest);

            // Assert
            Assert.IsTrue(output.Contains("ID: 1; Nome: Prima"));
            Assert.IsTrue(output.Contains("ID: 2; Nome: Seconda"));
            Assert.IsTrue(output.Contains("ID: 3; Nome: Terza"));

        }

        1 riferimento | 1/1 superati
        public static string VisualizzaClassi(List<Classe> classi)
        {
            StringBuilder sb = new StringBuilder();
            sb.AppendLine("=== Classi Disponibili ===");
            foreach (var classe in classi)
            {
                sb.AppendLine($"ID: {classe.IDClasse}; Nome: {classe.NomeClasse}");
            }

            return sb.ToString();
        }
```

Test del metodo "AggiungiClasse_AddClass":

1) Scopo: Verificare se il metodo "AggiungiClasse" aggiunge correttamente una nuova classe alla lista.
2) Descrizione: Si crea una lista vuota di oggetti "Classe". Si simula l'input dell'utente fornendo un ID e un nome di classe di test. Si richiama il metodo "AggiungiClasse" passando la lista di classi vuota. Si verifica che la lista di classi contenga esattamente un elemento, corrispondente all'ID e al nome forniti.
3) Esito atteso: Il test è considerato superato se la lista di classi contiene esattamente un elemento con l'ID e il nome corretti.

```
[TestMethod]
0 riferimenti
public void AggiungiClasse_AddClass()
{
    // Arrange
    List<Classe> classiTest = new List<Classe>();

    // Simulazione dell'input dell'utente
    string input = "1\nQuinta\n";

    // Simulazione della console input/output
    using (StringReader sr = new StringReader(input))
    {
        Console.SetIn(sr);

        // Act
        AggiungiClasse(classiTest);

        // Assert
        Assert.AreEqual(1, classiTest.Count);
        Assert.AreEqual(1, classiTest[0].IDClasse);
        Assert.AreEqual("Quinta", classiTest[0].NomeClasse);
    }
}

1 riferimento | 1/1 superati
public static void AggiungiClasse(List<Classe> classi)
{
    Console.WriteLine("=== Aggiungi Classe ===");
    int id = 0;
    bool isValidId = false;
    while (!isValidId)
    {
        Console.Write("Inserisci l'ID della classe: ");
        string idInput = Console.ReadLine();
        if (int.TryParse(idInput, out id))
        {
            isValidId = true;
        }
        else
        {
            Console.WriteLine("ID non valido. Inserire un valore numerico intero");
        }
    }

    Console.Write("Inserisci il nome della classe: ");
    string nome = Console.ReadLine();

    Classe nuovaClasse = new Classe(id, nome);
    classi.Add(nuovaClasse);

    Console.WriteLine("Classe aggiunta con successo.");
}
```

Test del metodo "ModificaClasse_ChangeClass":

1) Scopo: Verificare se il metodo "ModificaClasse" modifica correttamente il nome di una classe esistente.
2) Descrizione: Si crea una lista di oggetti "Classe" con un elemento di test. Si simula l'input dell'utente fornendo l'ID della classe di test e un nuovo nome di classe. Si richiama il metodo "ModificaClasse" passando la lista di classi di test. Si verifica che il nome della classe nella lista sia stato modificato correttamente.
3) Esito atteso: Il test è considerato superato se il nome della classe nella lista è stato modificato correttamente.

```
[TestMethod]
0 riferimenti
public void ModificaClasse_ChangeClass()
{
    // Arrange
    List<Classe> classiTest = new List<Classe>();
    classiTest.Add(new Classe(1, "Prima"));

    // Simulazione dell'input dell'utente
    string input = "1\nSeconda\n";

    // Simulazione della console input/output
    using (StringReader sr = new StringReader(input))
    {
        Console.SetIn(sr);

        // Act
        ModificaClasse(classiTest);

        // Assert
        Assert.AreEqual("Seconda", classiTest[0].NomeClasse);
    }
}

1 riferimento | 1/1 superati
public static void ModificaClasse(List<Classe> classi)
{
    Console.WriteLine("=== Modifica Classe ===");
    Console.Write("Inserisci l'ID della classe da modificare: ");
    int id = int.Parse(Console.ReadLine());

    Classe classeDaModificare = classi.Find(c => c.IDClasse == id);
    if (classeDaModificare != null)
    {
        Console.Write("Inserisci il nuovo nome della classe: ");
        string nome = Console.ReadLine();

        classeDaModificare.NomeClasse = nome;

        Console.WriteLine("Classe modificata con successo.");
    }
    else
    {
        Console.WriteLine("Classe non trovata.");
    }
}
```

Test del metodo "CancellaClasse_DeleteClass":

1) Scopo: Verificare se il metodo "CancellaClasse" cancella correttamente una classe esistente dalla lista.

2) Descrizione: Si crea una lista di oggetti "Classe" con un elemento di test. Si simula l'input dell'utente fornendo l'ID della classe di test. Si richiama il metodo "CancellaClasse" passando la lista di classi di test. Si verifica che la classe sia stata correttamente cancellata dalla lista.
3) Esito atteso: Il test è considerato superato se la classe è stata correttamente cancellata dalla lista.

```
[TestMethod]
| 0 riferimenti
public void CancellaClasse_DeleteClass()
{
    // Arrange
    List<Classe> classiTest = new List<Classe>();
    classiTest.Add(new Classe(1, "Prima"));

    // Simulazione dell'input dell'utente
    string input = "1\n";

    // Simulazione della console input/output
    using (StringReader sr = new StringReader(input))
    {
        Console.SetIn(sr);

        // Act
        CancellaClasse(classiTest);

        // Assert
        Assert.AreEqual(0, classiTest.Count);
    }
}
1 riferimento | 1/1 superati
public static void CancellaClasse(List<Classe> classi)
{
    Console.WriteLine("=== Cancella Classe ===");
    Console.Write("Inserisci l'ID della classe da cancellare: ");
    int id = int.Parse(Console.ReadLine());

    Classe classeDaCancellare = classi.Find(c => c.IDClasse == id);
    if (classeDaCancellare != null)
    {
        classi.Remove(classeDaCancellare);
        Console.WriteLine("Classe cancellata con successo.");
    }
    else
    {
        Console.WriteLine("Classe non trovata.");
    }
}
```

I seguenti test sono stati eseguiti sulla classe "TestStudente" nel namespace "TestGestione". La classe contiene metodi per la gestione degli studenti, inclusi i metodi per visualizzare, aggiungere, modificare e cancellare studenti.

Test del metodo "VisualizzaStudenti_DisplayStudents":

1) Scopo: Verificare se il metodo "VisualizzaStudenti" visualizza correttamente le informazioni degli studenti.
2) Descrizione: Si crea una lista di oggetti "Studente" con un elemento di test. Si richiama il metodo "VisualizzaStudenti" passando la lista di studenti di test. Si verifica che l'output contenga le informazioni corrette per lo studente nella lista di test.
3) Esito atteso: Il test è considerato superato se l'output contiene le informazioni corrette per lo studente nella lista di test.

```
[TestClass]
0 riferimenti
public class TestStudente
{
    [TestMethod]
    0 riferimenti
    public void VisualizzaStudenti_DisplayStudents()
    {
        // Arrange
        List<Studente> studentiTest = new List<Studente>();
        studentiTest.Add(new Studente(1, "Riccardo", "Gentile", new DateOnly(1998, 06, 19), 2023, "Prima"));

        // Act
        string output = VisualizzaStudenti(studentiTest);

        // Assert
        Assert.IsTrue(output.Contains("ID: 1; Nome: Riccardo; Cognome: Gentile; Data di Nascita: 1998-06-19; Anno di Corso: 2023; Classe: Prima"));

    }

    1 riferimento | 1/1 superati
    public static string VisualizzaStudenti(List<Studente> studenti)
    {
        StringBuilder sb = new StringBuilder();
        sb.AppendLine("=== Studenti Disponibili ===");
        foreach (var studente in studenti)
        {
            sb.AppendLine($"ID: {studente.IDStudente}; Nome: {studente.Nome}; Cognome: {studente.Cognome}; Data di Nascita: {studente.DataNascita.ToString("yyyy-MM-dd")}; Anno di Corso: {studente.AnnoCorso}; Classe: {studente.NomeClasse}");
        }

        return sb.ToString();
    }
```

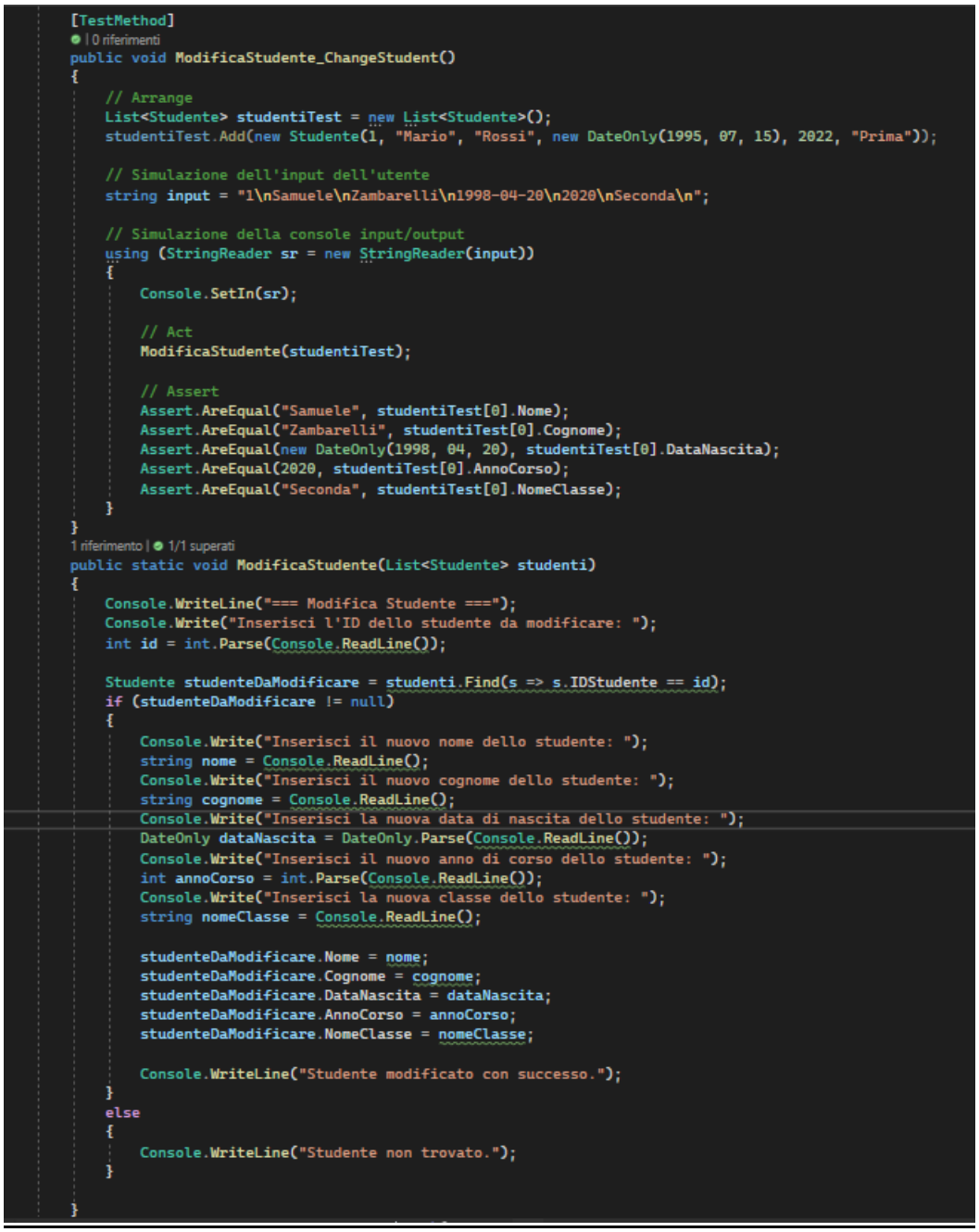
Test del metodo "AggiungiStudente_AddStudent":

1) Scopo: Verificare se il metodo "AggiungiStudente" aggiunge correttamente un nuovo studente alla lista.
2) Descrizione: Si crea una lista vuota di oggetti "Studente". Si simula l'input dell'utente fornendo un ID, nome, cognome, data di nascita, anno di corso e classe di test. Si richiama il metodo "AggiungiStudente" passando la lista di studenti vuota. Si verifica che la lista di studenti contenga esattamente un elemento, corrispondente ai dati forniti.
3) Esito atteso: Il test è considerato superato se la lista di studenti contiene esattamente un elemento con i dati corretti.

```
[TestMethod]
0 riferimenti
public void AggiungiStudente_AddStudent()
{
    // Arrange
    List<Studente> studentiTest = new List<Studente>();

    // Simulazione dell'input dell'utente
    string input = "1\nMario\nRossi\n1995-07-15\n2022\nSeconda\n";

    // Simulazione della console input/output
    using (StringReader sr = new StringReader(input))
    {
        Console.SetIn(sr);

        // Act
        AggiungiStudente(studentiTest);

        // Assert
        Assert.AreEqual(1, studentiTest.Count);
        Assert.AreEqual(1, studentiTest[0].IDStudente);
        Assert.AreEqual("Mario", studentiTest[0].Nome);
        Assert.AreEqual("Rossi", studentiTest[0].Cognome);
        Assert.AreEqual(new DateOnly(1995, 7, 15), studentiTest[0].DataNascita);
        Assert.AreEqual(2022, studentiTest[0].AnnoCorso);
        Assert.AreEqual("Seconda", studentiTest[0].NomeClasse);
    }
}
1 riferimento | 1/1 superati
public static void AggiungiStudente(List<Studente> studenti)
{
    Console.WriteLine("=== Aggiungi Studente ===");
    Console.Write("Inserisci l'ID dello studente: ");
    int id;
    while (!int.TryParse(Console.ReadLine(), out id))
    {
        Console.WriteLine("Inserisci un ID valido (numero intero).");
        Console.Write("Inserisci l'ID dello studente: ");
    }

    Console.Write("Inserisci il nome dello studente: ");
    string nome = Console.ReadLine();
    Console.Write("Inserisci il cognome dello studente: ");
    string cognome = Console.ReadLine();
    Console.Write("Inserisci la data di nascita dello studente: ");
    DateOnly dataNascita = DateOnly.Parse(Console.ReadLine());
    Console.Write("Inserisci l'anno di corso dello studente: ");
    int annoCorso = int.Parse(Console.ReadLine());
    Console.Write("Inserisci la classe dello studente: ");
    string nomeClasse = Console.ReadLine();

    Studente nuovoStudente = new Studente(id, nome, cognome, dataNascita, annoCorso, nomeClasse);
    studenti.Add(nuovoStudente);

    Console.WriteLine("Studente aggiunto con successo.");
}
```

Test del metodo "ModificaStudente_ChangeStudent":

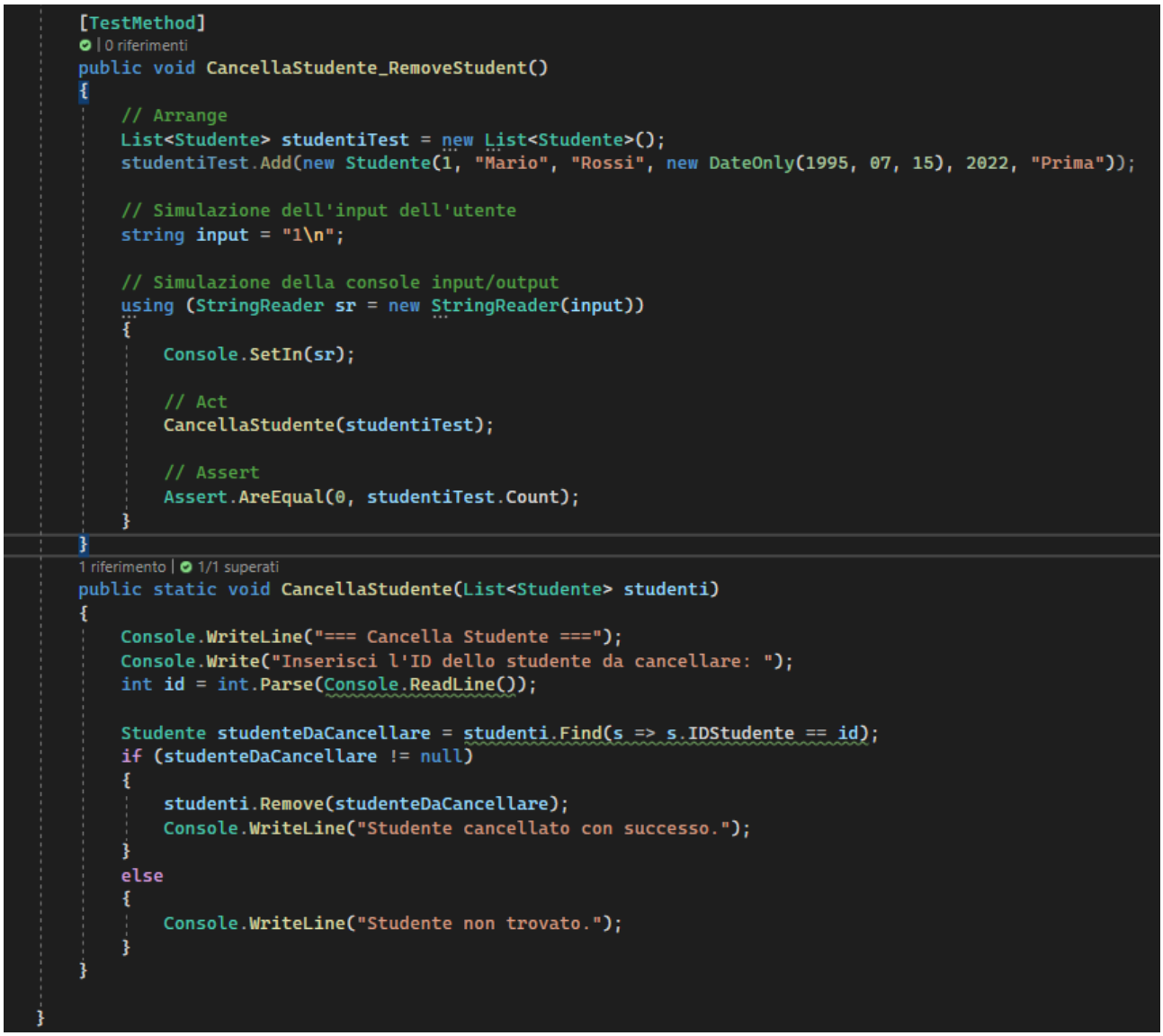
1) Scopo: Verificare se il metodo "ModificaStudente" modifica correttamente i dati di uno studente esistente.
2) Descrizione: Si crea una lista di oggetti "Studente" con un elemento di test. Si simula l'input dell'utente fornendo l'ID dello studente di test e nuovi dati per nome, cognome, data di nascita, anno di corso e classe. Si richiama il metodo "ModificaStudente" passando la lista di studenti di test. Si verifica che i dati dello studente nella lista siano stati modificati correttamente.
3) Esito atteso: Il test è considerato superato se i dati dello studente nella lista sono stati modificati correttamente.



```
[TestMethod]
● | 0 riferimenti
public void ModificaStudente_ChangeStudent()
{
    // Arrange
    List<Studente> studentiTest = new List<Studente>();
    studentiTest.Add(new Studente(1, "Mario", "Rossi", new DateOnly(1995, 07, 15), 2022, "Prima"));

    // Simulazione dell'input dell'utente
    string input = "1\nSamuele\nZambarelli\n1998-04-20\n2020\nSeconda\n";

    // Simulazione della console input/output
    using (StringReader sr = new StringReader(input))
    {
        Console.SetIn(sr);

        // Act
        ModificaStudente(studentiTest);

        // Assert
        Assert.AreEqual("Samuele", studentiTest[0].Nome);
        Assert.AreEqual("Zambarelli", studentiTest[0].Cognome);
        Assert.AreEqual(new DateOnly(1998, 04, 20), studentiTest[0].DataNascita);
        Assert.AreEqual(2020, studentiTest[0].AnnoCorso);
        Assert.AreEqual("Seconda", studentiTest[0].NomeClasse);
    }
}
1 riferimento | ● 1/1 superati
public static void ModificaStudente(List<Studente> studenti)
{
    Console.WriteLine("=== Modifica Studente ===");
    Console.Write("Inserisci l'ID dello studente da modificare: ");
    int id = int.Parse(Console.ReadLine());

    Studente studenteDaModificare = studenti.Find(s => s.IDStudente == id);
    if (studenteDaModificare != null)
    {
        Console.Write("Inserisci il nuovo nome dello studente: ");
        string nome = Console.ReadLine();
        Console.Write("Inserisci il nuovo cognome dello studente: ");
        string cognome = Console.ReadLine();
        Console.Write("Inserisci la nuova data di nascita dello studente: ");
        DateOnly dataNascita = DateOnly.Parse(Console.ReadLine());
        Console.Write("Inserisci il nuovo anno di corso dello studente: ");
        int annoCorso = int.Parse(Console.ReadLine());
        Console.Write("Inserisci la nuova classe dello studente: ");
        string nomeClasse = Console.ReadLine();

        studenteDaModificare.Nome = nome;
        studenteDaModificare.Cognome = cognome;
        studenteDaModificare.DataNascita = dataNascita;
        studenteDaModificare.AnnoCorso = annoCorso;
        studenteDaModificare.NomeClasse = nomeClasse;

        Console.WriteLine("Studente modificato con successo.");
    }
    else
    {
        Console.WriteLine("Studente non trovato.");
    }

}
```

Test del metodo "CancellaStudente_RemoveStudent":

1) Scopo: Verificare se il metodo "CancellaStudente" cancella correttamente uno studente esistente dalla lista.
2) Descrizione: Si crea una lista di oggetti "Studente" con un elemento di test. Si simula l'input dell'utente fornendo l'ID dello studente di test. Si richiama il metodo "CancellaStudente" passando la lista di studenti di test. Si verifica che lo studente sia stato correttamente cancellato dalla lista.
3) Esito atteso: Il test è considerato superato se lo studente è stato correttamente cancellato dalla lista.

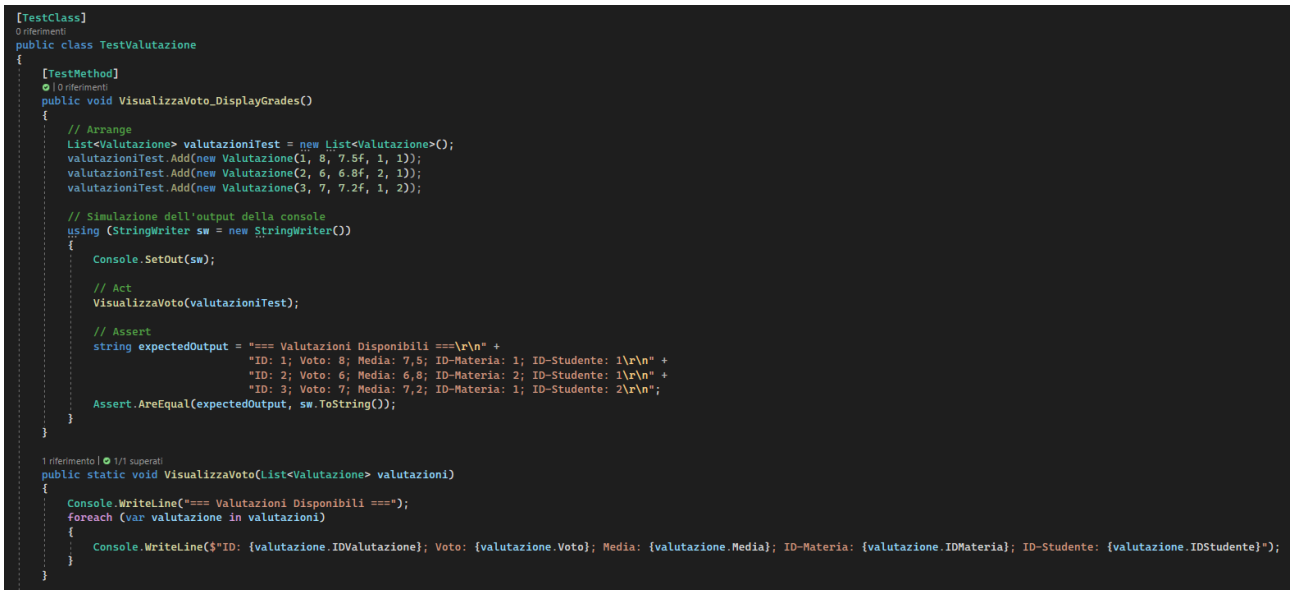


```
    [TestMethod]
    | 0 riferimenti
    public void CancellaStudente_RemoveStudent()
    {
        // Arrange
        List<Studente> studentiTest = new List<Studente>();
        studentiTest.Add(new Studente(1, "Mario", "Rossi", new DateOnly(1995, 07, 15), 2022, "Prima"));

        // Simulazione dell'input dell'utente
        string input = "1\n";

        // Simulazione della console input/output
        using (StringReader sr = new StringReader(input))
        {
            Console.SetIn(sr);

            // Act
            CancellaStudente(studentiTest);

            // Assert
            Assert.AreEqual(0, studentiTest.Count);
        }
    }
    1 riferimento | 1/1 superati
    public static void CancellaStudente(List<Studente> studenti)
    {
        Console.WriteLine("=== Cancella Studente ===");
        Console.Write("Inserisci l'ID dello studente da cancellare: ");
        int id = int.Parse(Console.ReadLine());

        Studente studenteDaCancellare = studenti.Find(s => s.IDStudente == id);
        if (studenteDaCancellare != null)
        {
            studenti.Remove(studenteDaCancellare);
            Console.WriteLine("Studente cancellato con successo.");
        }
        else
        {
            Console.WriteLine("Studente non trovato.");
        }
    }

}
```

I seguenti test sono stati eseguiti sulla classe "TestValutazione" nel namespace "TestGestione". La classe contiene metodi per la gestione delle valutazioni degli studenti, inclusi i metodi per visualizzare, aggiungere, modificare e cancellare le valutazioni.

Test del metodo "VisualizzaVoto_DisplayGrades":

1) Scopo: Verificare se il metodo "VisualizzaVoto" visualizza correttamente le informazioni delle valutazioni.
2) Descrizione: Si crea una lista di oggetti "Valutazione" con alcuni elementi di test. Si simula l'output della console. Si richiama il metodo "VisualizzaVoto" passando la lista di valutazioni di test. Si verifica che l'output contenga le informazioni corrette per le valutazioni nella lista di test.
3) Esito atteso: Il test è considerato superato se l'output contiene le informazioni corrette per le valutazioni nella lista di test.



```
[TestClass]
0 riferimenti
public class TestValutazione
{
    [TestMethod]
    | 0 riferimenti
    public void VisualizzaVoto_DisplayGrades()
    {
        // Arrange
        List<Valutazione> valutazioniTest = new List<Valutazione>();
        valutazioniTest.Add(new Valutazione(1, 8, 7.5f, 1, 1));
        valutazioniTest.Add(new Valutazione(2, 6, 6.8f, 2, 1));
        valutazioniTest.Add(new Valutazione(3, 7, 7.2f, 1, 2));

        // Simulazione dell'output della console
        using (StringWriter sw = new StringWriter())
        {
            Console.SetOut(sw);

            // Act
            VisualizzaVoto(valutazioniTest);

            // Assert
            string expectedOutput = "=== Valutazioni Disponibili ===\r\n" +
                                    "ID: 1; Voto: 8; Media: 7,5; ID-Materia: 1; ID-Studente: 1\r\n" +
                                    "ID: 2; Voto: 6; Media: 6,8; ID-Materia: 2; ID-Studente: 1\r\n" +
                                    "ID: 3; Voto: 7; Media: 7,2; ID-Materia: 1; ID-Studente: 2\r\n";
            Assert.AreEqual(expectedOutput, sw.ToString());
        }
    }

    1 riferimento | 1/1 superati
    public static void VisualizzaVoto(List<Valutazione> valutazioni)
    {
        Console.WriteLine("=== Valutazioni Disponibili ===");
        foreach (var valutazione in valutazioni)
        {
            Console.WriteLine($"ID: {valutazione.IDValutazione}; Voto: {valutazione.Voto}; Media: {valutazione.Media}; ID-Materia: {valutazione.IDMateria}; ID-Studente: {valutazione.IDStudente}");
        }
    }
```

Test del metodo "AggiungiValutazione_AddGrade":

1) Scopo: Verificare se il metodo "AggiungiVoto" aggiunge correttamente una nuova valutazione alla lista.
2) Descrizione: Si crea una lista vuota di oggetti "Valutazione". Si simula l'input dell'utente fornendo l'ID dello studente, l'ID della materia e il voto di test. Si simula l'input/output della console. Si richiama il metodo "AggiungiVoto" passando la lista di valutazioni vuota. Si verifica che la lista di valutazioni contenga esattamente un elemento, corrispondente ai dati forniti.
3) Esito atteso: Il test è considerato superato se la lista di valutazioni contiene esattamente un elemento con i dati corretti.

```
[TestMethod]
0 riferimenti
public void AggiungiValutazione_AddGrade()
{
    // Arrange
    List<Valutazione> valutazioniTest = new List<Valutazione>();

    // Simulazione dell'input dell'utente
    string input = "1\n1\n8\n";

    // Simulazione della console input/output
    using (StringReader sr = new StringReader(input))
    {
        Console.SetIn(sr);

        // Act
        AggiungiVoto(valutazioniTest);

        // Assert
        Assert.AreEqual(1, valutazioniTest.Count);
        Assert.AreEqual(1, valutazioniTest[0].IDValutazione);
        Assert.AreEqual(8, valutazioniTest[0].Voto);
        Assert.AreEqual(0, valutazioniTest[0].Media);
        Assert.AreEqual(1, valutazioniTest[0].IDMateria);
        Assert.AreEqual(1, valutazioniTest[0].IDStudente);

    }
}
1 riferimento | 1/1 superati
private void AggiungiVoto(List<Valutazione> valutazioni)
{
    Console.WriteLine("=== Aggiungi Voto ===");
    Console.Write("Inserisci l'ID dello studente: ");
    int idStudente = int.Parse(Console.ReadLine());
    Console.Write("Inserisci l'ID della materia: ");
    int idMateria = int.Parse(Console.ReadLine());
    Console.Write("Inserisci il voto: ");
    float voto = float.Parse(Console.ReadLine());

    // Controllo per il limite massimo del voto
    if (voto > 100)
    {
        voto = 100;
        Console.WriteLine("Il voto inserito supera il limite massimo consentito. Verrà assegnato il valore massimo (100).");
    }

    // Trova la valutazione corrispondente allo studente e alla materia
    Valutazione valutazioneEsistente = valutazioni.Find(v => v.IDStudente == idStudente && v.IDMateria == idMateria);

    if (valutazioneEsistente != null)
    {
        // Modifica la valutazione esistente
        valutazioneEsistente.Voto = voto;
        Console.WriteLine("Voto aggiornato con successo.");
    }
    else
    {
        // Crea una nuova valutazione
        Valutazione nuovaValutazione = new Valutazione(valutazioni.Count + 1, voto, 0, idMateria, idStudente);
        valutazioni.Add(nuovaValutazione);
        Console.WriteLine("Voto aggiunto con successo.");
    }

}
```

Test del metodo "ModificaVoto_ChangeGrade":

1) Scopo: Verificare se il metodo "ModificaVoto" modifica correttamente il voto di una valutazione esistente.
2) Descrizione: Si crea una lista di oggetti "Valutazione" con un elemento di test. Si simula l'input dell'utente fornendo l'ID della valutazione di test e il nuovo voto. Si simula l'input/output della console. Si richiama il metodo "ModificaVoto" passando la lista di valutazioni di test. Si verifica che il voto della valutazione nella lista sia stato modificato correttamente.
3) Esito atteso: Il test è considerato superato se il voto della valutazione nella lista è stato modificato correttamente.

```
[TestMethod]
0 riferimenti
public void ModificaVoto_ChangeGrade()
{
    // Arrange
    List<Valutazione> valutazioniTest = new List<Valutazione>();
    valutazioniTest.Add(new Valutazione(1, 8, 0, 1, 1));

    // Simulazione dell'input dell'utente
    string input = "1\n9\n";

    // Simulazione della console input/output
    using (StringReader sr = new StringReader(input))
    {
        Console.SetIn(sr);

        // Act
        ModificaVoto(valutazioniTest);

        // Assert
        Assert.AreEqual(9, valutazioniTest[0].Voto);
    }
}

1 riferimento | 1/1 superati
public static void ModificaVoto(List<Valutazione> valutazioni)
{
    Console.WriteLine("=== Modifica Voto ===");
    Console.Write("Inserisci l'ID del voto da modificare: ");
    int idVoto = int.Parse(Console.ReadLine());

    Valutazione votoDaModificare = valutazioni.Find(v => v.IDValutazione == idVoto);
    if (votoDaModificare != null)
    {
        Console.Write("Inserisci il nuovo voto: ");
        float nuovoVoto = float.Parse(Console.ReadLine());

        // Trova la valutazione corrispondente e aggiorna il voto
        votoDaModificare.Voto = nuovoVoto;
        Console.WriteLine("Voto modificato con successo.");
    }
    else
    {
        Console.WriteLine("Voto non trovato.");
    }
}
```

Test del metodo "CancellaValutazione_RemoveGrade":

1) Scopo: Verificare se il metodo "CancellaVoto" cancella correttamente una valutazione esistente dalla lista.
2) Descrizione: Si crea una lista di oggetti "Valutazione" con un elemento di test. Si simula l'input dell'utente fornendo l'ID della valutazione di test. Si simula l'input/output della console. Si richiama il metodo "CancellaVoto" passando la lista di valutazioni di test. Si verifica che la lista di valutazioni sia vuota dopo la cancellazione.
3) Esito atteso: Il test è considerato superato se la lista di valutazioni è vuota dopo la cancellazione.

```
    [TestMethod]
        0 riferimenti
        public void CancellaValutazione_RemoveGrade()
        {
        // Arrange
        List<Valutazione> valutazioniTest = new List<Valutazione>();
        valutazioniTest.Add(new Valutazione(1, 80, 0, 1, 1));

        // Simulazione dell'input dell'utente
        string input = "1\n";

        // Simulazione della console input/output
        using (StringReader sr = new StringReader(input))
        {
            Console.SetIn(sr);

            // Act
            CancellaVoto(valutazioniTest);

            // Assert
            Assert.AreEqual(0, valutazioniTest.Count);
        }
    }

    1 riferimento | 1/1 superati
    private void CancellaVoto(List<Valutazione> valutazioni)
    {
        Console.WriteLine("=== Cancella Voto ===");
        Console.Write("Inserisci l'ID del voto da cancellare: ");
        int idVoto = int.Parse(Console.ReadLine());

        Valutazione votoDaCancellare = valutazioni.Find(v => v.IDValutazione == idVoto);
        if (votoDaCancellare != null)
        {
            int idMateria = votoDaCancellare.IDMateria;
            valutazioni.Remove(votoDaCancellare);
            Console.WriteLine("Voto cancellato con successo.");
        }
        else
        {
            Console.WriteLine("Voto non trovato.");
        }
    }
}
```

Note:

- I test sono implementati utilizzando il framework di testing "Microsoft.VisualStudio.TestTools.UnitTesting".
- I test utilizzano la simulazione dell'input e dell'output della console tramite l'utilizzo di "StringReader" e "Console.SetIn" per i test interattivi.
- I test controllano le asserzioni utilizzando il metodo "Assert" del framework di testing per confrontare i risultati attesi con i risultati ottenuti.
- Ogni test è indipendente dagli altri e non dipende dall'ordine di esecuzione.

In conclusione, il processo di testing e documentazione per il progetto di gestione di una scuola ha garantito la verifica accurata delle funzionalità dell'applicazione e la conformità ai requisiti utente e di sistema. Seguendo i principi fondamentali e eseguendo test

approfonditi, è stato possibile sviluppare un'applicazione affidabile e di successo per la gestione delle attività scolastiche.

SEZIONE 7:

7.1 MANUALE DI UTILIZZO:

L'interfaccia per la gestione della scuola è molto semplice ed intuitiva. Una volta avviato il programma si verrà riportati ad un menù con le seguenti voci:

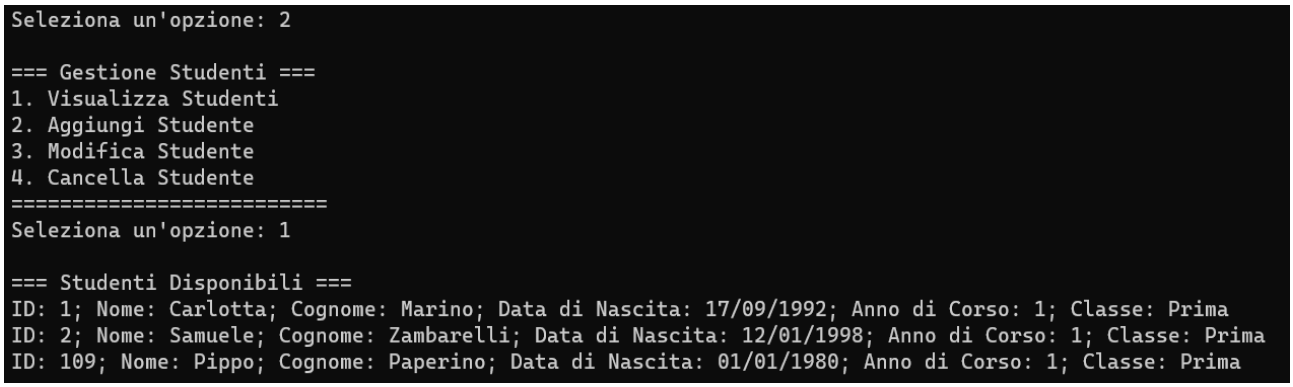
```
Dati caricati con successo.
=== Menu ===
1. Gestione Classi
2. Gestione Studenti
3. Gestione Insegnanti
4. Gestione Valutazioni
5. Gestione Materie
6. Uscita
============
Seleziona un'opzione:
```

Basta digitare il numero associato all'azione che si vuole eseguire ed il lavoro è presto che fatto.

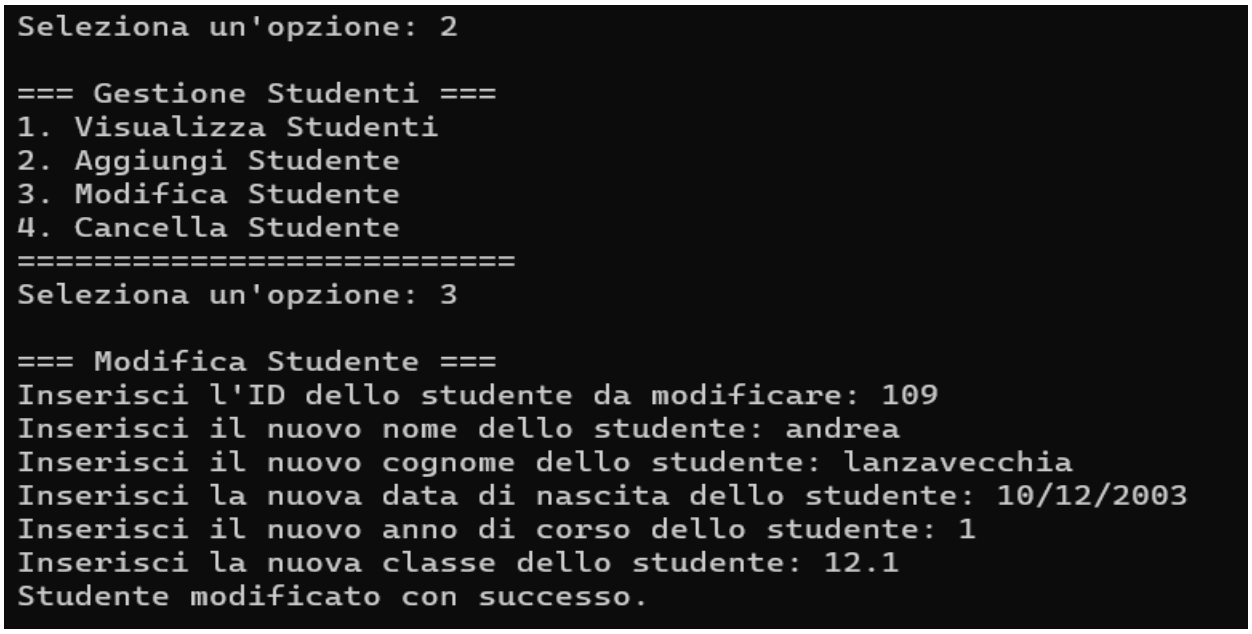
```
Seleziona un'opzione: 1

=== Gestione Classi ===
1. Visualizza Classi
2. Aggiungi Classe
3. Modifica Classe
4. Cancella Classe
======================
Seleziona un'opzione: 1

=== Classi Disponibili ===
ID: 1; Nome: 1^D
ID: 2; Nome: 2^C
ID: 3; Nome: 3^A
```

Per ogni gestione sono disponibili le opzioni di: **visualizzazione, aggiunta, modifica** e **rimozione.**

Nel caso soprastante è stata richiesta la visualizzazione delle classi. Ogni classe è associata ad un ID per poterla recuperare facilmente nel database ed, eventualmente, modificarla.



Per gli studenti vengono specificati: **nome, cognome, data di nascita, anno del corso** che sta conseguendo e la **classe**. Anche gli studenti naturalmente vengono associati ad un ID per rendere la modifica immediata.



Per gli insegnanti invece vengono visualizzati: **nome, cognome, le classi in cui insegna e la materia.**

Anche loro sempre identificati tramite un ID.

```
=== Gestione Insegnanti ===
1. Visualizza Insegnanti
2. Aggiungi Insegnante
3. Modifica Insegnante
4. Cancella Insegnante
==========================
Seleziona un'opzione: 1

=== Insegnanti Disponibili ===
ID: 1; Nome: Olimpia; Cognome: Di Stefano; Classe: Prima; Materia: SQL
ID: 2; Nome: Ignazio; Cognome: Selvaggio; Classe: Prima; Materia: Ingegneria del Software
ID: 3; Nome: giacomo; Cognome: pastorino; Classe: 12.1; Materia: programmazione in C
```

Per quanto riguarda invece le valutazioni e le materie:

```
Seleziona un'opzione: 4

=== Gestione Valutazioni ===
1. Visualizza Voto
2. Aggiungi Voto
3. Modifica Voto
4. Cancella Voto
==========================
Seleziona un'opzione: 1

=== Valutazioni Disponibili ===
ID: 1; Voto: 45; Media: 60,5; ID-Materia: 1; ID-Studente: 1
ID: 2; Voto: 76; Media: 60,5; ID-Materia: 1; ID-Studente: 1
ID: 3; Voto: 45; Media: 45; ID-Materia: 2; ID-Studente: 1
```

```
=== Gestione Materie ===
1. Visualizza Materie
2. Aggiungi Materia
3. Modifica Materia
4. Cancella Materia
====================
Seleziona un'opzione: 1

=== Materie Disponibili ===
ID: 1; Nome: ingegneria del software
ID: 2; Nome: base di dati
ID: 3; Nome: programmazione in C
```

Ogni materia è associata allo stesso ID degli insegnanti, e ogni valutazione ha l'ID dello studente che l'ha preso; in aggiunta è anche presente la media dei voti per ogni materia.

È fondamentale, una volta eseguita qualsiasi modifica, di eseguire l'azione di uscita (digitare 6 dal menù principale) per apportare in modo definitivo le modifiche.