ЧАСТНА ПРОФЕСИОНАЛНА ГИМНАЗИЯ
ПО ДИГИТАЛНИ НАУКИ „СОФТУНИ СВЕТЛИНА“

ДИПЛОМНА РАБОТА

ЗА ПРИДОБИВАНЕ НА ПРОФЕСИОНАЛНА КВАЛИФИКАЦИЯ

НА ТЕМА:

„ЕЛЕКТРОНЕН УЧИЛИЩЕН ДНЕВНИК“

ИЗГОТВИЛ:
Васил Димитров Райчев
ученик от 12В клас в
ЧПГДН "СофтУни Светлина"

НАУЧЕН РЪКОВОДИТЕЛ:
Мариян Йорданов

Дата: 3 май 2023
Сесия: май - юни 2023 г.

Гр. София

Съдържание

Увод

Електронните учебни дневници стават все по-популярен инструмент за оценяване на учениците през последните години. Тези дигитални платформи предлагат централизирано пространство за учителите, за да документират и проследяват напредъка на учениците във времето, предоставяйки ценна информация за ученето и растежа на учениците. Като позволяват на учителите да записват наблюдения, да документират важни етапи и да споделят обратна връзка с ученици и родители, електронните учебни дневници улесняват по-смислена и персонализирана комуникация между всички заинтересовани страни в образователния процес.

Освен това, електронните учебни дневници предлагат по-ефективна и сигурна алтернатива на традиционните системи на хартиен носител, позволявайки лесен достъп до записите на учениците, данните за представянето и оценките. Като такива, електронните учебни дневници представляват мощен инструмент за насърчаване на успеха на учениците и подпомагане на индивидуализираното обучение в съвременната класна стая.

Проблеми

Докато електронните учебни дневници предлагат много предимства, има и някои потенциални проблеми, свързани с тяхното използване. Едно от основните предизвикателства е потенциалът за технологични проблеми, като системни сривове или загуба на данни, които могат да компрометират целостта на записите на учениците. Освен това електронните учебни дневници може да са уязвими за хакерство или пробиви на данни, което може да компрометира сигурността и поверителността на чувствителната информация за учениците.

Друг потенциален проблем с електронните учебни дневници е рискът от пристрастност на учителите. Тъй като учителите имат контрол върху съдържанието на тези платформи, съществува риск те да представят предубедена представа за напредъка или представянето на учениците. Това може да доведе до несправедливи оценки и липса на прозрачност, което може да подкопае ефективността на процеса на оценяване.

Цели на дипломния проект

Целта на настоящата дипломна работа е да се даде началото на изграждането на един добре работещ софтуер, който да улеснява работата на учителите и да помага на учениците по време на тяхното обучение. В момента софтуера е в начален етап на разработка и ще бъде представен на конзолно приложение, тепърва ще предстоят много подобрения по него, основната цел е да бъде изцяло създаден от мен (Васил Райчев) и да бъде пуснат за общо ползване в близката 1 година.

Задачи, произтичащи от целите

- Извършване на проучване и подготовка на обзор на проблемната област
- Избиране на технологиите чрез които ще се изгради софтуерът
- Изграждане на сървърна част
- Изграждане на Backend част
- Изграждане на Frontend част
- Изграждане на приятна визия на софтуера
- Извеждане на изводи и заключения на базата на разработения проект

Глава 1. Проучване

Въведение:

Училищните електронни дневници стават все по-популярни в образователния сектор. С широкото използване на технологиите училищата включват електронни дневници, за да улеснят общуването на ученици и учители, да наблюдават напредъка и да проследяват представянето. Електронните дневници обикновено са уеб-базирани приложения или мобилни приложения, които позволяват на учители и ученици да влизат в системата и да имат достъп до информация за академичния напредък. В това изследване ще проучим разработването на училищен електронен дневник за оценяване на учениците, ползите от него и предизвикателствата, които идват с него.

Предимства:

Електронните дневници имат редица предимства пред традиционните дневници на хартиен носител. Първо, те са много по-ефективни, тъй като позволяват на учителите лесно да наблюдават напредъка на учениците и да предоставят незабавна обратна връзка. Това може да бъде особено полезно при идентифициране на области, в които учениците може да изпитват затруднения и предоставяне на подкрепа, преди проблемите да ескалират. Освен това електронните дневници са много по-сигурни от дневниците на хартиен носител, тъй като са защитени с пароли и могат да бъдат достъпни само от упълномощени лица.

Друго предимство на електронните дневници е, че те могат лесно да бъдат персонализирани, за да отговарят на нуждите на отделните училища. Например едно училище може да избере да използва определена система за проследяване на присъствието на учениците, докато друго училище може да използва различна система за проследяване на забележките и похвалите. Тази гъвкавост означава, че електронните дневници могат да бъдат пригодени да отговарят на специфичните нужди на всяко училище, което може да бъде особено полезно за училища с уникални изисквания.

Разработване:

За да се разработи електронен дневник за оценяване на учениците, има няколко ключови характеристики, които трябва да се вземат предвид. Те включват:

Потребителски интерфейс:

Потребителският интерфейс трябва да бъде лесен за използване и навигация, с ясни инструкции за учители и ученици. Тя трябва да бъде интуитивна и да осигурява бърз достъп до информацията, от която се нуждаят учителите и учениците.

Проследяване на оценките:

Системата трябва да позволява на учителите да проследяват представянето на учениците при оценяване, включително изпити, тестове и задачи. Това ще позволи на учителите бързо да идентифицират областите, в които учениците изпитват трудности, и да предоставят допълнителна подкрепа, ако е необходимо.

Мониторинг на напредъка:

Системата трябва да позволи на учителите да наблюдават напредъка на учениците с течение на времето, включително оценки, присъствие и поведенчески проблеми. Това ще позволи на учителите да идентифицират модели и тенденции и да предоставят подкрепа, когато е необходимо.

Предизвикателства:

Докато електронните дневници имат много предимства, те също така представляват редица предизвикателства. Едно от основните предизвикателства е да се гарантира, че системата е сигурна и защитена от неоторизиран достъп. Това изисква стабилни мерки за сигурност, включително защитни стени, криптиране и удостоверяване на потребителя.

Друго предизвикателство е да се гарантира, че системата е надеждна и може да обработва големи количества данни. Това изисква стабилна инфраструктура, която може

да обработва големи обеми трафик и да осигурява надежден достъп до системата по всяко време.

И накрая, има предизвикателство да се гарантира, че системата е лесна за използване и достъпна за всички ученици, независимо от тяхното ниво на техническа компетентност.

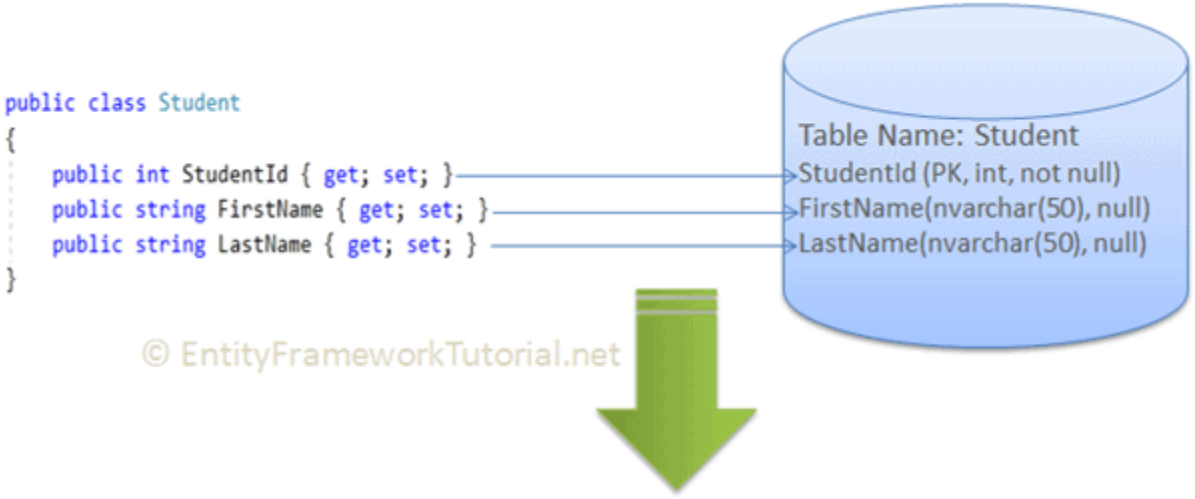
Глава 2. Използвани технологии

За да създадем електронния учебен дневник са ни необходими доста компоненти, които трябва да работят в синхрон за да се постигне безпроблемно извършване на работа. Един от тези компоненти е гореспоменатия Backend, който ще служи за съхранение на цялата информация.

Backend

Какво е и как работи ORM

Първата стъпка при създаването на електронния учебен дневник е изграждането на Backend, в случая той е базиран на ORM (Обектно-релационно картографиране) на Entity Framework, този framework прави C# класовете на таблици в базата данни, това се случва по следния начин:



Фигура 1.1: Class and Database Table

ORM (Обектно-релационно картографиране) е техника, използвана за съпоставяне между обектно-ориентирани езици за програмиране и системи за управление на релационни бази данни. Entity Framework е популярна ORM рамка, използвана в .NET Core приложения. Той предоставя удобен начин за взаимодействие с бази данни, без да се налага да пишете необработен SQL код.

В Entity Framework можете да създавате класове, които представляват таблици в база данни, които се наричат обекти. Тези обекти могат да имат свойства, които се съпоставят с колони в таблицата и връзки с други обекти.

За да използвате Entity Framework в приложение .NET Core, първо трябва да създадете context class на база данни, който наследява от класа DbContext, предоставен от Entity Framework. Този context class дефинира набор от обекти, които могат да се използват за запитване и манипулиране на данни в базата данни.

След като сте дефинирали вашите обекти и контекстен клас, можете да използвате LINQ (езикова интегрирана заявка), за да направите заявка в базата данни. LINQ предоставя начин за писане на заявки на C# или други .NET езици, които след това се превеждат в SQL изрази от Entity Framework.

Какво са и как работят Миграциите

Entity Framework Migrations е функция на рамката Entity Framework ORM (Обектно-релационно картографиране)), която позволява на разработчиците да управляват промените в схемата на базата данни с течение на времето. С Entity Framework Migrations разработчиците могат да създават, актуализират и връщат промените в схемата на базата данни по контролиран и последователен начин, без да се налага да пишат ръчно SQL скриптове или сами да управляват схемата на базата данни.

Работният поток за миграции на Entity Framework се състои от следните стъпки:

Активиране на миграции: Първо, трябва да активирате миграции за вашия проект, като изпълните командата "Enable-Migrations" в конзолата на Package Manager. Това създава папка Migrations във вашия проект и добавя клас Configuration, който управлява процеса на мигриране.

Създаване на миграция: След като миграциите са активирани, можете да създадете нова миграция, като изпълните командата „Add-Migration“ в конзолата на Package Manager. Това генерира нов файл за миграция в папката Migrations, който съдържа промените в схемата на базата данни.

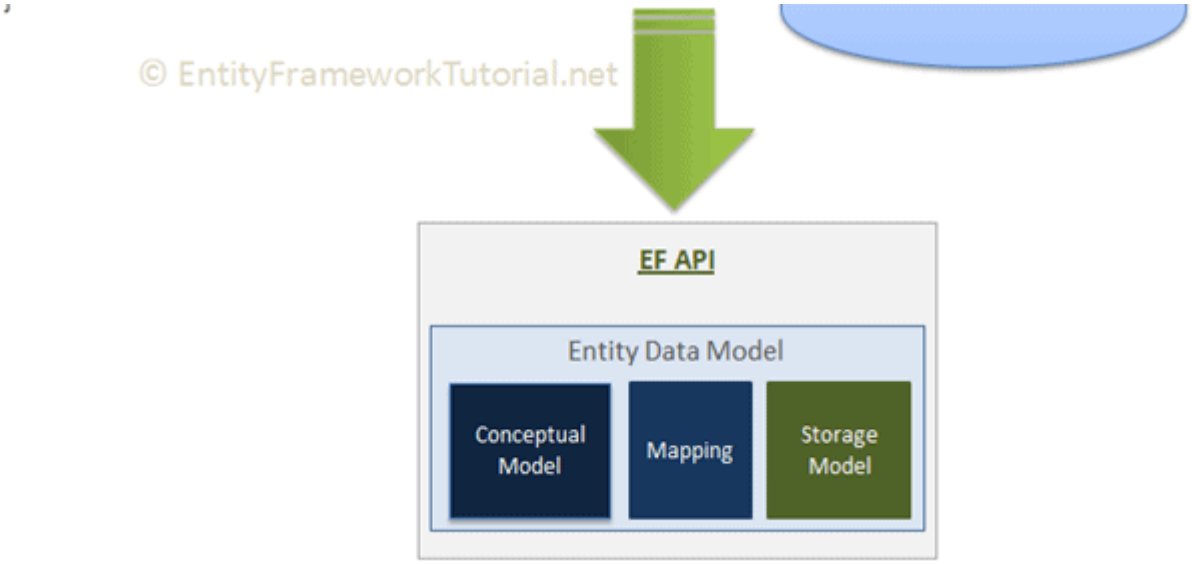
Прилагане на миграцията: След като създадете нова миграция, можете да я приложите към базата данни, като изпълните командата "Update-Database" в конзолата

на Package Manager. Това изпълнява файла за миграция и актуализира схемата на базата данни, за да съответства на промените.

Възстановяване на миграция: Ако трябва да върнете миграция, можете да изпълните командата „Update-Database“ с конкретно име на миграция, за да се върнете към предишна версия на схемата на базата данни.

Всеки файл за миграция съдържа набор от инструкции за модифициране на схемата на базата данни. Тези инструкции се генерират автоматично от Entity Framework въз основа на промените, направени във вашия модел на първо място на кода. Например, ако добавите ново свойство към клас обект, Entity Framework ще генерира миграционен файл, който добавя нова колона към съответната таблица на базата данни.

Миграциите могат също да се използват за актуализиране или зареждане на базата данни с първоначални данни. Например, можете да създадете миграция, която добавя нов запис към таблица или актуализира съществуващ запис с нови данни.



Фигура 1.2: Entity Framework

Защо програмния език C#

Едно от основните предимства на C# е неговата лекота на използване. C# е проектиран да бъде прост и лесен за научаване, дори и за начинаещи. Има ясен и кратък синтаксис, който улеснява писането на чист и четим код. C# идва и с мощен набор от

библиотеки и инструменти, включително .NET Framework и Visual Studio, които помагат на разработчиците да пишат код по-бързо и по-ефективно.

C# е безопасен за типове език, което означава, че всички променливи и изрази се проверяват за съвместимост на типове по време на компилиране. Това помага за улавяне на грешки в началото на процеса на разработка и улеснява поддръжката и отстраняването на грешки в кода. C# също включва функции като автоматично управление на паметта, което освобождава разработчиците от необходимостта да управляват паметта ръчно, намалявайки риска от грешки и увеличавайки продуктивността.

C# е многопарадигмен език, което означава, че поддържа както обектно-ориентирани, така и функционални стилове на програмиране. Това го прави универсален език, който може да се използва за широк спектър от приложения. C# също поддържа модерни функции за програмиране като асинхронно програмиране, което позволява на разработчиците да пишат код, който работи ефективно на модерен хардуер.

Една от ключовите силни страни на C# е неговата интеграция с .NET Framework. .NET Framework е мощна платформа за разработване на приложения, а C# е основният език, използван за разработване на .NET приложения. .NET Framework предоставя богат набор от библиотеки и инструменти, които улесняват изграждането на надеждни, мащабируеми и сигурни приложения.

C# също има силна поддръжка за програмиране на бази данни, което улеснява работата с бази данни като SQL Server и MySQL.

Какво е конзолно приложение

В C# конзолното приложение е програма, която се изпълнява в конзолен прозорец, който е текстово базиран потребителски интерфейс, който позволява на потребителите да взаимодействат с програмата, като въвеждат команди и преглеждат текстово базиран изход. Конзолните приложения често се използват за задачи, които не изискват графичен потребителски интерфейс, като помощни програми за команден ред и приложения от страна на сървъра.

За да създадете конзолно приложение в C#, можете да използвате Visual Studio, което е популярна интегрирана среда за разработка (IDE) за C#. Visual Studio предоставя шаблон за създаване на конзолни приложения, който включва основна програмна структура, която можете да персонализирате, за да отговаря на вашите нужди.

C# конзолните приложения предоставят лесен и ефективен начин за създаване на текстови програми, които могат да се изпълняват от прозорец на конзолата. Те са много подходящи за задачи, които не изискват графичен потребителски интерфейс, и могат да се използват за създаване на помощни програми от командния ред, сървърни приложения и други видове програми.

Глава 3. Проектиране и имплементация

Имплементация на Backend

Имплементацията започва от гореспоменатия Backend. Той ще бъде създаден чрез т.нар Code-First approach, което означава че таблиците в базата данни ще представляват C# класове, които по-късно ще бъдат „преведени“ към SQL таблици. Това „превеждане“ ще бъде направено от Entity Framework чрез Обектно-релационното картографиране, на което се основава целия framework. Тези C# класове служещи като template за SQL таблици са т.нар „Модели“.

Модели

Моделите в Entity Framework се използват за описание на структурата на данните в базата данни, както и за управление на тези данни чрез код. Моделите предоставят абстракция върху базата данни, като позволяват да се работи с обекти, които са по-близо до бизнес логиката на приложението, вместо да се занимаваме директно със записите в базата данни.

Модел AdminsAuthentication

```
3 references
public class AdminsAuthentication
{
    2 references
    public int AdminId { get; set; }

    3 references
    public string AdminUsername { get; set; } = null!;

    3 references
    public string AdminPassword { get; set; } = null!;
}
```

Фигура 2: Class AdminsAuthentication

Този код дефинира клас, наречен AdminsAuthentication, в който има три полета: AdminId, AdminUsername и AdminPassword.

Полето AdminId е int, което представлява уникалния идентификационен номер на администраторския потребител.

Полето AdminUsername е string, което представлява потребителското име на администраторския потребител. The = null! в края на декларацията на полето показва, че това поле е инициализирано като null, но се очаква да му бъде присвоена различна от null стойност в даден момент по време на изпълнението на програмата.

Полето AdminPassword е string, което представлява паролата на администраторския потребител. Подобно на полето AdminUsername, то също се инициализира като null, но се очаква да му бъде присвоена различна от null стойност по време на изпълнение на програмата.

Този код дефинира плана за обект, който съхранява детайлите за удостоверяване на администраторски потребител, включително неговия уникален идентификационен номер, потребителско име и парола.

Модел Grade

```
6 references
public class Grade : IComparable<Grade>
{
    private int gradeNumber;
    private string gradeName;

    15 references
    public int GradeId { get; set; }

    6 references
    public int GradeNumber
    {
        get
        {
            return this.gradeNumber;
        }
        set
        {
            if (value < 1)
            {
                throw new ArgumentException("The lowest grade is 1!");
            }
            if (value > 12)
            {
                throw new ArgumentException("The highest grade is 12!");
            }
            this.gradeNumber = value;
        }
    }
```

Фигура 3.1: Class Grade

Този код дефинира клас Grade, който имплементира интерфейса IComparable. Той има няколко полета, включително GradeId, GradeNumber, GradeName, IsDelete и Students. Полетата GradeNumber и GradeName имат логика за валидиране, за да гарантират, че стойностите са валидни, преди да бъдат зададени.

```
13 references
public string PrintGrade()
{
    return $"{this.GradeNumber} {this.GradeName}";
}

0 references
public int CompareTo([AllowNull] Grade other)
{
    int result = this.GradeNumber.CompareTo(other.GradeNumber);
    if (result == 0)
    {
        result = this.GradeName.CompareTo(other.GradeName);
    }
    return result;
}
```

Фигура 3.2: Class Grade

Методът PrintGrade връща string представяне на номера и името на класа. Методът CompareTo е внедрен, за да позволи сравнение на два обекта Grade. Първо сравнява полетата GradeNumber на двата обекта и ако са равни, сравнява техните полета GradeName.

Модел Student

Класа "Student", който имплементира интерфейса "IComparable<Student>". Класът дефинира полета за различни атрибути на ученика като име, дата на раждане, пол и др.

Полетата имат getter и setter методи, които гарантират, че задаваните стойности отговарят на определени условия. Например свойството "FirstName" не трябва да е нула или празно, първата буква трябва да е главна, трябва да е с дължина поне 2 букви и най-много 20 букви. Също така трябва да съдържа само букви или интервали.

По същия начин полето „DateOfBirth“ трябва да е между 1 и 31, полето „MonthOfBirth“ трябва да е между 1 и 12, а полето „YearOfBirth“ трябва да е между 1900 и 2023.

Полето „Gender“ трябва да бъде или „Male“, или „Female“, а полето „Country“ не трябва да е празно и трябва да отговаря на подобни условия като полето „FirstName“.

Класът също има две допълнителни полета: "StudentId" и "GradeId", които са цели числа и имат getter и setter. Те са създадени с цел да се следи взаимодействието между обект „Student” и обект „Grade”.

И накрая, класът имплементира интерфейса "IComparable<Student>", който изисква класът да имплементира метод "CompareTo". Този метод се използва за сравняване на два обекта "Student".

Модел StudentAuthentication

Класът има две private полета, наречени „studentAuthenticationUsername“ и „studentAuthenticationPassword“, които могат да бъдат достъпни чрез съответните им публични полета „StudentAuthenticationUsername“ и „StudentAuthenticationPassword“.

Има и две публични цели числа, „StudentAuthenticationId“ и „StudentId“. Първото е идентификатор за удостоверяване на ученика, докато второто е идентификатор за ученика, свързан с „StudentAuthentication”.

Полетата „StudentAuthenticationUsername“ и „StudentAuthenticationPassword“ имат настройки, които валидират входните стойности за съответните им полета. Ако проверката е неуспешна, се хвърля изключение с подходящо съобщение за грешка.

Полето "IsDelete" е булево, което може да се използва за маркиране на удостоверяването на ученик за изтриване.

И накрая, има метод с име "PrintStudentAuthentication", който връща форматиран string, съдържащ потребителското име на ученика за удостоверяване. Този метод не променя никакви данни в обекта, той просто връща низово представяне на него.

Модел Subject

```
8 references
public class Subject : IComparable<Subject>
{
    private string subjectName;

    12 references
    public int SubjectId { get; set; }

    13 references
    public int StudentId { get; set; }

    6 references
    public string SubjectName
    {
        get
        {
            return this.subjectName;
        }
        set
        {
            if (string.IsNullOrEmpty(value))
            {
                throw new ArgumentException("Subject cannot be empty!");
            }
            if (!char.IsUpper(value[0]))
            {
                throw new ArgumentException("The first letter on subject name cannot be lower!");
            }
            if (value.Length < 2)
```

Фигура 4: Class Subject

Този код дефинира клас Subject, който имплементира интерфейса IComparable<Subject>. Този интерфейс позволява екземплярите на класа Subject да бъдат сравнени с други екземпляри от същия клас.

Класът Subject има няколко полета, включително SubjectId, StudentId, SubjectName, IsDelete, Marks и Student.

Полето SubjectName има getter и setter. Инсталаторът включва поредица от проверки, за да гарантира, че името на темата отговаря на определени критерии, като например да е дълго поне два знака, започвайки с главна буква, състоящо се само от букви (без цифри или специални знаци) и не надвишаващо 50 знака в дължина. Ако някой от тези критерии не е изпълнен, се хвърля изключение.

Методът PrintSubject() връща полето SubjectName като форматиран string.

Модел Mark

```
7 references
public partial class Mark : IComparable<Mark>
{
    private decimal markLevel;
    private int dateOfAssessment;
    private int monthOfAssessment;
    private int yearOfAssessment;
    4 references
    public int MarkId { get; set; }

    11 references
    public int SubjectId { get; set; }

    3 references
    public decimal MarkLevel
    {
        get
        {
            return this.markLevel;
        }
        set
        {
            if (value < 2)
            {
                throw new ArgumentException("The lowest mark is 2!");
            }
            if (value > 6)
            {
                throw new ArgumentException("The highest mark is 6!");
            }
            this.markLevel = value;
        }
    }
```

Фигура 5: Class Mark

Този код дефинира клас, наречен Mark. Този клас имплементира интерфейса IComparable<Mark>, който позволява сравнение на обекти от този клас с други обекти от същия клас.

Полетата MarkLevel, DateOfAssessment, MonthOfAssessment и YearOfAssessment имат методи за настройка, които валидират входните стойности. Ако стойностите са извън определения диапазон, се хвърля ArgumentException.

Методът CompareTo() сравнява два обекта от класа Mark въз основа на годината на оценяване, месеца на оценяването и датата на оценяването. Ако обектите са с една и съща година на оценка, сравнението се извършва по месеца на оценка, а ако обектите са с еднакъв месец на оценка, сравнението се извършва по датата на оценка. Методът CompareTo() връща целочислена стойност, която показва резултата от сравнението. Ако

резултатът е отрицателен, това означава, че текущият обект е по-малък от другия обект. Ако резултатът е нула, това означава, че двата обекта са равни. Ако резултатът е положителен, това означава, че текущият обект е по-голям от другия обект.

Свързване на Entity Framework с база данни

След създаването на моделите е необходимо да се запазват в база данни. За да могат да бъдат тези запазвани моделите в база данни трябва Entity Framework да се свърже с базата данни(в случая SQL Server).

Class SchoolDiaryContext

SchoolDiaryContext е клас, който наследява DbContext. Този клас трябва да съдържа DbSet пропъртита, които представят таблиците в базата данни. Именно тези DbSet-ове превръщат моделите в таблици от базата данни като им се предостави класа на модела.

```
eferences
lic partial class SchoolDiaryContext : DbContext

    1 reference
    public SchoolDiaryContext()
    {
    }

    0 references
    public SchoolDiaryContext(DbContextOptions<SchoolDiaryContext> options)
        : base(options)
    {
    }
    4 references
    public virtual DbSet<AdminsAuthentication> AdminsAuthentications { get; set; }

    17 references
    public virtual DbSet<Grade> Grades { get; set; }

    10 references
    public virtual DbSet<Mark> Marks { get; set; }

    21 references
    public virtual DbSet<Student> Students { get; set; }

    9 references
    public virtual DbSet<StudentsAuthentication> StudentsAuthentications { get; set; }

    16 references
    public virtual DbSet<Subject> Subjects { get; set; }
```

Фигура 6.1: Class DbContext and DbSets

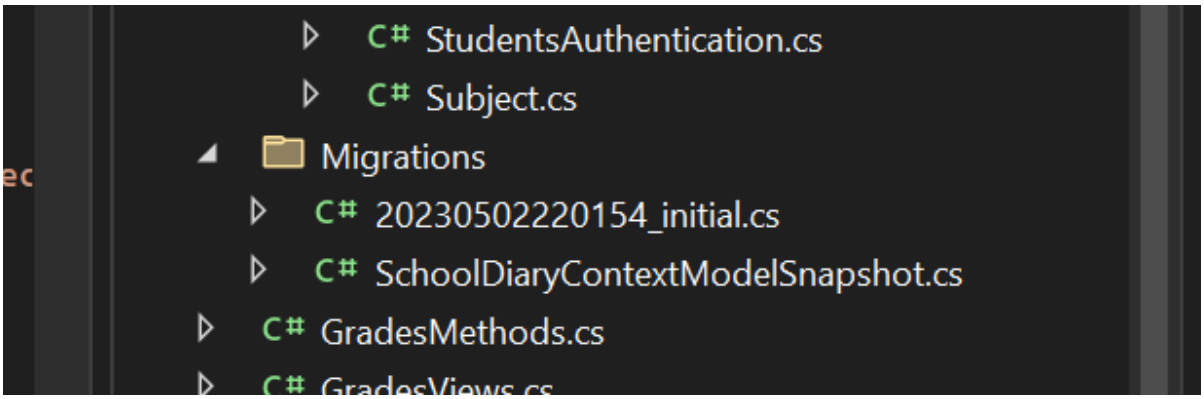
Конфигурираме връзката към базата данни.

```
0 references
protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
    => optionsBuilder.UseSqlServer("Server=(localdb)\\MSSQLLocalDB;Database=School_Diary;Integrated Security=True;");
```

Фигура 6.2: Връзка към SQL Server

Миграция

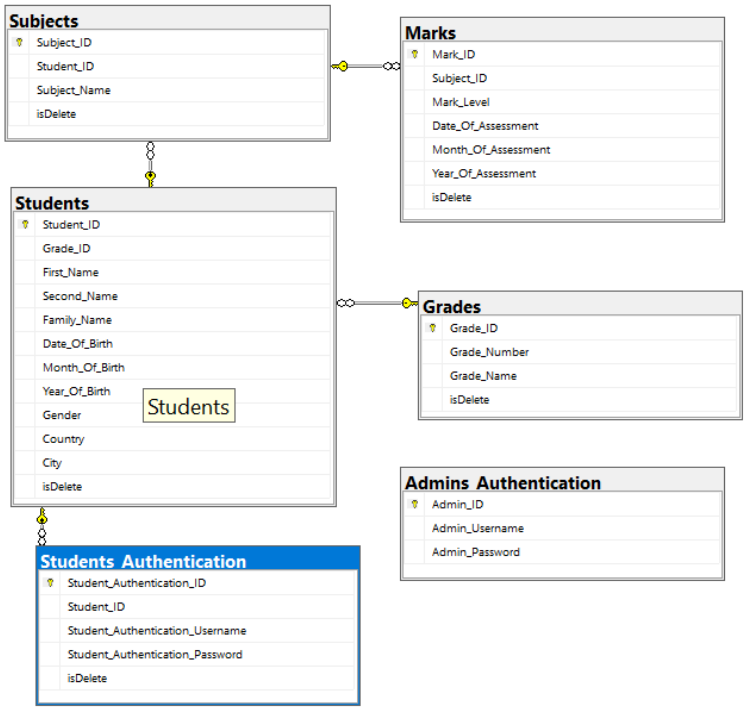
За да създадем първата си миграция трябва да отворим конзолата за мениджър на пакети (Package Manager Console), като трябва да напишем следното в нея: „add-migration initial", така вече създаваме нашата миграция.



Фигура 7: Миграция

След това прехвърлянето към базата данни пак се осъществява чрез конзолата, като трябва да напишем следното в нея: „update-database", така вече нашата миграция е в базата.

База данни



Фигура 8: Диаграма на база данни

Така вече изглежда нашата база данни с която ще работим.

Конзолна апликация

Тъй като проекта е все още в начален етап ще демонстрирам неговата работа с конзолна апликация. Сега ще проследим работата на софтуера от началото до края след като се стартира конзолата.

Login

Кодът започва със създаване на нов екземпляр на класа "SchoolDiaryContext", който е инструмент за обектно-релационно картографиране (ORM) за работа с база данни. След това проверява дали има някакви записи в таблицата „AdminsAuthentications“. Ако няма записи, той създава нов запис за администратор с потребителско име и парола по подразбиране и го записва в базата данни. За момента имаме възможност да се логнем само като администратори или като ученици, разбира се за да се логнем като ученик вече трябва да имаме изграден профил в базата.

```
public class Login
{
    0 references
    static void Main(string[] args)
    {
        SchoolDiaryContext data = new SchoolDiaryContext();
        if (data.AdminsAuthentications.ToList().Count == 0)
        {
            var admin = new AdminsAuthentication();
            admin.AdminUsername = "admin";
            admin.AdminPassword = "admin";
            data.AdminsAuthentications.Add(admin);
            data.SaveChanges();
        }
```

Фигура 9.1: Създаване на админ профил

След това кодът подканва потребителя да въведе своето потребителско име и парола в цикъл, докато не бъдат предоставени валидни идентификационни данни или потребителят избере да излезе. Цикълът преминава през таблицата „StudentsAuthentications“, за да провери дали въведеното потребителско име и парола съвпадат с някой от акаунтите на ученик. Ако се намери съвпадение, той показва оценките и предметите за удостоверения студент. Ако въведените идентификационни данни съвпадат с администраторския акаунт по подразбиране, той показва оценките и предметите за всички ученици.

Ако цикълът завърши без успешно удостоверяване, се хвърля изключение със съобщение за грешка. Накрая приложението показва съобщение за довиждане и излиза.

```
Console.WriteLine("Hello, This is School Diary!");
bool logout = true;
while (logout != false)
{
    Console.WriteLine();
    Console.Write("Username: ");
    string username = Console.ReadLine();
    Console.Write("Password: ");
    string password = Console.ReadLine();
    var allStudentsAuthentications = data.StudentsAuthentications.Where(x => x.IsDelete == false).ToList();
    try
    {
        if (username == data.AdminsAuthentications.FirstOrDefault().AdminUsername && password == data.AdminsAuthentications.FirstOrDefault().AdminPassword)
        {
            Console.Clear();
            bool leave = true;
            while (leave != false)
            {
                var gradesCount = data.Grades.Where(x => x.IsDelete == false).ToList().Count;
                if (gradesCount == 0)
                {
                    GradesViews.GradesEmpty(data, ref leave);
                }
                else
                {
                    GradesViews.GradesNotEmpty(data, ref leave);
                }
            }
            logout = false;
        }
        else if (allStudentsAuthentications.Count != 0)
        {
```

Фигура 9.2: Логин операции

Функционалности на админ

Когато влезем с профил админ ние имаме достъп до всички функционалности на софтуера.

Създаване на класове

В клас GradesMethods се намери метода AddGrade, чрез този метод администът може да създава класове по следни начин:

Създава се нов екземпляр на класа "Grade" и се съхранява в променливата "currentGrade". След това потребителят е подканен да въведе новия номер на класа и име на класа с помощта на конзолата. Въведеното от потребителя се проверява, за да се гарантира, че въведения номер и име на класа не са идентични със съществуващ клас. Ако въведеното е невалидно, се показва подходящо съобщение за грешка и потребителят получава шанс отново да въвежда.

След като бъде въведен валиден вход, свойствата „GradeNumber“ и „GradeName“ на обекта „currentGrade“ се задават на въведеното от потребителя. След това обектът "currentGrade" се добавя към колекцията "Grades" на контекста. Накрая, направените промени в контекста се записват с помощта на метода "SaveChanges".

Триене на класове

В клас GradesMethods се намери метода RemoveGrade, чрез този метод администът може да трие класове по следни начин:

```
    int currentGradeId = allGrades[number - 1].GradeId;
    data.Grades.Where(x => x.GradeId == currentGradeId).FirstOrDefault().IsDelete = true;
    var currentGradeStudents = data.Students
        .Where(x => x.GradeId == currentGradeId && x.IsDelete == false)
        .ToList();
    for (int y = 0; y < currentGradeStudents.Count; y++)
    {
        currentGradeStudents[y].IsDelete = true;
        data.StudentsAuthentications.Where(x => x.StudentId == currentGradeStudents[y].StudentId).FirstOrDefault().IsDelete = true;
        var currentGradeStudentSubjects = data.Subjects
            .Where(x => x.StudentId == currentGradeStudents[y].StudentId && x.IsDelete == false)
            .ToList();
        for (int u = 0; u < currentGradeStudentSubjects.Count; u++)
        {
            currentGradeStudentSubjects[u].IsDelete = true;
            var currentGradeStudentSubjectMarks = data.Marks
                .Where(x => x.SubjectId == currentGradeStudentSubjects[u].SubjectId && x.IsDelete == false)
                .ToList();
            for (int q = 0; q < currentGradeStudentSubjectMarks.Count; q++)
            {
                currentGradeStudentSubjectMarks[q].IsDelete = true;
            }
        }
    }
    data.SaveChanges();
}
```

Фигура 10: Изтриване на клас

Тази част от кода започва с получаване на идентификатора на избраната оценка от списъка с всички оценки. Той прави това, като извлича свойството GradeId на обекта Grade в списъка allGrades, който съответства на избраното число.

След това използва тази стойност currentGradeId, за да намери и маркира свойството IsDelete на съответния обект Grade в контекста на данните като true.

След това кодът извлича списък с всички обекти на ученик в същата оценка, които все още не са маркирани като изтрити. След това преминава през всеки от тези студенти и ги маркира като изтрити, като зададе свойството IsDelete на true. Той също така задава свойството IsDelete на съответния обект StudentAuthentication на true.

За всеки от тези ученици кодът след това извлича списък с всички предметни обекти, които са свързани с този студент и все още не са маркирани като изтрити. След това маркира всички тези обекти Subject като изтрити, като зададе свойството им IsDelete на true.

И накрая, за всеки от тези обекти Subject, кодът извлича списък с всички обекти Mark, които са свързани с този обект и все още не са маркирани като изтрити. След това маркира всички тези обекти за маркиране като изтрити, като зададе свойството им IsDelete на true.

Кодът завършва с извикване на метода SaveChanges в контекста на данните, който записва всички промени, направени в обектите в контекста, в основната база данни.

Останали функционалности

Администратора също така може да създава и трие ученици, което включва и създаването на техните профили с които после те се логват в програмата, може да създава и трие училищни предмети както и да пише оценки на учениците и да ги трие. Начина по който работят тези функции е сходен с начина по който работят функциите които разгледахме по – горе.

```
}
Console.WriteLine("Username");
Console.WriteLine("For example: vasilRaychev");
while (true)
{
    Console.WriteLine("");
    Console.Write("Type: ");
    try
    {
        string username = Console.ReadLine();
        if (username == "admin")
        {
            throw new ArgumentException("Username cannot be admin!");
        }
        for (int i = 0; i < allStudentAuthentication.Count; i++)
        {
            if (username == allStudentAuthentication[i].StudentAuthenticationUsername)
            {
                throw new ArgumentException("There cannot be two identical usernames!");
            }
        }
        currentStudentAuthentication.StudentAuthenticationUsername = username;
        Console.Clear();
        break;
    }
    catch (ArgumentOutOfRangeException)
    {
        Console.WriteLine("");
        Console.WriteLine("Write all information!");
        Console.WriteLine("Try Again!");
    }
    catch (ArgumentException e)
    {
        Console.WriteLine("");
        Console.WriteLine(e.Message);
        Console.WriteLine("Try Again!");
    }
}
```

Фигура 11: Създаване на ученически профил

Функционалности на ученик

Когато влезем с профил ученик, който предварително е създаден от профил админ ние имаме ограничен достъп до функционалностите на софтуера.

Ученика получава достъп само и единствено до своите предмети и до своите оценки без да може да променя нищо по тях. Той също така няма абсолютно никакъв достъп до другите ученици и техните оценки.

```
}
else if (allStudentsAuthentications.Count != 0)
{
    for (int i = 0; i < allStudentsAuthentications.Count; i++)
    {
        if (username == allStudentsAuthentications[i].StudentAuthenticationUsername && password == allStudentsAuthentications[i].StudentAuthenticationPassword)
        {
            Console.Clear();
            bool leave = true;
            while (leave != false)
            {
                var currentStudent = data.Students.Where(x => x.StudentId == allStudentsAuthentications[i].StudentId && x.IsDelete == false).FirstOrDefault();
                var currentStudentSubjectsCount = data.Subjects.Where(x => x.StudentId == currentStudent.StudentId && x.IsDelete == false).ToList().Count;
                if (currentStudentSubjectsCount == 0)
                {
                    SubjectsViews.SubjectsEmptyStudentAuth(currentStudent.StudentId, currentStudent.GradeId, data, ref leave);
                }
                else
                {
                    SubjectsViews.SubjectsNotEmptyStudentAuth(currentStudent.StudentId, currentStudent.GradeId, data, ref leave);
                }
            }
            logout = false;
            break;
        }
    }
}
```

Фигура 12: Логин като ученик

В рамките на цикъла програмата проверява дали потребителското име и паролата, въведени от потребителя, съответстват на съответните стойности в текущия обект StudentAuthentication в списъка. Ако има съвпадение, програмата продължава да изчиства екрана на конзолата и влиза в цикъл while.

В този цикъл while програмата първо извлича информацията за текущия студент от базата данни, използвайки променливата currentStudent. След това отчита броя на неизтритите предмети, свързани със студента, като използва променливата currentStudentSubjectsCount.

Ако броят на предметите е нула, тогава се извиква методът SubjectsEmptyStudentAuth, за да се справи със случая, когато студентът няма предмети. В противен случай се извиква методът SubjectsNotEmptyStudentAuth, за да обработи случая, когато студентът има предмети.

Параметърът ref leave се използва за контрол дали програмата излиза от цикъла while и се връща към главното меню или продължава да показва информацията за предмета за текущия студент.

И накрая, ако се намери съвпадение за въведеното потребителско име и парола, променливата за излизане се настройва на false, за да попречи на програмата да излезе от потребителя и вместо това му позволява да продължи да навигира в програмата. Командата break се използва за излизане от for цикъла, след като бъде намерено успешно съвпадение.

```
1 reference
public static void SubjectsNotEmptyStudentAuth(int currentStudentId, int currentGradeId, SchoolDiaryContext data, ref bool leave)
{
    Console.WriteLine(data.Grades.Where(x => x.GradeId == currentGradeId).FirstOrDefault().PrintGrade());
    Console.WriteLine(data.Students.Where(x => x.StudentId == currentStudentId).FirstOrDefault().PrintStudent());
    Console.WriteLine(data.Students.Where(x => x.StudentId == currentStudentId).FirstOrDefault().PrintStudentInfo());
    Console.WriteLine(data.StudentsAuthentications.Where(x => x.StudentId == currentStudentId).FirstOrDefault().PrintStudentAuthentication());
    Console.WriteLine("ALL SUBJECTS:");
    var allSubjects = data.Subjects.Where(x => x.StudentId == currentStudentId && x.IsDelete == false).ToList();
    allSubjects.Sort();
    for (int i = 0; i < allSubjects.Count; i++)
    {
        Console.WriteLine($"{i + 1}. {allSubjects[i].PrintSubject()}");
    }
    Console.WriteLine("");
    Console.WriteLine("1. Open Subject");
    Console.WriteLine("2. Leave");
```

Фигура 13: Информация за ученика

Този код ни показва как ще изглежда нашата конзола след като влезем като ученик и ученика има наследени предмети в дневника.

Глава 4. Ръководство за потребителя

Проектът е достъпен като сорс код от следното Github хранилище: School_Diary

Системни изисквания

За да стартирате проекта са ви нужни няколко определени системи, които сега ще опишем:

- Операционна система: Windows, Linux или MacOS.
- Инсталиране на Visual Studio
- Инсталиране на SQL Server Management Studio

Стартиране на проекта

След като отворим файла **.sln**, трябва да отворим Package Manager Console и да въведем няколко команди:

- Install-Package Microsoft.EntityFrameworkCore.Tools
- Install-Package Microsoft.EntityFrameworkCore.SqlServer
- Install-Package Microsoft.EntityFrameworkCore.SqlServer.Design
- Update-database

След като приключим с въвеждането на командите вече може да стартираме нашия проект с ctrl + f5.

Заключение

Успешно е завършена първата част от проекта School_Diary, която беше представена като дипломна работа. Използвайки Entity Framework, беше създадена нашата база данни, която позволява на администраторите и учениците да достъпват нужната им информация.

Проектът съдържа множество функционалности като: добавяне, премахване и отваряне на различните нива на нашата база, както и напълно имплементирана автентикационна система.

Има още много работа по проекта за да достигне своя изцяло завършен вид, например създаване на уеб апликация към него с красив и удобен интерфейс. Може да се добавят и допълнителни функционалности, като възможност за отзиви и присъствия на ученика.

Информационни източници

- Entity Framework documentation
- Csharp Help
- Csharp Forums
- Migrations Overview
- Stack Overflow

Рецензия на дипломен проект

Тема на дипломния проект	
Ученик	
Клас	
Професия	
Специалност	
Ръководител- консултант	
Рецензент	

Критерии за допускане до защита на дипломен проект	**Да**	**Не**
Съответствие на съдържанието и точките от заданието		
Съответствие между тема и съдържание		
Спазване на препоръчителния обем на дипломния проект		
Спазване на изискванията за оформление на дипломния проект		
Готовност за защита на дипломния проект		

Силни страни на дипломния проект	
Допуснати основни слабости	
Въпроси и препоръки към дипломния проект	

ЗАКЛЮЧЕНИЕ:

Качествата на дипломния проект дават основание ученикът/ученичката ..... да бъде допуснат/а до защита пред членовете на комисията за подготовка, провеждане и оценяване на изпит чрез защита на дипломен проект- част по теория на професията.

.....05.2023г. Рецензент:.....

Гр. София