להלן מבנה מחלקות ב-**C# עם Entity Framework Code First**, כולל קשרי גומלין ברורים:

**מחלקת Student (תלמיד - נתונים בסיסיים)**

csharp

CopyEdit

public class Student

{

public int Id { get; set; }

public string FirstName { get; set; } = string.Empty;

public string LastName { get; set; } = string.Empty;

public DateTime BirthDate { get; set; }

public string Gender { get; set; }

public string Class { get; set; }

// קשר 1:1 לטבלת פרטי התלמיד הנוספים

public StudentDetails? StudentDetails { get; set; }

// קשר 1 לרבים למסמכים

public ICollection<Document> Documents { get; set; } = new List<Document>();

public DateTime CreatedAt { get; set; } = DateTime.UtcNow;

}

* קשר **1:1** לטבלת פרטי התלמיד (StudentDetails).
* קשר **1 לרבים** לטבלת המסמכים (Documents).

**מחלקת StudentDetails (פרטים נוספים על התלמיד)**

csharp

CopyEdit

public class StudentDetails

{

[Key]

public int StudentId { get; set; } // מפתח ראשי הוא גם מפתח זר

public string Address { get; set; } = string.Empty;

public string Phone { get; set; } = string.Empty;

public string MedicalInfo { get; set; } = string.Empty;

public string AdditionalNotes { get; set; } = string.Empty;

// קשר 1:1 חזרה לתלמיד

public Student Student { get; set; } = null!;

}

* קשר **1:1** מול Student.

**מחלקת Document (מסמכים - תמונת פספורט ות.ז.)**

csharp

CopyEdit

public class Document

{

public int Id { get; set; }

public int StudentId { get; set; } // קשר לסטודנט

[Required]

public string DocumentType { get; set; } = string.Empty; // "passport", "id\_card"

[Required]

public string FileUrl { get; set; } = string.Empty;

public DateTime UploadedAt { get; set; } = DateTime.UtcNow;

// קשר 1 לרבים חזרה לתלמיד

public Student Student { get; set; } = null!;

}

* קשר **רל"ר** (Many-to-One) לתלמיד.

**מחלקת User (משתמש - צוות המוסד)**

csharp

CopyEdit

public class User

{

public int Id { get; set; }

[Required, MaxLength(100)]

public string Name { get; set; } = string.Empty;

[Required, EmailAddress, MaxLength(100)]

public string Email { get; set; } = string.Empty;

[Required]

public string PasswordHash { get; set; } = string.Empty;

[Required]

public string Role { get; set; } = "staff"; // "admin", "staff"

public DateTime CreatedAt { get; set; } = DateTime.UtcNow;

// קשר 1 לרבים לטבלת ההרשאות

public ICollection<Permission> Permissions { get; set; } = new List<Permission>();

}

* לכל משתמש יש **רל"ר** (One-to-Many) מול Permission.

**מחלקת Permission (הרשאות גישה)**

csharp

CopyEdit

public class Permission

{

public int Id { get; set; }

public int UserId { get; set; }

[Required]

public string PermissionType { get; set; } = string.Empty; // "view\_students", "edit\_students", "delete\_students"

public DateTime GrantedAt { get; set; } = DateTime.UtcNow;

// קשר חזרה למשתמש

public User User { get; set; } = null!;

}

* קשר **רל"ר** (Many-to-One) מול User.

**הגדרת ה-DbContext**

csharp

CopyEdit

public class SchoolDbContext : DbContext

{

public DbSet<Student> Students { get; set; }

public DbSet<StudentDetails> StudentDetails { get; set; }

public DbSet<Document> Documents { get; set; }

public DbSet<User> Users { get; set; }

public DbSet<Permission> Permissions { get; set; }

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

// קשר 1:1 בין Student ל-StudentDetails

modelBuilder.Entity<Student>()

.HasOne(s => s.StudentDetails)

.WithOne(sd => sd.Student)

.HasForeignKey<StudentDetails>(sd => sd.StudentId);

// קשר 1 לרבים בין Student ל-Documents

modelBuilder.Entity<Student>()

.HasMany(s => s.Documents)

.WithOne(d => d.Student)

.HasForeignKey(d => d.StudentId);

// קשר 1 לרבים בין User ל-Permissions

modelBuilder.Entity<User>()

.HasMany(u => u.Permissions)

.WithOne(p => p.User)

.HasForeignKey(p => p.UserId);

}

}

**סיכום קשרי הגומלין**

| **קשר** | **מחלקה 1** | **מחלקה 2** | **סוג הקשר** |
| --- | --- | --- | --- |
| 1:1 | Student | StudentDetails | One-to-One |
| 1 לרבים | Student | Document | One-to-Many |
| 1 לרבים | User | Permission | One-to-Many |

**יתרונות המבנה**

✅ **ביצועים גבוהים** - נתוני התלמיד הבסיסיים נפרדים מהנתונים המורחבים.  
✅ **פשטות ואופטימיזציה** - שמירת קשרים נכונים עם שליפות יעילות.  
✅ **ניהול הרשאות גישה ברור** - לכל משתמש יש יכולת שליטה מדורגת.

אילו סוגים לבחור:

**📌 לסיכום: הבחירות הנכונות עבורך**

| **נושא** | **בחירה מומלצת** | **סיבה** |
| --- | --- | --- |
| ארגון קבצי הפרויקט | **Solution עם פרויקטים נפרדים** | הפרדה בין API, מודלים וגישה למסד נתונים |
| מבנה ה-API | **Controllers (MVC API)** | ניהול מסודר של ניתובים, הרשאות ו- Middleware |
| מידול נתונים | **Code First** | שליטה על ישויות, קשרים ומיגרציות |

📌 **🚀 מוכנה לשלב הבא?**  
✔️ רוצים להוסיף אימות JWT?  
✔️ רוצים להוסיף Middleware להגנת API?  
✔️ רוצים לחבר את ה-API לאנגולר וריאקט?

תכנון ההיררכיה:

bash

CopyEdit

SchoolManagement.sln

├── Backend.API/ # פרויקט ה-API (Controllers, Middleware)

│ ├── Controllers/

│ ├── DTOs/

│ ├── Program.cs

│ ├── appsettings.json

│ └── Backend.API.csproj

├── Backend.Core/ # פרויקט הליבה (Entities, Interfaces)

│ ├── Entities/ # מחלקות הנתונים (מודלים)

│ ├── Interfaces/ # ממשקים ל-Repository ול-Services

│ ├── Enums/ # סוגי נתונים קבועים

│ └── Backend.Core.csproj

├── Backend.Infrastructure/ # פרויקט הגישה לנתונים (מימוש ה-Repository)

│ ├── Data/ # DbContext ומיגרציות

│ ├── Repositories/ # מימוש ה-Repository

│ ├── SchoolDbContext.cs # הקשר למסד הנתונים

│ └── Backend.Infrastructure.csproj

├── Backend.Services/ # פרויקט השירותים (לוגיקה עסקית)

│ ├── Services/ # מימוש הלוגיקה העסקית

│ ├── DTOs/ # מודלים להעברת נתונים

│ ├── Validations/ # אימות נתונים

│ └── Backend.Services.csproj

└── Backend.Tests/ # פרויקט בדיקות (Unit & Integration Tests)

├── StudentServiceTests.cs

├── UserServiceTests.cs

└── Backend.Tests.csproj