Entity Framework Implementation

Software Requirements Specifications (SRS)

Requirement gathering is the first step in any software project. There are two main ways to organize these requirements:

1. SRS Document – A formal, detailed document (50–100 pages).
2. Project Management Tool (e.g., JIRA) – Requirements written as user stories.

Project Example

Project Name: TechNova E-Commerce Platform

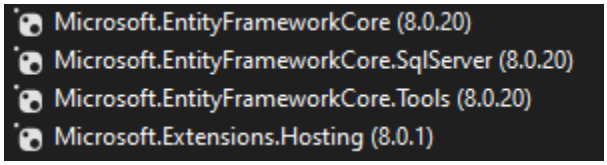
Technology Stack:
- Backend: .NET 8
- Database: Microsoft SQL Server
- ORM: Entity Framework Core

Core Modules We Will Cover:
- User Management
- Product Management
- Cart and Checkout

What is End-to-End?

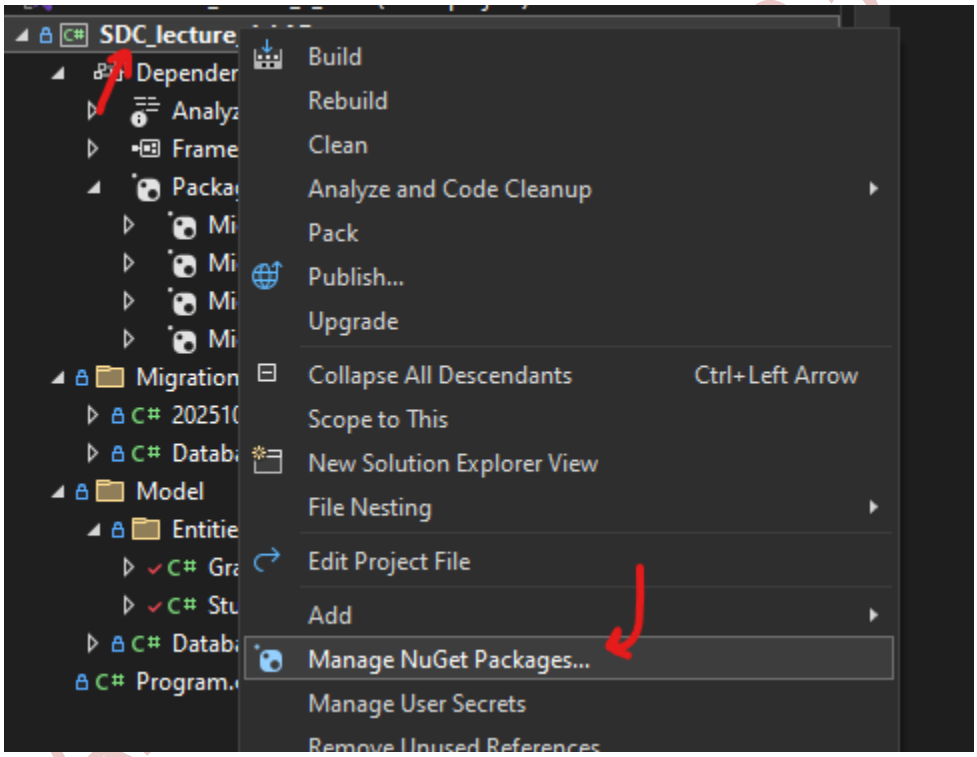
"End-to-end" means the entire process flow from user interaction to backend logic and data storage in the database. Example: Adding or updating students and grades via EF Core.

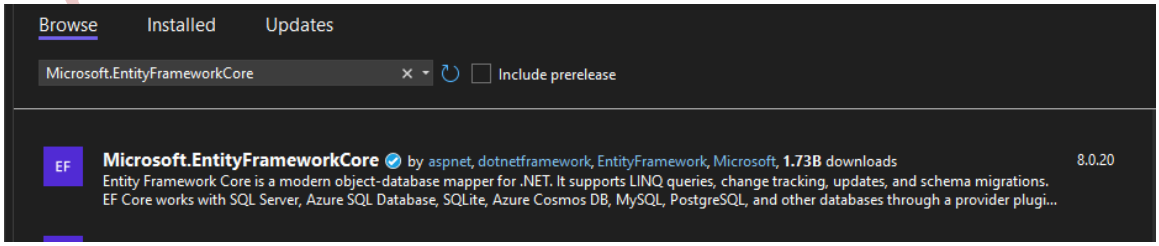
Packages Needed



To install them:

Right Click Solution -> Manage Nuget Packages -> Browse -> Search and install the package with the correct version (If using .Net Core 8 install the latest 8.x.x version)





Entity Framework (EF) Core — Code First Approach

Entities are C# classes that represent database tables. Each property represents a column.

You will be creating a separate class for each table (Entity)

[Table("Students")]

When you create or connect to the database, use a table named Students for this class. Without it EF would automatically name the table Student (same as the class name, or pluralized automatically)

[Column("id")]

Each [Column] attribute specifies the exact column name in the database table that this property should map to.

```
[Table("Students")]
public class Student
{
   [Column("id")]
   [Key]
   [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
   public int Id { get; set; }

   [Column("name")]
   public string Name { get; set; }

   [Column("age")]
   public string Age { get; set; }
}
```

DatabaseContext Class (DbContext)

```
using System;
using System.Collections.Generic;
using System.Data.Common;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Microsoft.EntityFrameworkCore;
using Microsoft.Extensions.Options;
using SDC_lecture_4_LAB.Model.Entities;

namespace SDC_lecture_4_LAB.Model
{
    5 references
    public class DatabaseContext : DbContext
    {
        0 references
        public DatabaseContext(DbContextOptions<DatabaseContext> options) : base(options)
        {
        }

        0 references
        public DbSet<Student> Students { get; set; }
        0 references
        public DbSet<Grade> Grades { get; set; }
    }
}
```

The DbContext manages all communication between C# code and the database.

Entity Classes

Student.cs and Grade.cs represent database tables.

Entity classes are simple C# classes that represent tables in your database.

Each property inside the class represents a column in that table.

So, for example:

```
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations.Schema;
using System.ComponentModel.DataAnnotations;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace SDC_lecture_4_LAB.Model.Entities
{
    [Table("Students")]
    2 references
    public class Student
    {
        [Column("id")]
        [Key]
        [DatabaseGenerated(DatabaseGeneratedOption.Identity)] // For making it identity key
        0 references
        public int Id { get; set; }


        [Column("name")]
        0 references
        public string Name { get; set; }


        [Column("age")]
        0 references
        public string Age { get; set; }


        [Column("grade_id")] // Foreign key (GradeId) links to the Grade table
        0 references
        public string GradeID { get; set; }


        [Column("grade")] //Navigation property - one Student belongs to one Grade
        0 references
        public Grade Grade { get; set; }


    }
}
```

This class tells EF,

Create a database table called Students with columns Id, name, and age etc.

SO when you run a migration, EF reads these entity classes and builds tables accordingly.

What Is ICollection ?

```
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace SDC_lecture_4_LAB.Model.Entities
{
    [Table("Grades")]
    2 references
    public class Grade
    {
        [Column("id")]
        [Key]
        [DatabaseGenerated(DatabaseGeneratedOption.Identity)] // For making it identity key
        0 references
        public int Id { get; set; }

        [Column("grade_name")]
        0 references
        public string GradeName { get; set; }

        0 references
        public ICollection<Student> Students { get; set; } // Navigation property - one Grade can have many Students
    }
}
```

ICollection<T> is a generic collection interface in C#

In EF, it represents a one-to-many relationship between entities.

Other words,

ICollection<Student> means a group (collection) of students.

So,

Grade has a list (ICollection) of Students.

Student has one Grade.

Database Connection and Hosting (Program.cs)

```
using Microsoft.Extensions.DependencyInjection;
using Microsoft.EntityFrameworkCore;
using Microsoft.Extensions.Hosting;
using SDC_lecture_4_LAB.Model;

//Data Source=(local);Initial Catalog=School-DB;Integrated Security=True;Encrypt=False


using var host = Host.CreateDefaultBuilder(args)
    .ConfigureServices((context, services) =>
    {
        services.AddDbContext<DatabaseContext>(options =>
        options.UseSqlServer("Data Source=(local);Initial Catalog=School-DB;Integrated Security=True;Encrypt=False"));

    })
    .Build();
```

This initializes your project and connects everything

you can now use DatabaseContext anywhere in your code to:

- Add new records
- Read data
- Update data
- Delete data

SQL Server Setup

1. Open SQL Server Management Studio (SSMS) and connect to (local). If using on personal desktop use your desktop name (dropdown -> browse -> you'll see your desktop name)

2. Create database by right clicking on the database folder and new database.

3. In Visual Studio -> View -> Server Explorer -> Connect to Database -> Microsoft SQL Server -> if you get a dialog just press yes to install missing packages and install using visual studio installer -> Put Encrypt=False -> Server name would be (local) or your desktop name -> database name is the name of the database you created.

4. Once connected, right click the connection -> properties -> copy and paste the connection string in Program.cs as a comment (make sure the DATABASE NAME has no spaces)

Database Migration Steps (EF Core)

Open View -> Other windows -> Package Manager Console.

In the console below run:

add-migration (migration name example AddingTableForOwners)

Then run:

update-database -verbose

(MAKE SURE YOU DON'T HAVE ANY SOLUTION ERRORS!

IF YOU DO FIX AND DO THE MIGRATION AGAIN!)

Key .NET Concepts

- Dependency Injection (DI) – Provides class dependencies automatically.
- Lifetime – Defines service instance duration.
- Request-Response Pipeline – Handles requests via middleware.
- Host Process – Manages app lifecycle and configuration.

NOTE:

In our example we are manually doing the hosting with

using var host = Host.CreateDefaultBuilder(args)

Steps:

1. Create a “Model” folder inside your project
2. Create “Entities” folder inside “Model” folder
3. Create classes (entities) inside the “Entities” folder
4. Inside the “Model” folder create a DatabaseContext.cs file
5. Setup SQL server and Database
6. Connect and do migrations