Week 4 – Day 2 Report **Date:** 31-10-2025
Module: Full Stack Project – Student Management System

1. Objective of the Day

The goal for Day 2 was to **connect the ASP.NET Core Web API backend with the SQL Server database** using the **Entity Framework Core (Database-First approach)**.
The main aim was to generate models from the existing SQL database and validate the data communication between the backend and SQL Server through the API.

2. Topics Covered / Tasks Completed

a. Entity Framework Core Integration

- Installed the necessary NuGet packages for EF Core:
    - Microsoft.EntityFrameworkCore.SqlServer
    - Microsoft.EntityFrameworkCore.Tools
    - Microsoft.EntityFrameworkCore.Design

b. Database-First Model Generation

- Opened **Package Manager Console** in Visual Studio.
- Used the **Scaffold-DbContext** command to generate models and DbContext from the existing SQL database:

Scaffold-DbContext
"Server=YOUR_SERVER_NAME;Database=StudentManagementDB;Trusted_Connection=True;TrustServerCertificate=True;" Microsoft.EntityFrameworkCore.SqlServer -OutputDir Models

This command automatically generated:

- ApplicationDbContext.cs (database context class)
- Student.cs and Course.cs (entity models)

3. Generated Model Files

Student.cs

```
namespace StudentApi.Models
{
  public partial class Student
  {
    public int Id { get; set; }
```

```
        public string Name { get; set; } = null!;
        public int Age { get; set; }
        public string Grade { get; set; } = null!;
        public int CourseId { get; set; }
        public virtual Course? Course { get; set; }
    }
}
```

Course.cs

```
namespace StudentApi.Models
{
    public partial class Course
    {
        public int CourseId { get; set; }
        public string CourseName { get; set; } = null!;
        public virtual ICollection<Student> Students { get; set; } = new List<Student>();
    }
}
```

4. ApplicationDbContext Configuration

ApplicationDbContext.cs

```
using Microsoft.EntityFrameworkCore;
namespace StudentApi.Models
{
    public partial class ApplicationDbContext : DbContext
    {
        public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options)
            : base(options)
        {
        }
```

```
    public virtual DbSet<Student> Students { get; set; }
    public virtual DbSet<Course> Courses { get; set; }
    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<Student>(entity =>
        {
            entity.HasOne(d => d.Course)
                .WithMany(p => p.Students)
                .HasForeignKey(d => d.CourseId);
        });
    }
  }
}
```

5. Dependency Injection Configuration

In Program.cs, the database context was registered for dependency injection:

```
using Microsoft.EntityFrameworkCore;
using StudentApi.Models;
var builder = WebApplication.CreateBuilder(args);
builder.Services.AddControllers();
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen();
// Database connection
builder.Services.AddDbContext<ApplicationDbContext>(options =>
   options.UseSqlServer(builder.Configuration.GetConnectionString("DefaultConnection")));
var app = builder.Build();
if (app.Environment.IsDevelopment())
{
```

```
    app.UseSwagger();
    app.UseSwaggerUI();
}
app.UseHttpsRedirection();
app.UseAuthorization();
app.MapControllers();
app.Run();
```

6. Database Connectivity Test

- Ran the application (dotnet run)
- Swagger UI opened at: http://localhost:5205/swagger/index.html
- Confirmed that the database connection was successful.
- Tested initial **GET request** to verify that the Student table data loads correctly.

7. Challenges Faced

- Initially encountered connection string issues due to TrustServerCertificate not being set to True.
- Solved by updating the connection string in appsettings.json.
- Minor issue with namespaces when auto-generating EF models, fixed by adding using StudentApi.Models; in the controller.