# ASP.NET Core Web API - Integrating Entity Framework Core (Code First)

## 📌 Steps to Integrate EF Core with Database (Code First Approach)

### 📝 Step 1: Install Required Packages

Run the following commands in the terminal to install EF Core and its SQL Server provider:

dotnet add package Microsoft.EntityFrameworkCore  
dotnet add package Microsoft.EntityFrameworkCore.SqlServer  
dotnet add package Microsoft.EntityFrameworkCore.Tools

### 📝 Step 2: Configure the Database Context

Create a new `Data/AppDbContext.cs` file:

using Microsoft.EntityFrameworkCore;  
using MyWebAPI.Models;  
  
namespace MyWebAPI.Data  
{  
 public class AppDbContext : DbContext  
 {  
 public AppDbContext(DbContextOptions<AppDbContext> options) : base(options) { }  
  
 public DbSet<Product> Products { get; set; }  
 }  
}

### 📝 Step 3: Configure Connection String in appsettings.json

Open `appsettings.json` and add your database connection string:

"ConnectionStrings": {  
 "DefaultConnection": "Server=.;Database=ProductDB;Trusted\_Connection=True;TrustServerCertificate=True;"  
}

### 📝 Step 4: Register EF Core in Program.cs

Modify `Program.cs` to register the `AppDbContext` with dependency injection:

var builder = WebApplication.CreateBuilder(args);  
  
// Add services to the container  
builder.Services.AddControllers();  
builder.Services.AddEndpointsApiExplorer();  
builder.Services.AddSwaggerGen();  
  
// Configure EF Core  
builder.Services.AddDbContext<AppDbContext>(options =>  
 options.UseSqlServer(builder.Configuration.GetConnectionString("DefaultConnection")));  
  
var app = builder.Build();  
  
app.UseHttpsRedirection();  
app.UseAuthorization();  
app.MapControllers();  
app.UseSwagger();  
app.UseSwaggerUI();  
  
app.Run();

### 📝 Step 5: Update ProductController to Use Database

Modify `ProductController.cs` to fetch and modify data using EF Core:

using Microsoft.AspNetCore.Mvc;  
using MyWebAPI.Data;  
using MyWebAPI.Models;  
using System.Collections.Generic;  
using System.Linq;  
using System.Threading.Tasks;  
using Microsoft.EntityFrameworkCore;  
  
namespace MyWebAPI.Controllers  
{  
 [Route("api/products")]  
 [ApiController]  
 private readonly AppDbContext \_context;

public ProductsController(AppDbContext context)

{

\_context = context;

}

[HttpGet]

public ActionResult<IEnumerable<Product>> GetAllProducts()

{

return \_context.Products.ToList();

}

[HttpGet("{id}")]

public ActionResult<Product> GetProduct(int id)

{

var product= \_context.Products.Find(id);

if(product == null)

{

return NotFound();

}

return product;

}

[HttpPost]

public ActionResult<Product> AddProduct(Product product)

{

\_context.Products.Add(product);

\_context.SaveChanges();

return CreatedAtAction(nameof(GetProduct), new {id=product.Id},product);

}

[HttpPut("{id}")]

public ActionResult UpdateProduct(int id, [FromBody] Product updatedProduct)

{

var product = \_context.Products.Find(id);

if (product == null)

return NotFound("Product not found");

product.Name = updatedProduct.Name;

product.Price = updatedProduct.Price;

\_context.SaveChangesAsync();

return NoContent();

}

[HttpDelete("{id}")]

public ActionResult DeleteProduct(int id)

{

var product = \_context.Products.Find(id);

if (product == null)

return NotFound("Product not found");

\_context.Products.Remove(product);

\_context.SaveChanges();

return NoContent();

}

}

### 📝 Step 6: Apply Migrations and Create Database

Run the following commands to apply migrations and create the database:

dotnet ef migrations add InitialCreate  
dotnet ef database update

## 📌 Summary

✅ Entity Framework Core (EF Core) integrated using Code First approach

✅ Database context (`AppDbContext`) added and registered in `Program.cs`

✅ ProductController modified to interact with the database

✅ Migrations applied to create the database