Step-by-step Guide — Create an ASP.NET Core Web API project with EF Core (Code-first) using SQL Server / LocalDB

0 — Project structure (what we'll end up with)

```
MyEcomApi/
■■ Controllers/
■ ■■ ProductsController.cs
■■ Data/
■ ■■ AppDbContext.cs
■■ Models/
■ ■■ Product.cs
■■ appsettings.json
■■ Program.cs
```

---

1 — Create the project

```
dotnet new webapi -n MyEcomApi
cd MyEcomApi
```

2 — Add EF Core packages

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

(Optional) Install dotnet ef tool:

```
dotnet tool install --global dotnet-ef
```

---

3 — Add a model (Models/Product.cs)

```
public class Product
{
public int Id { get; set; }
public string Name { get; set; }
public string Description { get; set; }
public decimal Price { get; set; }
}
```

---

4 — Create DbContext (Data/AppDbContext.cs)

```
public class AppDbContext : DbContext
{
public AppDbContext(DbContextOptions options) : base(options) { }
public DbSet Products { get; set; }
}
```

---

5 — Add connection string (appsettings.json)

```
"ConnectionStrings": {
"DefaultConnection": "Server=(localdb)\MSSQLLocalDB;Database=MyEcomDb;Trusted_Connection=True;TrustServerCertificate=True;"
}
```

---

6 — Register DbContext (Program.cs)

```
builder.Services.AddDbContext(options =>
options.UseSqlServer(builder.Configuration.GetConnectionString("DefaultConnection")));
```

---

7 — Create initial migration

```
dotnet ef migrations add InitialCreate
```

8 — Apply migration

```
dotnet ef database update
```

---

9 — Create ProductsController (Controllers/ProductsController.cs)

```
[ApiController]
[Route("api/[controller]")]
public class ProductsController : ControllerBase
{
private readonly AppDbContext _context;
public ProductsController(AppDbContext context) => _context = context;

[HttpGet]
public async Task>> GetAll() =>
await _context.Products.ToListAsync();

[HttpGet("{id}")]
public async Task> Get(int id)
{
var product = await _context.Products.FindAsync(id);
if (product == null) return NotFound();
return product;
}

[HttpPost]
public async Task> Create(Product product)
{
_context.Products.Add(product);
await _context.SaveChangesAsync();
return CreatedAtAction(nameof(Get), new { id = product.Id }, product);
}

[HttpPut("{id}")]
public async Task Update(int id, Product product)
{
if (id != product.Id) return BadRequest();
_context.Entry(product).State = EntityState.Modified;
await _context.SaveChangesAsync();
return NoContent();
}

[HttpDelete("{id}")]
public async Task Delete(int id)
{
```

```
var product = await _context.Products.FindAsync(id);
if (product == null) return NotFound();
_context.Products.Remove(product);
await _context.SaveChangesAsync();
return NoContent();
}
}
```

---

10 — Run the app and test

```
dotnet run
```

Swagger UI available at: https://localhost:xxxx/swagger

---

11 — Optional: Scaffold existing DB

```
dotnet ef dbcontext scaffold
"Server=(localdb)\MSSQLLocalDB;Database=ExistingDb;Trusted_Connection=True;"
Microsoft.EntityFrameworkCore.SqlServer -o Models -c ExistingDbContext
```

---

12 — Troubleshooting tips

- Ensure `Microsoft.EntityFrameworkCore.Design` is installed
- Ensure `dotnet-ef` tool is available
- Check LocalDB/SQL Server instance is running
- Run migrations after model changes

---

13 — Next steps

- Add DTOs & Automapper
- Add validation (FluentValidation)
- Add authentication/authorization (JWT)
- Add paging/filtering to endpoints
- Write tests and deploy