1️⃣ Builder Design Pattern (في ASP.NET Core Startup/Program.cs)

الـ Builder هنا مش الـ Builder pattern الكلاسيكي في الـ GoF، لكن الفكرة قريبة.

في Program.cs، الكود:

```
var builder = WebApplication.CreateBuilder(args);
```

بيديك كائن Builder مسؤول عن تجميع كل الإعدادات (Services, Middleware, Configurations).

هو اللي بيبني التطبيق النهائي بعد ما تخلص إضافة الخدمات.

زي ما بتجهز بيت:

Builder = المقاول اللي يجهز التصميم.

WebApplication = البيت النهائي الجاهز للسكن.

2️⃣ Controller Factory

في MVC، لما يجي Request زي /Products/Details/1، النظام محتاج ينشئ Controller مناسب.

هنا بييجي دور Controller Factory:

يحدد أي Controller يستدعى.

ينشئ نسخة منه (عن طريق الـ DI Container).

يعني بدل ما تكتب new ProductsController() يدويًا، الـ Factory بيعملها أوتوماتيك، وبيحقن Dependencies.

3️⃣ builder.Services.AddControllersWithViews();

دي Extension Method مهمة.

بتقول للـ Builder: "ضيف دعم للـ MVC Pattern (Controllers + Views)".

من غيرها المشروع مش هيعرف يرندر Views أو يتعامل مع Controllers.

داخليًا:

بيسجل الـ MVC Middleware.

بيسجل الـ Routing (Attribute + Conventional).

بيسجل الـ View Engine (Razor).

4️⃣ DI (Dependency Injection)

مفهوم أساسي في ASP.NET Core.

بدل ما الكود جوه Controller يعمل:

```
var service = new ProductService();
```

النظام بيحقنها أوتوماتيك عن طريق Constructor Injection.

مثال:

```
public class ProductsController : Controller
{
    private readonly IProductService _service;
    public ProductsController(IProductService service)
    {
        _service = service;
    }
}
```

علشان تشتغل: لازم تسجل الخدمة في الـ Builder:

```
builder.Services.AddScoped<IProductService, ProductService>();
```

كده لما تعمل Request، الـ Controller Factory يستدعي الـ Controller ويحقنله IProductService.

📌 الخلاصة:

Builder: يجهز البيئة والخدمات.

Controller Factory: ينشئ الـ Controller المناسب.

AddControllersWithViews(): تفعل دعم MVC.

DI: يحل مشكلة الـ new ويوفر مرونة واختبارية عالية.