|  |  |
| --- | --- |
| **ASP.NET Web App MVC** | là một mô hình phát triển ứng dụng web trong ASP.NET dựa trên kiến trúc Model-View-Controller (MVC). Đây là một cách tiếp cận phổ biến để xây dựng các ứng dụng web có cấu trúc rõ ràng, dễ bảo trì và mở rộng.  MVC là một mô hình thiết kế phần mềm chia ứng dụng thành 3 thành phần chính:   * Model: Chứa dữ liệu và logic xử lý dữ liệu (thường kết nối với cơ sở dữ liệu). * View: Chịu trách nhiệm hiển thị giao diện người dùng. * Controller: Xử lý yêu cầu từ người dùng, cập nhật Model và chọn View phù hợp để hiển thị.   Khi người dùng gửi một yêu cầu HTTP (ví dụ: truy cập một trang web), luồng xử lý trong ASP.NET MVC như sau:   * Người dùng gửi yêu cầu ➝ Trình duyệt gửi HTTP Request. * Bộ định tuyến (Routing) xác định Controller phù hợp dựa trên URL. * Controller xử lý logic và gọi Model để lấy dữ liệu (nếu cần). * Model truy xuất dữ liệu từ cơ sở dữ liệu và trả về Controller. * Controller chọn View phù hợp và truyền dữ liệu vào View. * View hiển thị dữ liệu và trả kết quả về trình duyệt. |
| Razor | **Razor** là **một template engine**, cho phép kết hợp **HTML + C#** để tạo giao diện web. ✅ **Chạy trên server**, render HTML hoàn chỉnh rồi gửi về trình duyệt. ✅ **Dùng trong ASP.NET Core MVC và Razor Pages**.  📌 **Cách hoạt động:** 1️⃣ **Server xử lý Razor (**.cshtml**)** → 2️⃣ **Chuyển Razor thành HTML hoàn chỉnh** → 3️⃣ **Gửi HTML về trình duyệt để hiển thị.** |
| Blazor | ✅ **Blazor** là **một framework UI** giúp xây dựng web bằng **C# thay vì JavaScript**. ✅ Hỗ trợ **SPA (Single Page Application)** như React/Vue nhưng dùng C#. ✅ Có 2 phiên bản chính:  **Blazor Server** → Chạy trên **server**, cập nhật UI qua **SignalR**.  **Blazor WebAssembly** → Chạy **hoàn toàn trên trình duyệt** bằng WebAssembly.    Hiểu đơn giản là chỉ cần code gì đó ở phía server thì có thể cho đoạn code đó dòng lện để server có thể tự gửi dữ liệu cần cập nhật lên cho người dùng mà không cần người dùng gửi request để có thể cập nhật dữ liệu |
| **ADO.Net** |  |
| File Program.cs | trong ASP.NET Core MVC, Program.cs là điểm khởi động chính của ứng dụng. Nó chịu trách nhiệm cấu hình, khởi chạy web server, đăng ký các dịch vụ (Dependency Injection), và thiết lập pipeline xử lý HTTP requests. |
| ILogger | là một **giao diện (interface) logging** trong ASP.NET Core, giúp ghi lại **log hoạt động, lỗi, cảnh báo,...** trong ứng dụng. 🔹 **Tích hợp sẵn trong ASP.NET Core**, hỗ trợ nhiều loại **provider** (Console, File, Database, Cloud, v.v.). 🔹 **Dependency Injection (DI)** cung cấp ILogger<T> mà không cần khởi tạo thủ công.  - hiểu đơn giản thì lớp này giúp ghi lại các lỗi và có thể gọi lớp này để in lỗi ra màn hình console |
| IActionResult | là **một interface** đại diện cho **mọi kiểu kết quả trả về từ Controller**. 🔹 Giúp Controller **linh hoạt hơn**, có thể trả về nhiều loại kết quả khác nhau như: ✔ **View (HTML):** return View(); ✔ **JSON:** return Json(data); ✔ **Redirect:** return RedirectToAction("Index"); ✔ **Trạng thái HTTP:** return StatusCode(404); |
| ActionResult | là **một class kế thừa** IActionResult, nhưng **chỉ dùng cho kết quả dạng View**. 🔹 **Không linh hoạt bằng** IActionResult, chủ yếu dùng khi chỉ cần trả về **View hoặc PartialView**. |
| NonController | → Đánh dấu **một class KHÔNG phải là Controller**, tránh bị truy cập từ trình duyệt. |
| NonAction | → Đánh dấu **một phương thức KHÔNG phải là action**, ngăn không cho truy cập từ URL.  **Dùng khi một method trong Controller không phải là action, chỉ là một method nội bộ.** |
|  |  |
|  |  |
| **WebApplicationBuilder** | WebApplicationBuilder là một lớp trong ASP.NET Core được sử dụng để cấu hình và khởi tạo một ứng dụng web. Nó giúp bạn:  ✔ Cấu hình môi trường chạy ứng dụng (Development, Production, Staging). ✔ Đăng ký các dịch vụ (Dependency Injection - DI) như MVC, Entity Framework, Authentication. ✔ Thiết lập máy chủ web (Kestrel hoặc IIS). ✔ Đọc các cấu hình từ appsettings.json.  **Cách WebApplicationBuilder hoạt động**  📌 Khi chạy WebApplication.CreateBuilder(args), ASP.NET Core thực hiện các bước sau:  Tạo đối tượng WebApplicationBuilder.  Đọc cấu hình ứng dụng (appsettings.json, biến môi trường, args).  Khởi tạo hệ thống Dependency Injection (builder.Services).  Thiết lập máy chủ web (builder.WebHost).  Tạo ứng dụng (var app = builder.Build();).  **WebApplicationBuilder giúp cấu hình và khởi động ứng dụng ASP.NET Core.** ✔ Có 3 thành phần chính: builder.Configuration, builder.Services, builder.WebHost. ✔ Cho phép đăng ký dịch vụ, thiết lập máy chủ web, và quản lý cấu hình.  🚀 Nếu bạn làm việc với ASP.NET Core,  WebApplicationBuilder là phần quan trọng giúp ứng dụng hoạt động! 💡 |
| **CreateBuilder** | là một phương thức **tĩnh (static)** của lớp WebApplication trong **ASP.NET Core**, dùng để **tạo một đối tượng** WebApplicationBuilder.   CreateBuilder **làm những gì?** Khi gọi WebApplication.CreateBuilder(args);, ASP.NET Core thực hiện **các bước sau**: **1. Thiết lập cấu hình ứng dụng (**Configuration**)** ✔ Đọc cấu hình từ:   * appsettings.json * appsettings.{Environment}.json * Biến môi trường (Environment Variables) * Tham số dòng lệnh (Command Line Arguments)    **Đăng ký hệ thống Dependency Injection (**Services**)** ✔ Tạo một **DI Container** để quản lý các **dịch vụ (services)** như:   * **MVC (**AddControllersWithViews()**)** * **Entity Framework Core (**AddDbContext()**)** * **Authentication, Logging, Caching**    **3. Cấu hình máy chủ Web (**WebHost**)** ✔ CreateBuilder tự động thiết lập máy chủ web **Kestrel** hoặc **IIS**. ✔ Hỗ trợ **cấu hình Web Server** tùy chỉnh.   **2️⃣ Luồng hoạt động của** CreateBuilder 📌 **Khi gọi** WebApplication.CreateBuilder(args);**, ASP.NET Core thực hiện các bước sau**:   1. **Tạo đối tượng** WebApplicationBuilder. 2. **Đọc cấu hình ứng dụng (**appsettings.json**, biến môi trường, args)**. 3. **Khởi tạo hệ thống Dependency Injection (**builder.Services**)**. 4. **Thiết lập máy chủ web (**builder.WebHost**)**. 5. **Tạo ứng dụng (**var app = builder.Build();**)**. |
| builder.Services | là một **thuộc tính của** WebApplicationBuilder, cung cấp một **Dependency Injection (DI) container** trong ASP.NET Core. Nó được dùng để **đăng ký và quản lý các dịch vụ** trong ứng dụng.   **1️⃣** builder.Services **hoạt động như thế nào?** ASP.NET Core sử dụng **Dependency Injection (DI)** để quản lý các dịch vụ.   * **Dịch vụ** là các đối tượng được tạo một lần và có thể tái sử dụng. * Khi bạn **đăng ký một dịch vụ trong** builder.Services, ASP.NET Core sẽ tự động **cung cấp nó** cho các thành phần cần dùng.    **2️⃣ Các phương thức đăng ký dịch vụ** Có 3 cách chính để đăng ký dịch vụ trong builder.Services:   1. AddSingleton<T>() – Dịch vụ tồn tại **xuyên suốt vòng đời ứng dụng**. 2. AddScoped<T>() – Dịch vụ tồn tại **trong mỗi request HTTP**. 3. AddTransient<T>() – Dịch vụ được **tạo mới mỗi lần gọi**.     builder.Services **là nơi đăng ký dịch vụ (Dependency Injection) trong ASP.NET Core.** ✔ **Giúp quản lý Controller, Database, Authentication, Logging,...** ✔ **Có 3 loại dịch vụ chính:** Singleton**,** Scoped**,** Transient**.** ✔ **Là phần quan trọng giúp ứng dụng ASP.NET Core mở rộng linh hoạt và dễ bảo trì.**  🚀 **Tóm lại, nếu bạn làm việc với ASP.NET Core,** builder.Services **là nền tảng giúp ứng dụng chạy mượt mà!** 💡  Vì sao phải đăng ký Repository trong ASP.NET Core?  **Trong Java Servlet**, bạn có thể **tạo trực tiếp đối tượng Repository** bằng new. 🔹 **Trong ASP.NET Core**, bạn cần **đăng ký Repository vào** builder.Services **trước khi sử dụng**. |
| AddControllersWithViews | là một phương thức trong **ASP.NET Core**, dùng để **đăng ký MVC (Model-View-Controller)** vào hệ thống **Dependency Injection (DI)**.  🔹 **Chức năng chính:** ✔ **Hỗ trợ Controller để xử lý request** ✔ **Hỗ trợ View (**.cshtml**) để hiển thị giao diện** ✔ **Cho phép Routing để điều hướng URL** |
| Build() |  |
|  | **Kiểm tra nếu ứng dụng KHÔNG chạy ở môi trường Development**, thì: ✔ **Chuyển hướng lỗi về** /Home/Error để tránh lộ thông tin lỗi ra ngoài. ✔ **Bật HSTS (**HTTP Strict Transport Security**)** để tăng cường bảo mật cho HTTPS.  📌 **Tóm lại:** Đoạn code này giúp ứng dụng **bảo mật hơn trong môi trường Production** bằng cách xử lý lỗi và buộc sử dụng HTTPS. 🚀 **1️⃣** app.Environment.IsDevelopment() **là gì?** ✔ app.Environment.IsDevelopment() kiểm tra xem ứng dụng có đang chạy trong môi trường **Development** không. ✔ Nếu true, ứng dụng đang chạy trên máy lập trình viên (DEV). ✔ Nếu false, ứng dụng đang chạy trên **Production hoặc Staging**.  📌 **Môi trường ứng dụng được xác định bằng** ASPNETCORE\_ENVIRONMENT, có thể là:   * Development (Môi trường phát triển) * Staging (Kiểm thử trước khi lên Production) * Production (Môi trường thực tế)    **2️⃣** app.UseExceptionHandler("/Home/Error") **là gì?** ✔ UseExceptionHandler("/Home/Error") giúp **xử lý lỗi** trong Production. ✔ Nếu xảy ra lỗi, người dùng sẽ được **chuyển hướng** đến /Home/Error. ✔ Giúp tránh lộ thông tin lỗi nhạy cảm ra ngoài.   **3️⃣** app.UseHsts(); **là gì?** ✔ **HSTS (HTTP Strict Transport Security)** giúp trình duyệt **chỉ cho phép kết nối HTTPS**. ✔ Ngăn chặn **tấn công downgrade HTTP**, bảo vệ dữ liệu người dùng. ✔ Mặc định, ASP.NET Core đặt thời gian HSTS là **30 ngày**.  📌 **Nếu ứng dụng chạy trên HTTP, dòng này sẽ không có tác dụng.** |
|  | ✅ **Đây là pipeline xử lý request trong ASP.NET Core MVC**, giúp: ✔ **Chuyển hướng HTTP sang HTTPS** ✔ **Cho phép truy cập file tĩnh (CSS, JS, hình ảnh, v.v.)** ✔ **Kích hoạt hệ thống định tuyến (Routing)** ✔ **Thực hiện phân quyền người dùng (Authorization)** ✔ **Xử lý yêu cầu và gửi phản hồi về trình duyệt**  📌 **Tóm lại:** Mỗi dòng code này là một **Middleware**, giúp kiểm soát luồng xử lý request trước khi gửi phản hồi. 🚀 **1️⃣** app.UseHttpsRedirection(); 🔹 **Chuyển hướng tất cả request HTTP sang HTTPS.** 🔹 Giúp bảo mật ứng dụng bằng cách ngăn truyền dữ liệu qua HTTP không an toàn.  📌 **Lưu ý:**   * Chỉ có tác dụng nếu ứng dụng được host trên một server có **chứng chỉ SSL**. * Nếu server không hỗ trợ HTTPS, dòng này sẽ bị bỏ qua.  **2️⃣** app.UseStaticFiles(); 🔹 **Cho phép phục vụ file tĩnh từ thư mục** wwwroot/. 🔹 Hỗ trợ **CSS, JavaScript, hình ảnh, fonts, v.v.**   **3️⃣** app.UseRouting(); 🔹 **Kích hoạt hệ thống định tuyến (Routing)**, giúp xác định request đi đâu. 🔹 Middleware này phải đặt **trước** app.UseAuthorization();.  📌 **Tóm lại:**   * Cho phép sử dụng **Controller, API, Razor Pages, SignalR**. * Nếu thiếu app.UseRouting();, request không thể tìm đến Controller.  **4️⃣** app.UseAuthorization(); 🔹 **Bật kiểm tra phân quyền người dùng**. 🔹 Nếu ứng dụng có hệ thống đăng nhập (Identity, JWT, OAuth), Middleware này sẽ kiểm tra quyền truy cập.  📌 **Lưu ý:**   * **Không kiểm tra đăng nhập** – chỉ kiểm tra **quyền truy cập**. * Nếu không có app.UseAuthorization();, mọi request sẽ bỏ qua kiểm tra quyền.    **6️⃣** app.Run(); 🔹 **Chạy ứng dụng và bắt đầu lắng nghe request từ trình duyệt**. 🔹 Nếu không có app.Run();, ứng dụng sẽ không hoạt động.  📌 **Tóm lại:** Đây là bước cuối cùng để khởi động ứng dụng ASP.NET Core. |
| **Controller** | là một thành phần quan trọng trong mô hình **MVC (Model-View-Controller)** của ASP.NET. Nó chịu trách nhiệm xử lý các yêu cầu từ trình duyệt, tương tác với Model (dữ liệu), và chọn View (giao diện) để hiển thị nội dung cho người dùng.    🔹 Nhận yêu cầu từ trình duyệt (HTTP Request). 🔹 Xử lý logic nghiệp vụ, tương tác với Model để lấy hoặc cập nhật dữ liệu. 🔹 Chọn View phù hợp và trả kết quả về trình duyệt (HTTP Response).  Controller là **cầu nối giữa người dùng và hệ thống**, đảm bảo rằng yêu cầu được xử lý đúng cách. **Routing – Cách Controller xử lý yêu cầu** **Controller nhận request thông qua hệ thống Routing của ASP.NET MVC.** Mặc định, route có dạng:   **Action Method trong Controller** Một Controller có nhiều **Action Method**, mỗi phương thức tương ứng với một yêu cầu HTTP.      - Mỗi khi tạo controller thì lớp đó sẽ kế thừa controller của asp và mỗi phương thức sẽ là một hướng để |
|  |  |