|  |  |
| --- | --- |
| **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** |  |
| scaffolded | **"scaffolded"** nghĩa là **tạo sẵn tự động** các phần của ứng dụng dựa trên model hoặc dữ liệu có sẵn.  Hiểu đơn giản thì thì scaffolded là cách gọi của từ một đối tượng thì nó tự sinh ra controller và folder + view của model đó mà không cần phải tạo thủ công view và controller cho model đó. |
| 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.** |
| Content | **là một phương thức trong Controller của ASP.NET Core MVC, dùng để trả về nội dung văn bản đơn giản (text, HTML, JSON, v.v.).**  📌 **Tóm lại:** ✔ **Dùng** Content("chuỗi") **để trả về text hoặc HTML mà không cần View.** ✔ **Dùng khi cần phản hồi nội dung đơn giản thay vì trả về View hoặc JSON.** |
| HttpGet | **là một attribute trong ASP.NET Core dùng để đánh dấu một phương thức trong Controller chỉ xử lý các request HTTP GET.**  📌 **Tóm lại:** ✔ **Giúp xác định rằng method chỉ phản hồi với request GET.** ✔ **Dùng để lấy dữ liệu từ server (không thay đổi dữ liệu).** ✔ **Thường dùng trong API hoặc Controller để phục vụ nội dung.** |
| Route | **là một attribute trong ASP.NET Core giúp định nghĩa đường dẫn URL cho Controller hoặc Action một cách tùy chỉnh.**  📌 **Tóm lại:** ✔ **Dùng** [Route] **trên Controller để định nghĩa base URL cho tất cả actions.** ✔ **Dùng** [Route] **trên Action để tùy chỉnh đường dẫn riêng cho từng API.** ✔ **Có thể kết hợp với** [HttpGet]**,** [HttpPost]**,** [HttpPut]**,** [HttpDelete] **để định nghĩa API chính xác hơn.**    Hiểu đơn giản thì nó đặt lại đường dẫn theo ý muốn mà không cần phải theo đường dẫn mặt định là tên controller + tên method. Một action method có thể có nhiều hơn 1 route |
| Các thuộc tính bắt đầu bằng From... | **Các thuộc tính** FromBody**,** FromQuery**,** FromRoute**,** FromForm**,** FromHeader**,** FromServices **giúp xác định cách ASP.NET Core lấy dữ liệu từ HTTP request để truyền vào method của Controller.**  📌 **Tóm lại:** ✔ [FromBody] → Lấy dữ liệu từ **body của request (JSON, XML, v.v.)**. ✔ [FromQuery] → Lấy dữ liệu từ **chuỗi truy vấn (query string)**. ✔ [FromRoute] → Lấy dữ liệu từ **route parameter trên URL**. ✔ [FromForm] → Lấy dữ liệu từ **form-data (POST form, file upload)**. ✔ [FromHeader] → Lấy dữ liệu từ **HTTP header**. ✔ [FromServices] → Lấy dịch vụ từ **Dependency Injection (DI)**. |
| View() | Trong **ASP.NET Core MVC**, phương thức View() được sử dụng trong controller để **trả về giao diện (HTML)** cho người dùng.  📌 **Cách hoạt động:**  Khi gọi View(), ASP.NET Core sẽ tìm kiếm file **.cshtml** tương ứng để render.  Sau khi render xong, nội dung HTML sẽ được gửi về trình duyệt.      lưu ý: nếu không đặt file giao diện ở trong thư mục có tên của controller sử dụng action method đó và file đó phải có tên giống với action medthod gọi đến thì asp sẽ tìm trong thư mục shared, nếu trong shared không có ( dù đặt ở bất cứ đâu khác ) thì asp sẽ báo lỗi chứ không có tự động tìm file có cùng tên với action method đó. Nếu muốn thì phải đưa đường dẫn cụ thể cho view |
|  |  |
| **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.     Đoạn này sẽ đặt đường dẫn mặc định nếu ta không nhập controller và action method trong url thì asp sẽ mặc định dùng đường dẫn home/index **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. |
|  | đây là đặt địa chỉ tắt cho product/details/123 thành p/123, có cách khác là dùng route để đặt địa chỉ khác cho action method đó |
| Routing | ✅ **Routing (Định tuyến)** trong ASP.NET Core là cơ chế **xác định URL nào sẽ gọi đến Controller và Action nào**. ✅ **Routing giúp ánh xạ request HTTP đến phương thức xử lý phù hợp trong ứng dụng**. ✅ **Có thể tùy chỉnh route để tạo URL thân thiện hơn** (SEO-friendly, RESTful API, v.v.).  📌 **Tóm lại:** ✔ Khi người dùng truy cập https://example.com/products/5, hệ thống biết cần gọi **Action** GetProduct(5) trong ProductController. ✔ Có 2 cách định nghĩa route: **Convention-based Routing** (trong Program.cs) và **Attribute Routing** (dùng [Route]). |
| **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 |
|  | |
| @model | là một **directive** trong Razor dùng để **định nghĩa kiểu dữ liệu (Model) mà View sẽ sử dụng**. Nó cho phép View nhận dữ liệu từ Controller một cách mạnh mẽ và có kiểu (strongly-typed).     📌 **Lưu ý khi dùng** @model ✔ **Mỗi file** .cshtml **chỉ được có một** @model. Nếu có nhiều, sẽ gặp lỗi RZ2001. ✔ **Không được nhầm với** @Model (Chữ M viết hoa). @model là khai báo kiểu, còn @Model là biến chứa dữ liệu. ✔ **Nếu không khai báo** @model**,** @Model **sẽ là kiểu** dynamic (ít an toàn hơn). |
| Các cách model binding trong asp | Model binding trong ASP.NET Core là một cơ chế giúp tự động gán dữ liệu từ HTTP request (như form, query string, route data) vào các đối tượng model trong controller. Dưới đây là các cách phổ biến của model binding và ví dụ cho cả frontend và backend. Các cách Model Binding trong ASP.NET Core: **Query String**: Dữ liệu trong URL sẽ được gán vào các parameter của action method.  **Route Data**: Dữ liệu trong URL pattern (route) sẽ được gán vào parameter của action.  **Form Data (POST)**: Dữ liệu trong form khi gửi request POST sẽ được gán vào model.  **Body**: Dữ liệu trong body của request sẽ được chuyển thành đối tượng model, thường dùng với các API.  **Header**: Dữ liệu trong HTTP headers có thể được gán vào parameter của action.           Tổng kết **Query String**: Thường dùng trong GET requests, dữ liệu gửi qua URL.  **Route Data**: Dùng khi dữ liệu là một phần của URL pattern.  **Form Data**: Dùng khi gửi form với phương thức POST.  **Body**: Thường dùng trong các API, dữ liệu gửi qua request body.  **Header**: Gửi dữ liệu qua HTTP headers.  Model Binding giúp việc phát triển ứng dụng trở nên dễ dàng hơn, vì nó tự động xử lý việc chuyển dữ liệu từ request thành đối tượng model mà không cần phải làm thủ công. |
| Fe model binding |  |
| \_ViewStart | \_ViewStart.cshtml là một file **đặc biệt** trong ASP.NET MVC (và Razor Pages), dùng để **cấu hình mặc định cho các view**, đặc biệt là **layout**. Mục đích chính của \_ViewStart.cshtml: ✅ **Tự động áp dụng layout cho mọi view** mà không cần chỉ định lại ở từng file .cshtml.  - hiểu đơn giản thì nó dùng để cài đặt layout mặc định áp dụng cho tất cả các view mà kh cần gọi tới layout đó ở từng trang một. Nó sự tự động sử dụng layout đó cho tất cả các trang kh có yêu cầu layout cụ thể |
| Db Context | |
| File DbContext được sinh ra từ DB First | **Vì** DbContext **và ánh xạ (mapping) giúp:**  Viết code nhanh hơn (không phải viết SQL tay nhiều)  Dễ bảo trì, dễ tái sử dụng  An toàn hơn, giảm lỗi SQL injection  Tận dụng sức mạnh của LINQ & mô hình hướng đối tượng  Nếu bạn dùng EF mà không ánh xạ trước → mất hết lợi thế của nó luôn 😄 Nếu bạn **muốn tự viết SQL khi cần**, thì đã có Dapper hoặc ADO.NET rồi!  Bạn hoàn toàn **có thể tự viết SQL và kết nối DB thủ công**, giống như JDBC:  Dùng SqlConnection, SqlCommand, SqlDataReader  Hoặc dùng Dapper (micro ORM rất nhẹ, dùng SQL trực tiếp)  → Nhưng bạn sẽ mất:  **Tự động hóa**  **Quản lý quan hệ**  **LINQ**  **Object mapping**  **Tính hướng đối tượng**  - Hiểu đơn giản thì lớp DBContext được sinh ra từ DataBase First sẽ là lớp để mô tả với ASP cấu trúc bảng, quan hệ giữa các bảng là gì và từ đó ASP có thể hiểu và lập trình viên có thể sử dụng các hàm và phương thức hổ trợ để thực hiện truy vấn trong DB mà không cần phải viết nguyên câu lệnh query để truy vấn dữ liệu như trong JAVA Severlet và phải tách từng cột thành một biến để có thể tạo thành 1 đối tượng trong code. |
| Giải thích file File DbContext được sinh ra từ DB First | 1. DbSet<T> là gì? Cách sử dụng Trong class DbContext, bạn sẽ thấy: 👉 Tác dụng: Mỗi DbSet<T> tương ứng với **một bảng trong database**.  Nó cho phép bạn thực hiện các thao tác:  Truy vấn (SELECT)  Thêm (INSERT)  Sửa (UPDATE)  Xoá (DELETE) 🔍 Ví dụ dùng trong Controller hoặc Repository:    1. HasOne().WithMany() là gì?  👉 Dòng này nghĩa là: **Mỗi chiếc xe (**Car**) chỉ có 1 màu (**Colord**)**  **Một màu (**Colord**) có thể thuộc về nhiều chiếc xe (**Car**)**  Đây là ánh xạ mối quan hệ **1-nhiều (One-to-Many)**:  Một Colord → nhiều Car  Một Car → một Colord ⚙️ Phải có trong entity model:    1. Property(...) ánh xạ chi tiết từng cột     Property() dùng để ánh xạ từng thuộc tính C# với cột DB.  HasColumnName(...): đặt tên cột trong DB.  HasMaxLength(...): đặt độ dài tối đa (có thể ánh xạ sang VARCHAR, NVARCHAR trong DB).  IsUnicode(false): ánh xạ kiểu VARCHAR thay vì NVARCHAR. 4. Cách dùng DbContext trong ASP.NET Controller Ví dụ với CarController:      Include(...) dùng để lấy cả entity liên quan (giống JOIN trong SQL). 5. Ưu điểm của DbContext (so với viết truy vấn tay thủ công)  | **Viết bằng DbContext** | **Viết bằng JDBC hoặc ADO.NET** | | --- | --- | | Gọn gàng, ít code | Nhiều đoạn xử lý thủ công | | Có tracking thay đổi | Không có, phải tự làm | | Tự động mapping entity | Phải ánh xạ thủ công | | Bảo mật SQL injection | Dễ sai sót nếu dùng chuỗi | | Dễ unit test / mock | Khó test / bảo trì |  Tổng kết DbContext là lớp quản lý kết nối và ánh xạ dữ liệu.  DbSet<T> đại diện bảng, giúp thao tác dữ liệu dễ dàng.  OnModelCreating ánh xạ chi tiết cột, khóa, quan hệ.  Mọi thao tác CRUD đều thực hiện dễ hơn và an toàn hơn nhờ EF Core. |
|  |  |