Cách hoạt động của ứng dụng ASP.NET Core

Ứng dụng ASP.NET Core có cách thức hoạt động khác biệt với ứng dụng ASP.NET truyền thống cũng nhưng các loại ứng dụng web xây dựng trên các nền tảng khác. Sự khác biệt này giúp ứng dụng ASP.NET Core hoạt động đa nền tảng, hiệu quả và an toàn hơn.

NỘI DUNG CỦA BÀI [Ẩn]

Cách thức hoạt động của hệ thống web

Một hệ thống web có mô hình hoạt động tương đối phức tạp và khác biệt rất nhiều với các loại ứng dụng desktop quen thuộc. Mặc dù hằng ngày bạn vẫn sử dụng trình duyệt – một bộ phận của hệ thống web – bạn có thể chưa hiểu được cách thức hoạt động của cả hệ thống này.

Nếu muốn phát triển ứng dụng web (và những loại ứng dụng có liên quan), bạn càng cần biết rõ hơn về cách thức hoạt động của hệ thống web nói chung.

Do đó, nếu bạn chưa từng học hoặc xây dựng ứng dụng web, bạn nên bỏ chút thời gian để tìm hiểu cách thức hoạt động của hệ thống web, trước khi đi sâu vào tìm hiểu cách thức hoạt động của ứng dụng ASP.NET Core.

> Nếu bạn đã quen thuộc với ASP.NET trước đây, hoặc đã quen thuộc với hệ thống web, bạn có thể bỏ qua nội dung này để đọc phần tiếp theo về cách hoạt động của ASP.NET Core.

Một hệ thống web thông thường bao gồm hai loại chương trình chạy trên các máy tính khác nhau:

- trình duyệt – loại chương trình chạy trên máy tính của người dùng;
- chương trình máy chủ web (web server) – loại chương trình chạy trên một máy tính riêng rất mạnh.
- trình duyệt và chương trình máy chủ web tương tác với nhau qua một mạng truyền thông (thường là mạng tcp/ip), sử dụng giao thức HTTP (Hypertext Transfer Protocol).

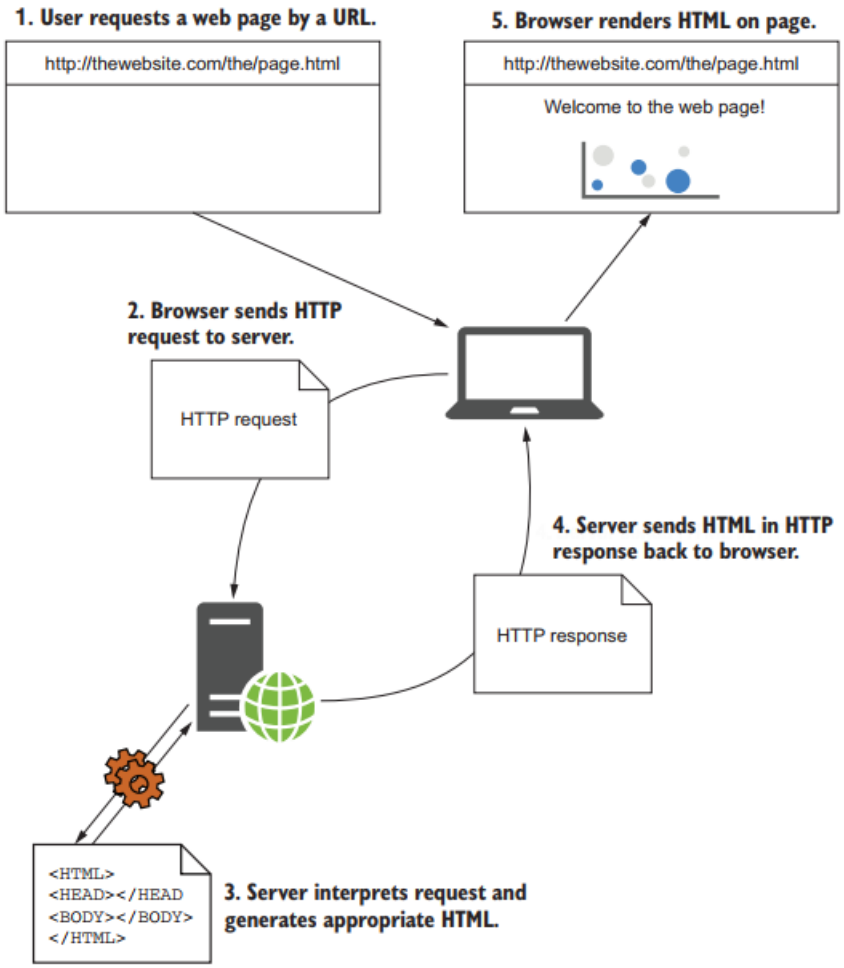
> Trình duyệt phổ biến bạn có thể đang sử dụng hàng ngày như Chrome, Firefox, Edge, Opera, Safari. Chương trình máy chủ ít người biết đến hơn. Có thể kể tên những chương trình máy chủ phổ biến như IIS, Apache, NGinx.

Trình duyệt và chương trình máy chủ "nói chuyện" với nhau bằng cách:

1. trình duyệt gửi đi các "lệnh" (gọi là truy vấn – *HTTP Request*);
2. chương trình máy chủ nhận "lệnh" và sinh ra dữ liệu tương ứng (ở dạng văn bản viết bằng ngôn ngữ HTML), thu thập các file (mã javascript, css, hình ảnh, v.v.);
3. dữ liệu được "đóng gói" (thành các phản hồi – *HTTP Response*) và trả lại cho trình duyệt.
4. trình duyệt sẽ hiển thị dữ liệu nhận được, cũng như thực thi các lệnh (do các file javascript phát ra).

Tổ hợp của dữ liệu (HTML), cách thức hiển thị (CSS) và lệnh (javascript) tạo ra những gì người dùng nhìn thấy và có thể tương tác trên trình duyệt.

Để dễ hình dung, dưới đây là sơ đồ minh họa hoạt động của hệ thống web "truyền thống".



Tất cả quy trình phức tạp trên được thực hiện tự động và bắt đầu khi người dùng nhập một "địa chỉ web" – **Unified Resource Locator hay URL** – vào thanh địa chỉ của trình duyệt.

> **Địa chỉ web** hay **URL** bao gồm một *hostname* độc nhất trên Internet và đường dẫn tới "tài nguyên" tương ứng (ví dụ, một file) trên máy chủ. **Hostname**, còn gọi là *tên miền* (Domain Name), được *Dịch vụ Tên miền* (Domain Name Service hay DNS) ánh xạ với địa chỉ IP của máy chủ, giúp gói tin HTTP Request đi được từ máy người dùng tới máy chủ.

Trong quá trình trên, nếu trong bất kỳ một thành phần nào (HTML, Javascript, CSS) có "tham chiếu" tới một file khác, một quy trình mới sẽ được tự động bắt đầu để lấy file tương ứng.

Nói cách khác, để "tải" trọn vẹn những gì cần thiết từ máy chủ về trình duyệt, chương trình trình duyệt có thể phải phát phát đi hàng trăm HTTP Request (và nhận về hàng trăm HTTP Response tương ứng).

Ví dụ, để tải trang web bạn đang xem, nếu không thực hiện bất kỳ kỹ thuật ghép file nào, trình duyệt của bạn sẽ phát về server của chúng tôi khoảng 150 HTTP request để lấy chừng ấy file.

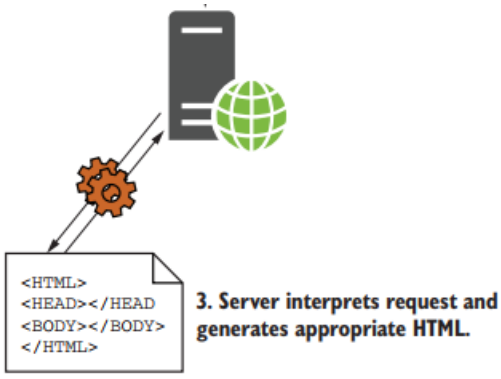
> Để xem các truy vấn, bạn có thể mở cửa sổ Developer (F12) của trình duyệt, tab Network và ấn F5 để tải lại trang.

Vai trò và vị trí của ứng dụng web

Sau khi đọc phần trên bạn hẳn phân vân, vậy "ứng dụng web" mà bạn viết ra, dù là dùng PHP hay ASP.NET, nằm ở đâu?

> Ở phần trên bạn có thể để ý chúng ta đang dùng từ "hệ thống web" để chỉ tổ hợp trình duyệt – chương trình máy chủ web. Chúng ta dùng thuật ngữ đó để giúp bạn phân biệt với "ứng dụng web" mà bạn sẽ xem xét trong phần này.

Hãy nhìn vào ***bước số 3*** trong sơ đồ của hệ thống web: Server giải mã truy vấn và sinh ra HTML tương ứng. Trên thực tế, mọi chuyện khá phức tạp.



Bạn cần biết rằng, các chương trình máy chủ web (Apache, IIS, NGinx) thực ra không hề biết gì về chương trình của bạn! Chúng cũng không thể thực thi được các lệnh do bạn viết ra, dù là bạn dùng ngôn ngữ lập trình nào đi chăng nữa! Các chương trình này chỉ có khả năng đọc các file (văn bản và ảnh) trên ổ đĩa của máy chủ, đóng gói vào HTTP Response và trả lại trình duyệt!

Đây cũng là cách thức hoạt động thuở ban đầu của web: lưu trữ và trả các file HTML cho trình duyệt khi được yêu cầu. Ngày nay, các trang web như vậy thường được gọi là trang

web tĩnh (static page), nhằm phân biệt với các trang web được tính ra tự động bởi các script mà chúng ta quen gọi là các ứng dụng web.

Tuy nhiên, các chương trình máy chủ đều hỗ trợ các chương trình mở rộng (extension hoặc plugin). Các cơ chế thực thi lệnh của người dùng được thực hiện bởi các chương trình mở rộng. Lấy ví dụ, chương trình PHP là một chương trình mở rộng như vậy.

Căn cứ vào yêu cầu của người dùng (thông qua URL), chương trình máy chủ sẽ biết cần kích hoạt chương trình mở rộng nào và chuyển tiếp truy vấn cho nó. Khi này, chương trình mở rộng sẽ thực thi lệnh (thường là các script – file lệnh mã mở, hoặc các file thư viện đã biên dịch) do người lập trình cung cấp để tự động sinh ra dữ liệu (HTML). Các script/thư viện do người dùng viết như thế chính là ứng dụng web.

Với cơ chế như trên, chương trình máy chủ Apache có thể chạy các script viết bằng ngôn ngữ PHP (qua chương trình PHP) hoặc ngôn ngữ Pearl (qua chương trình Pearl), v.v.. Chương trình IIS có thể thực thi các script viết bằng PHP, VbScript, cũng có thể chạy các phần mềm (thư viện) của .NET Framework.

Ứng dụng web cũng dùng để chỉ một loại "chương trình" viết hoàn toàn bằng Javascript và thực thi trong trình duyệt. Khi này, các chương trình máy chủ web chỉ làm nhiệm vụ cung cấp file javascript cho trình duyệt.

Trong hệ thống trên, ở vị trí của trình duyệt có thể là một ứng dụng desktop, ứng dụng mobile. Dữ liệu HTML, CSS và Javascript có thể được thay thế bằng chuỗi văn bản theo định dạng JSON/XML. Tuy nhiên, mô hình làm việc Request – Response là không đổi. Mô hình ứng dụng này thường được gọi là *ứng dụng hướng dịch vụ* (Service-Oriented Application hay SOA).

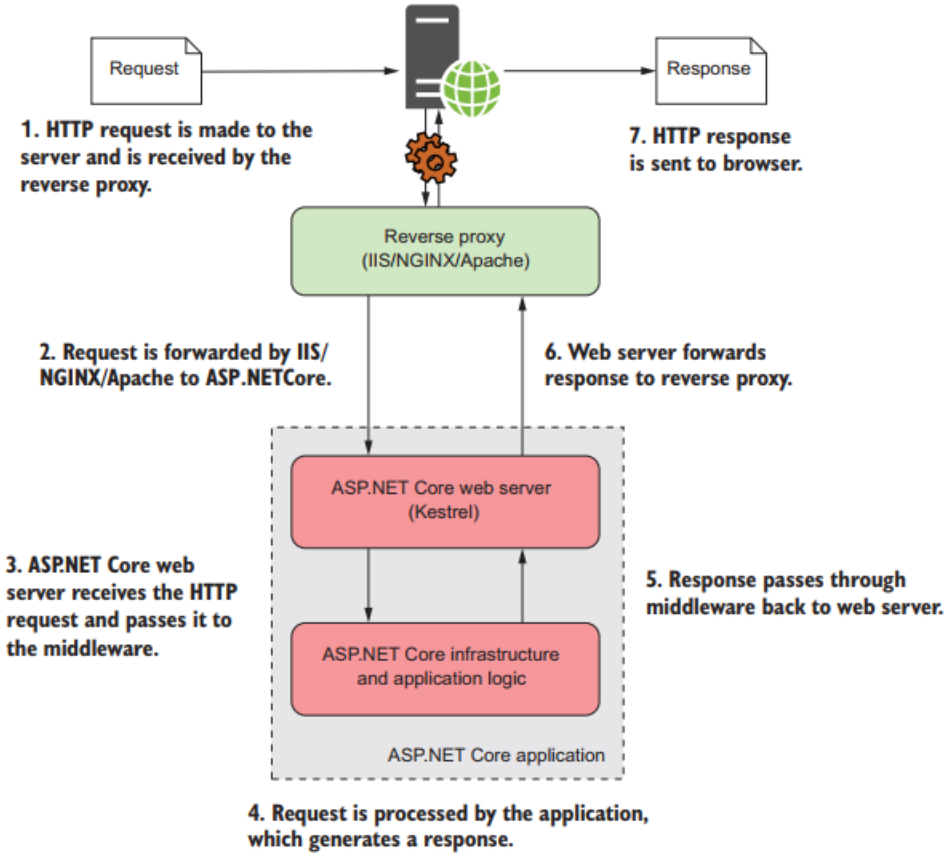
Cũng trong hệ thống trên, nếu HTML/CSS/Javascript chỉ được tải một lần duy nhất, và những lần sau server chỉ trả lại dữ liệu ở dạng JSON/XML (tức là trang không cần tải lại trọn vẹn), bạn thu được một loại ứng dụng chạy trên trình duyệt gọi là *ứng dụng đơn trang* (Single-page application).

ASP.NET Core cho phép bạn viết tất cả các loại ứng dụng kể trên. Sau đây chúng ta sẽ xem xét cách thực hoạt động riêng của ASP.NET Core.

Mô hình hoạt động của ứng dụng ASP.NET Core

Như đã phân tích ở phần trên, ASP.NET Core cũng tham gia vào bước thứ 3 trong mô hình hoạt động của cả hệ thống web. Tuy nhiên, trong trường hợp ASP.NET Core, mọi chuyện phức tạp hơn.

Hãy xem minh họa mô hình hoạt động của ASP.NET Core dưới đây:



Trước hết cần lưu ý, trong mỗi ứng dụng ASP.NET Core tích hợp sẵn một chương trình web server của riêng mình (built-in web server) có tên gọi là **Kestrel**. Đây là một chương trình web server thực sự, độc lập và được xây dựng dành riêng cho ASP.NET Core. Kestrel có thể hoạt động đa nền tảng (trên Windows, Linux và MacOs).

Phần code do bạn tự viết (dưới dạng thư viện đã biên dịch cùng các file khác) chỉ tương tác với Kestrel, cụ thể là: (1) nhận dữ liệu đầu vào từ Kestrel; (2) thực thi logic để sinh ra dữ liệu mới (HTML, JSON, XML, v.v.); (3) dữ liệu sinh ra được trả về cho Kestrel.

Bạn hoàn toàn có thể cảm nhận được, như vậy thì bản thân bộ đôi Kestrel và code bạn viết đã hoạt động giống hệt như mô hình web thông thường rồi. Nhưng trong sơ đồ trên, bạn vẫn nhìn thấy IIS, Apache, NGinX. Như vậy trong mô hình này có tới 2 chương trình web server cùng hoạt động!

Đúng là như vậy. Trong mô hình triển khai của ASP.NET Core bên trên có 2 chương trình web server. Trong đó, chương trình web server thứ nhất là những chương trình truyền thống (IIS, Apache, NGinX), giờ được gọi là *reverse proxy*. Server thứ hai là Kestrel, web server riêng của ASP.NET Core, còn gọi là *built-in server*.

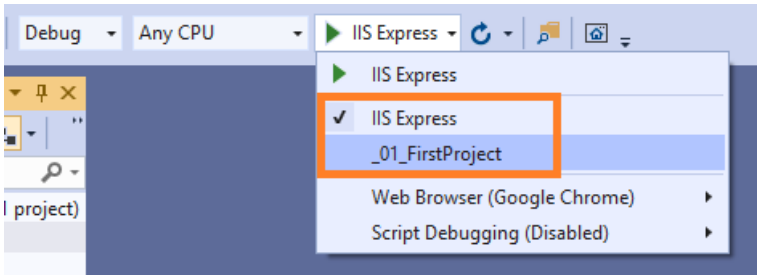
> Kestrel không phải là web server duy nhất cho ASP.NET Core. Bạn cũng có thể sử dụng HTTP.sys thay thế. Tuy nhiên, Kestrel là nhanh nhất và đa nền tảng.

Reverse proxy chịu trách nhiệm tương tác trực tiếp với client (trình duyệt hoặc chương trình desktop/mobile) qua HTTP. Nói theo cách khác, trình duyệt của bạn nhìn thấy reverse proxy như trong mô hình web thông thường. Tuy nhiên, reverse proxy không xử lý truy vấn

mà chuyển tiếp truy vấn cho Kestrel và nhận lại kết quả từ Kestrel. Mô hình triển khai này đem đến ưu điểm về tính bảo mật và hiệu suất.

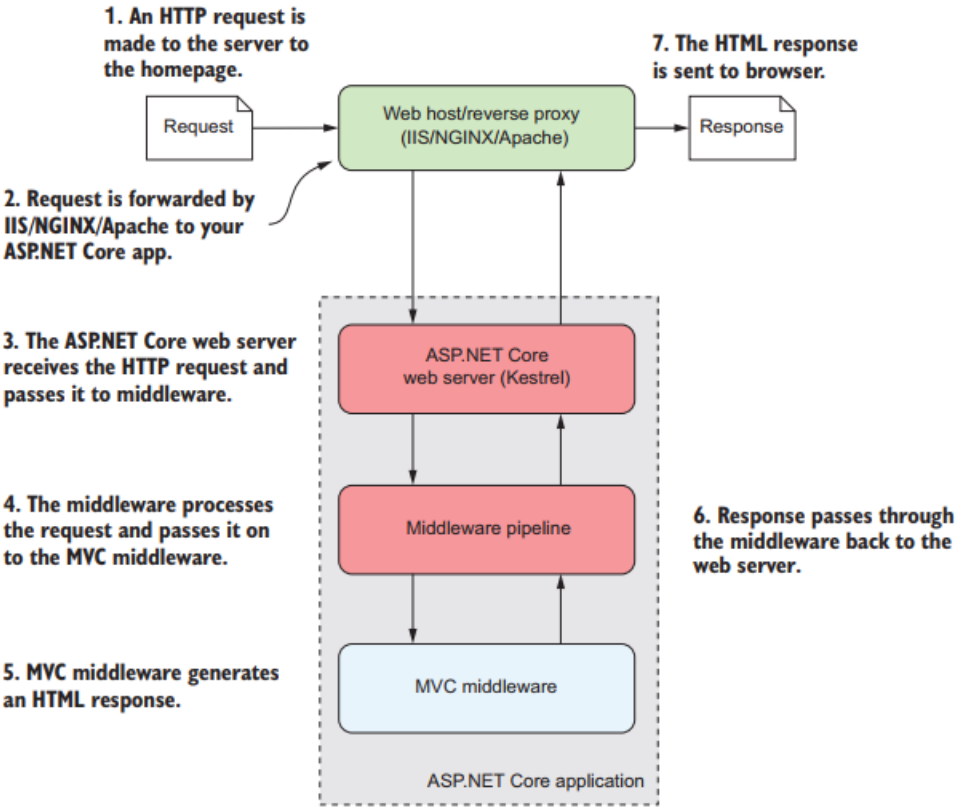
Reverse proxy không bắt buộc trong mô hình triển khai của ASP.NET Core. Bản thân Kestrel đã là một chương trình web server thực sự và độc lập. Nó có thể tự mình tiếp nhận và xử lý truy vấn HTTP Request đến từ client. Do vậy, chương trình ASP.NET Core bạn viết ra hoàn toàn có thể tự chạy như một ứng dụng console độc lập thông thường (vì đã có built-in Kestrel bên trong) trên tất cả các platform được .NET Core hỗ trợ.

> Nếu nhớ lại bài học trước về cách dịch và chạy ứng dụng ASP.NET Core bạn hẳn đã thấy Visual Studio cung cấp chế độ chạy bên trong IIS Express và chế độ chạy độc lập (Console). IIS Express, nếu được sử dụng, sẽ đóng vai trò *Reverse Proxy*. Nếu chạy độc lập, Kestrel sẽ được sử dụng và mặc định sẽ lắng nghe các HTTP Request ở cổng số 5000.



Chi tiết hoạt động của ứng dụng ASP.NET Core

Trong phần này chúng ta phân tích kỹ hơn nữa cấu trúc của ứng dụng ASP.NET Core nằm trong quan hệ với các thành phần khác của một hệ thống web.



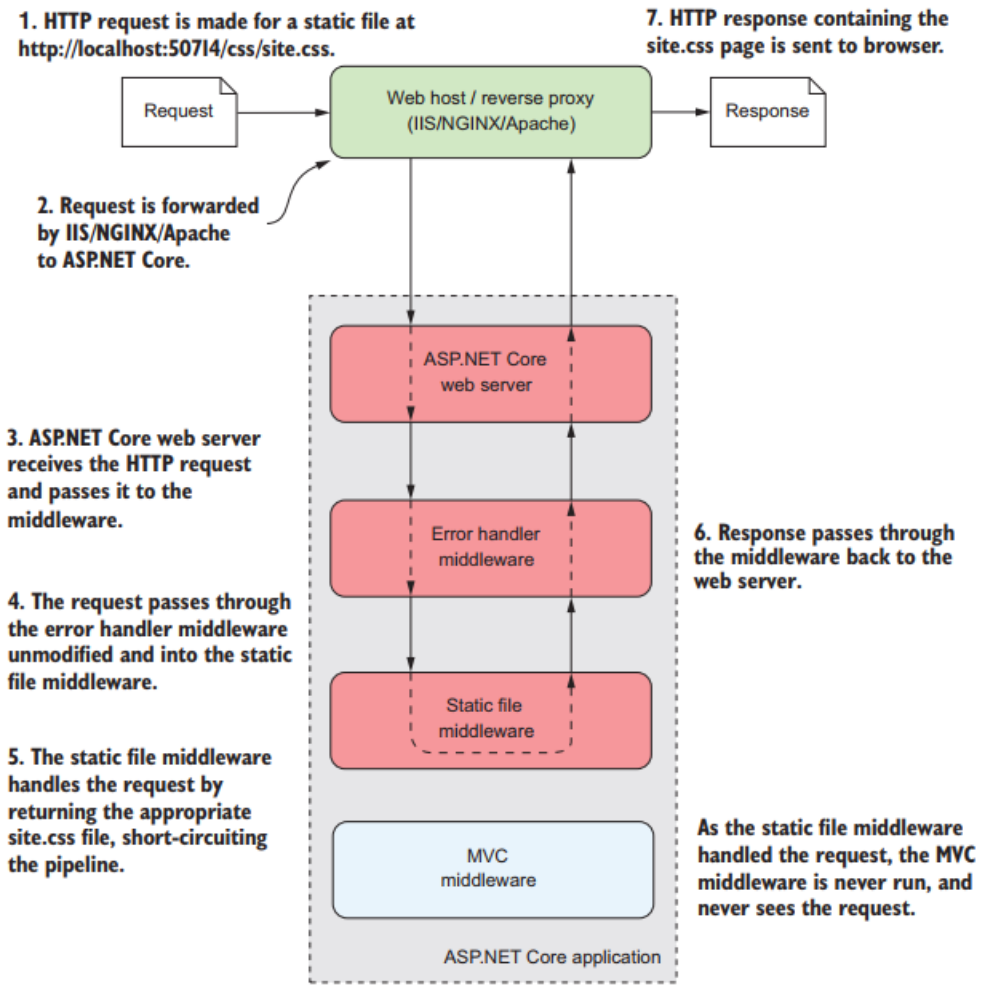
So với vô hình tổng quát đã trình bày ở phần trước, chúng ta tiếp tục mở rộng ứng dụng ra để nhìn cấu trúc bên trong nó, bao gồm phần *Middleware pipeline* và *MVC middleware* nằm dưới Kestrel web server.

Middleware là một khái niệm mới trong ASP.NET Core dùng để chỉ các thư viện/thành phần có khả năng xử lý truy vấn HTTP và trả lại kết quả. Các Middleware được ghép lại thành chuỗi (*pipeline*) cho từng nhiệm vụ cụ thể. Bạn, với vai trò người phát triển ứng dụng, tự quyết định sử dụng những Middleware nào và cách ghép nối chúng.

Ở dưới đáy của chuỗi pipeline là **MVC Middleware** – nơi bạn viết các logic của riêng mình để xử lý truy vấn nếu như không có Middleware phù hợp dọc đường di chuyển của truy vấn.

Nếu một truy vấn đáp ứng điều kiện xử lý ở một Middleware trong pipeline, nó sẽ được xử lý tại đó và trả lại kết quả, đồng thời kết thúc chuỗi xử lý truy vấn. Middleware tiếp theo trong pipeline sẽ không được kích hoạt (cũng đồng nghĩa truy vấn sẽ không đi đến tận cùng của chuỗi).

Lấy ví dụ, nếu trình duyệt yêu cầu lấy file tĩnh site.css. ASP.NET Core cung cấp sẵn *Static File Middleware* chuyên môn cho việc này. Middleware này sẽ xử lý yêu cầu, trả lại kết quả (file site.css) và kết thúc chuỗi di chuyển của truy vấn. Xem sơ đồ hoạt động được minh họa dưới đây.



Ví dụ xử lý truy vấn lấy file tĩnh

Kết luận

Bài học cung cấp cho bạn cái nhìn tổng thể về cách thức hoạt động của hệ thống web, ứng dụng web và ASP.NET Core. Bài học có ý nghĩa quan trọng giúp bạn hình dung được tổng thể vị trí của ASP.NET Core trước khi bắt đầu đi sâu vào nó.

> + Nếu bạn thấy site hữu ích, trước khi rời đi hãy **giúp đỡ** site bằng một hành động nhỏ để site có thể phát triển và phục vụ bạn tốt hơn.
> + Nếu bạn thấy bài viết hữu ích, hãy giúp **chia sẻ** tới mọi người.
> + Nếu có thắc mắc hoặc cần trao đổi thêm, mời bạn viết trong phần **thảo luận** cuối trang.
> Cảm ơn bạn!