Bài giảng tích hợp:

ASP NET CORE

Faculty of IT

Email: smdat@hueic.edu.vn

ASP.NET CORE

Chương 1. Giới thiệu chung về ASP.NET Core

Chương 2. ASP.NET Core Razor Pages

Chương 3. ASP.NET CORE MVC

Chương 1. Giới thiệu chung về ASP.NET Core

1.1. Giới thiệu ASP.Net Core

1.2. Cài đặt, tạo dự án, dịch và chạy ứng dụng ASP.NET Core

1.3. Cấu trúc ASP.NET Core

1.4. Cách hoạt động của ứng dụng ASP.NET Core

1.5. Cấu trúc dự án, cấu hình ứng dụng, middleware trong ASP.NET Core

1.6. Sử dụng LibMan – công cụ hỗ trợ tải thư viện

- **ASP.Net Core** là một **framework** dành cho phát triển ứng dụng web mới nhất của **Microsoft**
- Nhu cầu học và việc làm với **ASP.NET Core** tăng rất nhanh.
- Phiên bản đầu tiên phát hành tháng 6 năm 2016
- **ASP.Net Core** được thiết kế lại hoàn toàn để phù hợp cho phát triển các ứng dụng web hiện đại. Trong **ASP.Net Core**, cả **framework** và **platform** cho thực thi ứng dụng đều được xây dựng lại. Nó đồng thời bổ sung thêm những tính năng mới không có trong **ASP.Net** truyền thống
- Tuy được xây dựng lại từ đầu nhưng **ASP.Net Core** vẫn kế thừa những ưu điểm của **ASP.Net**. Vì vậy, **ASP.Net Core** cũng được xem như người kế tục của **ASP.Net**

- **ASP.Net Core** là một framework mã nguồn mở. Do đó cộng đồng **ASP.Net Core** phát triển nhanh chóng
- Các ứng dụng web sẵn có cũng có thể chuyển đổi sang **ASP.Net Core** để tận dụng những ưu thế của ***framework/platform*** mới này.
- Hiện nay nhiều công ty đã và đang chuyển dịch sang **ASP.NET Core**, nhất là khi phát triển các dự án mới. Do vậy, nhu cầu học và làm việc với công nghệ mới này đang tăng lên rất nhanh.

- Để hiểu được vì sao **Microsoft** quyết định xây dựng một framework mới, cũng như thuyết phục bạn chuyển sang học **ASP.NET Core**, chúng ta sẽ nói qua một vài vấn đề của **ASP.NET** truyền thống.
  - **ASP.NET** ra đời từ 2002 với vai trò là một bộ phận của **.NET Framework 1.0** nhằm thay thế cho **ASP** (**Active Server Pages**, ra đời từ 1996) cổ điển (sử dụng **VBScript**) và cạnh tranh với **PHP**.
  - Bên trên **ASP.NET**, **Microsoft** xây dựng một số mô hình lập trình khác nhau để hỗ trợ lập trình viên: (**1**) **Web Forms**, (**2**) **MVC**.
  - **Web Forms** ra đời năm 2002 hướng tới mô hình web "**stateful**" dựa trên sự kiện tương tự như **Windows Forms**. Mô hình lập trình của **Web Forms** có quá nhiều vấn đề, nhất là đối với các ứng dụng lớn, bao gồm hạn chế khi ***test***, mô hình ***stateful*** quá phức tạp, không kiểm soát được **HTML** gây khó khăn cho xây dựng client. **Web Forms** dần bị thay thế khi **MVC** ra đời.

- **ASP.NET MVC** ra đời năm 2009 dựa trên mô hình kiến trúc **MVC** (***Model – View – Controller***), tương tự như **Ruby on Rails**, **Django** hay **Java Spring**. Mô hình này đặc biệt thành công và được sử dụng rất rộng rãi thay thế cho **Web Forms**.
- Năm 2012, **ASP.NET Web API** ra đời giúp phát triển ứng dụng dạng dịch vụ **REST** và dần thay thế **WCF** (**Windows Communication Foundation**, xuất hiện từ 2006).

- Vấn đề của **ASP.NET**
  - Tất cả các mô hình lập trình của **ASP.NET** (**Web Forms** và **MVC**) đều được xây dựng trên cùng một **framework** chung sử dụng thư viện **System.Web.dll** – vốn là một bộ phận của **.NET Framework** lớn. Điều này mang tới cả ưu và nhược điểm.
  - Các ứng dụng web có thể sử dụng tất cả các tính năng của **.NET Framework** và **ASP.NET** chung – vốn vô cùng đa dạng, phong phú, có tính tin cậy và ổn định cao đã trải qua thử thách của thời gian.
  - **ASP.NET** cũng gắn chặt với dịch vụ hosting của **Windows** sử dụng **IIS** (***Internet Information Service***). Điều này khiến **ASP.NET** không thể mở rộng ra hoạt động trên các hệ điều hành khác – vốn là một yêu cầu rất quan trọng hiện nay.
  - Những lý do trên dẫn tới việc **Microsoft** quyết định xây dựng một ***framework/platform*** hoàn toàn mới cho phát triển ứng dụng web và đặt tên nó là **ASP.NET Core**.
  - Như vậy, không nên nhầm lẫn giữa **ASP.NET** và **ASP.NET Core**. Chúng là hai **framework** hoàn toàn khác nhau.

- **ASP.NET Core** được thiết kế nhằm đáp ứng các yêu cầu:
  - Phát triển và hoạt động đa nền tảng;
  - Có kiến trúc dựa trên các **module**;
  - Phát triển hoàn toàn ở dạng mã nguồn mở;
  - Phù hợp với xu hướng hiện đại của ứng dụng web.
- Để đạt được các yêu cầu trên, **Microsoft** xây dựng một ***platform*** mới đảm bảo nhẹ – nhanh – đa nền tảng. ***Platform*** mới này được đặt tên là **.NET Core**. Hiện nay **.NET Core** hoạt động được trên **Windows**, **macOS** và **Linux**.
- Tuy nhiên, **.NET Core** chứa rất nhiều **API** giống như của **.NET Framework**. Điều này giúp lập trình viên dễ dàng chuyển đổi sang **.NET Core** mà không cần phải học lại mọi thứ từ đầu. Nếu nhìn từ phía các **API** thì có thể hình dung **.NET Core** tương tự như một bộ phận (độc lập) tách ra từ **.NET Framework**.

1.1. Giới thiệu ASP.Net Core

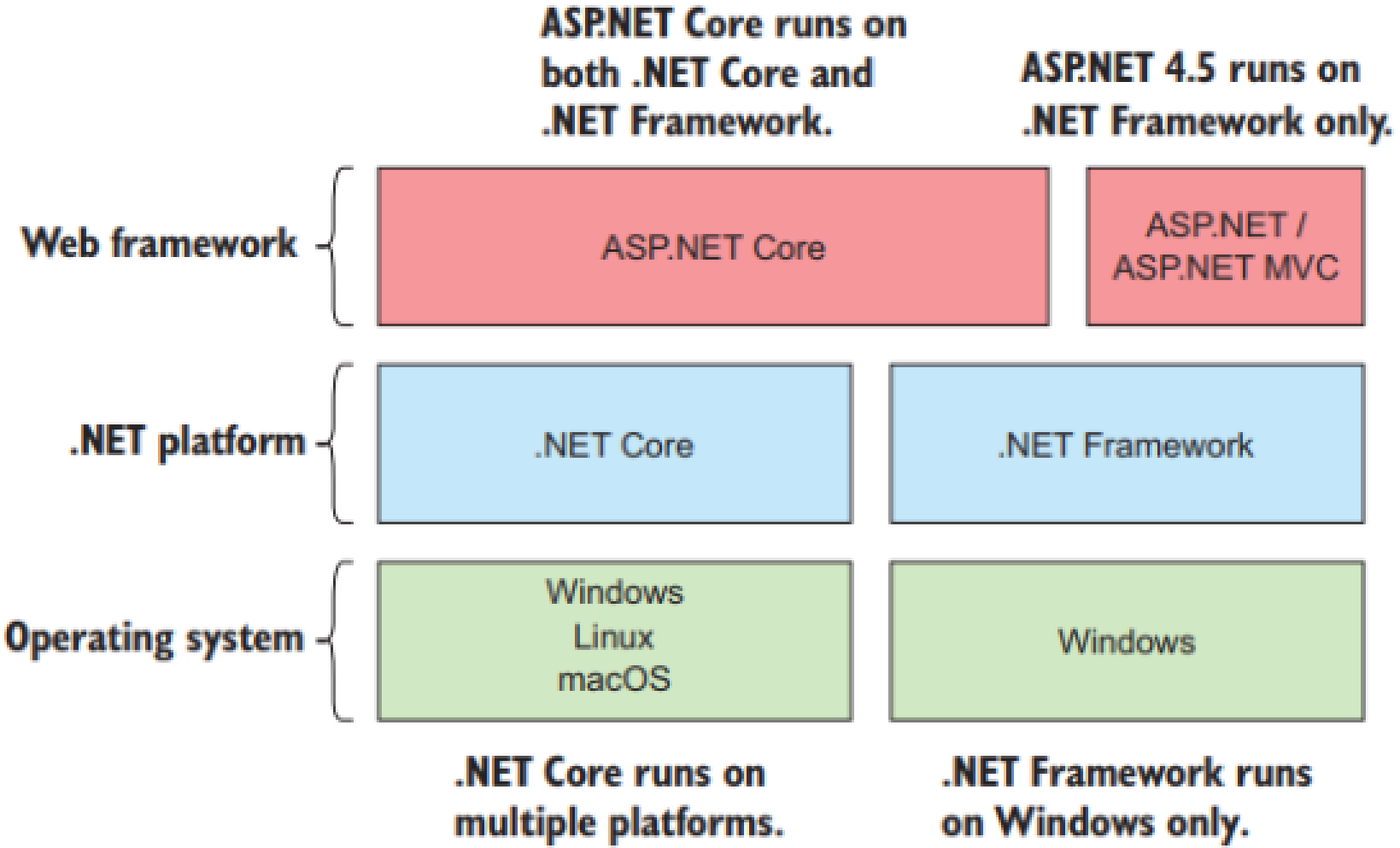
- **Ưu nhược điểm của ASP.NET cũ**
- **.NET Core và ASP.NET**
- **ASP.NET Core và .NET Framework**

- Một điều khiến rất nhiều người nhầm lẫn là mối quan hệ giữa **ASP.NET Core** và **.NET Framework**: **ASP.NET Core** có thể hoạt động trên **.NET Framework** (giống như **ASP.NET** cũ).
- Để hiểu vấn đề này bạn cần hình dung **.NET Framework** (và cả **.NET Core**) theo hai khía cạnh: **(1)** hệ thống thư viện **API** hỗ trợ phát triển ứng dụng; **(2)** **runtime** dành cho thực thi ứng dụng.
- **ASP.NET Core** chứa hệ thống **API** của riêng nó. Hệ thống **API** này sử dụng các **API** cơ bản của **.NET**. Thêm vào đó, **.NET Core** và **.NET Framework** có chung hệ thống **API** cơ bản.
- **Runtime** có thể hình dung như chương trình máy ảo sẽ nạp ứng dụng vào để thực thi. Ứng dụng và tất cả các thư viện của cả **.NET Core** và **.NET Framework** đều nằm ở dạng mã trung gian **IL** (***Intermediate Language***).
- Hai yếu tố trên cho phép chương trình viết bằng **ASP.NET Core** có thể hoạt động trên **runtime** (máy ảo) của **.NET Framework**. Ở chiều ngược lại, **ASP.NET** truyền thống không thể hoạt động trên **.NET Core** do nó phụ thuộc vào **System.Web.dll** của **.NET Framework**, vốn không có trong **.NET Core**.
- **ASP.NET Core 2.0** tới **2.2** có thể chạy trên **.NET Framework 4.6.1** (và các phiên bản cao hơn), đồng thời có thể chạy trên **.NET Core 2.0** (và các bản cao hơn). Tuy nhiên **ASP.NET Core 3.0** chỉ chạy trên **.NET Core 3.0**.

1.1. Giới thiệu ASP.Net Core

- **Ưu nhược điểm của ASP.NET cũ**
- **.NET Core và ASP.NET**
- **ASP.NET Core và .NET Framework**

• Mối quan hệ giữa **ASP.NET Core** với **.NET Core** và **.NET Framework** được minh họa như hình dưới đây.



- Trong **ASP.NET** xây dựng các ứng dụng web sử dụng một trong số các mô hình lập trình mà **framework** này cung cấp như **Web Forms**, **MVC**, **Web API**, **Web Pages**. Các mô hình này có thể xem như những **framework** riêng biệt xây dựng bên trên framework lớn **ASP.NET**.
- Trong **ASP.NET Core**, các mô hình lập trình trong **ASP.NET Core** được thống nhất. Có thể lựa chọn phát triển ứng dụng web theo mô hình **MVC, Web API**, **Razor Pages**. Tuy nhiên, các mô hình này không tách rời (sử dụng các class/thư viện riêng biệt) như trong **ASP.NET** mà nằm trong một hệ thống thống nhất, sử dụng chung **class** và thư viện.

- **ASP.NET Core** cho phép bạn viết gần như bất kỳ loại ứng nào có liên quan đến **HTTP**, như ứng dụng web **HTML** truyền thống, **REST API** cho ứng dụng đơn trang (**Single Page Application**, **SPA**), dịch vụ gọi hàm từ xa (**Remote Procedure Call**, **RPC**).
- Ứng dụng web với mã **HTML** do server sinh ra là loại ứng dụng cơ bản và truyền thống mà **ASP.NET Core** hỗ trợ. Để phát triển các loại ứng dụng này bạn có thể sử dụng mô hình lập trình **MVC** hoặc **Razor Pages**
  - **ASP.NET Core MVC** là mô hình lập trình ứng dụng web tương tự như **ASP.NET MVC** quen thuộc. Các thành phần của ứng dụng được phân chia ra các thành phần tuân theo mẫu kiến trúc **MVC** (***Model – View – Controller***), tương tự như **Ruby on Rails**, **Java Spring** hoặc **Django**.
  - **Razor Pages** là mô hình đơn giản hóa của **MVC**, chỉ bao gồm thành phần **V** (**View**) viết bằng ngôn ngữ **Razor** – loại cấu trúc kết hợp **HTML** và **C#**. Bạn có thể hình dung **Razor** là một dạng ngôn ngữ tương tự **PHP** nhưng có cấu trúc của **C#**. **Razor Pages** tương tự như **Web Pages** của **ASP.NET**.

- Ứng dụng **ASP.NET Core** có thể được phát triển (và hoạt động) trên cả **Windows**, **Linux** và **macOS**.
- Để thực thi ứng dụng, cần cài đặt **.NET Core runtime**;
- Để phát triển ứng dụng, cần cài đặt **.NET Core SDK**.
- Ngoài ra, cũng cần có một công cụ hỗ trợ viết code – biên dịch mã - **debug**

- Để phát triển ứng dụng **ASP.NET Core** bạn cần các công cụ sau:
  - Bộ **ASP.NET Core SDK**;
  - Một môi trường phát triển ứng dụng tích hợp như **Visual Studio**, **JetBrains Rider** hoặc một code editor như **Visual Studio Code**.

Chương 1. Giới thiệu chung về ASP.NET Core

1.1. Giới thiệu ASP.Net Core

1.2. Cài đặt, tạo dự án, dịch và chạy ứng dụng ASP.NET Core

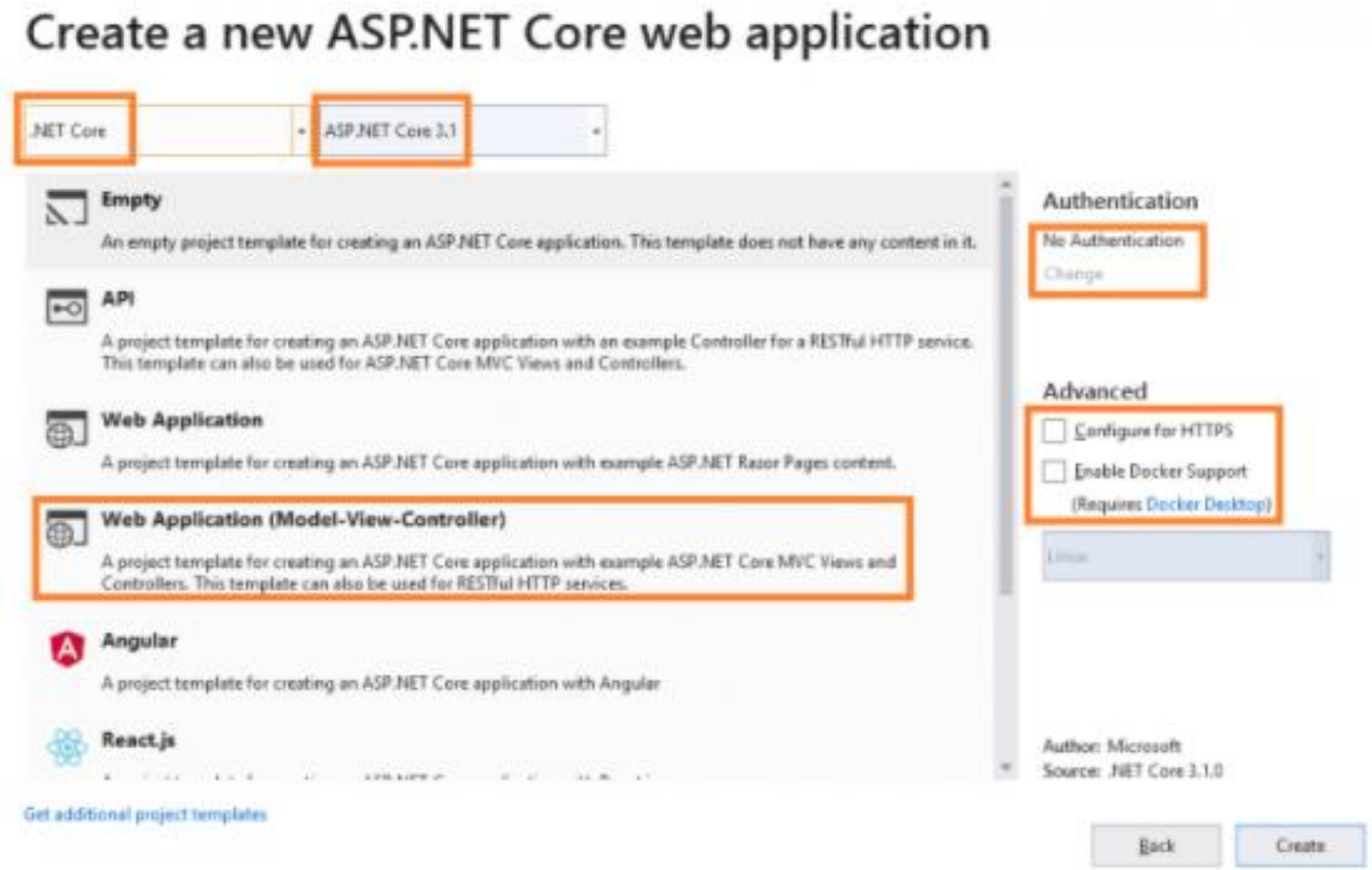
- **Giới thiệu chung về các công cụ cho ASP.NET Core**
- **Cài đặt ASP.NET Core SDK trên Windows**

- Trong **Visual Studio**, dự án **ASP.NET Core** được tạo ra như bất kỳ một dự án nào khác từ cửa sổ **Get Started -> Create a new project**, hoặc từ menu (**File ->New ->Project**).
- Chọn template **ASP.NET Core Web Application**
- Chọn tên project (**Project name**), nơi lưu (**Location**) và tên solution (**Solution name**).
- Tiếp theo bạn cần lựa chọn **project template** cho loại ứng dụng đang cần phát triển.
- Hãy chọn template **Web Application** (***Model-View-Controller***)

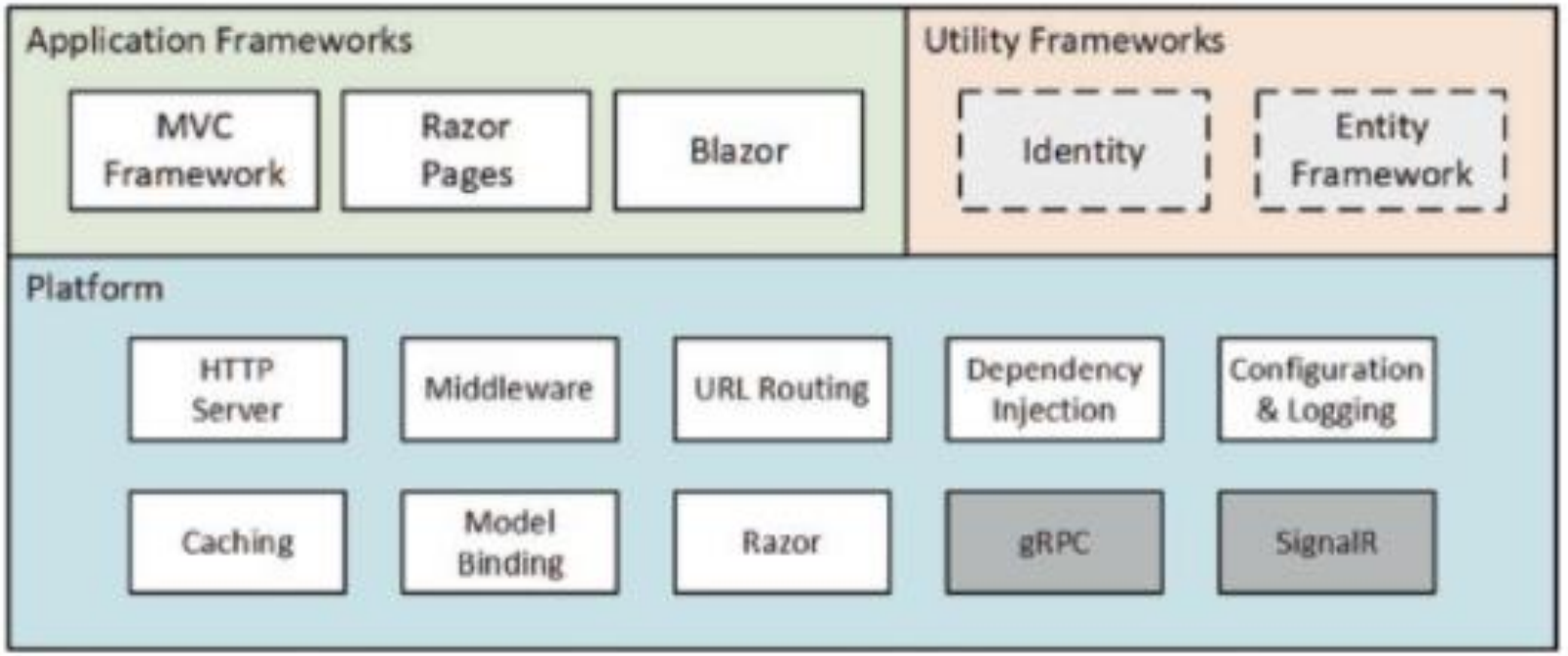
1.1. Giới thiệu ASP.Net Core

1.2. Cài đặt, tạo dự án, dịch và chạy ứng dụng ASP.NET Core

- **Giới thiệu chung về các công cụ cho ASP.NET Core**
- **Cài đặt ASP.NET Core SDK trên Windows**
- **Tạo dự án ASP.NET Core từ Visual Studio**



- **Application Frameworks** chứa những cái tên có lẽ tương đối quen thuộc như **MVC Framework**, **Razor Pages** hay **Blazor**. Đây là những framework giúp bạn xây dựng các dạng khác nhau của ứng dụng web.
- **Utility Frameworks** chứa những thứ cảm tưởng như không liên quan đến **Asp.net Core**: **Identity** và **Entity Framework**. Khối này chứa những framework hỗ trợ cho ứng dụng, bao gồm bảo mật và cơ sở dữ liệu.
- **Flatform** là những gì tạo nên nền tảng chung nhất mà mọi loại ứng dụng **Asp.net Core** đều cần sử dụng đến.



- **Application Frameworks**
  - **MVC**: là **framework** xây dựng theo nguyên lý phân chia nhiệm vụ (**Separation of Concerns**, **SoC**). **MVC** phân chia chức năng thành các phần của ứng dụng ra làm các nhóm độc lập, gọi là **Model**, **View** và **Controller**
  - **Razor Pages** là một **framework** đơn giản hướng tới xây dựng các trang web tự thân độc lập. Mô hình của **Razor Pages** đơn giản hơn so với **MVC**. **Razor Pages** có thể xem là **framework** tương tự với **Asp.net Web Pages**
  - **Blazor** là **framework** dành cho phát triển ứng dụng đơn trang (**SPA**) nhưng sử dụng ngôn ngữ **C#** thay cho **JavaScript** chạy trên trình duyệt. Đây là **framework** mới nhất và đang nhận được nhiều sự quan tâm.

- **Utility Frameworks:** chứa hai **framework** (không bắt buộc) nhưng lại được sử dụng gần như trong mọi ứng dụng **Asp.net Core**. Vì vậy, quá trình học **Asp.net Core** thực tế luôn gắn với hai **framework** này ở giai đoạn sau.
  - **Entity Framework Core** là một **ORM** (***Object-Relational Mapping***) giúp ứng dụng tương tác với cơ sở dữ liệu. **Entity Framework Core** giúp ánh xạ (hai chiều) giữa các bảng cơ sở dữ liệu (ví dụ, **SQL Server**) với các domain class của ứng dụng. **Entity Framework Core** không gắn với **Asp.net Core** mà có thể sử dụng trong bất kỳ ứng dụng **.NET** nào.
  - **Asp.net Core Identity** là framework dành cho xác thực (***authentication***) và xác minh quyền (***authorization***) người dùng trong ứng dụng.

- **Flatform:** chứa nhiều thành phần cấp thấp cần thiết cho việc nhận và xử lý truy vấn **HTTP**, cũng như tạo ra các phản hồi phù hợp.
  - **HTTP Server**, còn gọi là ***built-in server*** với tên gọi **Kestrel**, có nhiệm vụ tiếp nhận truy vấn **HTTP**. **Kestrel** có thể hoạt động độc lập (tích hợp trong một ứng dụng khác) hoặc phối hợp với một **web server** thông thường (**Apache**, **NGinx**, **IIS**).
  - **Middleware** là nhóm thành phần có nhiệm vụ xử lý truy vấn **HTTP**. Các **middleware** được xếp theo chuỗi. Khi truy vấn chạy qua mỗi khâu của chuỗi **middle** sẽ được xem xét xử lý.
  - **URL Routing** là cơ chế ánh xạ chuỗi truy vấn **HTTP** sang thực thi một phương thức nào đó. Do vậy, mỗi **URL** (địa chỉ bạn gửi về **server**) sẽ tương ứng với thực thi một phương thức trên server.
  - **Razor** là cơ chế sinh ra **HTML** từ dữ liệu và logic của chương trình. **Razor** được gọi là **view engine** trong **Asp.net Core**. **Razor** sử dụng loại cú pháp đặc biệt kết hợp giữa **C#** và **HTML**.
  - **Model Binding** là cơ chế ánh xạ dữ liệu chứa trong truy vấn **HTTP** sang tham số của phương thức cần thực thi. Nhờ cơ chế **Model Binding**, việc xây dựng các phương thức trong **Asp.net Core** đơn giản như bất kỳ phương thức **c#** thông thường nào.

- **Flatform:**
  - **Dependency Injection** là cơ chế cho phép tự động sinh và chèn object vào một object khác. **Asp.net Core** xây dựng sẵn cơ chế này cho bạn mà không cần đến một thư viện thứ ba (như **Ninject**, **Unity**).
  - **Configuration & Logging** là cơ chế hỗ trợ cấu hình và lưu vết quá trình thực thi ứng dụng.
  - **Caching** là cơ chế lưu tạm để tăng hiệu suất cho ứng dụng.

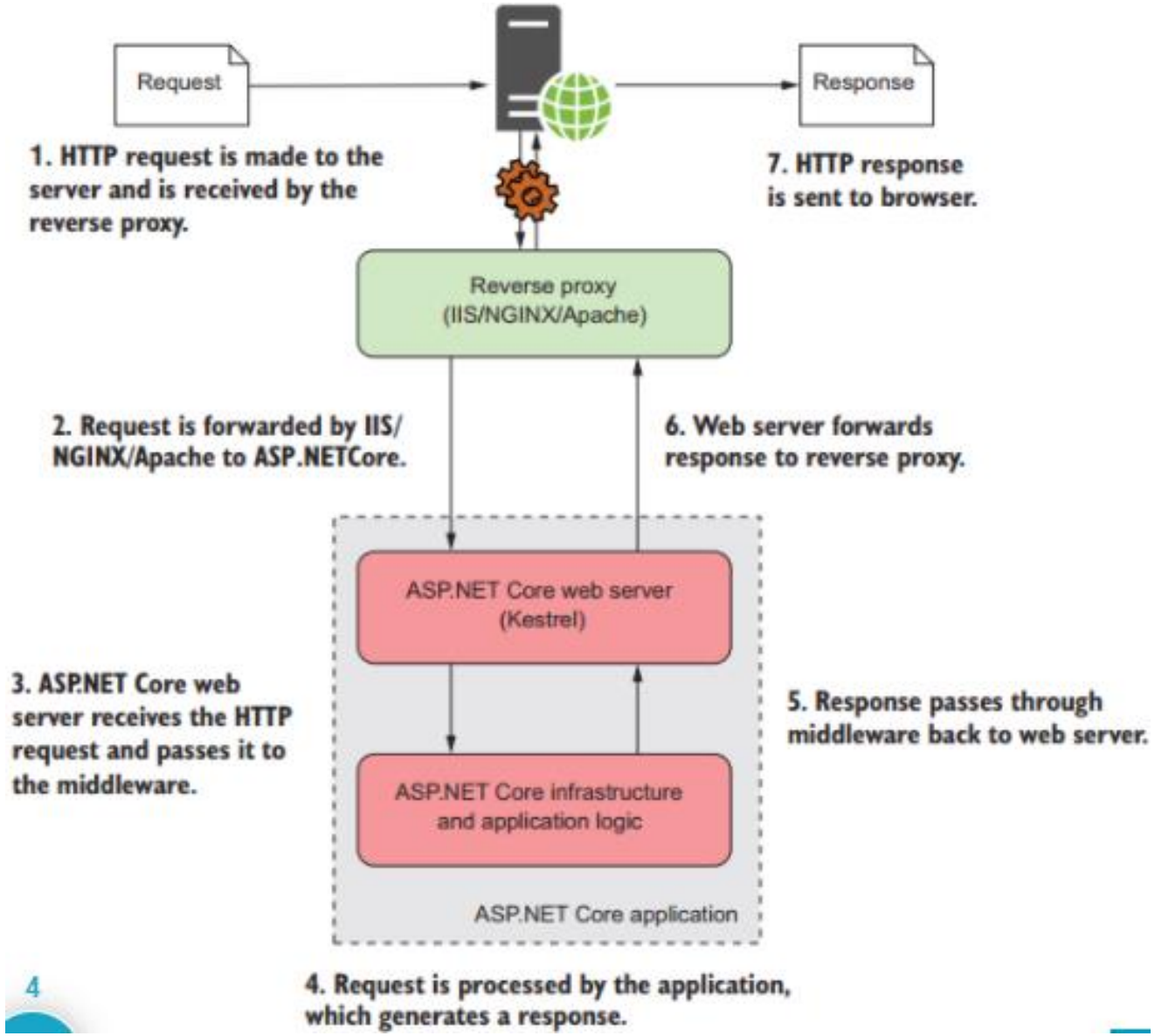
Chương 1. Giới thiệu chung về ASP.NET Core

1.1. Giới thiệu ASP.Net Core

1.2. Cài đặt, tạo dự án, dịch và chạy ứng dụng ASP.NET Core

1.3. Cấu trúc ASP.NET Core

1.4. Cách hoạt động của ứng dụng ASP.NET Core



- 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**.
- 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***

- 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. Khi đó sẽ 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.

- **Thực hành 1**: Tạo dự án **ASP.NET Core** trống

- **Connected Services:** danh sách các dịch vụ online mà project này sử dụng, ví dụ **WCF Web Service**, các dịch vụ trên đám mây **Azure**, v.v..
- **Dependencies:** danh sách các gói thư viện **NuGet** được cài đặt và sử dụng trong project. Dù là một project trống đơn giản nhất cũng phải sử dụng một số gói thư viện nhất định.
- **Properties:** chứa thông tin cấu hình của project. Nếu mở bằng **Visual Studio**, bạn sẽ gặp giao diện đồ họa quen thuộc vốn có trong tất cả các project **C#**. Trong **ASP.NET Core**, các thông tin này thực chất được lưu trong một file **json** mà bạn có thể trực tiếp điều chỉnh. Cách thức lưu mới này rất hữu ích nếu bạn sử dụng một trình viết code khác (như **Visual Studio Code**).
- **appsettings.json:** chứa các thông tin cấu hình cho hoạt động của ứng dụng, như ***connection string***, các biến môi trường, tham số dòng lệnh, v.v.. Bạn sẽ lần lượt gặp và làm việc với file này trong quá trình học.
- **Program.cs:** File này chứa class **Program** chịu trách nhiệm cấu hình nền tảng (**infrastructure**) của ứng dụng. Class này cũng chứa **entry point** của ứng dụng. Nhìn chung, các cấu hình bên trong **Program** hầu như không thay đổi trong mỗi **project**. Nếu không có yêu cầu gì đặc biệt, bạn sẽ không cần điều chỉnh gì trong class **Program** và file **Program.cs**.

- **Startup.cs:** File này chứa **class** với các phương thức cấu hình cho hoạt động của ứng dụng, ví dụ bạn sẽ sử dụng những **middleware** nào, thứ tự sắp xếp các **middleware** trong **pipeline** ra sao. Bạn cũng có thể cấu hình sử dụng các loại dịch vụ (**service**) nào, như **Dependency Injection**, **Logging**.
- **wwwroot**: Thư mục chứa các file tĩnh như **html**, **css**, **javascript**, hình ảnh. Các file tĩnh phải nằm trong thư mục này thì mới có thể cung cấp cho trình duyệt. Trong **ASP.NET Core**, thư mục này có tên gọi là riêng là **web root**

- **Startup** là class dùng cho cấu hình hoạt động của ứng dụng. Cụ thể hơn, class này chịu trách nhiệm (**1**) đăng ký các *dịch vụ* (***service***) được sử dụng trong ứng dụng và (**2**) đăng ký và ghép nối các ***middleware*** dùng để xử lý truy vấn **HTTP**.
- Bắt buộc phải có hai phương thức sau:
  - **ConfigureServices**: là phương thức dùng để đăng ký các dịch vụ,
  - **Configure:** là phương thức dùng để đăng ký các **middleware**.

- là một khái niệm mới trong **ASP.NET Core** dùng để chỉ một **module/class** có khả năng xử lý truy vấn **HTTP** và trả lại kết quả ở dạng **HTTP Response**.
- Sau khi **Kestrel** nhận truy vấn từ **reverse proxy** (**IIS**, **Apache**, **Nginx**), truy vấn này sẽ di chuyển dọc theo chuỗi (**pipeline**) class **middleware** cho đến khi một nó gặp một **middleware** phù hợp để xử lý. Khi **middleware** xử lý xong, kết quả sẽ trả ngược lại đường đi của truy vấn.
- Do một truy vấn có thể chứa những yêu cầu rất khác biệt, đồng thời mỗi ứng dụng có những mục đích khác nhau, tổ hợp **middleware** cho mỗi ứng dụng không hoàn toàn giống nhau. "***Tổ hợp middleware***" thể hiện qua loại **middleware** được sử dụng và thứ tự bố trí chúng trên **pipeline**.
- Cơ chế đăng ký và ghép nối middleware này giúp ứng dụng **ASP.NET Core** chỉ cần sử dụng những gì mình cần, không dùng những chức năng dư thừa, qua đó làm ứng dụng nhẹ và nhanh hơn.

- **Thực hành 2**: Cấu hình sử dụng ***static file***
  - File tĩnh (**static file**) là những file đã được tạo sẵn và được cung cấp nguyên vẹn cho trình duyệt khi được yêu cầu.
  - Các loại file sau đây được coi là static file: **html** (file dữ liệu chính của các trang web tĩnh), **css** (***cascading style sheet*** – chịu trách nhiệm định dạng trang), **js** (file mã nguồn ***Javascript***), các loại file ảnh.
  - Các file tĩnh kết hợp lại với nhau tạo ra các trang web tĩnh (**static page**) – dạng thức cơ bản nhất của web.
  - Phục vụ các file static là nhiệm vụ cơ bản nhất của các **web server**. Tuy nhiên trong **ASP.NET Core** tính năng này yêu cầu bạn phải thực hiện một số thao tác cấu hình. Nói cách khác, mặc định **ASP.NET Core** không bật tính năng này.

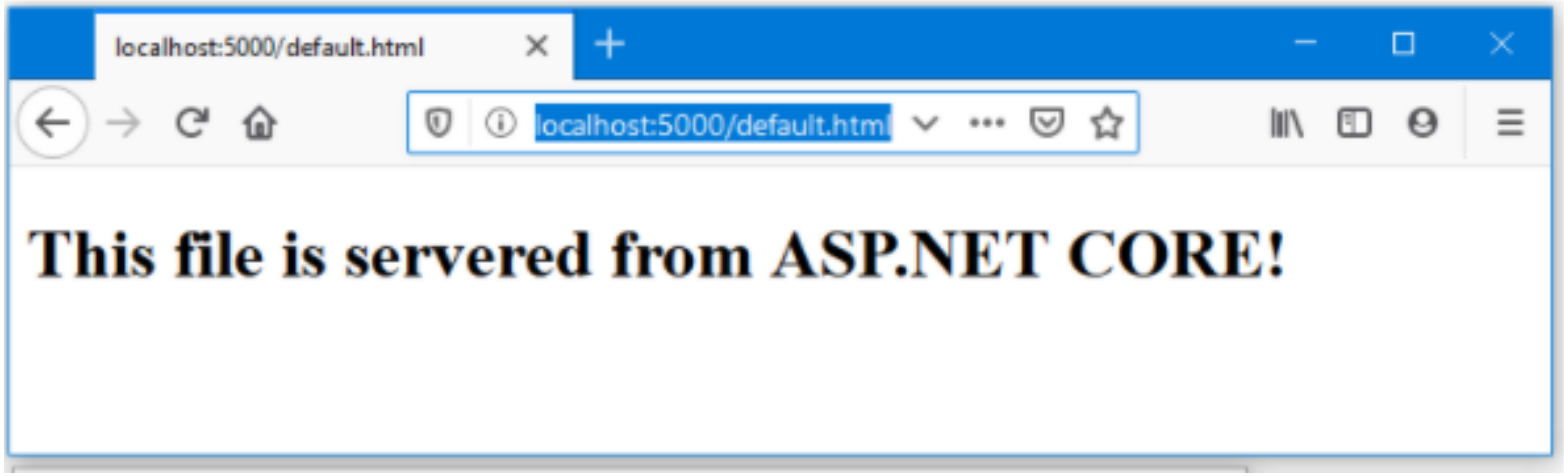
- **Thực hành 2**: Cấu hình sử dụng ***static file***
  - **Bước 1**: Tạo thư mục **wwwroot**.
    - Mặc định **ASP.NET Core** yêu cầu tất cả các file tĩnh phải nằm trong thư mục **wwwroot** trực thuộc thư mục dự án.
    - Vì vậy, nếu trong project bạn chưa có thư mục này, hãy tạo mới bằng cách click phải vào tên **project ->chọn Add ->New Folder** ->đặt tên cho thư mục vừa tạo là **wwwroot**.
  - **Bước 2**: Tạo file ***default.html***
    - Trong thư mục **wwwroot** tạo file ***default.html*** bằng cách: click phải vào thư mục **wwwroot -> Add ->New Item** ->chọn **template HTML Page** ->đặt tên file là ***default.html*** ->**OK**.
    - Mở file vừa tạo và điều chỉnh nội dung như sau:

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <title>Hello ASP.NET Core</title>
</head>
<body>
    <h1>This file is served from ASP.NET  CORE!</h1>
</body>
</html>
```

- **Thực hành 2**: Cấu hình sử dụng ***static file***
  - **Bước 3**: Điều chỉnh class ***Startup***.
    - Mở file ***Startup.cs*** và điều chỉnh phương thức **Configure** như sau (bổ sung dòng ***app.UseStaticFiles();***)
    - dòng ***app.UseStaticFiles();*** yêu cầu gắn thêm middleware xử lý **static** file vào đầu **pipeline** của ứng dụng. Nghĩa là nếu trình duyệt yêu cầu một file tĩnh, ứng dụng **ASP.NET Core** sẽ tìm kiếm file tương ứng trong **wwwroot**. Nếu tìm thấy sẽ trả lại ngay file tương ứng và kết thúc **pipeline** xử lý.

```
1.  public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
2.  {
3.      if (env.IsDevelopment())
4.      {
5.          app.UseDeveloperExceptionPage();
6.      }
7.
8.      app.UseStaticFiles();
9.
10.     app.UseRouting();
11.
12.     app.UseEndpoints(endpoints =>
13.     {
14.         endpoints.MapGet("/", async context =>
15.         {
16.             await context.Response.WriteAsync("Hello World!");
17.         });
18.     });
19. }
```

- **Thực hành 2**: Cấu hình sử dụng ***static file***
  - **Bước 4:** Chạy thử ứng dụng.
    - Bạn gõ **URL** yêu cầu lấy file ***default.html*** vào thanh địa chỉ của trình duyệt. Kết quả như dưới đây.



• Thực hành 3: JavaScript và CSS

• Bước 1: *JavaScript*.

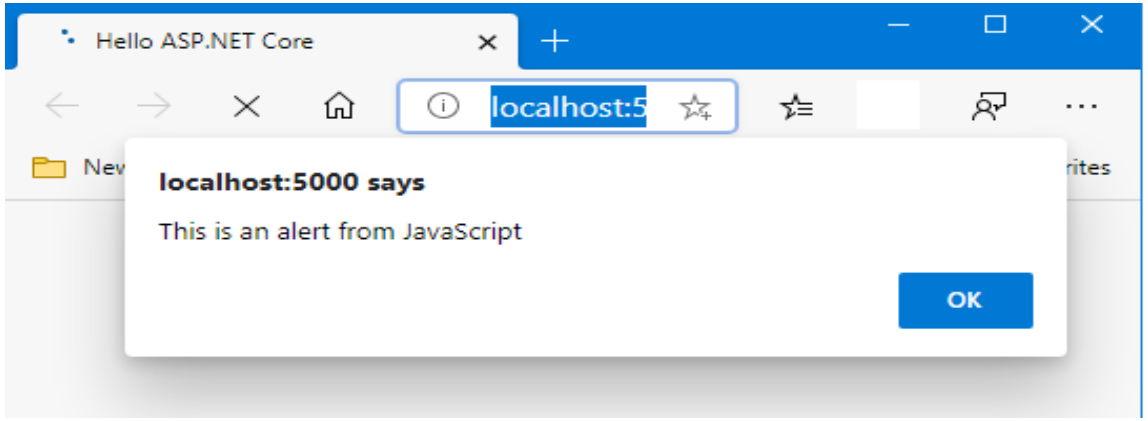
- Thêm file ***script.js*** vào thư mục **wwwroot** và viết code như sau:

```
1.  alert("This is an alert from JavaScript");
```

- Điều chỉnh file ***default.html*** như sau:

```
1.  <!DOCTYPE html>
2.  <html>
3.  <head>
4.      <meta charset="utf-8" />
5.      <title>Hello ASP.NET Core</title>
6.      <script src="script.js"></script>
7.  </head>
8.  <body>
9.      <h1>This file is served from ASP.NET  CORE!</h1>
10. </body>
11. </html>
```

- Khi chạy thử bạn sẽ thu được kết quả như sau:



• Thực hành 3: JavaScript và CSS

• Bước 2: *CSS*.

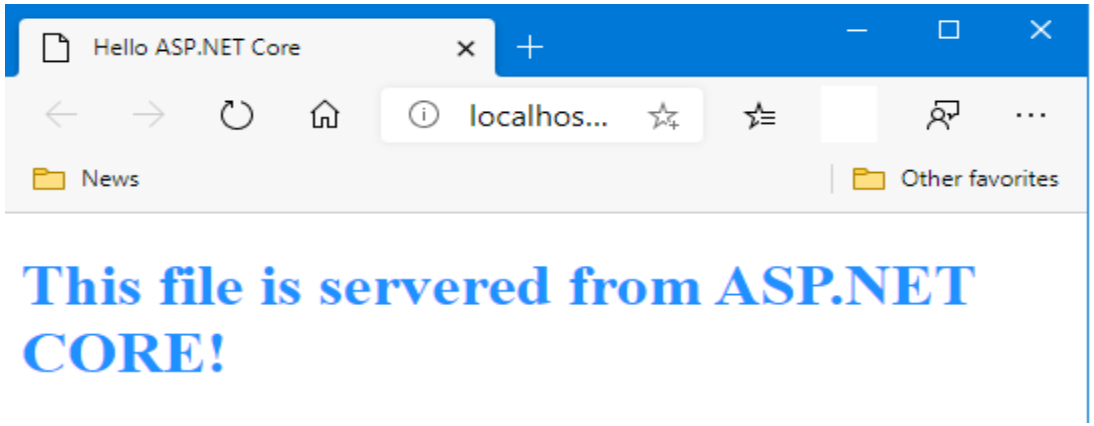
- Thêm file ***styles.css*** vào thư mục **wwwroot** và viết code như sau:

```
1.  h1 {color: dodgerblue;}
```

- Điều chỉnh file ***default.html*** như sau:

```
1.  <!DOCTYPE html>
2.  <html>
3.  <head>
4.      <meta charset="utf-8" />
5.      <title>Hello ASP.NET Core</title>
6.      <link rel="stylesheet" type="text/css" href="styles.css" />
7.      <script src="script.js"></script>
8.  </head>
9.  <body>
10.     <h1>This file is served from ASP.NET  CORE!</h1>
11. </body>
12. </html>
```

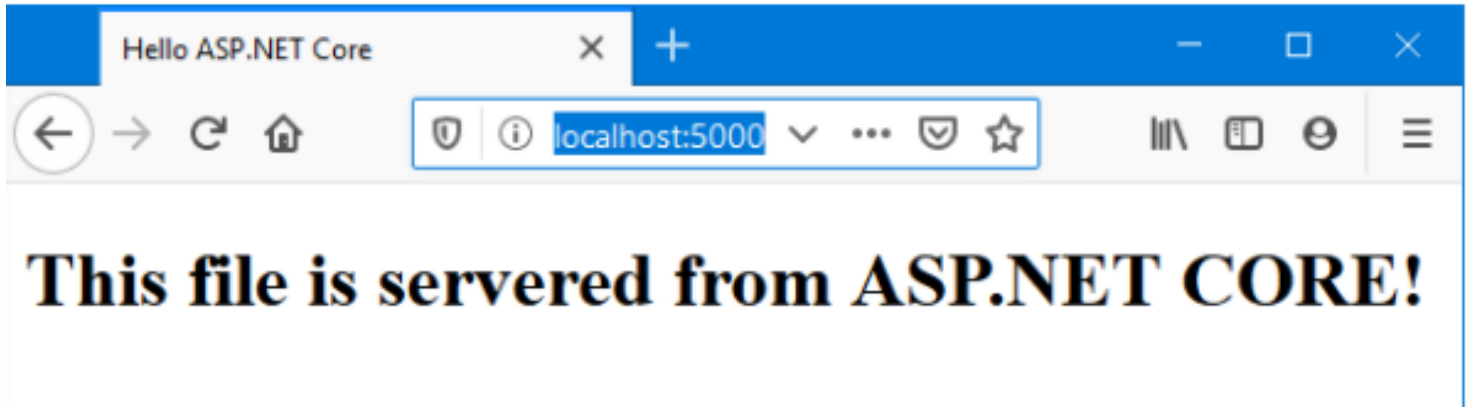
- Khi chạy thử bạn sẽ thu được kết quả như sau:



- **Thực hành 4: Cấu hình file mặc định**
  - **Bước 1:** Bổ sung **middleware DefaultFiles**.
    - Mở file **Startup.cs** và điều chỉnh phương thức **Configure** như sau:

```
22.  public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
23.  {
24.      if (env.IsDevelopment())
25.      {
26.          app.UseDeveloperExceptionPage();
27.      }
28.
29.      var options = new DefaultFilesOptions();
30.      options.DefaultFileNames.Add("home.html");
31.      options.DefaultFileNames.Add("home.htm");
32.      app.UseDefaultFiles(options);
33.
34.      app.UseStaticFiles();
35.
36.      app.UseRouting();
37.
38.      app.UseEndpoints(endpoints =>
39.      {
40.          endpoints.MapGet("/", async context =>
41.          {
42.              await context.Response.WriteAsync("Hello World!");
43.          });
44.      });
45.  }
```

- **Thực hành 4: Cấu hình file mặc định**
  - **Bước 2:** Chạy thử chương trình.
    - Chạy chương trình và nhập **URL** là ***localhost:5000***. Bạn thu được cùng kết quả như trong ***phần thực hành 2***. Sự khác biệt là giờ bạn không cần chỉ định rõ file cần lấy nữa.

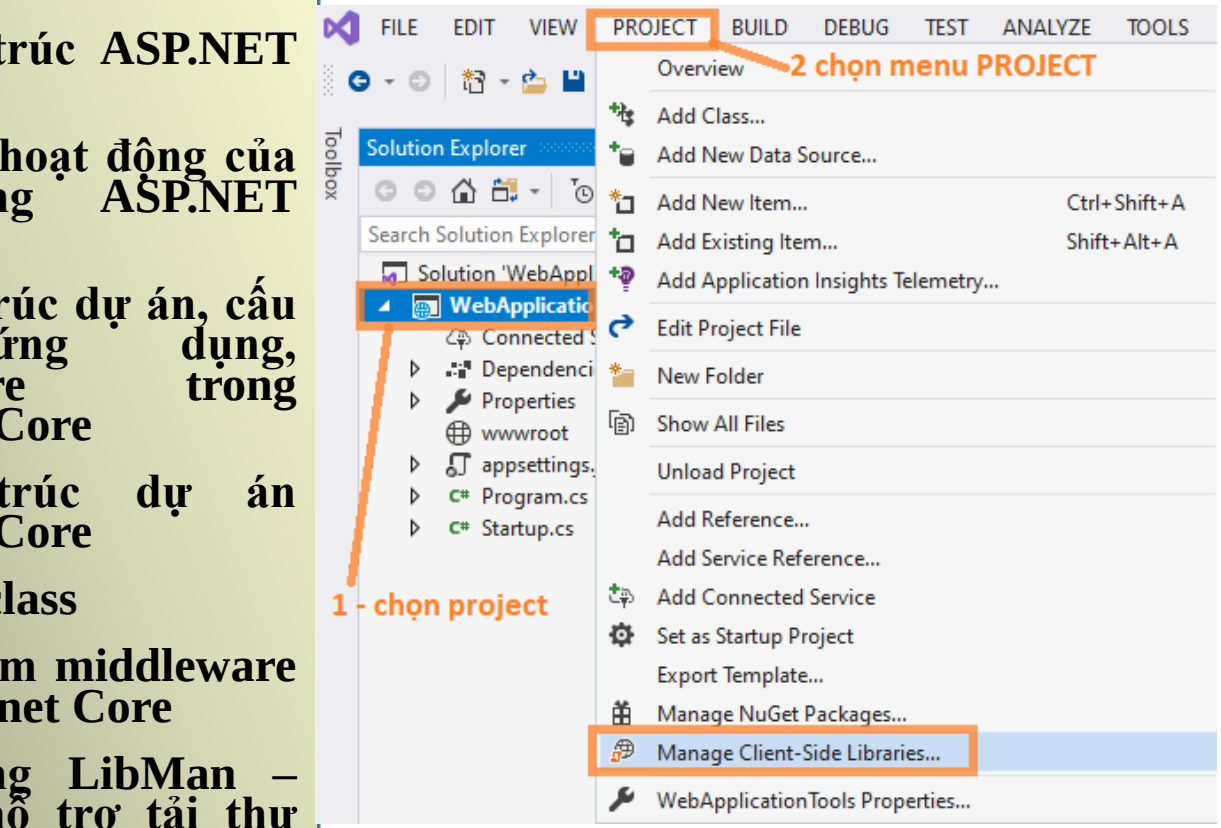


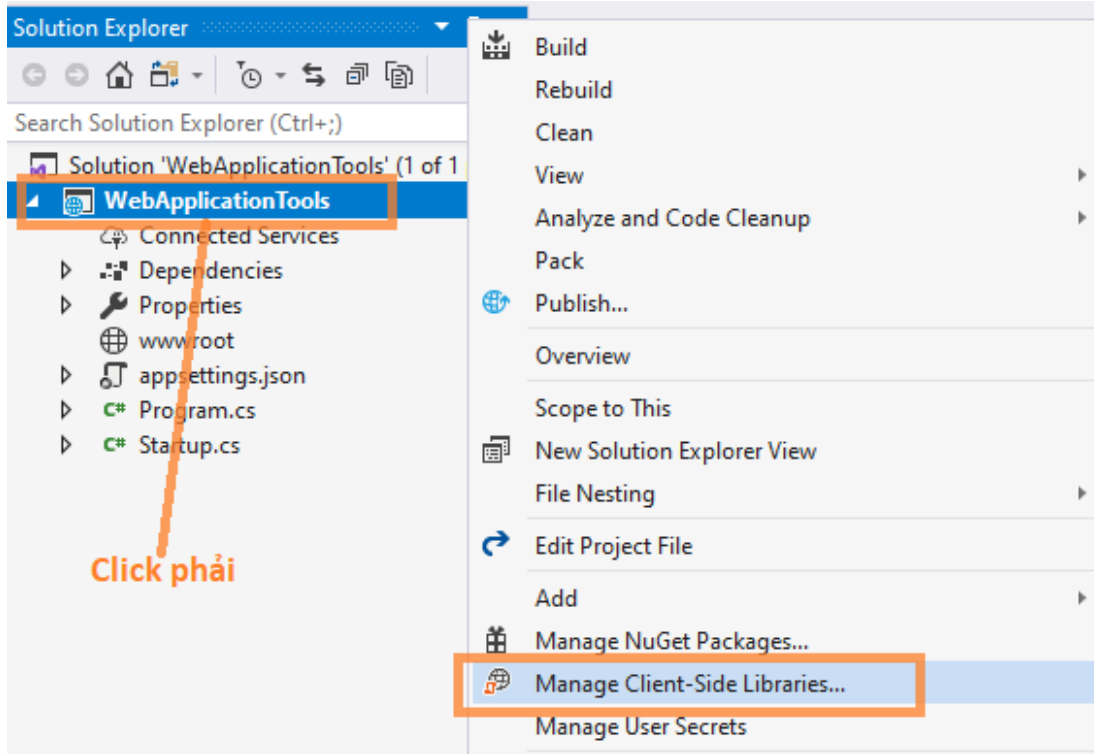
- **LibMan**: (***Library Manager***) là một công cụ của **Visual Studio** hỗ trợ tải các thư viện **CSS** và **JavaScript** cho ứng dụng **ASP.NET Core** được xây dựng với mục đích như vậy.
- **LibMan** là một công cụ đơn giản, nhẹ và tiện lợi giúp tìm và tải các file thư viện từ các nguồn trên Internet về dự án.

- **Tạo project thử nghiệm**
  - **Bước 1:** Tạo một **empty project** mới:
  - **Bước 2:** Cấu hình sử dụng ***StaticFiles middleware***:

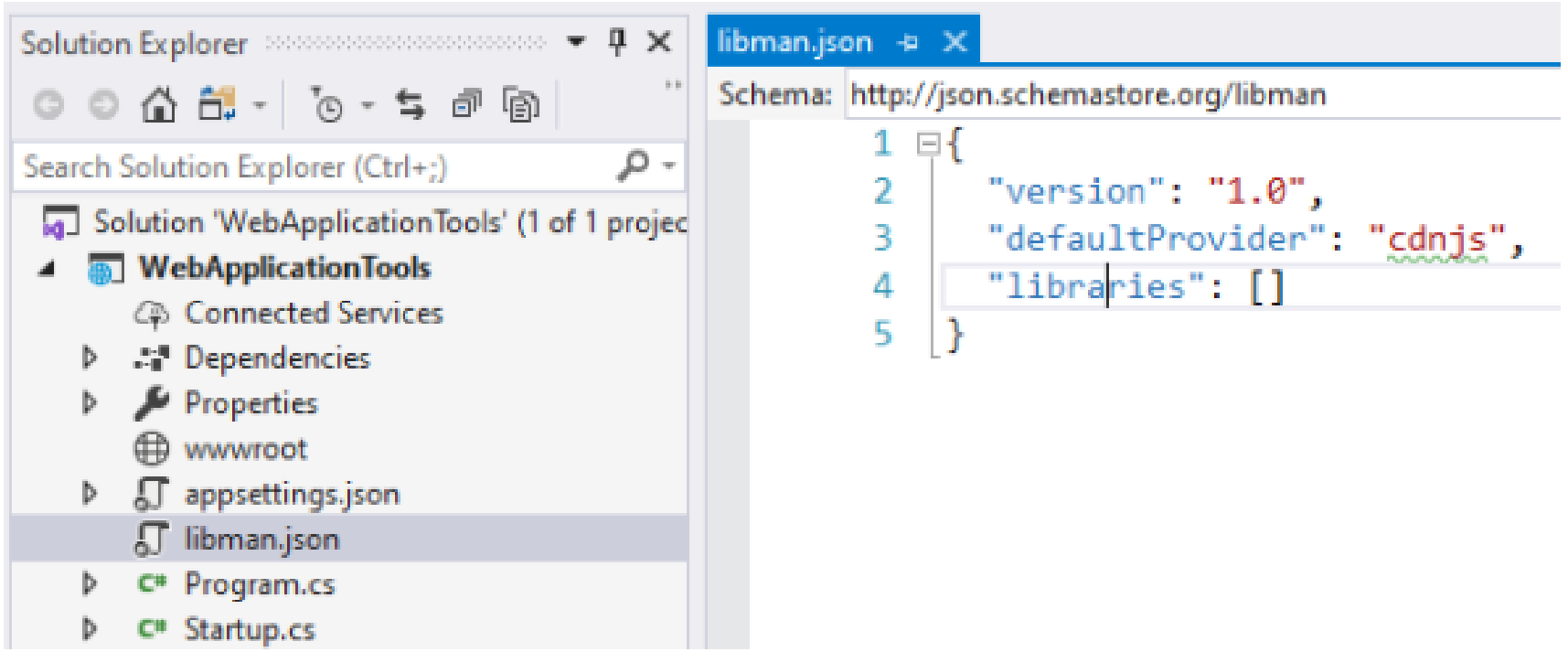
```
1.  public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
2.  {
3.      if (env.IsDevelopment())
4.      {
5.          app.UseDeveloperExceptionPage();
6.      }
7.
8.      app.UseStaticFiles();
9.
10.     app.UseRouting();
11.
12.     app.UseEndpoints(endpoints =>
13.     {
14.         endpoints.MapGet("/", async context =>
15.         {
16.             await context.Response.WriteAsync("Hello World!");
17.         });
18.     });
19. }
```

- **Tạo project thử nghiệm**
  - **Bước 3:** Tạo thư mục **wwwroot** trực thuộc project
  - **Bước 4:** Chọn chức năng ***Manage Client-Side Libraries*** theo một trong hai cách sau:





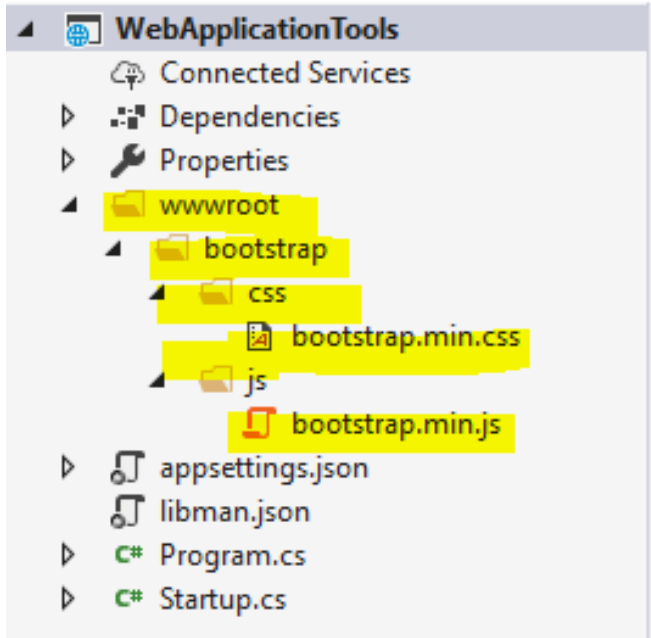
- **Tạo project thử nghiệm**
  - **Bước 3:** Tạo thư mục **wwwroot** trực thuộc project
  - **Bước 4:** Chọn chức năng **Manage Client-Side Libraries**
    - Khi này file ***libman.json*** với nội dung như sau sẽ được thêm vào **project**. Đây là file cấu hình của **LibMan**.



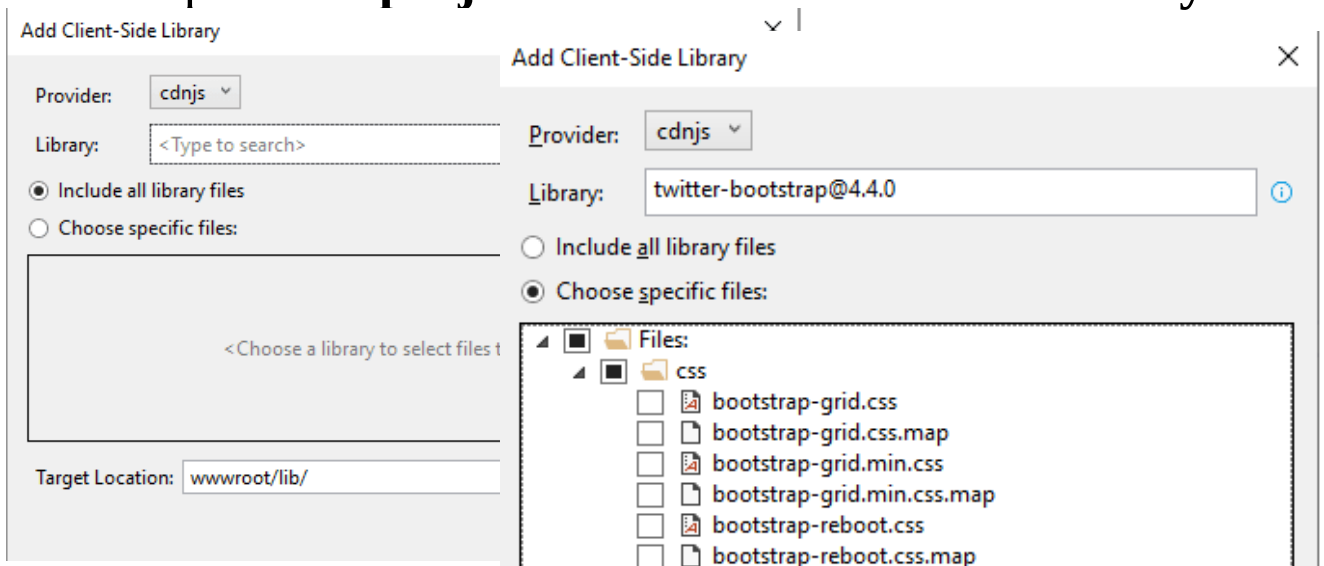
- **Cài đặt thư viện BootStrap:** 2 cách
  - **Option 1** – Sử dụng file cấu hình ***libman.json***
    - Nhập trực tiếp thông tin vào ***libman.json***:

```
{
  "version": "1.0",
  "defaultProvider": "cdnjs",
  "libraries": [
    {
      "library": "twitter-bootstrap@4.4.0",
      "destination": "wwwroot/bootstrap",
      "files": [ "css/bootstrap.min.css", "js/bootstrap.min.js" ]
    }
  ]
}
```

    - Ngay khi lưu file ***libman.json***, **LibMan** sẽ tải các file cần thiết vào đúng thư mục bạn chỉ định:



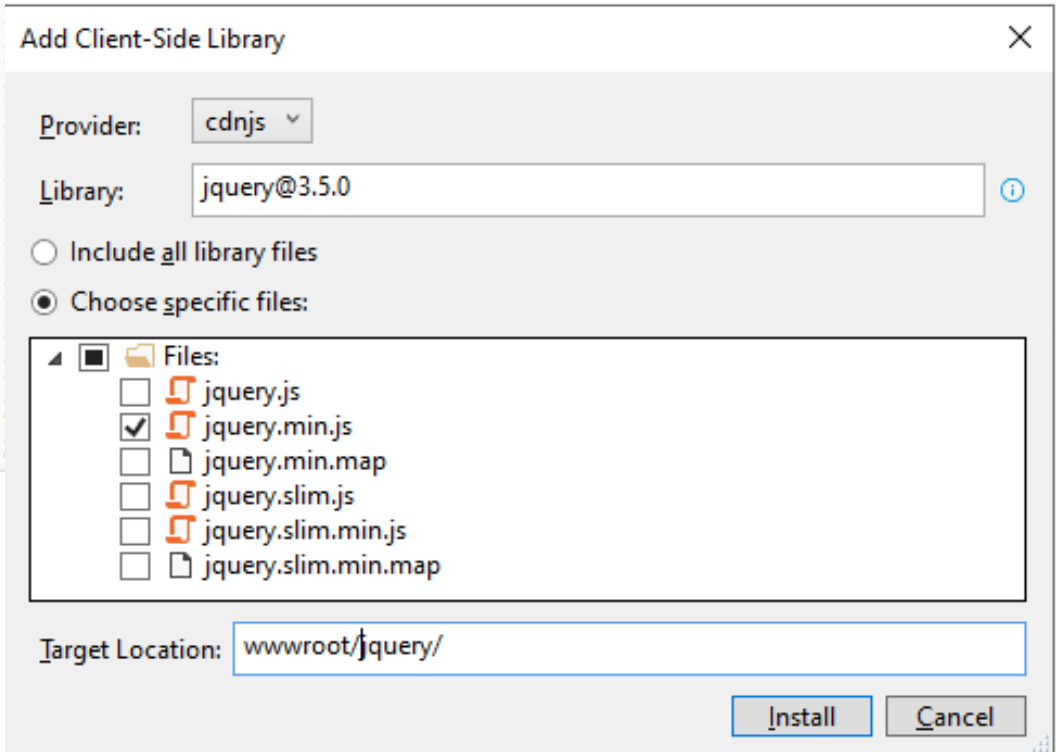
- **Cài đặt thư viện BootStrap:** 2 cách
  - **Option 2** – Sử dụng giao diện đồ họa
    - Click phải vào **project ->Add ->Client Side Library** để mở cửa sổ sau:



- Nhập tên thư viện **Library**: vào ô textbox

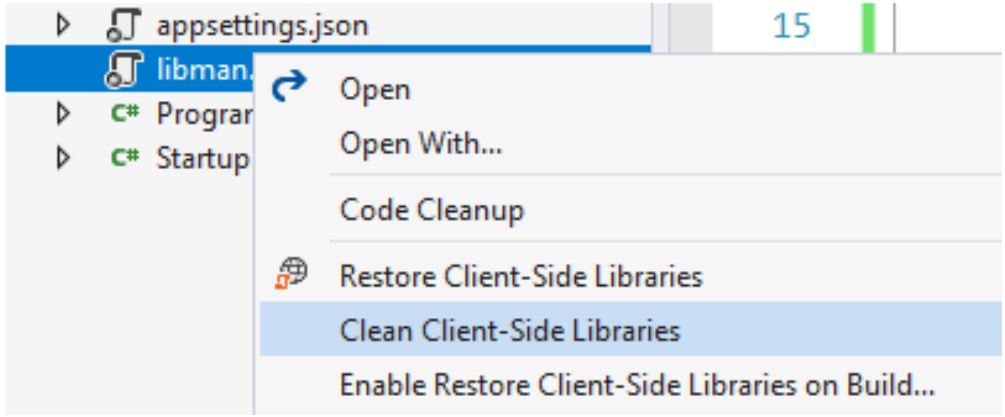
- **Cài đặt jQuery:** Do **BootStrap** yêu cầu phải có thêm **jQuery**, hãy thực hiện thao tác tương tự như trên để cài đặt **jQuery** vào thư mục **wwwroot/lib**:

```
{
  "version": "1.0",
  "defaultProvider": "cdnjs",
  "libraries": [
    {
      "library": "twitter-bootstrap@4.4.0",
      "destination": "wwwroot/bootstrap/",
      "files": [
        "js/bootstrap.min.js",
```

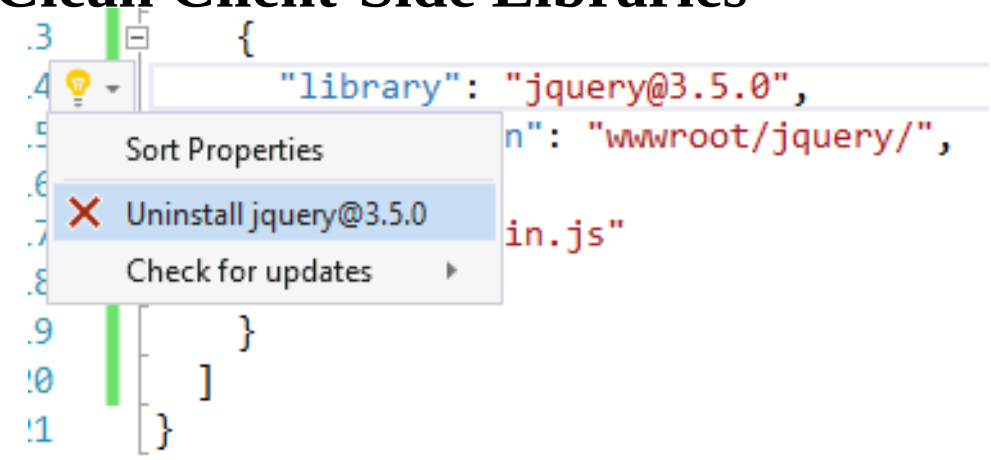


• Một số kỹ thuật khác với LibMan

- Xóa bỏ toàn bộ thư viện: Thao tác này sẽ xóa bỏ tất cả các file thư viện đã tải về **project** nhưng giữ nguyên file cấu hình ***libman.json***.



- Xóa hoặc cập một thư viện: Click phải chuột vào file ***libman.json*** và chọn **Clean Client-Side Libraries**



Thank you!