# LỜI CẢM ƠN

# MỤC LỤC

# DANH MỤC ẢNH

Ảnh 1 12

# DANH MỤC VIẾT TẮT

|  |  |  |  |
| --- | --- | --- | --- |
| STT | Phần viết tắt | Phần viết đầy đủ | Giải nghĩa |
| 01 | MVC | Model-View-Controler |  |
| 02 | API |  |  |
|  | DI | Dependency Injection |  |
|  | HTTP |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

# Chương I. GIỚI THIỆU

## PHẦN 1.1. GIỚI THIỆU ASP.NET Core

ASP.NET Core là một framework mã nguồn mở được phát triển bởi Microsoft, dùng để xây dựng các ứng dụng web, dịch vụ web và API. Đây là phiên bản tái cấu trúc của ASP.NET, với nhiều cải tiến về hiệu suất, khả năng mở rộng và tính linh hoạt.

**Các đặc điểm nổi bật:**

* **Đa nền tảng:** ASP.NET Core có thể chạy trên Windows, macOS và Linux, cho phép các nhà phát triển xây dựng ứng dụng mà không bị ràng buộc vào một hệ điều hành cụ thể.
* **Hiệu suất cao:** Với kiến trúc nhẹ và tối ưu hóa cho môi trường đám mây, ASP.NET Core mang lại hiệu suất xử lý tốt hơn so với các phiên bản trước.
* **Thiết kế mô-đun:** Các thành phần trong ASP.NET Core được thiết kế theo kiểu mô-đun, cho phép người dùng chọn lựa và thêm vào các thư viện cần thiết, giúp giảm kích thước ứng dụng và tăng tính hiệu quả.
* **Kiến trúc MVC:** ASP.NET Core hỗ trợ mô hình kiến trúc Model-View-Controller (MVC), giúp phân tách các thành phần trong ứng dụng, làm cho mã nguồn dễ quản lý và bảo trì hơn. MVC cho phép phát triển các ứng dụng web tương tác và dễ dàng mở rộng.
* **Hỗ trợ Dependency Injection:** ASP.NET Core tích hợp sẵn hỗ trợ Dependency Injection, giúp quản lý các phụ thuộc giữa các thành phần của ứng dụng một cách hiệu quả, nâng cao tính linh hoạt và khả năng kiểm thử.
* **Tích hợp với các công nghệ hiện đại:** Framework này hỗ trợ tích hợp với nhiều công nghệ hiện đại như Entity Framework Core cho việc truy xuất dữ liệu, SignalR cho ứng dụng thời gian thực, và Razor Pages để phát triển giao diện người dùng nhanh chóng.
* **Hỗ trợ tốt cho bảo mật:** ASP.NET Core cung cấp nhiều tính năng bảo mật mạnh mẽ như Xác thực (Authentication), Ủy quyền (Authorization), bảo vệ CSRF (Cross-Site Request Forgery) và XSS (Cross-Site Scripting). Điều này đảm bảo ứng dụng được bảo mật và an toàn hơn khi triển khai.
* **Hỗ trợ kiểm thử:** ASP.NET Core dễ dàng hỗ trợ việc kiểm thử ứng dụng (Unit Test và Integration Test), nhờ vào tính năng Dependency Injection, giúp tách biệt các lớp và kiểm soát được các phụ thuộc trong quá trình kiểm thử.
* **Cộng đồng và tài liệu phong phú:** ASP.NET Core được hỗ trợ bởi một cộng đồng lớn và tích cực, cung cấp nhiều tài liệu, hướng dẫn và thư viện mã nguồn mở, giúp các lập trình viên dễ dàng tìm hiểu và áp dụng.

**Ứng dụng thực tiễn:** ASP.NET Core được sử dụng rộng rãi trong việc phát triển các ứng dụng web doanh nghiệp, trang thương mại điện tử, API cho ứng dụng di động và nhiều loại ứng dụng khác, nhờ vào tính linh hoạt và hiệu suất của nó.

ASP.NET Core là một framework mạnh mẽ, linh hoạt và hiện đại, giúp các nhà phát triển xây dựng các ứng dụng web, dịch vụ API và ứng dụng đa nền tảng với hiệu suất cao. Kiến trúc nhẹ, mở rộng dễ dàng, và sự hỗ trợ từ cộng đồng là những lợi thế lớn mà nó mang lại.

## PHẦN 1.2. LÝ DO CHỌN ĐỀ TÀI

Trong bối cảnh công nghệ ngày càng phát triển, việc xây dựng các ứng dụng web nhanh chóng, hiệu quả và linh hoạt là một yêu cầu thiết yếu. ASP.NET Core nổi bật như một framework đáp ứng tốt những yêu cầu này, giúp nhà phát triển tạo ra các ứng dụng chất lượng cao.

**ASP.NET Core được chọn làm đề tài vì có những đặc điểm nổi trội như sau:**

* **Đa nền tảng:** ASP.NET Core cho phép phát triển ứng dụng trên nhiều hệ điều hành khác nhau như Windows, macOS và Linux. Điều này không chỉ mang lại tính linh hoạt mà còn mở rộng cơ hội cho các nhà phát triển trong việc triển khai ứng dụng.
* **Hiệu suất và khả năng mở rộng:** Với kiến trúc tối ưu hóa và khả năng xử lý cao, ASP.NET Core là lựa chọn lý tưởng cho các ứng dụng yêu cầu hiệu suất tốt và khả năng mở rộng dễ dàng. Việc nghiên cứu framework này sẽ cung cấp những kiến thức cần thiết để xây dựng các ứng dụng có quy mô lớn.
* **Hỗ trợ công nghệ hiện đại:** ASP.NET Core tích hợp dễ dàng với các công nghệ và công cụ hiện đại như Entity Framework Core, SignalR, và các dịch vụ đám mây. Điều này giúp nhà phát triển dễ dàng triển khai các giải pháp phong phú và đa dạng, từ đó nâng cao chất lượng ứng dụng.
* **Tài liệu và cộng đồng phong phú:** Framework này được hỗ trợ bởi một cộng đồng lớn và rất nhiều tài liệu hướng dẫn, giúp các lập trình viên dễ dàng tiếp cận và học hỏi. Việc chọn đề tài này không chỉ mang lại kiến thức về kỹ thuật mà còn tạo cơ hội kết nối với cộng đồng phát triển.

Ngoài ra việc nghiên cứu và thực hành với ASP.NET Core sẽ giúp cải thiện kỹ năng lập trình, hiểu rõ hơn về kiến trúc ứng dụng web, và nâng cao khả năng giải quyết vấn đề trong phát triển phần mềm.

# Chương II. TỔNG QUAN VỀ ASP.NET CORE

## PHẦN 1.1. ASP.NET Core là gì?

### 1.1.1. ASP.NET Core là gì?

ASP.NET Core là một tập hợp các thư viện chuẩn như một framework mã nguồn mở đa nền tảng *(cross-platform)* cho việc xây dựng những ứng dụng hiện tại dựa trên kết nối đám mây, giống như web apps, IoT và backend cho mobile.

ASP.NET Core không phải là phiên bản tiếp theo của ASP.NET. Nó là một cái tên mới được xây dựng từ đầu. Nó có một sự thay đổi lớn về kiến trúc và kết quả là nó gọn hơn, phân chia module tốt hơn.

Từ Core trong ASP.NET Core rất dễ nhầm. Tên đó gợi nhớ việc ứng dụng ASP.NET Core chỉ có thể được build trên .NET Core, điều đó là sai. Ứng dụng ASP.NET Core có thể được phát triển sử dụng .NET Core hoặc .NET Framework. Ứng dụng ASP.NET Core được xây dựng sử dụng .NET Core có thể chạy trên bất cứ hệ điều hành nào trong khi nếu được xây dựng trên .NET Framework chỉ có thể chạy trên Windows.

ASP.NET Core bao gồm các thành phần theo hướng module nhằm tối thiểu tài nguyên và chi phí phát triển.

### 1.1.2. .NET Core là gì?

.NET Core là môi trường thực thi. Nó được thiết kế lại hoàn toàn của .NET Framework. Mục tiêu chính của .NET Core là hỗ trợ phát triển ứng dụng đa nền tảng cho ứng dụng .NET. Nó được hỗ trợ trên Windows, Mac OS và Linux. .NET Core là một framework mã nguồn mở được xây dựng và phát triển bởi Microsoft và cộng đồng .NET trên Github

NET Core là một tập con của Full .NET Framework. WebForms, Windows Forms, WPF không phải là một phần của .NET Core.

Nó cũng triển khai đặc điểm của .NET Standard.

### 1.1.3. .NET Standard là gì?

.NET Standard là một đặc tả chuẩn của .NET API hướng tới hỗ trợ trên tất cả các triển khai của nền tảng .NET. Nó định nghĩa một tập các quy tắc thống nhất cần thiết để hỗ trợ tất cả các ứng dụng trên nền .NET.

### 1.1.4. Sự khác nhau giữa .NET Core và .NET Framework

.NET Framework là môi trường cũ hơn và đã tồn tại trên Windows nhiều năm nay. .NET Core hỗ trợ các tính năng trong tập con của .NET Framwork. Các tính năng như WebForms, Windows Forms, WPF chưa được đưa lên .NET Core. .NET Framwork chỉ chạy trên Windows trong khi .NET Core có thể chạy trên bất cứ nền tảng nào.

## PHẦN 1.2. Lịch sử phát triển của ASP.NET Core

Phiên bản đầu tiên của ASP.NET được gọi là ASP.NET 1.0. Chương trình đã được giới thiệu cùng với . NET Framework 1.0 vào năm 2002. Kể từ đó, Microsoft đã liên tục cải tiến, phát triển ASP.NET.

ASP.NET Core có một số thay đổi kiến trúc lớn, đó là kết quả của việc học hỏi rất nhiều từ các framework module hóa khác. ASP.NET Core không còn dựa trên System.Web.dll nữa. Nó được dựa trên một tập hợp các gói, các module hay cũng được gọi là các Nuget packages.

**Những cải tiến nền tảng của ASP.NET Core:**

* Hợp nhất việc xây dựng web UI và web APIs
* Tích hợp những client-side frameworks hiện đại và những luồng phát triển
* Hệ thống cấu hình dựa trên môi trường đám mây thật sự
* Dependency injection được xây dựng sẵn
* HTTP request được tối ưu nhẹ hơn
* Có thể host trên IIS hoặc self-host trong process của riêng bạn
* Được xây dựng trên .NET Core, hỗ trợ thực sự app versioning
* Chuyển các thực thể, thành phần, module như những NuGet packages
* Những công cụ mới để đơn giản hóa quá trình phát triển web hiện đại
* Xây dựng và chạy đa nền tảng (Windows, Mac và Linux)
* Mã nguồn mở và tập trung vào cộng đồng

ASP.NET đã trở thành nền tảng công nghệ phát triển web phổ biến trong việc xây dựng ứng dụng web mạnh mẽ, tăng độ bảo mật và hiệu suất hoạt động trên Windows. ASP.NET đã được sử dụng trong nhiều dự án phát triển web quan trọng. Từ đó hình thành cộng đồng phát triển khổng lồ trong lĩnh vực công nghệ.

## PHẦN 1.3. Các thành phần chính của ASP.NET

**Cấu trúc ASP.NET:**

* **Page (.aspx):** Đây là đơn vị cơ bản của một trang web ASP.NET. Mỗi trang có thể chứa mã HTML, CSS và mã lập trình để thể hiện nội dung và logic của trang.
* **Code-behind (.aspx.cs hoặc .aspx.vb):** Đây là file code phía sau của mỗi trang ASP.NET. Tiện ích chứa mã lập trình phục vụ cho trang tương ứng và được sử dụng để xử lý sự kiện và thao tác với dữ liệu.
* **Web.config:** Đây là file cấu hình chính cho ứng dụng ASP.NET. Công nghệ chứa thông tin về các cài đặt như chuỗi kết nối CSDL, cấu hình bảo mật và các thiết lập khác cho ứng dụng.

**Các thành phần của ASP.NET:**

* **Controls:** ASP.NET cung cấp một loạt các điều khiển web như textbox, Button, gridview, và nhiều hơn nữa. Các điều kiện này giúp xây dựng giao diện người dùng tương tác và thể hiện các dữ liệu.
* **Code-behind:** Mã lập trình phía sau mỗi trang được sử dụng để xử lý sự kiện, truy xuất và thực hiện các tác vụ logic phức tạp.
* **Data Access:** ASP.NET hỗ trợ nhiều phương pháp truy cập dữ liệu như ADO.NET, Entity Framework, LINQ để truy vấn và cập nhật cơ sở dữ liệu.
* **State Management:** ASP.NET cung cấp các cơ chế quản lý trạng thái như Viewstate, Session và Cookie để lưu trữ thông tin tạm thời hoặc duy trì trạng thái giữa các yêu cầu.
* **Authentication và Authorization:** ASP.NET cung cấp các cơ chế xác thực và phân quyền, cho phép ứng dụng kiểm tra danh tính người dùng và kiểm soát quyền truy cập vào các tài nguyên.
* **Caching:** ASP.NET hỗ trợ caching để tối ưu hóa hiệu suất ứng dụng bằng cách lưu trữ tạm thời dữ liệu hoặc mã được sử dụng thường xuyên để tránh thực hiện lại các tác vụ tốn kém.

## PHẦN 1.4. So sánh ASP.NET với PHP

ASP.NET và PHP là hai ngôn ngữ lập trình phổ biến được sử dụng trong việc phát triển các ứng dụng web.

**Điểm giống nhau:**

* **Hoạt động đa nền tảng:** Cả ASP.NET và PHP đều có khả năng chạy trên đa nền tảng, bao gồm Windows, Linux và MacOS.
* **Hỗ trợ cộng đồng lớn:** Cả hai đều có cộng đồng phát triển lớn với nhiều tài liệu, thư viện và tài nguyên hỗ trợ đầy đủ.
* **Tích hợp với cơ sở dữ liệu:** ASP.NET và PHP đều mang đến nhiều cơ sở dữ liệu như Mysql, SQL Server và Oracle để lưu trữ và truy xuất dữ liệu.
* **Cung cấp các framework và thư viện mạnh mẽ:** Các tiện ích đều cung cấp các framework và thư viện giúp xây dựng ứng dụng web nhanh chóng và dễ dàng.

**Điểm khác nhau:**

* **Ngôn ngữ lập trình:** ASP.NET sử dụng C#, VB.NET và F# làm ngôn ngữ chính. Trong khi PHP được thiết kế đặc biệt cho lập trình web.
* **Độ phổ biến:** PHP được coi là ngôn ngữ lập trình web phổ biến nhất trên thế giới, ASP.NET thường được sử dụng trong các dự án và hệ thống phát triển bởi Microsoft.
* **Cấu trúc và phương pháp phát triển:** ASP.NET thường sử dụng cấu trúc MVC (Model-View-Controller) hoặc mô hình Web Forms. PHP có tích hợp linh hoạt và khả năng phát triển các ứng dụng web theo nhiều cách khác nhau.
* **Hiệu suất:** Trong một số trường hợp, ASP.NET có thể có hiệu suất tốt hơn khi xử lý các ứng dụng web lớn và tải trọng cao. PHP thường được sử dụng cho các ứng dụng web nhỏ hơn và yêu cầu tài nguyên ít hơn.

Nhìn chung, ASP.NET và PHP đều là các công cụ mạnh mẽ để phát triển ứng dụng web. Sự lựa chọn giữa 2 ngôn ngữ này phụ thuộc vào yêu cầu dự án và sở thích cá nhân của người phát triển.

## PHẦN 1.5. Ứng dụng thực tế của ASP.NET

**Xây dựng các ứng dụng ứng viên việc làm:** Chương trình ASP.NET có tích hợp thuận tiện với các công nghệ tìm kiếm và công cụ phân tích dữ liệu. Phần mềm cho phép phát triển các trang web tuyển dụng, quản lý hồ sơ ứng viên và tìm kiếm việc làm.

**Phát triển ứng dụng e-commerce:** ASP.NET cung cấp các công cụ và thư viện tiện ích để phát triển các ứng dụng e-commerce. Người dùng có thể tạo ra các trang web mua sắm trực tuyến, quản lý giỏ hàng, thanh toán và tích hợp các hệ thống thanh toán trực tuyến.

**Phát triển ứng dụng di động:** ASP.NET hỗ trợ việc phát triển ứng dụng di động thông qua các framework như Xamarin và ASP.NET Core. Từ đó mà lập trình viên dễ dàng xây dựng các ứng dụng di động đa nền tảng sử dụng mã nguồn chung.

**Một số ứng dụng khác:**

* **Phát triển các trang web tương tác:** ASP.NET cho phép lập trình viên tạo ra các trang web tương tác, cho phép người dùng tương tác với dữ liệu và chức năng trực tuyến.
* **Quản lý dữ liệu:** ASP.NET cung cấp các công cụ và thư viện để quản lý dữ liệu trong web ứng dụng. Người lập trình có thể thao tác với cơ sở dữ liệu.
* **Phát triển ứng dụng quản lý dự án:** ASP.NET cung cấp khả năng xây dựng các ứng dụng cho phép quản lý các công việc, lịch trình, nguồn lực và tiến độ dự án.

## PHẦN 1.6. Cấu trúc dự án asp.net core

Trong một dự án ASP.NET Core MVC, các thành phần như: *Startup.cs, Program.cs, appsettings.json, Controllers, Views,* và *Models.*

### 1.6.1. Program.cs

Đây là điểm khởi đầu của ứng dụng ASP.NET Core. File Program.cs chứa phương thức *Main()* và sử dụng *WebHost.CreateDefaultBuilder()* để cấu hình máy chủ web cho ứng dụng.

**Các công việc chính:**

* Cấu hình máy chủ Kestrel (máy chủ web tích hợp của ASP.NET Core).
* Cấu hình logger để ghi nhận thông tin log.
* Chạy ứng dụng bằng cách gọi *.Build()* và *.Run()*

### 1.6.2. Startup.cs

Startup.cs là nơi cấu hình các dịch vụ và pipeline xử lý yêu cầu *(Request Pipeline)* của ứng dụng.

**Các phương thức chính:**

* *ConfigureServices(IServiceCollection services):* Nơi đăng ký các dịch vụ *(service)* mà ứng dụng cần, như Entity Framework, Identity, hoặc dịch vụ tùy chỉnh khác.
* *Configure(IApplicationBuilder app, IWebHostEnvironment env):* Cấu hình pipeline xử lý yêu cầu. Các middleware như UseRouting, UseEndpoints, và UseAuthorization được thêm ở đây để xử lý các yêu cầu.

### 1.6.3. appsettings.json

Đây là file cấu hình chính cho ứng dụng ASP.NET Core, nơi lưu các thiết lập như chuỗi kết nối cơ sở dữ liệu, cài đặt logging, và các cấu hình tùy chỉnh.

File này có thể chứa các cấu hình cho từng môi trường *(như appsettings.Development.json cho môi trường phát triển).*

Các giá trị từ *appsettings.json* được tải vào cấu hình bằng cách sử dụng *IConfiguration* trong ứng dụng và có thể truy cập dễ dàng từ các phần khác của ứng dụng.

### 1.6.4. Controllers

Thư mục *Controllers* chứa các lớp *controller*, là nơi xử lý các yêu cầu HTTP.

**Cách thức hoạt động:**

* Các *controller* sẽ nhận yêu cầu từ người dùng, tương tác với dữ liệu (thông qua *Models*), và trả về phản hồi (dữ liệu hoặc *view*).
* Mỗi *controller* có thể có nhiều action method tương ứng với các hành động khác nhau như GET, POST, PUT, và DELETE.

### 1.6.5. Views

*Views* là nơi lưu trữ các giao diện người dùng (UI) của ứng dụng.

**Cách thức hoạt động:**

* Các *view* là các tệp HTML động có thể kết hợp với dữ liệu từ *controller*, thường được viết bằng Razor *(.cshtml).*
* *Views* chứa các thư mục con theo tên của *controller* để tổ chức *view* cho từng *controller*, ngoài ra còn có các thư mục như *Shared* cho các *view* dùng chung.

### 1.6.6. Models

*Models* chứa các lớp định nghĩa dữ liệu và logic nghiệp vụ.

**Cách thức hoạt động:**

* *Models* đại diện cho dữ liệu của ứng dụng và các quy tắc nghiệp vụ liên quan.
* Các lớp *Model* thường bao gồm các thuộc tính đại diện cho dữ liệu và các phương thức để quản lý các quy tắc nghiệp vụ hoặc tương tác với cơ sở dữ liệu.

## PHẦN 1.7. Xây dựng ứng dụng đầu tiên

### 1.7.1. Cài đặt môi trường phát triển

#### 1.7.1.1. Cài đặt Visual Studio

**Bước 1:** Vào trang <https://visualstudio.microsoft.com/downloads/>

Ở đây chúng ta chọn bản Community. Nhấn vào Free download để tải về. Bỏ qua bước này nếu bạn đã có Visual Studio cài đặt trong máy.

**A screenshot of a computer

Description automatically generated**

Ảnh 1: Tải file cài đặt Visual Studio Community

**Bước 2:** Mở file vừa mới tải về, sau đó chọn Run để bắt đầu cài đặt Visual Studio.

**Bước 3:** Chọn Continue và bạn chờ vài phút để tải file cài đặt về.

**Bước 4:** Chọn workloads cần thiết:

* ASP.NET và web development
* .NET Core cross-platform development

**Bước 5:** Chọn vị trí lưu file của Visual Studio. Sau đó bạn Install để bắt đầu quá trình tải và cài đặt.

**A screenshot of a computer

Description automatically generated**

Ảnh 2: Thay đổi vị trí lưu Visual Studio

**Bước 6:** Chờ Visual Studio tải và cài đặt các gói tài nguyên, quá trình này nhanh hay chậm tùy vào tốc độ tải của máy. Chọn Start after installation để khởi động Visual Studio sau khi tải xong.

**Bước 7:** Sau khi cài đặt xong thì mở phần mềm, chọn Create a new project để bắt đầu.

#### 1.6.1.2. Cài đặt .NET SDK

**Bước 1: Tải xuống .NET SDK**

* **Truy cập trang tải xuống:** Mở trình duyệt và truy cập trang chính thức của .NET tại <https://dotnet.microsoft.com/download>
* **Chọn phiên bản:** Chọn phiên bản .NET SDK mới nhất hoặc phù hợp với nhu cầu của bạn (chẳng hạn .NET 6 hoặc .NET 7).
* **Tải về:** Chọn hệ điều hành của bạn (Windows, macOS, hoặc Linux) và tải gói cài đặt tương ứng.

**Bước 2: Cài đặt .NET SDK trên hệ điều hành**

* **Trên Windows:**
  + Mở tệp .exe bạn vừa tải xuống và làm theo hướng dẫn trên màn hình để hoàn tất quá trình cài đặt.
  + Sau khi cài đặt, khởi động lại PC nếu cần thiết.
* **Trên macOS:**
  + Mở tệp .pkg đã tải về và làm theo các bước hướng dẫn để cài đặt.
  + Sau khi cài đặt, khởi động lại Mac nếu cần thiết.
* **Trên Linux:**
  + Các bước cài đặt trên Linux phụ thuộc vào bản phân phối bạn dùng (Ubuntu, Fedora, v.v.). Trang tải xuống của .NET sẽ có hướng dẫn cụ thể cho từng hệ điều hành.

**Bước 3: Kiểm tra cài đặt .NET SDK**

* Mở Command Prompt (trên Windows) hoặc Terminal (trên macOS/Linux).
* Nhập lệnh sau để kiểm tra phiên bản .NET SDK đã cài đặt: dotnet --version
* Nếu lệnh trên hiển thị phiên bản .NET, bạn đã cài đặt thành công .NET SDK.

#### 1.6.1.3. Cài đặt Dotnet CLI (tùy chọn)

Dot Net CLI được cài đặt như một phần của .NET Core SDK. Tải về Dot Net CLI tại <https://dotnet.microsoft.com/en-us/download>

**Các lệnh thường dùng:**

*new* – Tạo mới project, file cấu hình hay solution

*restore* – Tải về các thành phần phụ thuộc trong project

*build* – Biên dịch dự án với các thành phần liên quan

*publish* – Đóng gói toàn bộ ứng dụng ra một thư mục

*run* – Chạy source code mà không cần lệnh biên dịch

*test* – Thực thi unit test

*vstest* – Thực thi unit test từ file cụ thể

*pack* – Đóng gói code vào NuGet package

*clean* – Làm sạch output của dự án

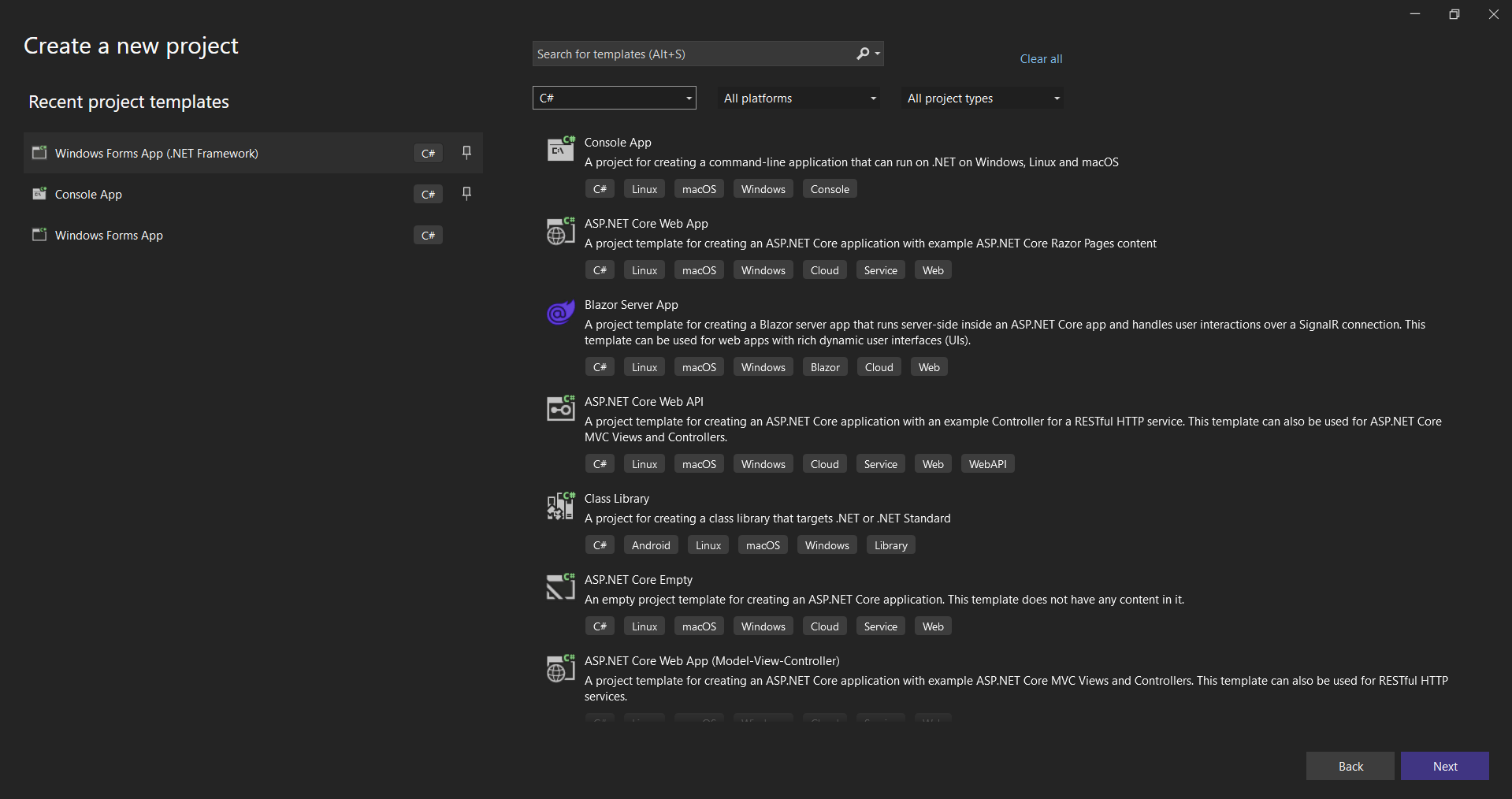
*sln* – Chỉnh sửa file .NET Core solution

*help* – Xem trợ giúp

*store* – Lưu trữ các assembly cụ thể

### 1.7.2. Tạo dự án ASP.NET Core Web Application (Phụ lục 1)

**Bước 1:** Mở Visual Studio > Chọn ASP.NET Core Web App

****

Ảnh 3: Chọn template project

**Bước 2:** Đặt tên Project và vị trí lưu sau đó bấm Next > Create

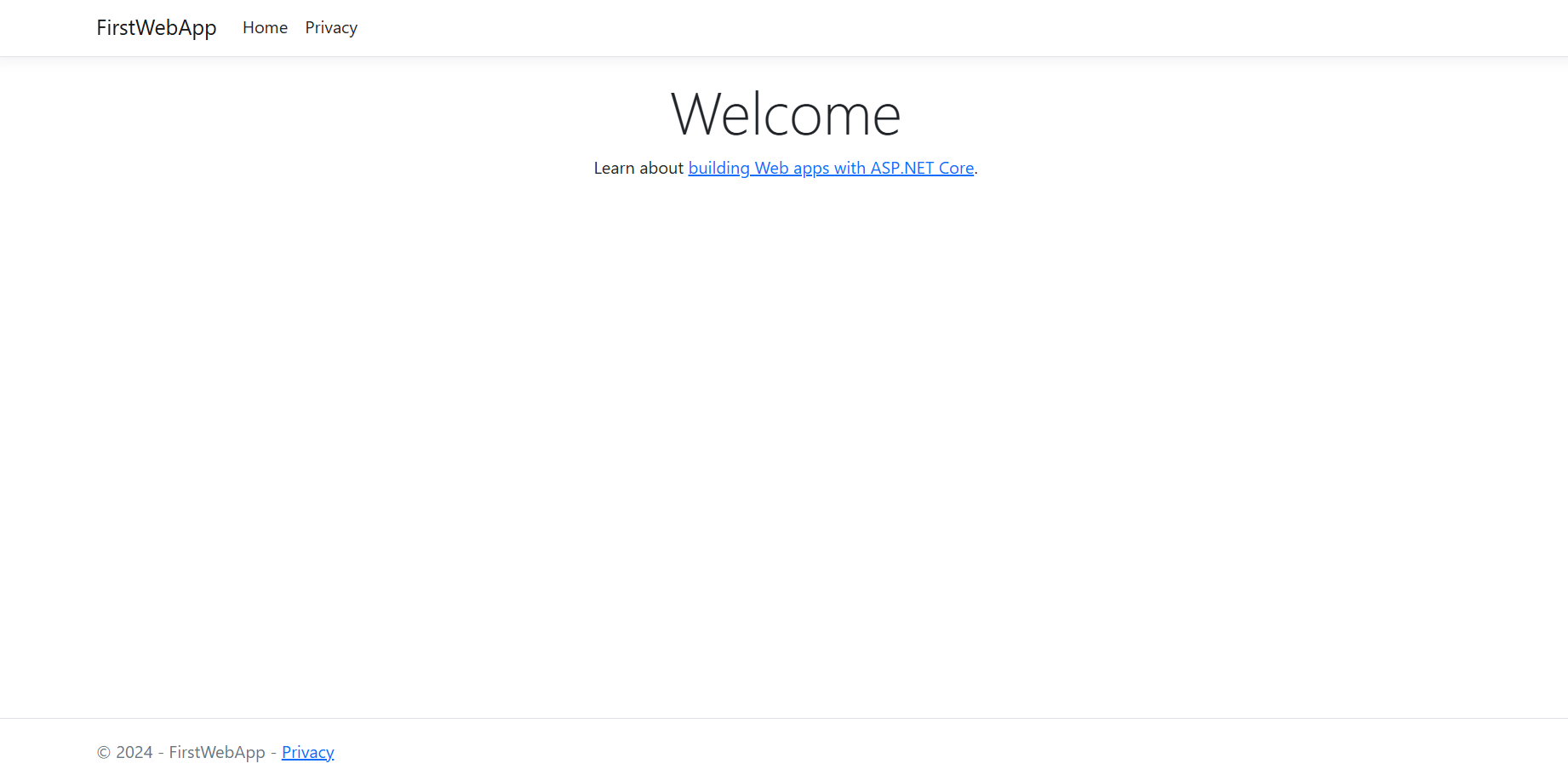
**A screenshot of a computer

Description automatically generated**

Ảnh 4: Tạo project mới

**Bước 3:** Bật tùy chọn *“Allow invalid certificates for resources loaded from localhost.”* trên trình duyệt của bạn.

**Bước 4:** Bấm nút chạy (F5 hoặc Ctrl+F5) chương trình và xem thành quả.



Ảnh 5: Web mẫu được tạo sẵn sau khi tạo project

Việc chạy ứng dụng với Ctrl-F5 *(non-debug)* cho phép bạn làm code thay đổi, lưu file, refresh trình duyệt và thấy ngay những thay đổi của code. Nhiều lập trình viên thích sử dụng chế độ non-debug để nhanh chóng chạy ứng dụng và xem những thay đổi. Điều này là tương tự với ngôn ngữ thông dịch như php, python hay ruby,..

# Chương III. Kestrel – ASP.NET Core Web Server

## PHẦN 3.1. Kestrel là gì?

Kestrel là một HTTP web server mã nguồn mở *(open source),* đa nền tảng *(cross-platform),* hướng sự kiện *(event-driven)* và bất đồng bộ *(asynchronous I/O).* Nó được phát triển để chạy ứng dụng ASP.NET Core trên bất cứ nền tảng nào. Nó được thêm vào mặc định trong ứng dụng ASP.NET Core. Có nghĩa là khi bạn build ứng dụng asp.net, chạy nó thì sẽ tự động chạy web server Kestrel.

Github repo: <https://github.com/aspnet/KestrelHttpServer>

## PHẦN 3.2. Tại sao sử dụng Kestrel

Ứng dụng ASP.NET cũ thường dính chặt vào IIS (Internet Information Service). IIS là một web server với tất cả các tính năng đầy đủ cần có. Nó được phát triển từ khá lâu và rất trưởng thành, nhưng nó cồng kềnh và nặng. Nó trở thành một trong những Web server tốt nhất ở thời điểm hiện tại nhưng nó cũng là một trong những thứ chậm nhất.

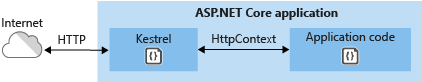
ASP.NET dính chặt vào IIS cũng là gánh nặng cho IIS.

Thiết kế mới của ứng dụng ASP.NET Core giờ đây hoàn toàn tách rời khỏi IIS. Điều này tạo cho ASP.NET Core có thể chạy trên bất cứ nền tảng nào. Nhưng nó vẫn có thể lắng nghe các HTTP Request và gửi lại response về cho client. Đó là Kestrel.

## PHẦN 3.3. Sử dụng Kestrel

Kestrel chạy in-process trong ứng dụng ASP.NET Core. Vì thế nó chạy độc lập với môi trường. Kestrel web server nằm trong thư viện Microsoft.AspNetCore.Server.Kestrel.

**Có 2 cách để sử dụng Kestrel:**

* **Tự host (Self Hosting):** Mô hình self hosting của ASP.NET Core là lắng nghe trực tiếp các HTTP Request từ internet như hình dưới đây:  
  
* **Đằng sau Web Server khác:** Kestrel không phải là một web server đầy đủ tính năng. Nhưng nó nhanh. Nó không đủ mạnh để chạy Kestrel một mình trên môi trường Production. Nó được đề xuất chạy đằng sau một Web Server Fully Feature như IIS, Nginx hay Apache…Trong kịch bản này các web server khác như là một reverse proxy server (máy chủ ủy quyền). Các Reverse proxy server sẽ lấy HTTP Request từ internet và gửi sang Kestrel. IIS có thể lấy HTTP Request và thực hiện một số các xử lý như ghi log, lọc request, rewrite-url trước khi gửi sang cho Kestrel. A diagram of a computer

  Description automatically generated

Có nhiều lý do tại sao chúng ta sử dụng mô hình này cho môi trường Production.

1. Bảo mật: Nó có thể hạn chế việc phơi ra ngoài những gì không cần thiết và nó cũng cho chúng ta các tùy chọn về cấu hình và bảo vệ ứng dụng.
2. Dễ dàng cài đặt cân bằng tải
3. Cài đặt SSL: Chỉ các máy chủ ủy quyền cần các chứng chỉ SSL, còn các server này có thể làm việc với ứng dụng của bạn trên mạng nội bộ sử dùng HTTP thuần.
4. Chia sẻ một IP với nhiều địa chỉ
5. Lọc các Request, ghi log và rewrite-URL…
6. Nó có thể restart ứng dụng nếu bị crash.

Phương thức CreateDefaultBuilder gọi UseIISIntegration, sẽ nói co ASP.NET rằng ứng dụng sử dụng IIS như là một reverse proxy server đằng trước Kestrel.

## PHẦN 3.4. Triển khai ứng dụng ASP.NET trên Server Linux với Kestrel Apache Nginx

Trong phần này, ta sẽ đi tìm hiểu một số cấu hình cơ bản để có thể áp dụng làm cơ sở chuẩn bị cho việc publish ứng dụng, triển khai ứng dụng trên môi trường product.

### 3.4.1. Cấu hình Kestrel trong ASP.NET core

Khi khởi tạo ứng dụng ASP.NET từ các template, bao giờ cũng có file Program.cs trong đó định nghĩa hàm *Main* - gọi *CreateHostBuilder* để chạy ứng dụng, code mặc định này đã sử dụng và chạy Kestrel.

|  |
| --- |
| **public class Program**{  **public static void Main**(**string**[] args){  BuildWebHost(args).Run();  }  **public static** IWebHost **BuildWebHost**(**string**[] args) =>  WebHost.CreateDefaultBuilder(args)  .UseStartup<Startup>()  .Build();  } |

Phương thức *Main* gọi đến *CreateDefaultBuilder*, có trách nhiệm tạo một host cho ứng dụng. *(Host là nơi chứa ứng dụng để chạy)*. *CreateDefaultBuilder* đăng ký Kestrel như là server sẽ sử dụng trong ứng dụng.

Phụ thuộc vào môi trường chạy ứng dụng mà Kestrel chạy và lắng nghe ở các cổng khác nhau theo cấu hình của môi trường. Trong môi trường phát triển, thường nó sẽ lắng nghe ở cổng 5000 *(http)* và cổng 5001 *(https)*. Cổng này có thể thiết lập qua biến môi trường hệ thống ASPNETCORE\_URLS (ví dụ trong khi đóng gói vào docker có thiết lập biến môi trường ASPNETCORE\_URLS=https://+:443;http://+:80 thì Kestrel chạy và lắng nghe cổng 443 và 80)

Để tùy biến Kestrel với các thiết lập cụ thể sử dụng đối tượng *KestrelServerOptions* như sau:

|  |
| --- |
| **public static** IHostBuilder **CreateHostBuilder**(**string**[] args) =>  **Host.CreateDefaultBuilder**(args)  **.ConfigureWebHostDefaults**(webBuilder =>  {  **webBuilder.UseKestrel**(kestrelServerOptions => {  // Các thiết lập cho Kestrel tại đây  // sử dụng KestrelServerOptons để thiết lập  })  **webBuilder.UseStartup**<**Startup**>();  }); |

**Các thiết lập Kestrel:**

**Bind IP - thiết lập Kestrel lắng nghe trên cổng và IP:** Có thể sử dụng các phương thức *KestrelServerOptions.Listen*, *KestrelServerOptions.ListenAnyIP*, *KestrelServerOptions.ListenLocalhost* ví dụ:

|  |
| --- |
| **webBuilder.UseKestrel**(kestrelServerOptions => {  // Thiết lập lắng nghe trên cổng 8090 với IP bất kỳ  kestrelServerOptions.Listen(IPAddress.Any, 8090);  // Lắng nghe trên cổng 8091 trên server chạy ứng dụng  kestrelServerOptions.ListenLocalhost(8091);  kestrelServerOptions.Listen(IPAddress.Loopback, 8092, listenOptions => {  // Thiết lập sử dụng SSL - file xác thực SSL testCert.pfx  listenOptions.UseHttps("testCert.pfx", "testPassword");  });  }); |

**Thiết lập một số giới hạn của Kestrel với KestrelServerLimits:** Nếu muốn thay đổi các giới hạn mặc định thì có thể dùng, *KestrelServerLimits* với các thuộc tính đặt giới hạn, ví dụ thiết lập *keep-alive timeout*

|  |
| --- |
| **webBuilder.UseKestrel**(kestrelServerOptions => {  /...  kestrelServerOptions.Limits.KeepAliveTimeout = TimeSpan.FromMinutes(2);  }); |

**Tham khảo thêm các thiết lập khác tại:** kestrelserverlimits [https://learn.microsoft.com/en-us/dotnet/api/microsoft.aspnetcore.server.kestrel.core.kestrelserverlimits]

* **Thiết lập Bind IP cho Server:**

Có thể nhanh chóng dùng *webBuilder.UseUrls* để thiết lập IP và cổng của Kestrel, dùng địa chỉ Url để thiết lập Kestrel lắng nghe trên nó. Trong đó có chỉ ra giao thức (http, https), địa chỉ IP, và cổng. Ví dụ:

|  |
| --- |
| **public static** IHostBuilder **CreateHostBuilder**(**string**[] args) =>  **Host.CreateDefaultBuilder**(args)  .**ConfigureWebHostDefaults**(webBuilder =>  {  **webBuilder.UseUrls**("http://0.0.0.0:8090", "https://0.0.0.0:8091");  **webBuilder.UseStartup**<**Startup**>();  }); |

Sử dụng Kestrel phía sau các Http Server khác (Http Apache, Nginx, IIS) thì nên sử dụng cách này (*UseUrls*), và chỉ cần lắng nghe ở giao thức http, còn https được cấu hình ở server phía trước (*nginx, apache*).

|  |
| --- |
| **public static** IHostBuilder **CreateHostBuilder**(string[] args) =>  **Host.CreateDefaultBuilder**(args)  .**ConfigureWebHostDefaults**(webBuilder =>  {  // Chỉ nhận http (không https)  **webBuilder.UseUrls**("http://0.0.0.0:5000");  **webBuilder.UseStartup**<**Startup**>();  }); |

Ngoài ra cũng chú ý, để sử dụng chính xác Kestrel với nginx, apache cần thêm vào đầu *Startup.configure*

|  |
| --- |
| ...  // using Microsoft.AspNetCore.HttpOverrides;  **app.UseForwardedHeaders**(new **ForwardedHeadersOptions**  {  ForwardedHeaders = **ForwardedHeaders.XForwardedFor** **| ForwardedHeaders.XForwardedProto**  });  ... |

### 3.4.2. Publish ứng dụng ASP.NET core

Để biên dịch mã nguồn chạy ở môi trường Product, có thể thực hiện các lệnh như sau:

|  |
| --- |
| # phục hồi các dependency từ Nuget  dotnet restore  # build  dotnet build -c Release -o app/build  # publish  dotnet publish -c Release -o app/publish |

Kết quả là xuất ra ứng dụng ra thư mục app/publish, bạn dùng thư mục này để phân phối - triển khai chạy ứng dụng. Trong thư mục có file dll tên ứng dụng dùng để chạy ứng dụng.

|  |
| --- |
| dotnet tên-ứng-dụng.dll |

Chú ý, server chạy ứng dụng phải cài .NET Core SDK và .NET Core Runtime.

**Kèm thư mục khi publish**

Trong dự án có thể có các thư mục tài nguyên, ví dụ thư mục Uploads, nếu muốn thư mục này được copy vào publish thì trong file: .csproj thêm vào đoạn mã:

|  |
| --- |
| <**ItemGroup**>  <**Content** Include="Uploads\\*\*">  <**CopyToOutputDirectory**>PreserveNewest</**CopyToOutputDirectory**>  </Content>  </**ItemGroup**> |

### 3.4.3. Giám sát ứng dụng ASP.NET với systemd

Giả sử ứng dụng lưu trên Server Linux ở đường dẫn */home/userasp/mvcblog* (đã copy nội dung thư mục *app/publish* vào *mvcblog*), file chạy ứng dụng là *mvcblog.dll*, nó được publish và lắng nghe ở cổng http 8090 (cổng này do bạn đặt trong ứng dụng ở phần trên)

Bạn hoàn toàn có thể vào thư mục, chạy ứng dụng với lệnh:

|  |
| --- |
| cd /home/userasp/mvcblog/  dotnet mvcblog.dll |

Khi bạn chạy trực tiếp như vậy, nếu ứng dụng bị crash vì một sự cố nào đó - thì nó sẽ dừng hẳn và không được khởi động lại. Giải quyết trường hợp này hãy sử dụng systemd của Linux, giúp giám sát tình trạng ứng dụng, khởi động ứng dụng nếu chưa chạy hoặc bị crash.

Tạo ra file dịch vụ trong thư mục */etc/systemd/system/,* ví dụ ứng dụng là *mvcblog*, tạo ra file */etc/systemd/system/mvcblog.service* (dùng lệnh vi), sau đó biên tập nội dung file này như sau:

|  |
| --- |
| [Unit]  Description=Ung dung ASP.NET MVC BLOG  [Service]  WorkingDirectory=/home/userasp/mvcblog  ExecStart=/usr/bin/dotnet /home/userasp/mvcblog/mvcblog.dll  Restart=always  # Khởi động lại ứng dụng sau 10 bị crash  RestartSec=10  KillSignal=SIGINT  SyslogIdentifier=asp-net-app  User=userasp  Environment=ASPNETCORE\_ENVIRONMENT=Production  [Install]  WantedBy=multi-user.target |

*/usr/bin/dotnet* là đường dẫn đầy đủ đến file binary của lệnh dotnet, có thể kiểm tra bằng lệnh:

|  |
| --- |
| which dotnet |

Thiết lập dịch vụ tự động chạy:

|  |
| --- |
| systemctl enable mvcblog |

Khởi chạy dịch vụ (ứng dụng):

|  |
| --- |
| systemctl start mvcblog |

Xem trạng thái

|  |
| --- |
| systemctl status mvcblog |

Lưu ý, ứng dụng ASP.NET chạy trên lắng nghe ở cổng 8090 hoặc cổng do bạn thiết lập, cổng này không public ra ngoài (không dùng firewall để mở). Cổng chỉ dùng cho các dịch vụ như apache, nginx chuyển hướng đến.

# Chương IV. Middleware và Request Pipeline trong ASP.NET Core

## 4.1. Middleware là gì?

Middleware là thành phần của phần mềm đóng vai trò tác động vào request pipeline *(luồng request)* để xử lý chúng và tạo ra response phản hồi lại client. Mỗi một tiến trình middleware thao tác với các request nhận được từ middleware trước nó. Nó cũng có thể quyết định gọi middleware tiếp theo trong pipeline hoặc trả về response cho middleware ngay trước nó *(ngắt pipeline).*

Tóm lại, Middleware là một module code nó nhận yêu cầu gửi đến Request và trả về Response. Cụ thể trong ASP.NET Core, middlewarre có thể:

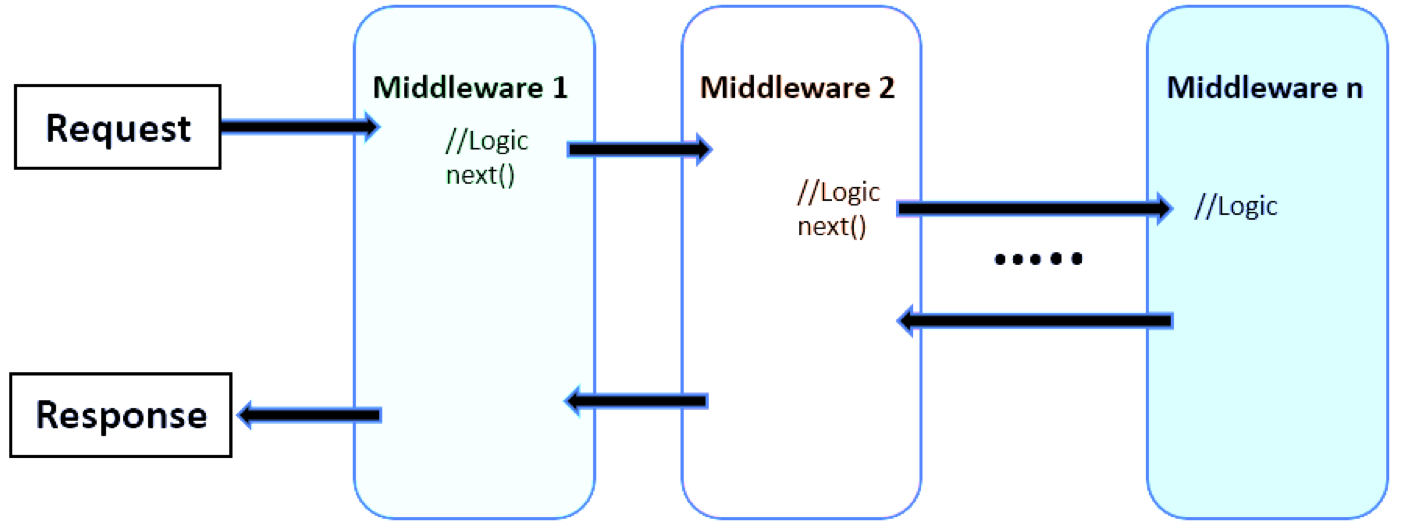
* Nhận một HTTP Request gửi đến và phát sinh ra HTTP Response để trả về
* Nhận một HTTP Request gửi đến, thi hành một số tác vụ (có thể là sửa đổi HTTP Request), sau đó chuyển đến một middleware khác.
* Nhận HTTP Response, sửa nó và chuyển đến một Middleware khác

## 4.2. Request Pipeline

Request Pipeline là cơ chế bắt đầu khi các request bắt đầu được xử lý với một đối tượng Request đầu vào và kết thúc với đầu ra là một response. Pipeline chỉ ra cách mà ứng dụng phản hồi với HTTP Request. Request đến từ trình duyệt đi qua pipeline và quay trở lại khi xử lý xong để trả về client. Các thành phần đơn lẻ tạo nên pipeline này được gọi là middleware.

Trong ứng dụng ASP.NET Core, các middlware kết nối lại với nhau thành một xích, middleware đầu tiên nhận HTTP Request, xử lý nó và có thể chuyển cho middleware tiếp theo hoặc trả về ngay HTTP Response. Chuỗi các middleware theo thứ tự như vậy gọi là pipeline.

Các middleware như là các dịch vụ nhỏ, đăng ký vào ứng dụng bằng cách sử dụng đối tượng IApplicationBuilder, sau đó ứng dụng căn sẽ xây dựng lên các pipeline (luồng xử lý) cho các truy vấn gửi đến.



Ảnh 6: Sơ đồ các middleware trong pipeline

Để thực hành, tạo ra ứng dụng web đơn giản với cách lệnh sau:

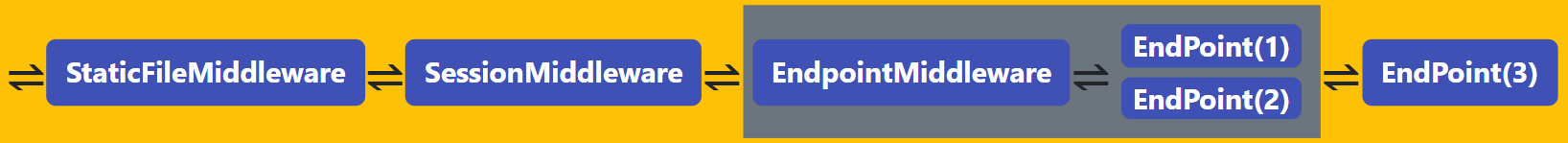
|  |
| --- |
| mkdir 02.middleware # tạo thư mục lưu code  cd 02.middleware # vào thư mục  dotnet new web # tạo dự án với template mẫu, đơn giản  code . # mở ra bằng Visual Studio Code, nhớ chọn .NET: Generate Assets for Build and Debug lần đầu mở ra |

Trong ứng dụng ASP.NET CORE, có sẵn một một loạt Middleware hoặc có thể xây dựng thêm để đưa vào một pipeline, luồng xử lý HttpRequest và trả về HttpResponse. Việc thêm Middleware vào pipeline được thực hiện ở phương thức Configure của lớp Startup. Các middleware trong pipeline xử lý các HTTP Message đều có dạng - nhận đầu vào là tham số kiểu HttpContext (có chứa HttpRequest và HttpRespone), thi hành code, sau đó chuyển đến cho middleware tiếp theo.

Ví dụ sửa mã *Startup.cs* có sử dụng một số Middleware như sau:

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Threading.Tasks;  using Microsoft.AspNetCore.Builder;  using Microsoft.AspNetCore.Hosting;  using Microsoft.AspNetCore.Http;  using Microsoft.Extensions.DependencyInjection;  using Microsoft.Extensions.Hosting;  **namespace \_02.middleware**{  **public class Startup** {  // Đăng ký các dịch vụ sử dụng bởi ứng dụng, services là một DI container  **public void ConfigureServices**(IServiceCollection services) {  // Thêm dịch vụ dùng bộ nhớ lưu cache (session sử dụng dịch vụ này)  **services.AddDistributedMemoryCache**();  // Thêm dịch vụ Session, dịch vụ này cung cấp Middleware  **services.AddSession**();  }  **public void Configure**(**IApplicationBuilder** app, **IWebHostEnvironment** env){  // Thêm StaticFileMiddleware - nếu Request là yêu cầu truy cập file tĩnh,  // Nó trả ngay về Response nội dung file và là điểm cuối pipeline, nếu khác  // nó gọi Middleware phía sau trong Pipeline  **app.UseStaticFiles**();  // Thêm SessionMiddleware: khôi phục, thiết lập - tạo ra session  // gán context.Session, sau đó chuyển gọi ngay middleware  // tiếp trong pipeline  **app.UseSession**();  // Thêm EndpointRoutingMiddleware: ánh xạ Request gọi đến Endpoint (Middleware cuối)  // phù hợp định nghĩa bởi EndpointMiddleware  **app.UseRouting**();  // app.UseEndpoint dùng để xây dựng các endpoint - điểm cuối của pipeline theo Url truy cập  **app.UseEndpoints**(endpoints =>  {  // EndPoint(2) khi truy vấn đến /Testpost với phương thức post hoặc put  **endpoints.MapMethods**("/Testpost" , **new** **string**[] {"post", "put"}, **async** **context** => {  **await context.Response.WriteAsync**("post/pust");  });  // EndPoint(2) - Middleware khi truy cập /Home với phương thức GET - nó làm Middleware cuối Pipeline  **endpoints.MapGet**("/Home", **async** context => {  **int**? count = **context.Session.GetInt32**("count");  count = (count != null) ? count + 1 : 1;  **context.Session.SetInt32**("count", count.Value);  **await context.Response.WriteAsync**($"Home page! {count}");  });  });  // EndPoint(3) app.Run tham số là hàm delegate tham số là HttpContex  // - nó tạo điểm cuối của pipeline.  **app.Run**(**async** context => {  context.Response.StatusCode = **StatusCodes.Status404NotFound**;  **await context.Response.WriteAsync**("Page not found");  });  }  }  } |

Với cấu hình trên khi chạy ứng có thể khởi tạo lên các pipleline để phân luồng đi của yêu cầu gửi đến (HttpContext), gồm có:



Với cấu hình trên, tùy thuộc vào truy vấn mà HttpContext có thể đi qua các Middleware: StaticFileMiddleware, SessionMiddleware, EndpointRoutingMiddleware, EndPoint(1), EndPoint (2), EndPoint (3), ví dụ:

* Nếu truy cập đến một file tĩnh (file chứa trong thư mục wwwroot), thì điểm cuối của pipeline là *StaticFileMiddleware* (các middleware khác không được sử dụng).
* Nếu truy cập địa chỉ /Home (http://localhost:5000/Home) thì HttpContext đi qua các Middleware: *StaticFileMiddleware, SessionMiddleware, EndpointMiddleware, EndPoint(2)*
* Nếu truy cập địa chỉ / thì HttpContext đi qua các Middleware: *StaticFileMiddleware, SessionMiddleware,EndpointMiddleware, EndPoint(3)*
* Nếu truy cập địa chỉ /Testpost với phương thức post, update thì HttpContext đi qua các Middleware: *StaticFileMiddleware, SessionMiddleware, EndpointMiddleware, EndPoint(1)*

## 4.3. Tạo Middleware riêng trong ASP.NET Core (Phụ lục 2)

Giờ ta sẽ thực hành tạo ra một Middleware như sau: tạo Middleware có tên là *CheckAcessMiddleware*, đăng ký Middleware này nằm sau *SessionMiddleware* trong pipeline của ứng dụng trên, *CheckAcessMiddleware* có chức năng - kiểm tra nếu truy cập là *URL /testxxx* thì không gọi Middleware tiếp theo mà thiết lập ngay HttpRespone hiện thị thông báo không được truy cập.

Còn nếu Url khác */testxxx* thì thêm vào HttpResponse một Header tên *throughCheckAcessMiddleware* với giá trị ngày tháng - để biết HttpRequest đã đi qua Middleware này sau đó gọi Middleware tiếp theo trong pipeline.

**Cấu trúc Middleware trong ASP.NET**

Một lớp (class) phù hợp là một Middleware trong ASP.NET nếu lớp đó có cấu trúc thỏa mãn những điều kiện sau:

* Có một phương thức khởi tạo public với tham số thứ nhất kiểu *RequestDelegate*, nếu có tham số thứ 2 thì các tham số tiếp theo này phải Inject được từ DI của hệ thống
* Phải có tổi thiểu một trong hai phương thức có tên *Invoke* hoặc *InvokeAsync* với tham số nhận là *HttpContext*, những phương thức này phải trả về *Task*. (Dùng *InvokeAsync* nếu muốn áp dụng kỹ thuật bất đồng bộ - nên làm)
* Trong *Invoke/InvokeAsync* bạn viết code xử lý tác vụ của Middleware, sau đó quyết định chuyển đến Middleware tiếp theo bằng cách gọi *RequestDeleage* đã truyền đến trong hàm tạo, hoặc không chuyến đến Middlware tiếp theo thì cần đảm bảo lúc này HttpResponse trrong HttpContext đã phù hợp để trả về cho Client.

**Bước 1: Tạo Middleware tên CheckAcessMiddleware**

Middleware/CheckAcessMiddleware.cs

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Threading.Tasks;  using Microsoft.AspNetCore.Hosting;  using Microsoft.AspNetCore.Http;  using Microsoft.Extensions.Configuration;  using Microsoft.Extensions.Hosting;  using Microsoft.Extensions.Logging;  **namespace \_02.middleware** {  **public class CheckAcessMiddleware** {  // Lưu middlewware tiếp theo trong Pipeline  **private readonly RequestDelegate \_next**;  **public CheckAcessMiddleware** (**RequestDelegate** next) => \_next = next;  **public async Task Invoke** (**HttpContext** httpContext) {  **if** (**httpContext.Request.Path** == "/testxxx") {  **Console.WriteLine** ("CheckAcessMiddleware: Cấm truy cập");  **await Task.Run** (  **async** () => {  **string** html = "<h1>CAM KHONG DUOC TRUY CAP</h1>";  httpContext.Response.StatusCode = **StatusCodes.Status403Forbidden**;  **await httpContext.Response.WriteAsync** (html);  }  );  } **else** {  // Thiết lập Header cho HttpResponse  **httpContext.Response.Headers.Add** ("throughCheckAcessMiddleware", new [] { DateTime.Now.ToString () });  **Console.WriteLine** ("CheckAcessMiddleware: Cho truy cập");  // Chuyển Middleware tiếp theo trong pipeline  **await \_next** (httpContext);  }  }  }  } |

**Bước 2: Đưa Middleware vào pipeline**

Như vậy đã có một Middleware theo đúng chuẩn để có thể đưa vào pipeline nhận HttpRequest của ASP.NET Core. Để đăng ký chỉ việc sử dụng phương thức *UseMiddleware* của *IapplicationBuilder*.

Ví dụ, tại phương thức *Configure* của *Startup*

|  |
| --- |
| **public void Configure**(**IApplicationBuilder** app, **IHostingEnvironment** env)  {  / ..  **app.UseStaticFiles**();  **app.UseSession**();  // Đưa Middleware vào pipeline - vị trí thứ 3  **app.UseMiddleware**<**CheckAcessMiddleware**>();  /..  } |

Tuy nhiên ta sẽ làm theo cách các Middleware mặc định được đăng ký, tức là thêm phương thức mở rộng vào *IApplicationBuilder*, ví dụ thêm vào nó phương thức *UseCheckAccess*.

Middleware/MyAppExtensions.cs

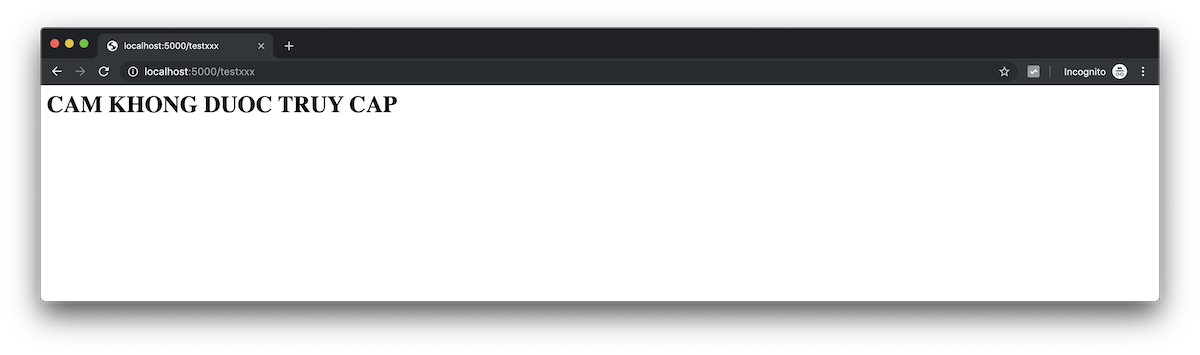
|  |
| --- |
| using Microsoft.AspNetCore.Builder;  **namespace \_02.middleware**{  **public static class MyAppExtensions**  {  // Mở rộng cho IApplicationBuilder phương thức UseCheckAccess  **public static IApplicationBuilder UseCheckAccess**(**this IApplicationBuilder** builder)  {  **return builder.UseMiddleware**<**CheckAcessMiddleware**>();  }  }  } |

Đã có phương thức mở rộng UseCheckAccess, để đăng ký Middleware vào pipeline thực hiện:

|  |
| --- |
| **public void Configure**(**IApplicationBuilder** app, **IHostingEnvironment** env)  {  / ..  **app.UseStaticFiles**(); // Đưa Middleware vào pipeline - vị trí đầu tiên  **app.UseSession**(); // Đưa Middleware vào pipeline - vị trí thứ 2  **app.UseCheckAccess**(); // Đưa Middleware vào pipeline - vị trí thứ 3  /..  } |

**Bước 3: Kiểm tra:**

Giờ truy cập vào địa chỉ /testxxx thì Request sau khi qua Middleware Session, đến Middleware *CheckAccess* thì bị chặn lại, các Middleware phía sau không được gọi.



Trong trường hợp, địa chỉ truy cập khác /testxxx, thì Request sau khi đi qua *CheckAccess* - thì Response có thêm một Header và chuyển các Request cho Middleware tiếp theo xử lý:

A screenshot of a computer

Description automatically generated

Header được thêm vào sau khi đi qua middleware *CheckAccess*

## 4.4. Xây dựng Middleware triển khai từ giao diện IMiddleware (Phụ lục 3)

Bạn có thể triển khai giao diện IMiddleware, trong đó chỉ việc xây dựng phương thức:

|  |
| --- |
| **Task InvokeAsync**(HttpContext context, RequestDelegate next) |

Ví dụ, xây dựng một Middleware - đặt tên *FrontMiddleware*, Middleware này không làm gì cả, nó chỉ in ra console thông báo cho biết Request đã đi qua nó - ta sẽ đặt Middleware này ở đầu pipeline, sau đó chuyển đến Middleware tiếp theo trong pipeline.

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Threading.Tasks;  using Microsoft.AspNetCore.Hosting;  using Microsoft.AspNetCore.Http;  using Microsoft.Extensions.Configuration;  using Microsoft.Extensions.Hosting;  using Microsoft.Extensions.Logging;  **namespace \_02.middleware** {  **public class FrontMiddleware : IMiddleware**  {  **public async Task InvokeAsync**(**HttpContext** context, **RequestDelegate** next)  {  **Console.Clear**();  **Console.WriteLine**("FrontMiddleware: " + context.Request.Path);  **await next**(context);  }  }  } |

Đối với loại Middleware này để sử dụng nó, phải đăng ký vào DI container trước (đăng ký dịch vụ) trong phươnng thức *ConfigureServices* của Startup

|  |
| --- |
| public void ConfigureServices(IServiceCollection services)  {  **services.AddTransient**<**FrontMiddleware**>();  //.. các đăg ký khác  } |

Sau đó thêm vào pipeline theo cách thông thường, khai báo đầu tiên trong Configure để nó đứng đầu pipeline:

|  |
| --- |
| public void Configure(IApplicationBuilder app, IWebHostEnvironment env)  {  **app.UseMiddleware**<**FrontMiddleware**>();  // ... các Middleware khác  } |

Chạy ứng dụng, để thấy HttpContext đi qua Middleware trên.

## 4.4. Truyền dữ liệu giữa các Middleware (PHỤ LỤC 4)

Để truyền dữ liệu giữa các Middleware hãy sử dụng *HttpContext.Items*, nó là dữ liệu kiểu *IDictionary*, bạn có thể lưu dữ liệu vào trong nó ở Middleware này và ở Middleware khác đọc được.

**Lưu dữ liệu:**

|  |
| --- |
| **context.Items.Add**("dulieu1", "Data Object ..."); |

**Đọc dữ liệu:**

|  |
| --- |
| ar data = **context.Items**["dulieu1"]; |

# Chương V. Làm việc với file cấu hình trong ASP.NET Core

# Chương VI. Sử dụng Static Files trong ASP.NET Core

# Chương VII. MVC Design Pattern trong ASP.NET Core

# Chương VIII. Xây dựng ứng dụng ASP.NET Core MVC đầu tiên

# Chương IX. Cơ bản về ASP.NET Core Controller

# Chương X. Cơ chế Routing trong ASP.NET Core

# Chương XI. Attribute Routing

# Chương XII. Route Constrains

# Chương XIII. Action Selectors & Action Verbs

# Chương XIV. Action Result

# Chương XV. View

# Chương XVI. Razor View Engine trong ASP.NET Core MVC

# Chương XVII. Sử dụng Layouts và Section

# Chương XV. ViewBag và ViewData

# Chương XVIII. Model và ViewModel

# Chương XIX. Truyền dữ liệu từ Controller sang View

# Chương XX. Xây dựng HTML Form

# Chương XXI. Strongly Typed View

# Chương XXII. Tag Helpers trong ASP.NET Core MVC

# Chương XXIII. Input Tag Helper

# Chương XIV. Environment Tag Helper

# Chương XXV. Cơ chế Model Binding: Truyền dữ liệu từ View lên Controller

# Chương XXVI. Model Validation

# Chương XXVII. Validation Tag Helper

# Chương XXVIII. Unobtrusive Client Validation

# Chương XXIX. Cơ chế Dependency Injection

# Chương XXX. Vòng đời của Dependency Injection: Transient, Singleton và Scoped

# TÀI LIỆU THAM KHẢO

[1] Học ASP.NET Core căn bản: <https://tedu.com.vn/series/hoc-aspnet-core-can-ban.html>

[2] ASP.NET Core là gì? <https://topdev.vn/blog/asp-net-core-la-gi/>

[3] C# cơ bản .NET Core: <https://xuanthulab.net/lap-trinh-c-co-ban/>

# PHỤ LỤC

**1. FirstWebApp Project:** Github: