KHOA KỸ THUẬT VÀ CÔNG NGHỆ

**BỘ MÔN CÔNG NGHỆ THÔNG TIN**

****

THỰC TẬP ĐỒ ÁN CHUYÊN NGÀNH

**HỌC KỲ I, NĂM HỌC 2024-2025**

**ỨNG DỤNG RESTFUL API VÀ**

**.NET CORE PHÁT TRIỂN HỆ THỐNG ĐẶT VÉ XE ULTRABUS**

*Giáo viên hướng dẫn:*

ThS. Phạm Minh Đương

*Sinh viên thực hiện:*

Họ tên: Dương Văn Hiệp

MSSV: 110121209

Lớp: DA21TTB

***Trà Vinh, tháng 01 năm 2025***

KHOA KỸ THUẬT VÀ CÔNG NGHỆ

**BỘ MÔN CÔNG NGHỆ THÔNG TIN**

****

THỰC TẬP ĐỒ ÁN CƠ SỞ NGÀNH

**HỌC KỲ I, NĂM HỌC 2024-2025**

**ỨNG DỤNG RESTFUL API VÀ**

**.NET CORE PHÁT TRIỂN HỆ THỐNG ĐẶT VÉ XE ULTRABUS**

*Giáo viên hướng dẫn:*ThS. Phạm Minh Đương

*Sinh viên thực hiện:*

Họ tên: Dương Văn Hiệp

MSSV: 110121209

Lớp: DA21TTB

***Trà Vinh, tháng 01 năm 2025***

++

**NHẬN XÉT CỦA GIÁO VIÊN HƯỚNG DẪN**

*Trà Vinh, ngày ….. tháng …… năm ……*

**Giáo viên hướng dẫn**

*(Ký tên và ghi rõ họ tên)*

**NHẬN XÉT CỦA THÀNH VIÊN HỘI ĐỒNG**

*Trà Vinh, ngày ….. tháng …… năm ……*

**Thành viên hội đồng**

*(Ký tên và ghi rõ họ tên)*

**LỜI CẢM ƠN**

Đầu tiên, tôi xin gửi lời cảm ơn chân thành đến Khoa Kỹ thuật và Công nghệ, Bộ môn Công nghệ Thông tin đã tạo điều kiện thuận lợi cho tôi trong suốt quá trình học tập và hoàn thành đồ án thực tập chuyên ngành. Tiếp theo, tôi xin bày tỏ lòng biết ơn sâu sắc đến thầy ThS. Phạm Minh Đương đã truyền đạt kiến thức và hướng dẫn tôi trong quá trình làm đồ án.

Tôi đã cố gắng vận dụng những kiến thức đã học được trong các học kỳ qua để hoàn thành đồ án. Kết quả đạt được là cả quá trình nổ lực của tôi, tuy nhiên vẫn không tránh khỏi những thiếu sót trong quá trình nghiên cứu và trình bày. Rất kính mong sự góp ý của quý thầy (cô) để đồ án thực tập chuyên ngành của tôi được hoàn thiện hơn.

Xin trân trọng cảm ơn!

**Sinh viên thực hiện**

**MỤC LỤC**

[CHƯƠNG 1. TỔNG QUAN 1](#_Toc183221107)

[CHƯƠNG 2. NGHIÊN CỨU LÝ THUYẾT 2](#_Toc183221108)

[2.1 Giới thiệu về NextJS 2](#_Toc183221109)

[2.1.1 Tổng quan về NextJS 2](#_Toc183221110)

[2.1.2 Cách cài đặt NextJS 2](#_Toc183221111)

[2.1.3 Cấu trúc của NextJS 2](#_Toc183221112)

[2.1.4 Các thành phần cơ bản trong NextJS 2](#_Toc183221113)

[2.2 Giới thiệu về ASP.NET Core 2](#_Toc183221114)

[2.2.1 Tổng quan về ASP.NET Core 2](#_Toc183221115)

[2.2.2 Các đặc tính quan trọng của ASP.NET Core 3](#_Toc183221116)

[2.2.3 Các nhánh của ASP.NET 4](#_Toc183221117)

[2.2.4 Khác biệt quan trọng của ASP.NET và ASP.NET Core 4](#_Toc183221118)

[2.2.5 Cài đặt và cấu hình môi trường phát triển ASP.NET Core 4](#_Toc183221119)

[2.2.6 Dotnet CLI giao diện dòng lệnh trong ASP.NET Core 7](#_Toc183221120)

[2.2.7 Khởi tạo ứng dụng ASP.NET Core 9](#_Toc183221121)

[2.2.8 Kestrel: Web Server dành cho ASP.NET Core 13](#_Toc183221122)

[2.2.9 Middleware và Request Pipeline trong ASP.NET Core 15](#_Toc183221123)

[2.2.10 Routing trong ASP.NET Core 18](#_Toc183221124)

[2.2.11 Controller 19](#_Toc183221125)

[2.2.12 Model và ViewModel trong ASP.NET Core MVC 22](#_Toc183221126)

[2.2.13 View 24](#_Toc183221127)

[CHƯƠNG 3. HIỆN THỰC HÓA NGHIÊN CỨU 25](#_Toc183221128)

[3.1 Mô tả bài toán 25](#_Toc183221129)

[3.2 Đặc tả các yêu cầu chức năng 25](#_Toc183221130)

[3.2.1 Yêu cầu chức năng 25](#_Toc183221131)

[3.2.2 Yêu cầu phi chức năng 25](#_Toc183221132)

[3.3 Thiết kế dữ liệu 25](#_Toc183221133)

[3.3.1 Lược đồ cơ sở dữ liệu 25](#_Toc183221134)

[3.3.2 Danh sách các thực thể 25](#_Toc183221135)

[3.3.3 Chi tiết các thực thể 25](#_Toc183221136)

[3.4 Thiết kế giao diện 25](#_Toc183221137)

[3.4.1 Sơ đồ website 25](#_Toc183221138)

[3.4.2 Giao diện website 25](#_Toc183221139)

[CHƯƠNG 4. KẾT QUẢ NGHIÊN CỨU 25](#_Toc183221140)

[4.1 Dữ liệu thử nghiệm 25](#_Toc183221141)

[4.2 Kết quả thử nghiệm 25](#_Toc183221142)

[CHƯƠNG 5. KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN 25](#_Toc183221143)

[5.1 Kết luận 25](#_Toc183221144)

[5.2 Hướng phát triển 25](#_Toc183221145)

[DANH MỤC TÀI LIỆU THAM KHẢO 26](#_Toc183221146)

[PHỤ LỤC 27](#_Toc183221147)

**DANH MỤC HÌNH ẢNH - BẢNG BIỂU**

No table of figures entries found.

No table of figures entries found.

**TÓM TẮT ĐỒ ÁN CHUYÊN NGÀNH**

Đề tài xây dựng một hệ thống cho hãng xe khách, bao gồm các chức năng quản lý cho nội bộ và chức năng dành cho khách hàng.

Cách giải quyết vấn đề: Nêu tác dụng của các công nghệ dùng để giải quyết về vấn đề đó

Một số kết quả đạt được:

**MỞ ĐẦU**

# 

# TỔNG QUAN

Giới thiệu về đề tài và các công nghệ sử dụng

# NGHIÊN CỨU LÝ THUYẾT

## Giới thiệu về NextJS

### Tổng quan về NextJS

NextJS là một framework mã nguồn mở được phát triển dựa trên React, cho phép xây dựng các ứng dụng web với nhiều tính năng tối ưu hóa, bao gồm Server-Side Rendering (SSR) và Static Site Generation (SSG). Được phát triển bởi công ty Vercel, NextJS giúp tăng cường hiệu suất và khả năng SEO cho các ứng dụng web bằng cách cho phép máy chủ xử lý dữ liệu và tạo HTML trước khi gửi đến trình duyệt.

### Ưu nhược điểm của NextJS

**Ưu điểm**

### Cấu trúc của NextJS

### Các thành phần cơ bản trong NextJS

## Giới thiệu về ASP.NET Core

### Tổng quan về ASP.NET Core

ASP.NET Core là một tập hợp các thư viện chuẩn như một framework để xây dựng ứng dụng web.

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.

ASP.NET Core có thể chạy trên cả .NET Core hoặc full .NET Framework.

**.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](https://github.com/dotnet/core)

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

Nó cũng triển khai đặc điểm của .NET Standard.

**.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.

Bạn có thể tìm hiểu thêm về [**.NET Standard**](https://docs.microsoft.com/en-us/dotnet/standard/net-standard) tại đây.

**Sự khác nhau giữa .NET Core và .NET Framework**

.NET Framwork 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.

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.

### Các đặc tính quan trọng của ASP.NET Core

- Có thể xây dựng và chạy ứng dụng ASP.NET đa nền tảng trên Windows, Mac và Linux (mã nguồn mở và cộng đồng phát triển)

- ASP.NET Core hợp nhất ASP.NET MVC và ASP.NET Web API.

- Có thể host trên IIS hoặc tự host.

- Có sẵn Dependency Injection.

- Dễ dàng tích hợp với các framework frontend như Angular, Knockout...

- Hỗ trợ cấu hình cho nhiều môi trường.

- Cơ chết HTTP Request pipeline mới.

- Hỗ trợ quản lý phiên bản

- Dùng chung toàn bộ Nuget Package.

### Các nhánh của ASP.NET

Có hai nhánh của ASP.NET cho đến hiện tại là ASP.NET và ASP.NETCore.  ASP.NET là phiên bản hiện tại của ASP.NET và nó cần .NET Framwork để chạy.

ASP.NET Core là cách mới để xây dựng ứng dụng web. Nó có thể chạy cả trên .NET Framework và .NET Core.

### Khác biệt quan trọng của ASP.NET và ASP.NET Core

| **ASP.NET** | **ASP.NET CORE** |
| --- | --- |
| Nền tảng đã có từ lâu | Hoàn toàn được thiết kế mới |
| Chạy trên .NET Framwork | Chạy trên cả .NET Core và .NET Framework |
| Chỉ trên Windows | Chạy trên tất cả các OS sử dụng .NET Core |
| Nền tảng ổn định với tính năng phong phú | Chưa hoàn chỉnh nhưng mong đợi sẽ hoàn chỉnh trong  tương lai |
| WebForms được hỗ trợ | Không hỗ trợ WebForms |
| System.web.dll cồng kềnh | Nhỏ, nhẹ và module hóa |
| Bản quyền của Microsoft | ASP.NET Core là mã nguồn mở |

### Cài đặt và cấu hình môi trường phát triển ASP.NET Core

Để cài đặt và cấu hình môi trường phát triển **ASP.NET Core**, cần thực hiện các bước sau:

**Bước 1: Cài đặt .NET SDK**

**Tải và cài đặt .NET SDK**:

Truy cập trang chính thức của **.NET Core** tại <https://dotnet.microsoft.com/download>.

Tải phiên bản mới nhất của **.NET SDK** (Software Development Kit), đây là công cụ cần thiết để phát triển các ứng dụng với .NET Core.

Chọn hệ điều hành của bạn (Windows, macOS, Linux) và làm theo hướng dẫn cài đặt tương ứng.

**Kiểm tra cài đặt**:

Sau khi cài đặt, mở Command Prompt (Windows) hoặc Terminal (macOS/Linux) và gõ lệnh:

dotnet --version

Lệnh này sẽ hiển thị phiên bản .NET Core hiện tại đã được cài đặt.

**Bước 2: Cài đặt IDE (Integrated Development Environment)**

1. **Visual Studio** (Windows):

**Visual Studio** là IDE phổ biến nhất để phát triển ứng dụng ASP.NET Core. Tải Visual Studio tại <https://visualstudio.microsoft.com/>.

Chọn **Visual Studio 2022** (hoặc phiên bản mới nhất) và trong quá trình cài đặt, chọn các **workloads** sau:

**ASP.NET and web development**.

**.NET Core cross-platform development** (nếu phát triển trên Linux hoặc macOS).

1. **Visual Studio Code** (Windows, macOS, Linux):

Nếu bạn muốn sử dụng một IDE nhẹ, bạn có thể chọn **Visual Studio Code** (VS Code). Tải tại <https://code.visualstudio.com/>.

Cài đặt thêm **C# Extension** từ **Marketplace** trong VS Code để hỗ trợ phát triển ASP.NET Core.

Đảm bảo rằng bạn đã cài đặt **.NET SDK** trước khi làm việc với VS Code.

**Bước 3: Tạo và phát triển ứng dụng ASP.NET Core**

**Tạo ứng dụng ASP.NET Core**:

Mở Command Prompt hoặc Terminal và điều hướng đến thư mục bạn muốn tạo dự án.

Chạy lệnh sau để tạo một ứng dụng ASP.NET Core mới:

dotnet new mvc -n MyAspNetCoreApp

Lệnh này tạo một ứng dụng ASP.NET Core MVC (Model-View-Controller) cơ bản.

**Chạy ứng dụng**:

Di chuyển vào thư mục dự án mới tạo:

cd MyAspNetCoreApp

Chạy ứng dụng với lệnh:

dotnet run

Ứng dụng sẽ chạy trên **http://localhost:5000** mặc định. Bạn có thể mở trình duyệt và truy cập vào địa chỉ này để xem ứng dụng.

**Bước 4: Cấu hình ứng dụng ASP.NET Core**

**Cấu hình file appsettings.json**:

Trong dự án ASP.NET Core, bạn có thể sử dụng file appsettings.json để cấu hình các thiết lập của ứng dụng, ví dụ như kết nối cơ sở dữ liệu, cài đặt môi trường, v.v.

Cấu hình môi trường cũng có thể được thực hiện trong file Startup.cs để thiết lập dịch vụ và middleware.

**Cấu hình các môi trường**:

**ASP.NET Core** hỗ trợ nhiều môi trường (Development, Production, Staging).

Bạn có thể cấu hình môi trường bằng cách thiết lập biến môi trường ASPNETCORE\_ENVIRONMENT trong hệ điều hành hoặc trong launchSettings.json.

**Bước 5: Debug và triển khai**

1. **Debug**:

Trong Visual Studio, bạn có thể dễ dàng debug ứng dụng bằng cách thiết lập các điểm dừng và kiểm tra mã nguồn.

Trong Visual Studio Code, bạn cần cài đặt **C# extension** và sử dụng các công cụ debug tích hợp sẵn.

1. **Triển khai**:

ASP.NET Core hỗ trợ nhiều phương thức triển khai ứng dụng, từ việc triển khai trên IIS (Windows) đến triển khai trên các dịch vụ đám mây như **Azure**, **AWS**, hoặc **Docker**.

### Dotnet CLI giao diện dòng lệnh trong ASP.NET Core

**Download Dotnet CLI**

Dot Net CLI được cài đặt như một phần của .NET Core SDK. Có thể download tại đây [**link**](https://dotnet.microsoft.com/download)

CLI cài đặt mặc định đi kèm và có thể có nhiều phiên bản được cài đặt chung 1 máy.

**Sử dụng Dotnet CLI**

Cú phá .NET CLI gồm 3 phần. Từ khóa chính, động từ, và các tham số

dotnet [verb] [arguments]

Tên từ khóa chính là “dotnet”

Động từ là lệnh muốn thực thi. Lệnh thực hiện một hành động

Các tham số sẽ được pass vào để cung cấp thêm thông tin cho hành động.

**Các lệnh thường dùng:**

Dưới đây là một số câu lệnh phổ biến thường dùng cho dotnet:

| **Lệnh** | **mô tả** |
| --- | --- |
| 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 them trợ giúp |
| store | Lưu trữ các assembly cụ thể |

1. **Tạo dự án mới**

Để tạo một ứng dụng ASP.NET Core mới, bạn sử dụng lệnh dotnet new. Có thể tạo các loại ứng dụng khác nhau như Web, API, MVC, hoặc Console.

Tạo ứng dụng MVC:

dotnet new mvc -n MyAspNetCoreApp

Tạo API:

dotnet new webapi -n MyApiApp

Tạo ứng dụng console:

dotnet new console -n MyConsoleApp

1. **Khôi phục và cài đặt các gói phụ thuộc**

Khi bạn thêm hoặc sửa đổi các gói NuGet trong dự án, có thể khôi phục các gói này bằng lệnh:

dotnet restore

1. **Biên dịch ứng dụng**

Để biên dịch ứng dụng, sử dụng lệnh dotnet build. Lệnh này sẽ biên dịch mã nguồn của bạn thành các tệp có thể thực thi.

dotnet build

1. **Chạy ứng dụng**

Để chạy ứng dụng đã biên dịch, sử dụng lệnh dotnet run:

dotnet run

1. **Tạo bản phát hành**

Để tạo bản phát hành của ứng dụng cho môi trường sản xuất, sử dụng lệnh dotnet publish:

dotnet publish --configuration Release

1. **Kiểm tra dự án (Test)**

Dotnet CLI hỗ trợ kiểm tra đơn vị (unit testing) với dotnet test. Lệnh này sẽ chạy các bài kiểm tra đã được định nghĩa trong dự án.

dotnet test

1. **Xem thông tin về môi trường và cấu hình**

Có thể kiểm tra thông tin về .NET SDK hiện tại và các công cụ đã cài đặt với lệnh:

dotnet --info

1. **Các lệnh khác**

dotnet publish: Triển khai ứng dụng vào môi trường sản xuất.

dotnet pack: Đóng gói dự án thành một gói NuGet để phát hành.

dotnet nuget: Quản lý gói NuGet.

### Khởi tạo ứng dụng ASP.NET Core

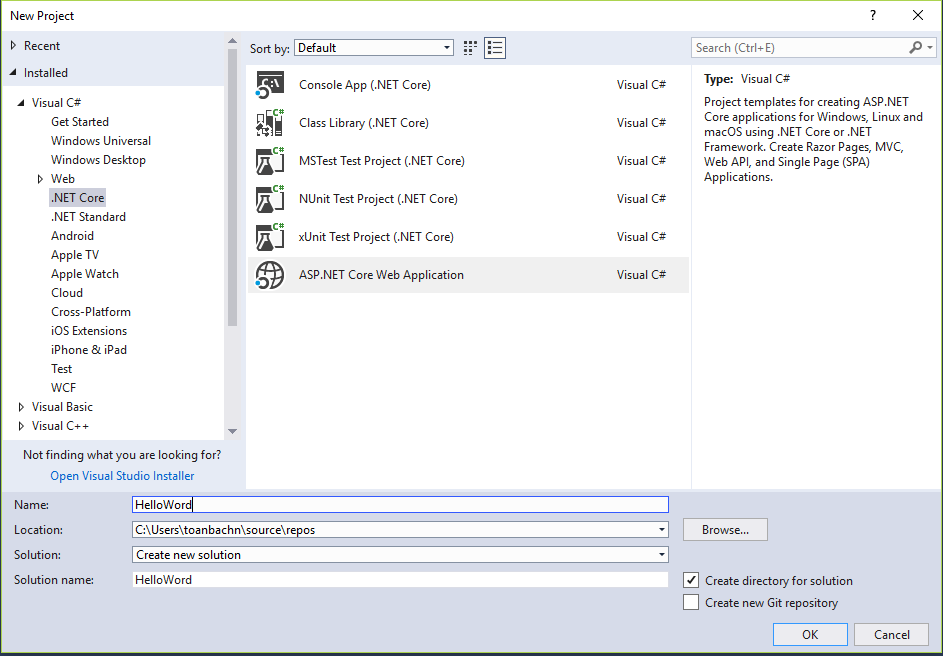
1. **Tạo một ứng dụng**

Khởi động Visual Studio 2017. Bạn cần bản 15.9.12 hoặc cao hơn. Nếu bạn chưa có nó có thể quay lại bài [cài đặt môi trường](https://tedu.com.vn/lap-trinh-aspnet-core/cai-dat-va-cau-hinh-moi-truong-phat-trien-aspnet-core-204.html):

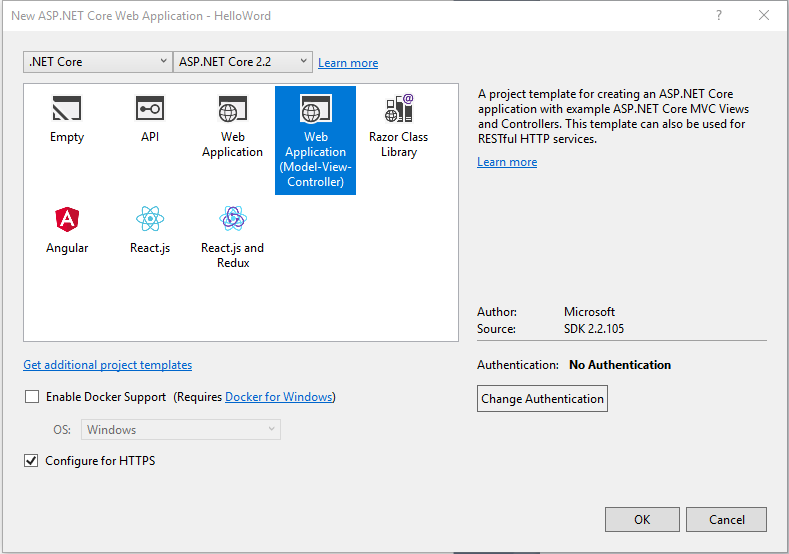
Vào menu Open -> File -> New -> Project

Chọn template ASP.NET Core Web Application. Template này nằm trong Visual C# -> .NET Core

Đặt tên project là HelloWorld.



Khi click nút OK, bạn sẽ sang dialog ASP.NET Core Web Application



Ở đây bạn có 1 số lựa chọn

1. **Runtime**

Dropdown đầu tiên là chọn môi trường, nó bao gồm 2 tùy chọn .NET Core & .NET Framework.

.NET Framework là framework cũ có nhiều tính năng nhưng chỉ giới hạn trên Windows

Tùy chọn số 2 là .NET Core, cho phép xây dựng ứng dụng chạy đa nền tảng

Chọn .Net Core

1. **Phiên bản của ASP.NET**

Dropdown 2 cho phép bạn chọn phiên bản của ASP.NET. Chúng ta có 3 lựa chọn ASP.NET Core 2.0, 2.1 và 2.2

Chọn .NET Core 2.2

1. **Project Template**

Tiếp theo chúng ta cần chọn Project Template. Có vài lựa chọn Angular, ReactJs template.

Chọn Empty

1. **Docker Support**

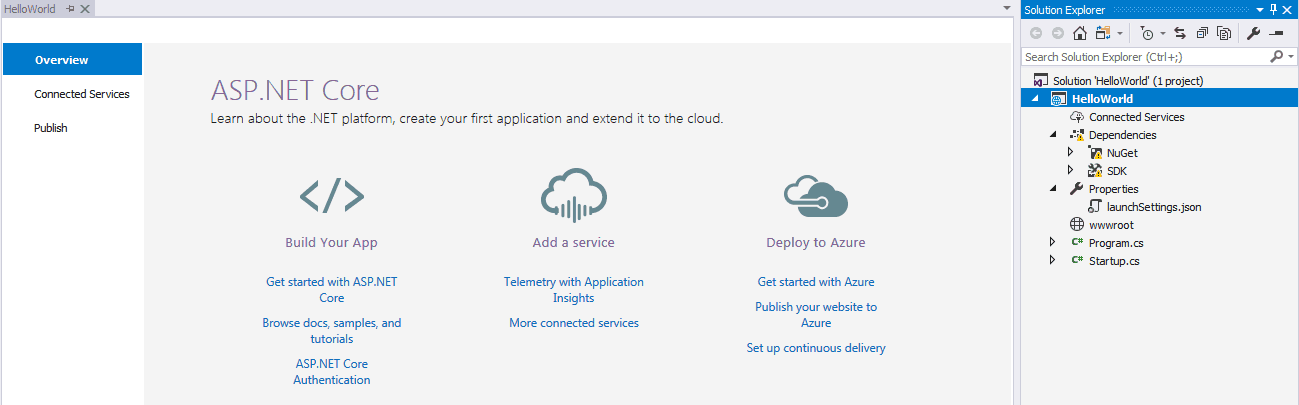
[**Docker**](https://www.docker.com/what-docker) support được them vào Visual Studio 2017 cho phép ứng dụng .NET chạy bên trong Docker container.

Tạm thời bỏ check nó đi

1. **Authentication**

Chúng ta có thể cài đặt individual, work or school hoặc windows authentication sử dụng tùy chọn này. Với Empty template,  chỉ có lựa chọn là No Authentication. Không sao cứ để vậy sau chúng ta thêm Authentication bằng  tay.

Click OK để tạo Project. Visual Studio tạo ra một project cơ bản nhỏ gọn nhất



1. **Chạy ứng dụng**

Nhấn F5 để chạy ứng dụng. Sẽ thấy dòng chữ “Hello World” trên trình duyệt

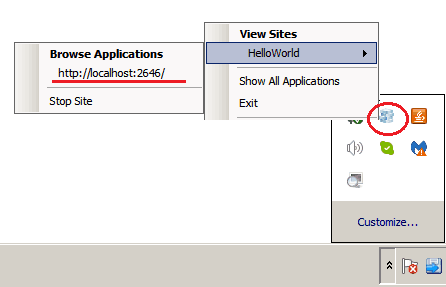
Bấm F5 để chạy ưng dụng ở mode Debug cho phép thay đổi ứng dụng khi nó đang chạy.

Bạn cũng có thể bấm Ctrl-F5, để chạy ứng dụng với mode không debug.

1. **Chạy ứng dụng trên IIS Express**

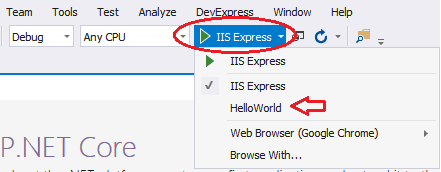
Visual Studio khởi động IIS Express và chạy ứng dụng của bạn. Nó chọn một cổng ngẫu nhiên để thực thi ứng dụng

Biểu tượng của IIS Express xuất hiện ở góc dưới bên phải theo hình dưới đây:



1. **Chạy ứng dụng với dotnet CLI**

Mặc định Visual Studio khởi động IIS Express để chạy ứng dụng. Có thể đổi từ IIS Express sang HelloWorld (hoặc tên Project) từ thanh standard



Đổi nó thành HelloWorld và bấm F5. Ứng dụng sẽ chạy bằng Dotnet CLI.

Có thể mở thư mục chứa project ( thư mục chứa file HelloWorld.csproj ) và chạy lệnh dotnet run cũng ra kết quả tương tự.

### Kestrel: Web Server dành cho ASP.NET Core

Kestrel là web server mặc định trong ASP.NET Core, được thiết kế để có hiệu suất cao và chạy trên nhiều nền tảng khác nhau như Windows, macOS và Linux. Nó hoạt động như một server nhẹ, nhanh và có khả năng xử lý nhiều kết nối đồng thời, rất phù hợp với các ứng dụng web hiện đại.

#### Đặc điểm của Kestrel:

1. **Hiệu suất cao**

Kestrel được xây dựng với mục tiêu đạt được hiệu suất tối đa, có khả năng xử lý hàng nghìn yêu cầu đồng thời với độ trễ thấp. Điều này là nhờ vào việc sử dụng cơ chế xử lý bất đồng bộ và quản lý kết nối rất hiệu quả.

1. **Đa nền tảng**

Kestrel chạy được trên cả Windows, macOS, và Linux, nhờ vào nền tảng .NET Core.

1. **Web Server nhẹ**

Kestrel là một web server nhẹ, có thể được sử dụng trực tiếp trong môi trường phát triển hoặc sản xuất mà không cần cài đặt các server phức tạp như IIS (Internet Information Services) hoặc Apache.

1. **Tích hợp với ASP.NET Core**

Kestrel tích hợp chặt chẽ với ASP.NET Core, hỗ trợ các yêu cầu HTTP/HTTPS và các tính năng khác của ứng dụng web như routing, middleware, và quản lý session.

**Kết nối Kestrel với Reverse Proxy (IIS, Nginx, Apache)**

Mặc dù Kestrel có thể hoạt động độc lập, nhưng trong môi trường sản xuất, nó thường được kết hợp với một reverse proxy server như IIS, Nginx, hoặc Apache để tận dụng các tính năng bảo mật, load balancing và xử lý kết nối HTTP/2. Reverse proxy này có thể bảo vệ Kestrel khỏi các cuộc tấn công và tối ưu hóa việc xử lý các kết nối đến.

**Cấu hình Kestrel**

Kestrel có thể được cấu hình trong Startup.cs hoặc Program.cs (tùy vào phiên bản .NET Core). Bạn có thể cấu hình port, sử dụng HTTPS, giới hạn kết nối, hoặc thiết lập các tùy chọn khác để tối ưu hóa server.

Ví dụ cấu hình Kestrel trong **Program.cs**:

public class Program

{

public static void Main(string[] args)

{

CreateHostBuilder(args).Build().Run();

}

public static IHostBuilder CreateHostBuilder(string[] args) =>

Host.CreateDefaultBuilder(args)

.ConfigureWebHostDefaults(webBuilder =>

{

webBuilder.UseKestrel(options =>

{

options.Listen(IPAddress.Any, 5000); // Lắng nghe tất cả các IP trên port 5000

})

.UseStartup<Startup>();

});

}

#### Ưu điểm và nhược điểm của Kestrel:

**Ưu điểm**:

* Hiệu suất cao, phù hợp với các ứng dụng web cần khả năng xử lý đồng thời lượng lớn yêu cầu.
* Khả năng hoạt động trên nhiều hệ điều hành.
* Cấu hình và triển khai dễ dàng.

**Nhược điểm**:

* Không tích hợp tính năng bảo mật (chẳng hạn như SSL/TLS) mặc định, cần phải cấu hình với reverse proxy để có đầy đủ bảo mật khi triển khai trên môi trường sản xuất.

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

Trong ASP.NET Core, Middleware và Request Pipeline là các khái niệm quan trọng giúp xử lý các yêu cầu HTTP, cho phép bạn xử lý, sửa đổi hoặc trả về phản hồi trước khi yêu cầu được gửi đến controller hoặc sau khi controller xử lý xong.

#### Middleware là gì?

Middleware là các lớp phần mềm giúp xử lý yêu cầu HTTP trong quá trình từ khi nó đến server cho đến khi phản hồi được trả về. Mỗi middleware trong ASP.NET Core có thể thực hiện một công việc cụ thể như xác thực, ghi log, xử lý lỗi, hoặc trả về dữ liệu.

Một middleware có thể:

* Chặn hoặc modifying yêu cầu HTTP trước khi chúng đến controller.
* Chặn hoặc modifying phản hồi HTTP trước khi chúng được gửi đến client.
* Điều khiển quá trình yêu cầu – ví dụ, có thể bỏ qua các middleware phía sau nếu cần.

Các middleware có thể được liên kết với nhau theo thứ tự nhất định trong Request Pipeline.

#### Request Pipeline là gì?

Request Pipeline là chuỗi các middleware mà một yêu cầu HTTP đi qua. Khi yêu cầu được gửi đến ứng dụng ASP.NET Core, nó sẽ đi qua các middleware trong pipeline này. Pipeline xác định thứ tự mà các middleware sẽ được thực thi.

#### Cách hoạt động của Middleware và Request Pipeline

1. **Xử lý yêu cầu HTTP**:
   * Khi một yêu cầu HTTP đến, nó đi qua các middleware trong pipeline theo thứ tự từ trên xuống dưới.
   * Mỗi middleware có thể làm một số công việc (như kiểm tra xác thực, ghi nhật ký, điều chỉnh yêu cầu, ...), và có thể **tiếp tục** hoặc **dừng** quá trình xử lý yêu cầu.
2. **Xử lý phản hồi HTTP**:
   * Khi yêu cầu được xử lý (ví dụ: controller trả về kết quả), phản hồi HTTP sẽ đi qua các middleware theo chiều ngược lại (từ dưới lên trên).
   * Các middleware có thể thay đổi hoặc thêm các thông tin vào phản hồi trước khi gửi lại client.

Ví dụ về Middleware trong ASP.NET Core

Một ví dụ đơn giản về middleware là logging middleware, có thể ghi lại thông tin của mỗi yêu cầu HTTP.

**Cấu hình middleware trong Startup.cs**

public class Startup

{

public void Configure(IApplicationBuilder app)

{

// Thêm middleware vào pipeline

app.UseMiddleware<MyCustomMiddleware>();

// Middleware xử lý các yêu cầu HTTP

app.UseRouting();

// Xử lý các yêu cầu tới endpoint của ứng dụng

app.UseEndpoints(endpoints =>

{

endpoints.MapControllers();

});

}

}

Middleware tùy chỉnh:

public class MyCustomMiddleware

{

private readonly RequestDelegate \_next;

public MyCustomMiddleware(RequestDelegate next)

{

\_next = next;

}

public async Task InvokeAsync(HttpContext context)

{

// Xử lý yêu cầu trước khi nó đến controller

Console.WriteLine("Middleware: Processing Request");

// Tiếp tục chuỗi middleware

await \_next(context);

// Xử lý phản hồi sau khi controller xử lý xong

Console.WriteLine("Middleware: Processing Response");

}

}

#### Một số loại Middleware phổ biến trong ASP.NET Core

1. UseRouting và UseEndpoints: Dùng để cấu hình các routes cho ứng dụng và kết nối yêu cầu HTTP với các controller.
2. UseAuthentication và UseAuthorization: Được dùng để xác thực và phân quyền người dùng.
3. UseStaticFiles: Middleware xử lý các file tĩnh (như hình ảnh, CSS, JavaScript).
4. UseCors: Cấu hình Cross-Origin Resource Sharing (CORS) để cho phép yêu cầu từ các domain khác.
5. UseExceptionHandler: Dùng để xử lý các lỗi toàn cục trong ứng dụng.

### Routing trong ASP.NET Core

Routing trong ASP.NET Core là cơ chế xử lý các yêu cầu HTTP và ánh xạ chúng đến các phương thức của controller hoặc các hành động tương ứng. Nó cho phép ứng dụng xác định và xử lý các URL mà người dùng gửi yêu cầu tới. Dưới đây là một số điểm chính về Routing trong ASP.NET Core:

1. **Cấu hình Routing**:

Routing trong ASP.NET Core có thể được cấu hình trong Startup.cs thông qua phương thức UseRouting() và UseEndpoints(). Đây là các phương thức cần thiết để thiết lập các tuyến đường cho các yêu cầu HTTP.

1. **Attribute Routing**:

ASP.NET Core hỗ trợ Attribute Routing, cho phép bạn gán trực tiếp các route vào các phương thức hành động trong controller bằng các thuộc tính như [Route("api/products")] hoặc [HttpGet].

1. **Conventional Routing**:

Với Conventional Routing, bạn định nghĩa các tuyến đường chung trong Configure method của Startup.cs, và các route này sẽ được ánh xạ đến các phương thức hành động dựa trên tên phương thức và mô hình URL.

1. **Routing Parameters**:

Bạn có thể sử dụng các tham số trong URL thông qua syntax như {id} hoặc {controller} trong route để định nghĩa các giá trị động, ví dụ: /products/{id}.

1. **Custom Routes**:

ASP.NET Core cho phép bạn tạo các route tùy chỉnh để hỗ trợ các yêu cầu cụ thể từ người dùng. Điều này có thể giúp thiết kế ứng dụng với cấu trúc URL dễ hiểu và có thể tùy chỉnh.

1. **Endpoints**:

Sau khi đã cấu hình routing, bạn có thể chỉ định các endpoint cụ thể thông qua UseEndpoints(), nơi bạn chỉ rõ các controller hoặc các hành động cần thực thi khi nhận yêu cầu từ người dùng.

Ví dụ về Attribute Routing

[Route("api/[controller]")]

public class ProductsController : Controller

{

[HttpGet("{id}")]

public IActionResult GetProduct(int id)

{

return Ok(new { ProductId = id });

}

}

Ví dụ về Conventional Routing trong Startup.cs:

public void Configure(IApplicationBuilder app, IWebHostEnvironment env)

{

app.UseRouting();

app.UseEndpoints(endpoints =>

{

endpoints.MapControllerRoute(

name: "default",

pattern: "{controller=Home}/{action=Index}/{id?}");

});

}

### Controller

1. **Định nghĩa Controller**

Controller trong ASP.NET Core là một lớp xử lý các yêu cầu HTTP từ người dùng và trả về các phản hồi phù hợp (như trang HTML, dữ liệu JSON, v.v.).

Mỗi Controller thường chứa nhiều phương thức hành động (action methods) được ánh xạ tới các URL cụ thể. Mỗi phương thức hành động này xử lý một loại yêu cầu HTTP (GET, POST, PUT, DELETE, v.v.).

1. **Tạo Controller**

Để tạo một Controller mới, bạn có thể sử dụng công cụ scaffolding của Visual Studio hoặc viết tay. Để sử dụng scaffolding, bạn có thể sử dụng lệnh dotnet aspnet-codegenerator controller hoặc qua công cụ menu của Visual Studio.

Tên của Controller thường được đặt theo nguyên tắc, ví dụ như ProductsController, với Products là danh từ số nhiều và Controller là hậu tố của lớp.

1. **Phương thức hành động:**

Mỗi phương thức hành động được định nghĩa với một chú thích HTTP tương ứng (ví dụ: [HttpGet], [HttpPost], [HttpPut], [HttpDelete]).

Các phương thức này xử lý các hành động HTTP cụ thể và có thể trả về các kiểu dữ liệu khác nhau, chẳng hạn như IActionResult, ViewResult, JsonResult, v.v.

1. **Dữ liệu Model Binding**

Controller sử dụng kỹ thuật model binding để chuyển dữ liệu từ yêu cầu HTTP (như dữ liệu form, dữ liệu JSON) vào các đối tượng model trong phương thức hành động.

Dữ liệu được tự động ánh xạ từ yêu cầu HTTP vào các thuộc tính của model mà không cần phải gán thủ công.

1. **Dependency Injection**

ASP.NET Core Controller hỗ trợ cơ chế Dependency Injection (DI) để quản lý các dịch vụ và dependency bên trong controller.

Các dịch vụ cần thiết có thể được inject vào controller thông qua constructor hoặc các phương thức khác.

1. **Cấu hình Controller**

Cấu hình cho các Controller thường được thực hiện trong Startup.cs bằng cách chỉ định các route, middleware, và endpoints cần thiết để xử lý yêu cầu HTTP.

Các route có thể được định nghĩa bằng các thuộc tính như [Route] hoặc thông qua MapControllerRoute trong Startup.cs.

Ví dụ về Controller cơ bản

using Microsoft.AspNetCore.Mvc;

namespace MyApp.Controllers

{

[Route("api/[controller]")]

public class ProductsController : Controller

{

private readonly IProductService \_productService;

public ProductsController(IProductService productService)

{

\_productService = productService;

}

[HttpGet]

public IActionResult GetAllProducts()

{

var products = \_productService.GetAll();

return Ok(products);

}

[HttpGet("{id}")]

public IActionResult GetProductById(int id)

{

var product = \_productService.GetById(id);

if (product == null)

{

return NotFound();

}

return Ok(product);

}

[HttpPost]

public IActionResult CreateProduct([FromBody] Product product)

{

\_productService.Create(product);

return CreatedAtAction(nameof(GetProductById), new { id = product.Id }, product);

}

[HttpPut("{id}")]

public IActionResult UpdateProduct(int id, [FromBody] Product product)

{

if (id != product.Id)

{

return BadRequest();

}

\_productService.Update(product);

return NoContent();

}

[HttpDelete("{id}")]

public IActionResult DeleteProduct(int id)

{

\_productService.Delete(id);

return NoContent();

}

}

}

### Model và ViewModel trong ASP.NET Core MVC

1. **Model trong ASP.NET Core MVC**

Model là lớp đại diện cho dữ liệu và các nghiệp vụ liên quan đến dữ liệu đó. Model chứa các thuộc tính mà bạn muốn lưu trữ trong cơ sở dữ liệu và là đối tượng mà các Controller xử lý.

Model không chỉ đại diện cho dữ liệu mà còn chứa các logic nghiệp vụ (business logic) cần thiết để xử lý dữ liệu trước khi trả về cho View hoặc lưu vào cơ sở dữ liệu.

Ví dụ về Model:

public class Product

{

public int Id { get; set; }

public string Name { get; set; }

public decimal Price { get; set; }

public string Description { get; set; }

}

Model này có thể được sử dụng để lưu trữ dữ liệu sản phẩm trong cơ sở dữ liệu và có thể được sử dụng trong View để hiển thị thông tin sản phẩm.

2. **ViewModel trong ASP.NET Core MVC**

ViewModel là lớp đại diện cho dữ liệu mà View cần để hiển thị, nhưng nó không nhất thiết phải giống với Model. ViewModel được sử dụng để kết hợp nhiều Model hoặc để chuẩn bị dữ liệu theo cách mà View cần.

ViewModel giúp truyền dữ liệu từ Controller đến View, có thể bao gồm các trường không tồn tại trong Model cơ sở dữ liệu, ví dụ như các trường được tính toán hoặc các thông tin bổ sung cần thiết chỉ để hiển thị trên giao diện người dùng.

Ví dụ về ViewModel:

public class ProductViewModel

{

public string Name { get; set; }

public string PriceFormatted { get; set; }

public string Description { get; set; }

public bool IsOnSale { get; set; }

}

Trong ví dụ này, ProductViewModel có thể chứa thông tin từ Product Model, nhưng có thể thêm các thông tin bổ sung như PriceFormatted để hiển thị giá dưới dạng tiền tệ.

**Sự khác biệt chính giữa Model và ViewModel**

|  |  |  |
| --- | --- | --- |
| **Yếu tố** | **Model** | **ViewModel** |
| **Mục đích** | Đại diện cho dữ liệu và logic nghiệp vụ của ứng dụng | Chuyển dữ liệu từ Controller đến View |
| **Dữ liệu** | Thường phản ánh cấu trúc dữ liệu trong cơ sở dữ liệu | Có thể chứa dữ liệu kết hợp từ nhiều Model hoặc dữ liệu tính toán cho View |
| **Định hướng** | Thường dành cho xử lý dữ liệu và lưu trữ trong CSDL | Dành cho việc trình bày dữ liệu và hiển thị trên giao diện người dùng |

### View

**View** là thành phần chịu trách nhiệm hiển thị giao diện người dùng (UI). View nhận dữ liệu từ Controller và trình bày chúng dưới dạng HTML hoặc bất kỳ định dạng phù hợp nào để người dùng cuối có thể tương tác. Dưới đây là các điểm cơ bản về View trong ASP.NET Core.

**Cấu trúc của View**

View trong ASP.NET Core MVC được tạo ra dưới dạng các tệp Razor (.cshtml). Tệp Razor có thể chứa mã C# cùng với HTML, và mã C# trong tệp Razor được đánh dấu bằng cú pháp @.

Tệp .cshtml được xử lý bởi Engine Razor để chuyển mã HTML động vào trong trang web khi người dùng truy cập.

Ví dụ về View (Index.cshtml):

@model IEnumerable<Product> <!-- Định nghĩa kiểu dữ liệu mà View sẽ nhận từ Controller -->

<h2>Product List</h2>

<table>

<tr>

<th>Name</th>

<th>Price</th>

</tr>

@foreach (var product in Model) <!-- Duyệt qua Model và hiển thị dữ liệu -->

{

<tr>

<td>@product.Name</td>

<td>@product.Price</td>

</tr>

}

</table>

# HIỆN THỰC HÓA NGHIÊN CỨU

## Mô tả bài toán

## Đặc tả các yêu cầu chức năng

### Yêu cầu chức năng

### Yêu cầu phi chức năng

## Thiết kế dữ liệu

### Lược đồ cơ sở dữ liệu

#### Mô hình thực thể kết hợp

#### Mô hình vật lý

### Danh sách các thực thể

### Chi tiết các thực thể

## Thiết kế giao diện

### Sơ đồ website

### Giao diện website

# KẾT QUẢ NGHIÊN CỨU

## Dữ liệu thử nghiệm

## Kết quả thử nghiệm

# KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

## Kết luận

## Hướng phát triển

# DANH MỤC TÀI LIỆU THAM KHẢO

# PHỤ LỤC

**Hướng dẫn cài đặt laragon**