# GIỚI THIỆU

> Bài hướng dẫn này sẽ chỉ bạn các bước cơ bản để tạo một Web API sử dụng ASP.net Core của Microsoft.

Tưởng tượng bạn đang làm việc cho một công ty bán bán đồ cho Thú cưng. Quản lý yêu cầu bạn phát triển một RESTful dịch vụ quản lý hàng cho cửa hàng Online của công ty. Dịch vụ API yêu cầu cần có hỗ trợ `Thêm` `Xoá` `Sửa` `Xem chi tiết` các sản phẩm ( Là hoạt động cơ bản của HTTP có thể gọi là `Read` `Update` `Delete` - **CRUD** )

Bài này sẽ hướng dẫn tạo nên một `cross-platform` RESTful với `ASP.NET Core web API` viết bằng `.NET core` và `C#`. Chúng ta cũng sẽ được học cách làm việc Database. `Entity Framework` (EF) sẽ được sử dụng như **O**bject-**R**elational **M**apper (O/RM) (Chuyển đổi CSDL giữa các hệ thống).

Chuỗi bài này sẽ .NET Core CLI để hướng dẫn phát triển một Web API. Sau khi hoàn thành chuỗi đề Hướng dẫn này, bạn có thể áp dụng để phát triển ở các môi trường khác như `Visual Studio` (Window), `Visual Studio for Mac` (MacOS) hoặc `Visual Studio Code` (Windows, Linux, & macOS).

## Mục tiêu

Trong chuỗi bài này bạn sẽ được học:

- Sử dụng .NET Core CLI để tạo một project ASP.NET Core web API.
- Tạo Database sản phẩm
- Thêm CRUD vào web API
- Test Web API

## Điều kiện tiên quyết

- Nắm được cơ bản về C#
- Nắm kiến thức về RESTful và HTTP

# TẠO PROJECT WEB API

Sử dụng .NET core CLI là cách đơn giản nhất để tạo ASP.NET Core web API. CLI đã được tích hợp sẵn vào trong Azure Cloud Shell, nên rất thuận tiện trong việc phát triện trên Azure.

# Tạo mới Project

## Mổ xẻ thành phần

1. Chạy dòng .NET Core CLI command trên CMD**

```
dotnet new webapi -o contoso-pets/src/ContosoPets.Api
```

Như vậy Project API mới đã được khởi tạo. Dòng lệnh trên khởi tạo dựa trên khung mẫu sẵn của ASP.NET Core. `webapi` dựng sẵn viết bằng *C#*. `contoso-pets/src/ContosoPets.Api` là đường dẫn khởi tạo. Tên của Project này là `ContosoPets.Api` được thể hiện trong đường dẫn.

1. Chạy dòng lệnh

```
cd ./contoso-pets/src/ContosoPets.Api
```

`cd` vào thư mục chưa project

1. Chạy dòng lệnh

```
code .
```

Project `ContosoPets.Api` sẽ được mở trực tiếp ở editor.

1. Phân tích cấu trúc file

| Name | Decription |
| --- | --- |
| *Controllers/* | Chứa các public Classes mô tả điểm cuối của HTTP |
| *Program.cs* | Chứa hàm `Main` quản lý xuất nhập của app |
| *ContosoPets.Api.csproj* | Chứa cấu hình metadata của project |
| *Startup.cs* | Cấu hình dịch vụ và là nơi nhận lệnh HTTP |

# Build và Test

1. Chạy dòng code trên CMD:

```
dotnet run > ContosoPets.Api.log &
```

Dòng code trên sẽ:

- Phục hồi các gói Nugget
- Build
- Host một web api với Kestel web server
- Hiện thị ID tác vụ đang chạy

.NET Core sẽ yêu cầu đăng nhập vào thông tin và khối lệnh. Ngoài ra `&` dùng để chạy code dưới dạng code chạy ở tiến trình nền

Web api được lưu trữ trên cả `http://localhost:5000` và `https://localhost:5001`

1. Gửi yêu HTTP GET để test API:

```
curl -k -s https://localhost:5001/api/values | jq
```

> `curl` là một CLI tool để dùng để test API và cả dùng để test các Web Entry point trả về.

Dòng code trên đã dùng:

- HTTPS để gửi yêu cầu tới web API chạy trên cổng 5001 của localhost. Class `ValuesController`
- `-k` là tuỳ chọn để biểu thị curl có thể kết nối không cần kiểm tra an toàn khi làm việc với HTTPS. Cần đó NET Core SDK và HTTPS development certificate để test.
- `-s` là tuỳ chọn để loại trừ tất cả các dạng trả về trừ JSON. JSON được gởi vào jq command-line JSON bộ xử lý JSON để biểu thị rõ ràng hơn

```
[
  "value1",
  "value2"
]
```

1. Dừng tất cả các tiến trình

```
kill $(pidof dotnet)
```

# THÊM DATA

Class `Model` cần thiết để mô tả đại diện *sản phẩm ở* trong *kho*. `Model` mô tả rõ ràng và đầy đủ tính chất của *sản phẩm* và được dùng truyền dữ liệu Web API. Nhờ `Model` giữ tính *Bất biến* cho *Sản phẩm* trong *Kho hàng*. Dữ liệu được tạo như là [linkin-memory EF Core database].

Trong bài này mình chỉ cần sử dụng in-memory database cho đơn giản. Khi phát triển sản phẩn nên chọn các dịch lưu trữ dữ liệu khác như là SQL Server or Azure SQL Database.

1. Chạy dòng code trên CMD tạo class Model

```
mkdir Models && touch $_/Product.cs
```

> `touch` là lệnh chỉ có trên Linux và Cloud Shell

Dòng code trên đã tạo folder `Models` tại `root`. Và tạo file .cs *Product* trong thư mục Models.

> Thư mục được đặt tên là `Models` là xuất phát từ quy ước của kiến trúc *Model*-View-Controller

1. Thêm code vào `Product.cs`

```
using System.ComponentModel.DataAnnotations;

namespace ContosoPets.Api.Models
{
    public class Product
    {
        public long Id { get; set; }

        [Required]
        public string Name { get; set; }

        [Required]
        [Range(minimum: 0.01, maximum: (double) decimal.MaxValue)]
        public decimal Price { get; set; }
    }
}
```

Product bao gồm:

| Thuộc tính | Ý nghĩa | Yêu cầu |
| --- | --- | --- |
| id | là id cảu sản phẩm | Kiểu `long` |
| Name | là tên sản phẩm | Kiểu `string`, không được NULL |
| Price | là giá của sản phẩm | Kiểu `decimal`, không được NULL, giá trị nhỏ nhất là 0.01 |

`Name` và `Price` được đánh dấu là là yêu cầu cần để đảm bảo giá trị được cung cấp. Ngoài ra `Price` được định giá trình `Max` và `Min`.

1. Thêm DataContext:

```
mkdir Data && touch $_/ContosoPetsContext.cs $_/SeedData.cs
```

Folder Data được tạo và thêm 2 file .cs trống *ContosoPetsContext.cs* và *SeedData*.

Thêm code vào `Data/ContosoPetsContext.cs`:

```
using Microsoft.EntityFrameworkCore;
using ContosoPets.Api.Models;

namespace ContosoPets.Api.Data
{
    public class ContosoPetsContext : DbContext
    {
        public ContosoPetsContext(DbContextOptions<ContosoPetsContext> options)
            : base(options)
        {
        }

        public DbSet<Product> Products { get; set; }
    }
}
```

`ContosoPetsContext` thực hiện EF Core từ `DbContext` yêu cầu truy cập dữ liệu in-memory database.

1. Chỉnh sử `Startup.cs`

Thêm dòng code sau vào method `ConfigureServices`

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddDbContext<ContosoPetsContext>(options =>  //this line
        options.UseInMemoryDatabase("ContosoPets"));      //this line
    services.AddMvc().SetCompatibilityVersion(CompatibilityVersion.Version_2_2);
}
```

Dòng code trên đã thực hiện:

- Đăng ký class `DbContext` với tên mới là `ContosoPetsContext` bằng ASP.net Core dependency injection
- Định nghĩa in-memory database tên là `ContosoPets` từ đó ta có thể thêm dữ liệu mẫu vào `ContosoPets`

1. Cập nhập Depencies trong file `startup.cs`

```
using Microsoft.EntityFrameworkCore;
using ContosoPets.Api.Data;
```

`UseInMemoryDatabase` được gọi từ `Microsoft.EntityFrameworkCore`.
`ContosoPetsContext` được gọi từ `ContosoPets.Api.Data`.

1. Thêm Data vào `Data/SeedData.cs.`

```
using System;
using System.Linq;
using Microsoft.EntityFrameworkCore;
using Microsoft.Extensions.DependencyInjection;
using ContosoPets.Api.Models;

namespace ContosoPets.Api.Data
{
    public static class SeedData
    {
        public static void Initialize(ContosoPetsContext context)
        {
        //Tạo dữ liệu ban đầu tại đây
            if (!context.Products.Any())
            {
                context.Products.AddRange(
                    new Product
                    {
                        Name = "Squeaky Bone",
                        Price = 20.99m
                    },
                    new Product
                    {
                        Name = "Knotted Rope",
                        Price = 12.99m
                    }
                );

                context.SaveChanges();
            }
        }
    }
}
```

Sử dụng 'EntityFrammeworkCore'. Method `Initialize` đã khởi tạo in-memory database 2 món sản phẩm.

1. Chỉnh sửa file `Program.cs`

```
using System;
using Microsoft.AspNetCore;
using Microsoft.AspNetCore.Hosting;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Logging;
```

```
using ContosoPets.Api.Data;

namespace ContosoPets.Api
{
    public class Program
    {
        public static void Main(string[] args)
        {
            var host = CreateWebHostBuilder(args).Build();
            SeedDatabase(host);
            host.Run();
        }

        public static IWebHostBuilder CreateWebHostBuilder(string[] args) =>
            WebHost.CreateDefaultBuilder(args)
                .UseStartup<Startup>();

        private static void SeedDatabase(IWebHost host)
        {
            var scopeFactory = host.Services.GetRequiredService<IServiceScopeFactory>()

            using (var scope = scopeFactory.CreateScope())
            {
                var context = scope.ServiceProvider.GetRequiredService<ContosoPetsConte

                if (context.Database.EnsureCreated())
                {
                    try
                    {
                        SeedData.Initialize(context);
                    }
                    catch (Exception ex)
                    {
                        var logger = scope.ServiceProvider.GetRequiredService<ILogger<P
                        logger.LogError(ex, "A database seeding error occurred.");
                    }
                }
            }
        }
    }
}
```

`Program.Main` là method được thực hiện đầu tiên khi App được khởi động. Data seed được gọi tại `SeedData.Initialize`

1. Build app

```
dotnet build
```