



Chương 5: ASP.NET Core MVC – Căn bản

Giảng viên: Tạ Việt Phương
E-mail: phuongtv@uit.edu.vn

Nội dung



ASP.NET Core



ASP.NET Core MVC

Nội dung



ASP.NET Core

Giới thiệu ASP.NET Core

- 📖 Là framework đa nền tảng (cross-platform) chạy trên nền tảng .NET Core.
- 📖 Là thư viện chuẩn để xây dựng ứng dụng web.
- 📖 Phiên bản đầu tiên 06/2016. Hiện tại là phiên bản 6.0 (11/2021, long-term support) và 7.0 (11/2022, standard-term support) – có trong Visual Studio 2022.
- 📖 Được thiết kế để đáp ứng các yêu cầu:
 - Phát triển ứng dụng trên đa nền tảng.
 - Nâng cao hiệu suất
 - Kiến trúc dựa trên các module.
 - Là mã nguồn mở.
 - Phù hợp với xu hướng hiện đại của ứng dụng web
- 📖 Hoạt động được trên Windows, macOS và Linux.

Giới thiệu ASP.NET Core

📖 Có thể triển khai ứng dụng viết trên ASP.NET Core như một web server độc lập hoặc kết hợp với các web server khác: IIS, Apache, Nginx.

📖 Có thể chạy trên được .NET Framework:

- Hệ thống API của ASP.NET Core sử dụng các API cơ bản của .NET. Trong đó .NET Core và .NET Framework có chung hệ thống API cơ bản.
- Cùng chung runtime thực thi cho ứng dụng, đều ở dạng mã trung gian IL (Intermediate Language).
- ASP.NET Core phiên bản 2.0 tới 2.2 chạy trên được .NET Framework 4.6.1 hoặc cao hơn và trên .NET Core 2.0 hoặc cao hơn.
- ASP.NET Core 3.0 chỉ chạy trên .NET Core 3.0

Giới thiệu ASP.NET Core

- 📖 Không còn dựa trên System.Web.dll, mà dựa trên các gói, các module (Nuget packages).
- 📖 Các class/thư viện của ASP.NET Core sử dụng chung cho các mô hình phát triển ứng dụng web: MVC, Web API, Razor Pages.
- 📖 Tích hợp liền mạch với các thư viện và framework phía client: Angular, React, Bootstrap...
- 📖 Các ứng dụng của ASP.NET Core:
 - Web HTML theo mô hình MVC hoặc Razor Pages.
 - REST API cho ứng dụng đơn trang (Single Page Application – SPA).
 - Dịch vụ gọi hàm từ xa: Remote Procedure Call – RPC.

Giới thiệu ASP.NET Core

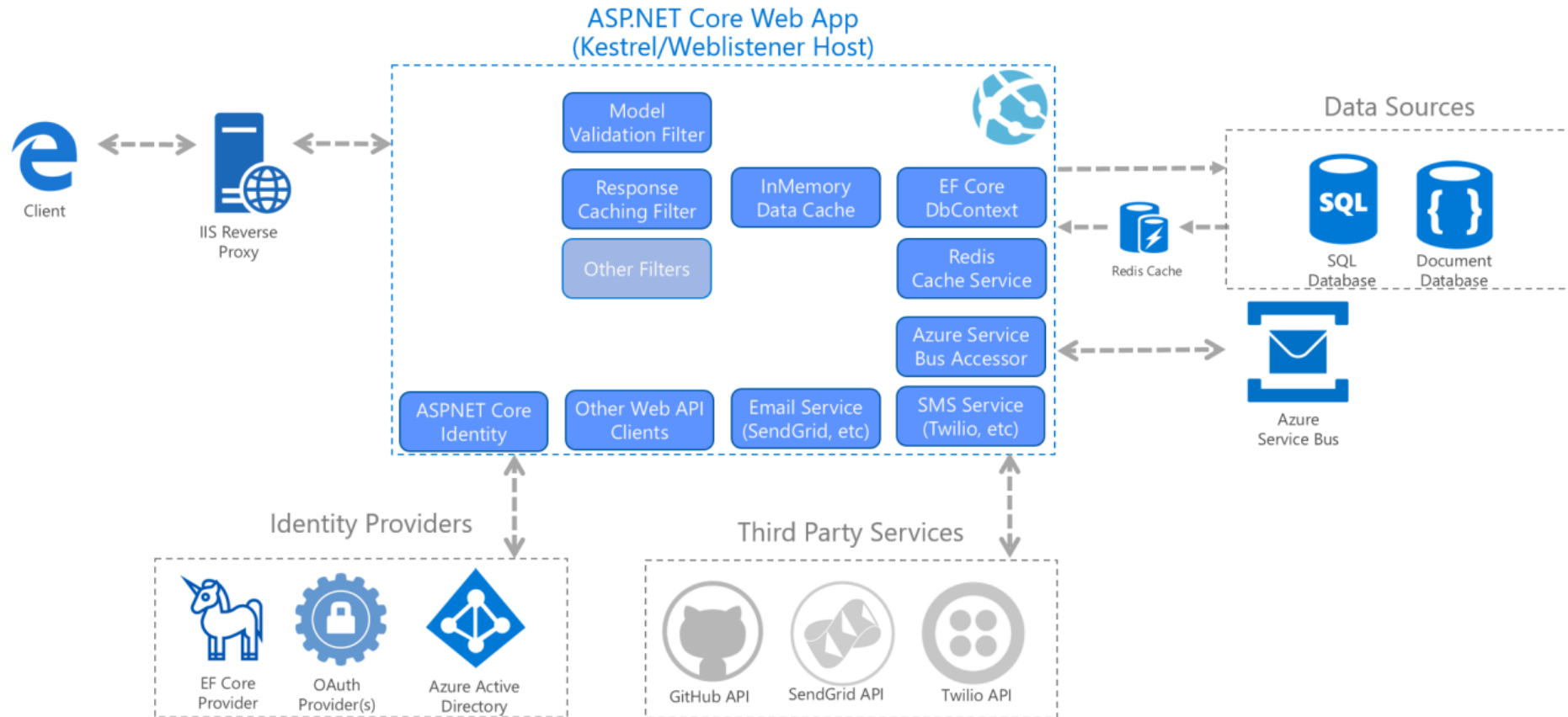


Các đặc điểm nổi bật của ASP.NET Core:

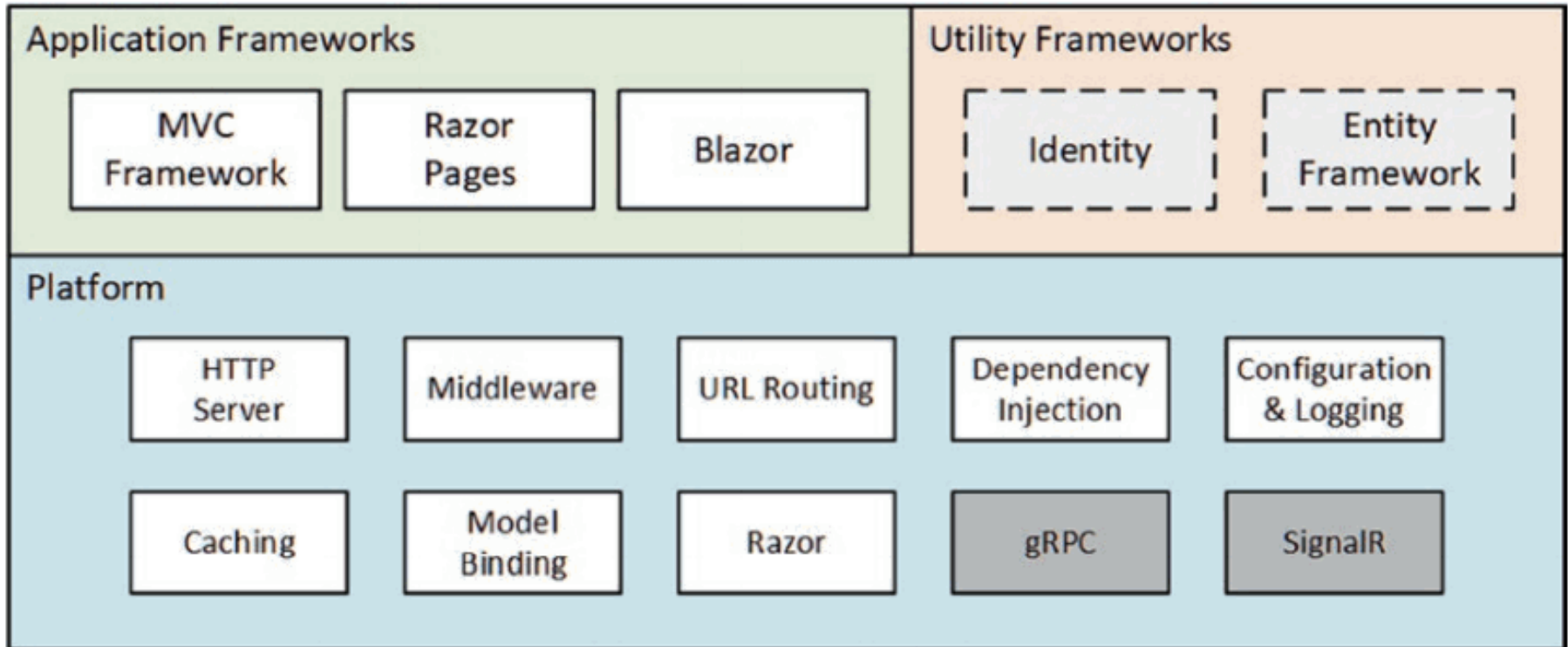
- Dùng để xây dựng giao diện web (Web UI) và các API Web.
- Tích hợp framework ở phía client.
- Hệ thống cấu hình sẵn có trên đám mây.
- Hiệu suất cao.
- Có khả năng chạy trên nhiều host như IIS, Nginx, Apache, Docker hoặc self-host
- Side-by-side App với .NET Core cho phép ứng dụng chạy đồng thời trên nhiều phiên bản.
- Chạy trên đa nền tảng: Windows, MacOS, Linux.
- Mã nguồn mở và có cộng đồng phát triển lớn.
- Được cung cấp dưới dạng các gói NuGet. Khi xây dựng ứng dụng, chỉ cần cài đặt các gói cần thiết.

Giới thiệu ASP.NET Core

ASP.NET Core Architecture



Giới thiệu ASP.NET Core



Cấu trúc của ASP.NET Core

Giới thiệu ASP.NET Core

- 📖 **Application Frameworks** chứa những thành phần tương đối quen thuộc như MVC Framework, Razor Pages hay Blazor. Đây là những framework giúp xây dựng các dạng khác nhau của ứng dụng web.
- 📖 **Khối Utility Frameworks** chứa Identity và Entity Framework. Khối này chứa những framework hỗ trợ cho ứng dụng, bao gồm bảo mật và cơ sở dữ liệu.
- 📖 **Khối Platform** là những gì tạo nên nền tảng chung nhất mà mọi loại ứng dụng Asp.net Core đều cần sử dụng đến.

Giới thiệu ASP.NET Core



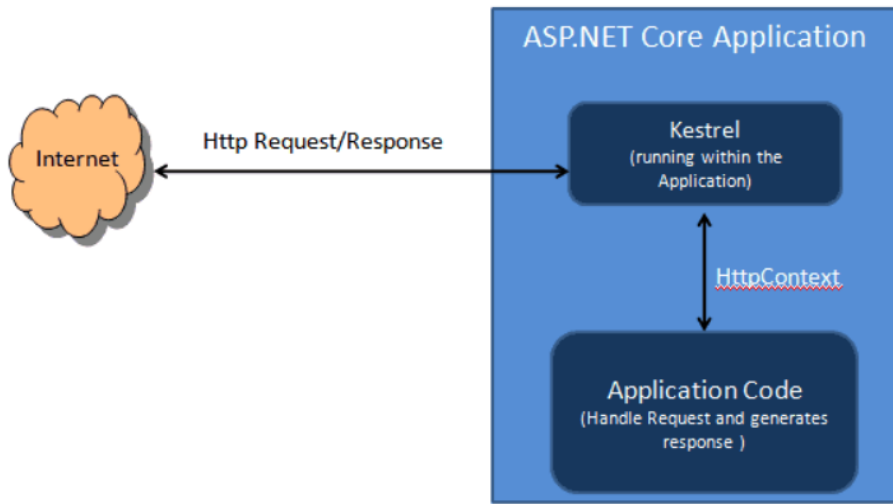
Kestrel

- 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.
- 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. 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.
- 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`.

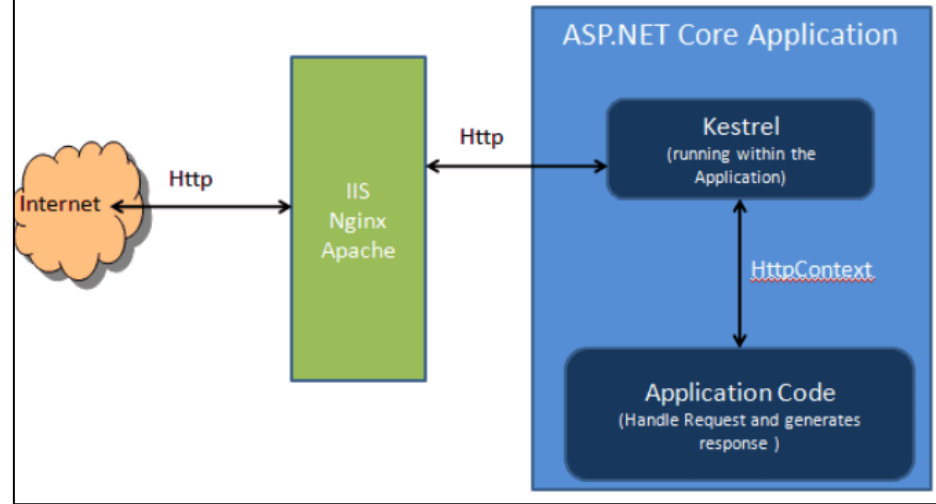
Giới thiệu ASP.NET Core

- 📖 **Kestrel**
 - Có 2 cách để sử dụng Kestrel
 - Tự host (Self Hosting)
 - Đằng sau một Web server khác

Self Hosting of ASP.NET Core Application



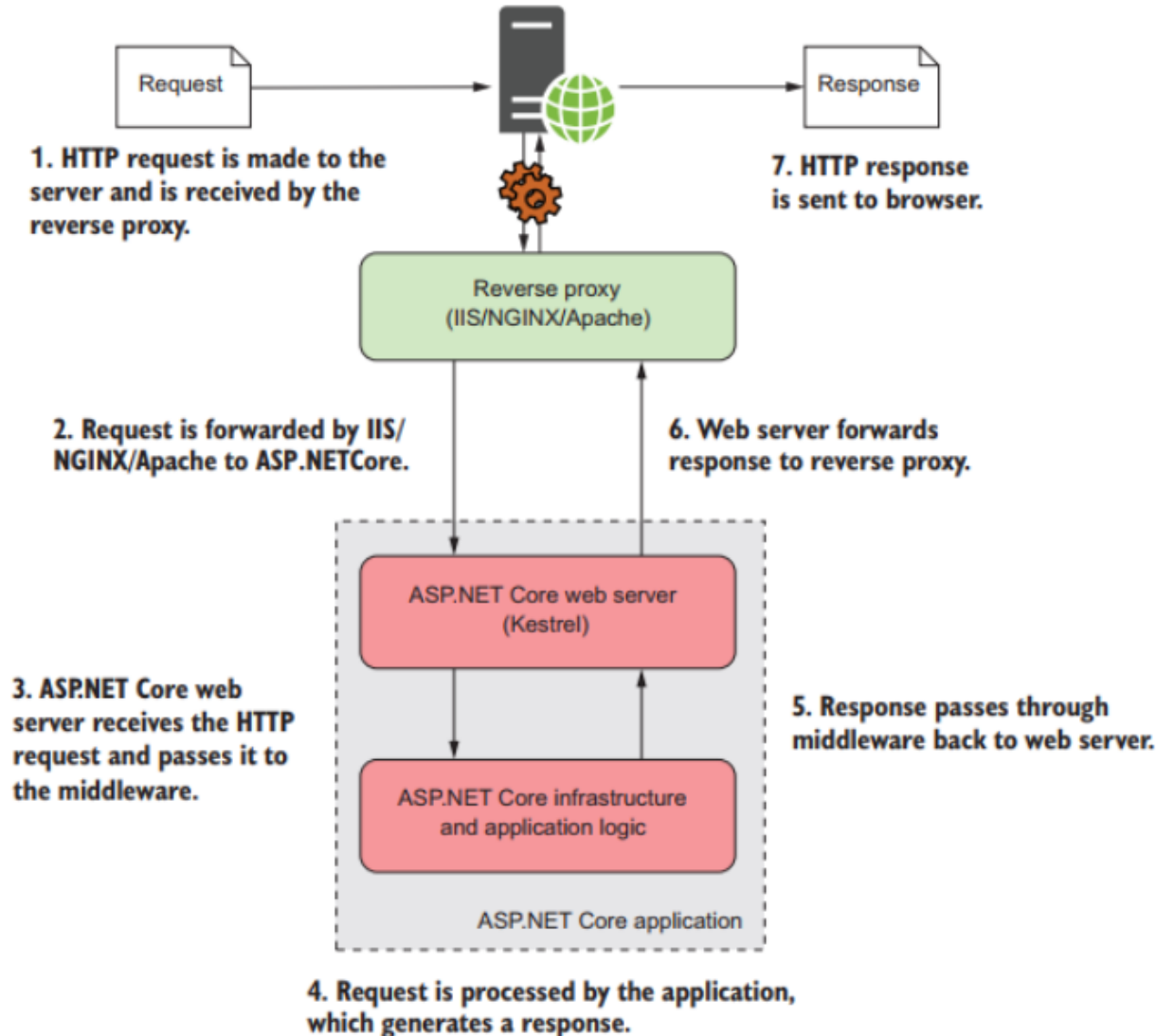
Hosting Model of ASP.NET Core Application



Giới thiệu ASP.NET Core



Mô hình hoạt động



Giới thiệu ASP.NET Core

📖 **Reverse proxy** chịu trách nhiệm tương tác trực tiếp với client (trình duyệt hoặc chương trình desktop/mobile) qua HTTP. Nói theo cách khác, trình duyệt nhìn thấy reverse proxy như trong mô hình web thông thường. Tuy nhiên, reverse proxy không xử lý truy vấn mà chuyển tiếp truy vấn cho Kestrel và nhận lại kết quả từ Kestrel. Mô hình triển khai này đem đến ưu điểm về tính bảo mật và hiệu suất.

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

Nội dung

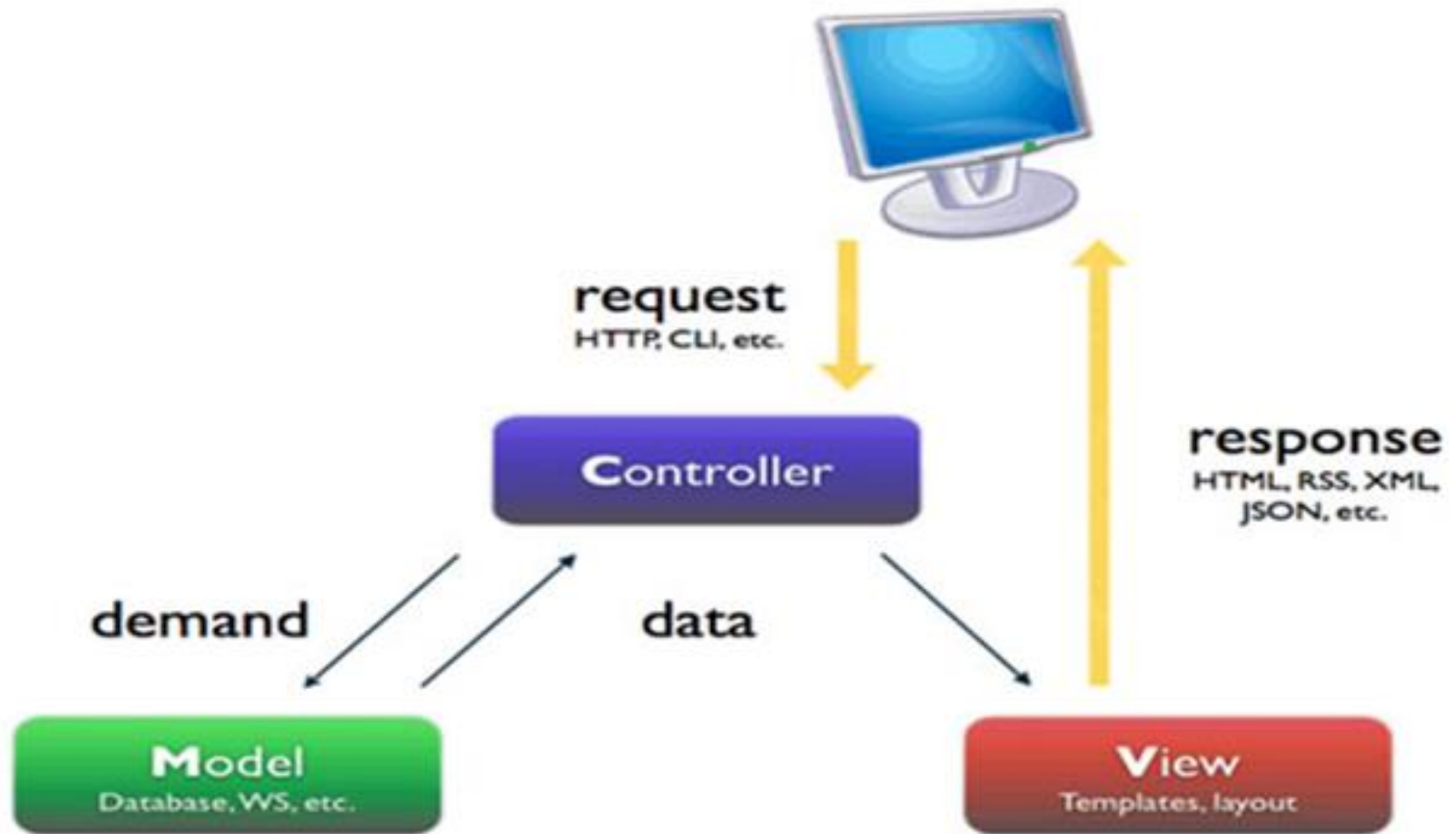


ASP.NET Core MVC

ASP.NET Core MVC

- 📖 Là framework cho việc xây dựng ứng dụng web.
- 📖 Sử dụng mô hình MVC để thiết kế và xây dựng web.
- 📖 MVC (Model – View - Controller) là mô hình tổ chức chương trình thành 3 thành phần khác nhau Model, View và Controller. Mỗi thành phần có một nhiệm vụ riêng biệt và độc lập với các thành phần khác.
 - **Model:** là thành phần chứa phương thức xử lý, truy xuất dữ liệu, hàm xử lý, chứa các đối tượng mô tả dữ liệu như các class và lưu trữ dữ liệu xuống hệ quản trị CSDL.
 - **View:** là thành phần nhận và hiển thị thông tin, tương tác với người dùng thông qua giao diện và gửi yêu cầu đến controller và nhận lại phản hồi từ controller.
 - **Controller:** là thành phần xử lý và điều hướng các hành động từ client; là thành phần kết nối giữa View và Model

ASP.NET Core MVC



ASP.NET Core MVC

📖 Triết lý **chia để trị** có nghĩa là mỗi thành phần trong ứng dụng chỉ nên có một trách nhiệm cụ thể. Chúng sẽ độc lập với các thành phần khác nhiều nhất có thể. Nói cách khác, các thành phần nên giảm bớt sự phụ thuộc vào nhau. Ứng dụng được xây dựng dựa trên triết lý này sẽ dễ dàng kiểm thử, bảo trì và mở rộng.

ASP.NET Core MVC

 Cung cấp các tính năng dựa trên mô hình xây dựng website động:

- *Routing*: xác định URL và điều hướng thông tin tương ứng với request do người dùng đưa vào. Cấu hình Routing được lưu trữ trong `RoutTable`.
- *Model binding*: dùng để chuyển đổi dữ liệu từ yêu cầu của client (form data, route data, query string, parameters, HTTP headers) thành đối tượng để controller xử lý. Tự ánh xạ dữ liệu từ HTTP Request tới tham số của method action.
- *Model validation*: ràng buộc dữ liệu cho các thuộc tính trong model, thuộc tính sẽ được kiểm tra ở client trước khi được gửi về server.
- *Dependency injection*: controller có thể gửi yêu cầu đến các service thông qua hàm khởi tạo của chúng.

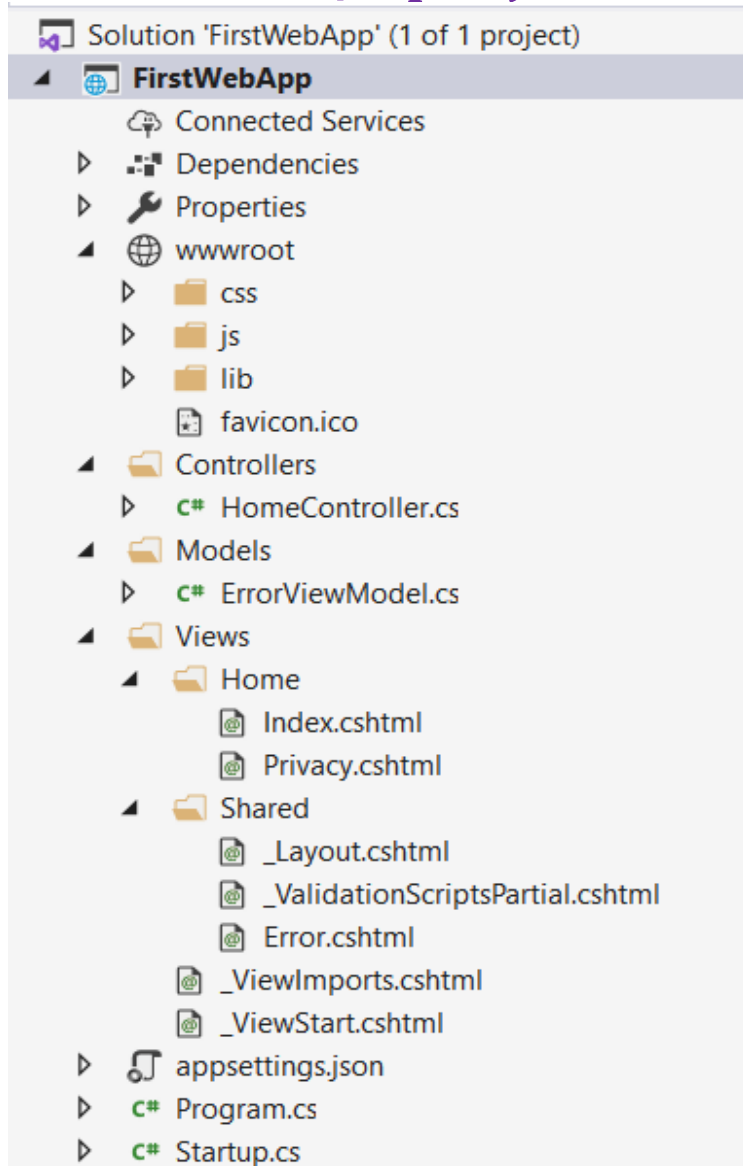
ASP.NET Core MVC

- *Filters*: cho phép kích hoạt trước hoặc sau các action của controller. VD: Phân quyền
- *Areas*: cho phép chia các controller, view và model thành từng nhóm riêng biệt nhau. VD: nhóm các controller cho việc xử lý quản lý Administrator.
- *Web APIs*: hỗ trợ cho xây dựng các API.
- *Testability*: hỗ trợ việc kiểm tra ứng dụng được xây bằng framework sử dụng interfaces và dependency injection dễ dàng.
- *Razor view engine*: được dùng để render code theo ngôn ngữ Razor để tạo View
- *Tag Helpers*: cho phép code ở server side tham gia vào việc tạo và hiển thị các phần tử HTML.

ASP.NET Core MVC



Cấu trúc thư mục project MVC trong VS



ASP.NET Core MVC


- 📖 **Connected Services:** danh sách các dịch vụ online mà project sử dụng
- 📖 **Dependencies:** danh sách gói thư viện NuGet được cài đặt và sử dụng trong project.
- 📖 **Properties:** cấu hình của project, nằm trong file json launchSetting.json, dùng để chạy ứng dụng từ VS hoặc .NET Core CLI.
- 📖 **appsettings.json:** thông tin cấu hình hoạt động của ứng dụng: connection string, biến môi trường, tham số dòng lệnh...
- 📖 **Program.cs:** file chứa class Program, là file cấu hình nền tảng (infrastructure) của ứng dụng, chứa entry point của ứng dụng và các cấu hình trong file hầu như không thay đổi trong mỗi project.

ASP.NET Core MVC

 **wwwroot:** chứa các file css, javascript, các thư viện, hình ảnh. Các file phải nằm trong thư mục này thì mới hoạt động.


Gồm các thư mục:

- css: chứa các file css.
- js: chứa các file javascript
- lib: các thư viện cho javascript như jQuery bootstrap
- favicon.ico: file biểu tượng của site

 **Startup.cs:** chứa class với các phương thức cấu hình cho hoạt động của ứng dụng, vd: sử dụng middleware nào, thứ tự các middleware trong pipeline, các cấu hình sử dụng service, Dependency Injection, Logging hoặc routing controller

ASP.NET Core MVC

 **Controllers:** chứa các file **.cs** của controller; tương ứng với mỗi controller sẽ có một thư mục view đi kèm.

 **Views:** chứa các file **.cshtml** của view; ứng với mỗi controller sẽ là một folder trong view. Các phương thức có **return View()** trong controller sẽ tương ứng với một file trong view.

- Thư mục **Shared** là nơi chứa các thành phần giao diện chung cho trang web: `_Layout.cshtml`, `_ValidationScriptsPartial.cshtml` và `Error.cshtml`

 **Models:** chứa các file **.cs** của model; là các lớp thể hiện dữ liệu của ứng dụng.

ASP.NET Core MVC



File Controller

```
HomeController.cs* - X
FirstWebApp FirstWebApp.Controllers.HomeController
1  using ...
9
10 namespace FirstWebApp.Controllers
11 {
12     3 references
13     public class HomeController : Controller
14     {
15         private readonly ILogger<HomeController> _logger;
16
17         0 references
18         public HomeController(ILogger<HomeController> logger)
19         {
20             _logger = logger;
21
22         0 references
23         public IActionResult Index()
24         {
25             return View();
26
27         0 references
28         public IActionResult Privacy()
29         {
30             return View();
31
32         [ResponseCache(Duration = 0, Location = ResponseCacheLocation.None, NoStore = true)]
33         0 references
34         public IActionResult Error()
35         {
36             return View(new ErrorViewModel { RequestId = Activity.Current?.Id ?? HttpContext.TraceIdentifier });
37         }
38     }
39 }
```

ASP.NET Core MVC



File View

Index.cshtml* ➔ ✕

```
1  @{
2      ViewData["Title"] = "Home Page";
3  }
4
5  <div class="text-center">
6      <h1 class="display-4">Welcome</h1>
7      <p>Learn about <a href="https://docs.microsoft.com/aspnet/core">
8          building Web apps with ASP.NET Core</a>.</p>
9  </div>
```

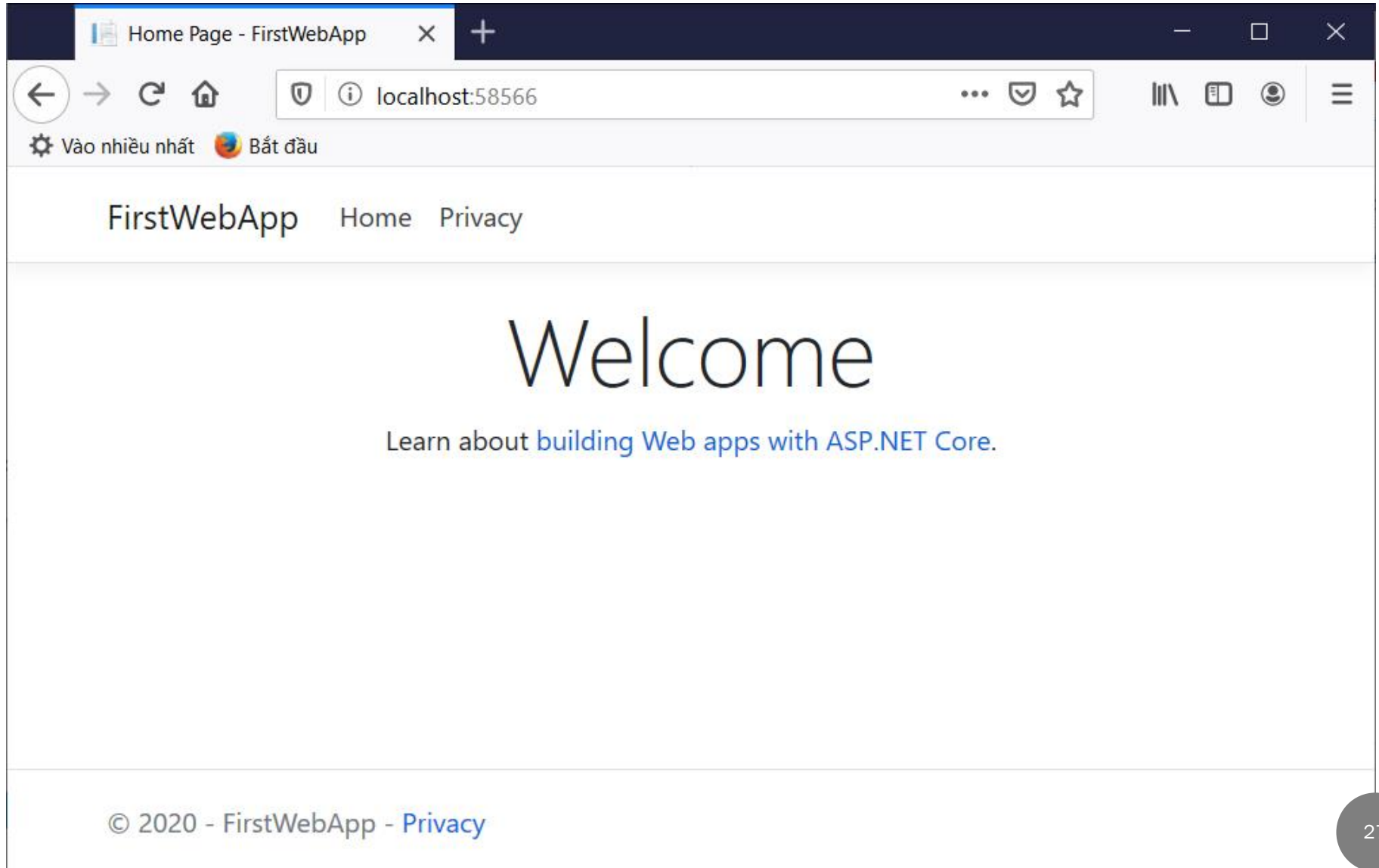
Privacy.cshtml ➔ ✕

```
1  @{
2      ViewData["Title"] = "Privacy Policy";
3  }
4  <h1>@ViewData["Title"]</h1>
5
6  <p>Use this page to detail your site's privacy policy.</p>
```

ASP.NET Core MVC



Kết quả:



Cấu hình file Startup.cs

- 📖 MVC thực thi controller (và các phương thức của chúng) dựa vào đường dẫn URL.
- 📖 Đường dẫn URL xác định phương thức thực thi trong controller với dạng:

- localhost:{Port}/[Controller]/[ActionName]/[Parameters]

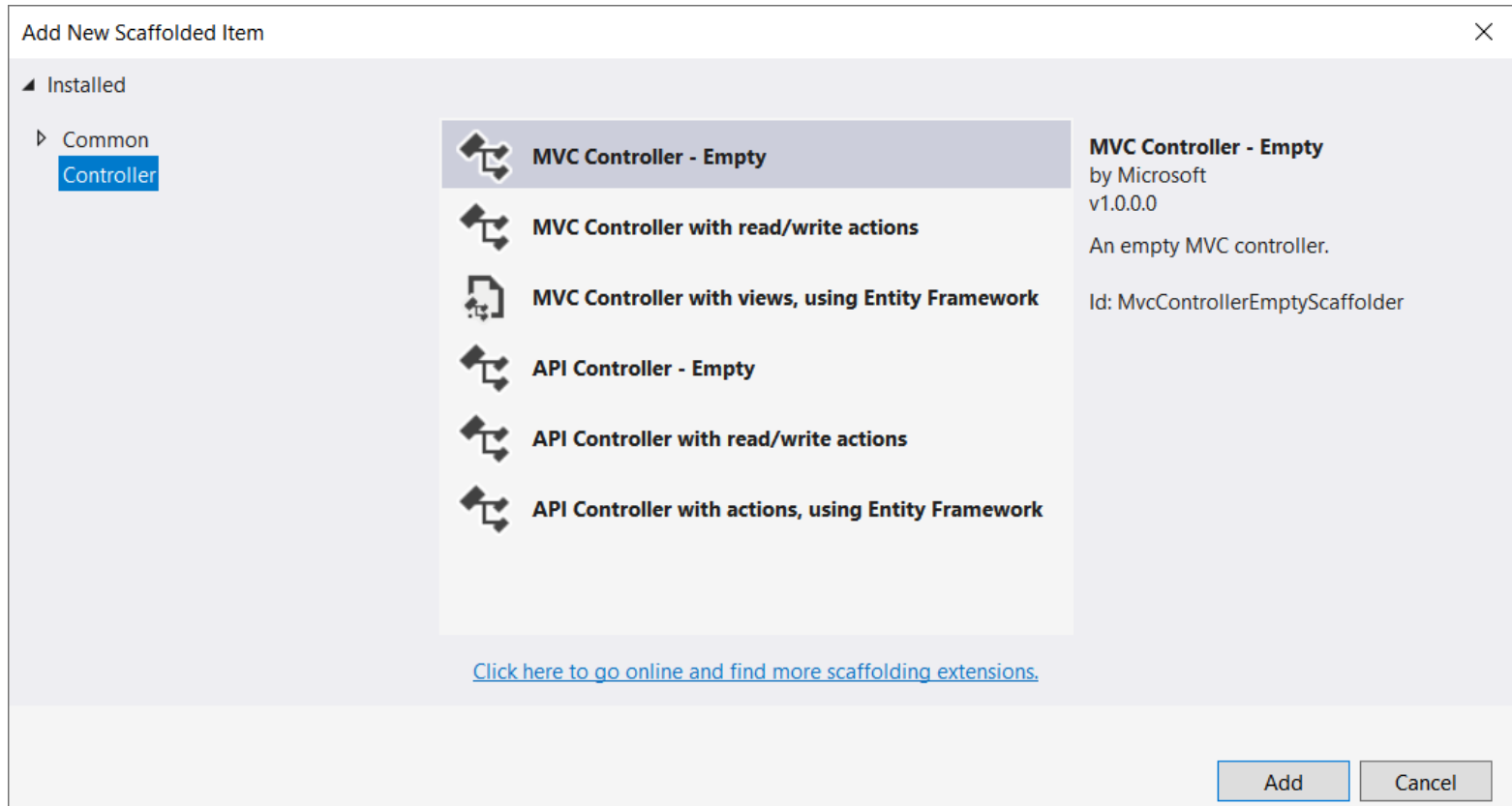
- 📖 Cấu hình đường dẫn controller trong phương thức **Configure** của file Startup.cs.

```
app.UseEndpoints(endpoints =>
{
    endpoints.MapControllerRoute(
        name: "default",
        pattern: "{controller=Home}/{action=Index}/{id?}");
});
```

- Mặc định là controller **Home** và phương thức **Index**. Tham số **id?** là tham số không bắt buộc phải xác định rõ.

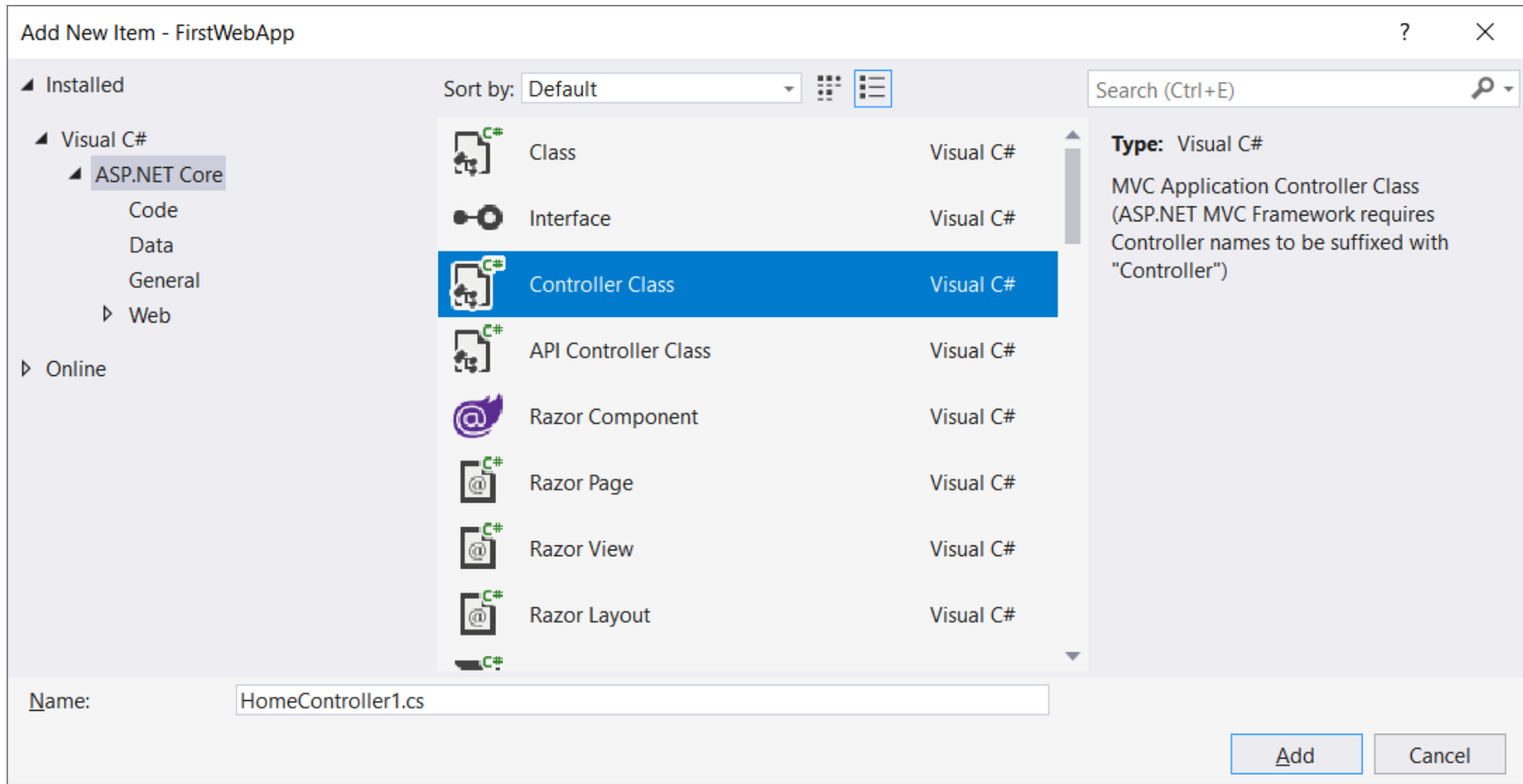
Thêm controller

- 📖 Nhấp chuột phải tại thư mục Controllers > Add > Controller hoặc New Item...
 - Tại cửa sổ Add Scaffold chọn MVC Controller – Empty



Thêm controller

- Với Controllers > Add > New Item... chọn Controller Class



Thêm controller

Thêm đoạn code sau vào Controller mới tạo

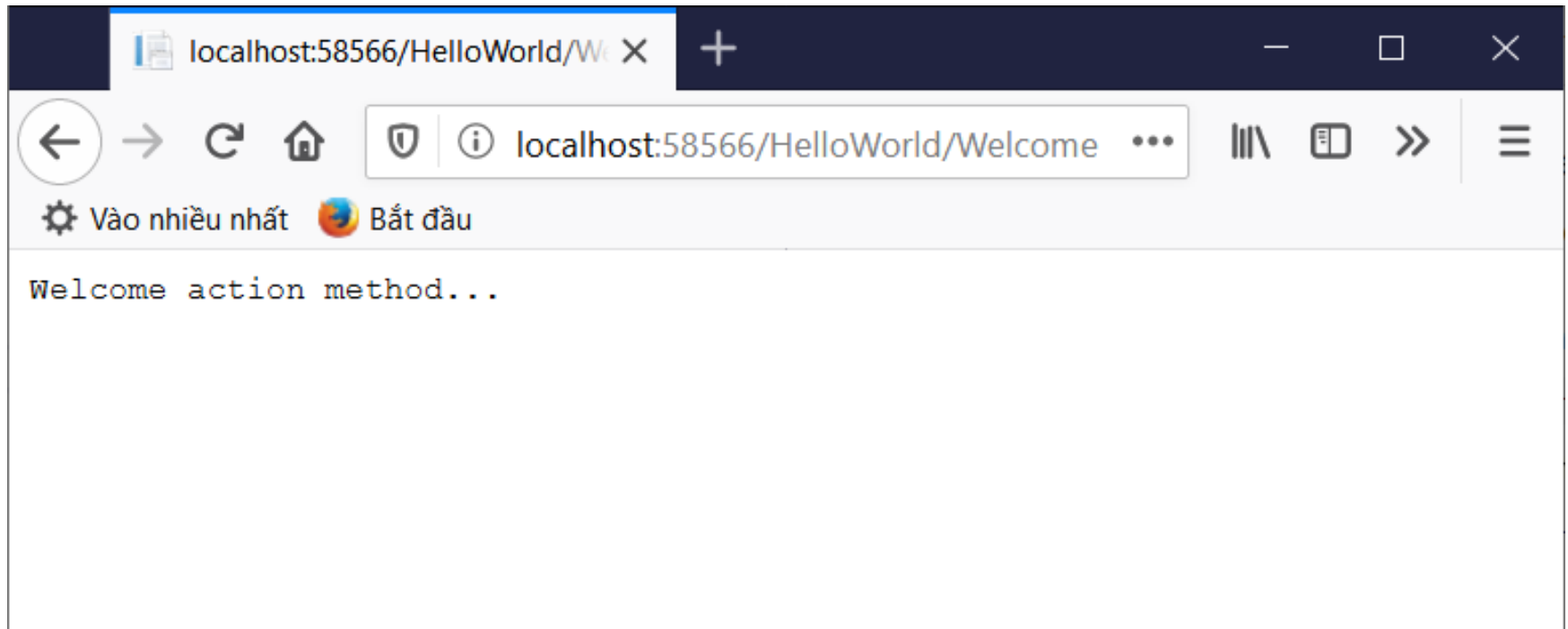
```
public class HelloWorldController : Controller
{
    // GET: HelloWorld/
    0 references
    public string Index()
    {
        return "Hello world default action";
    }

    // GET: HelloWorld/Welcome
    0 references
    public string Welcome()
    {
        return "Welcome action method...";
    }
}
```

Thêm controller

- Tất cả các phương thức public trong controller đều có thể được gọi như là một trang web.
- Đường dẫn URL để gọi một controller gồm tên lớp controller và tên phương thức cần gọi:

<http://localhost:58566/HelloWorld/Welcome>

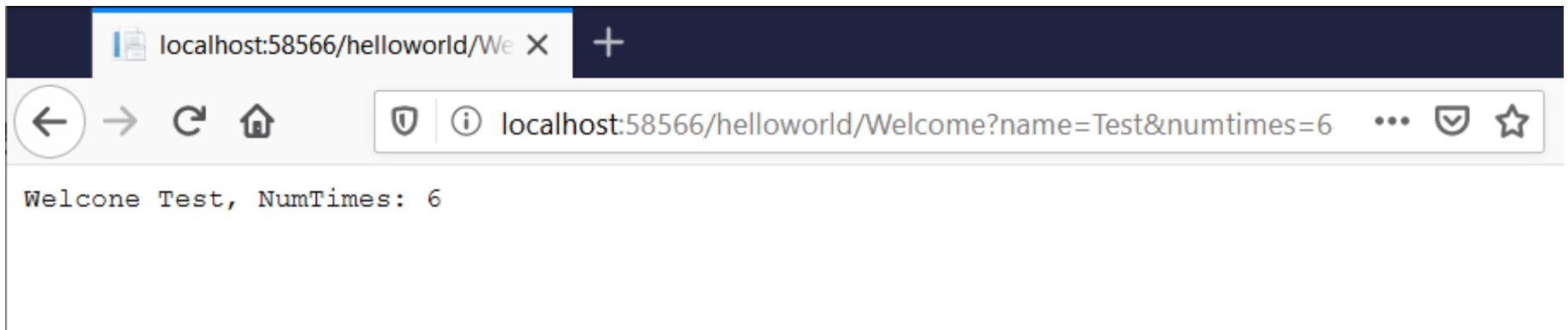


Thêm controller

 Controller với tham số truyền vào:

```
public string Welcome(string name, int numTimes = 1)
{
    return HtmlEncoder.Default.Encode($"Welcone {name}, NumTimes: {numTimes}");
}
```

<http://localhost:58566/helloworld/Welcome?name=Test&numtimes=6>

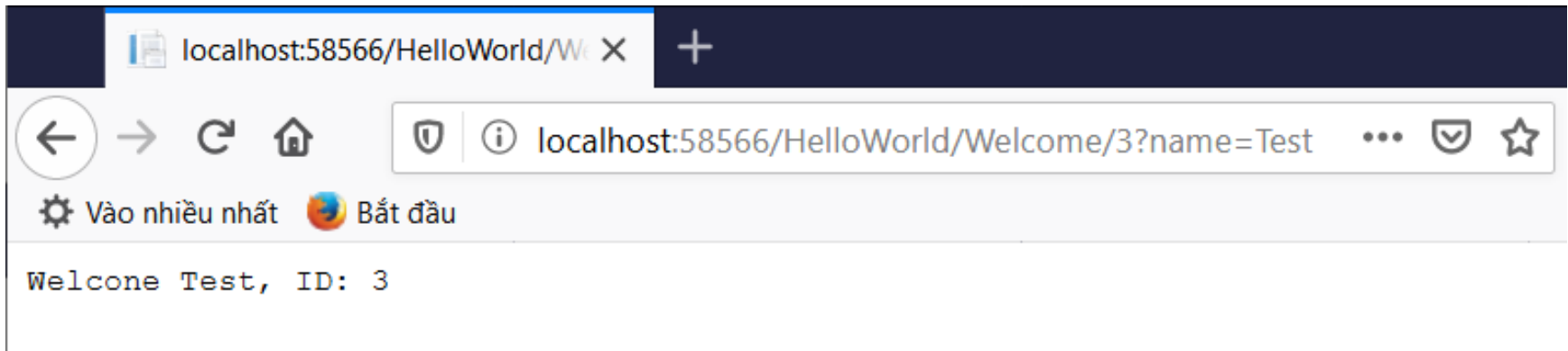


Thêm controller

📖 Controller với tham số truyền vào có tham số ID:

```
public string Welcome(string name, int ID = 1)
{
    return HtmlEncoder.Default.Encode($"Welcone {name}, ID: {ID}");
}
```

<http://localhost:58566/HelloWorld/Welcome/3?name=Test>



Thêm view

- 📖 Tạo template view sử dụng Razor View.
- 📖 Razor View Engine là ngôn ngữ ngắn gọn, rõ ràng và hữu ích cho việc tạo ra giao diện của ASP.NET Core MVC.
- 📖 Razor View template có định dạng file **.cshtml**, cung cấp cách tạo ra HTML bằng C#.
- 📖 Để gọi view tạo bằng Razor, phương thức trong controller (action method) phải gọi phương thức **View()** và trả về kiểu **ActionResult**.
- 📖 Nhấp chuột phải tại thư mục Views > Add > New folder: đặt tên thư mục theo tên controller.
- 📖 Nhấp chuột phải tại thư mục vừa tạo Add > New Item > Razor View.

Thêm view

 Bổ sung code như hình:

```
Index.cshtml  + X
1      @{
2          ViewData["Title"] = "Index";
3      }
4      <h2>Index</h2>
5      <p> Hello from View Razor</p>
-
```

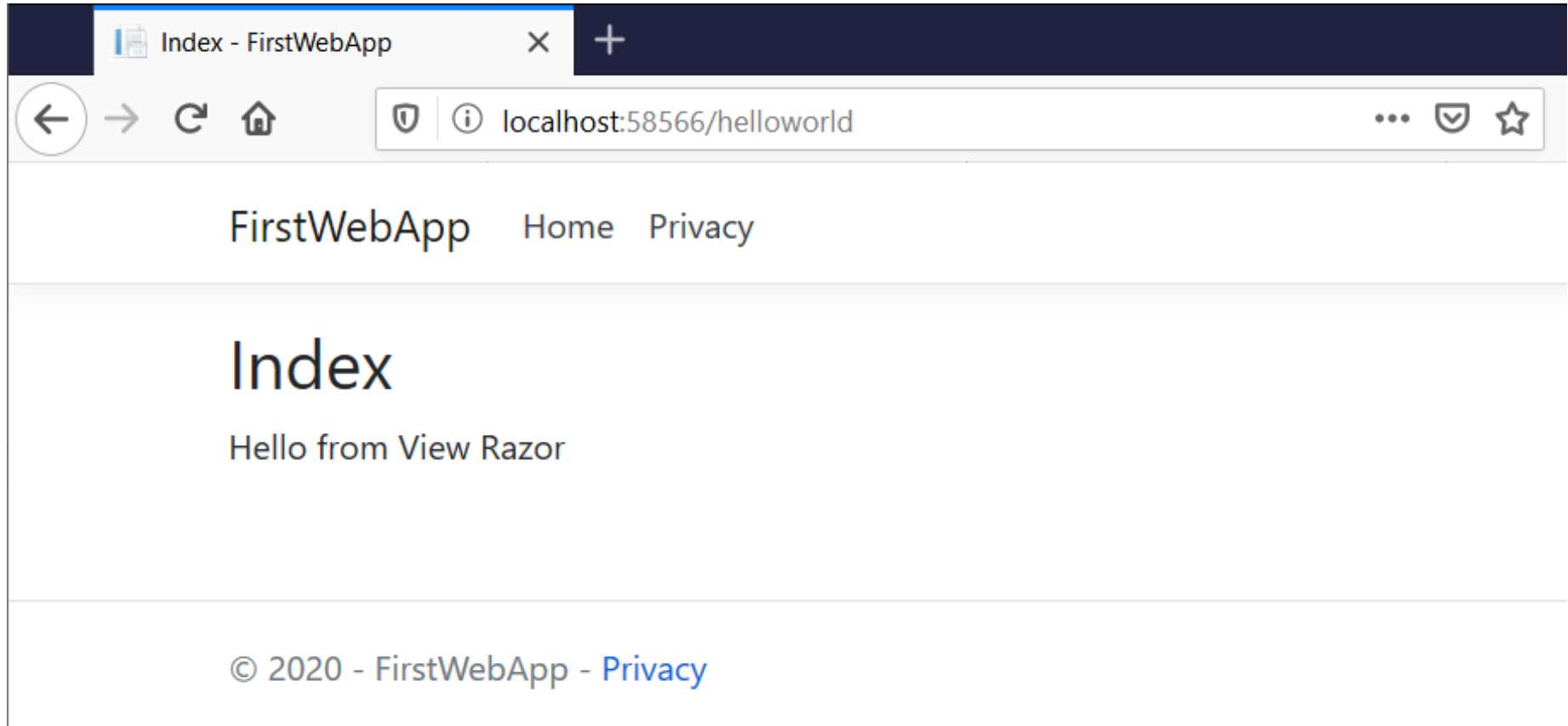
 Chỉ sửa phương thức Index trong controller

```
public IActionResult Index()
{
    return View();
}
```

Thêm view



Kết quả:



Thêm view

- 📖 Lưu trữ và truyền dữ liệu từ controller tới view bằng **ViewData**.
- 📖 **ViewData** là một kiểu đối tượng động (không cần định nghĩa trước kiểu dữ liệu) và là kiểu dictionary (lưu trữ theo dạng cặp key – value).
- 📖 Được Razor Pages tạo sẵn và có thể truy cập từ bất kỳ đâu trong ứng dụng Razor.
- 📖 Khi gọi phương thức **View()** trong Controller action **ViewData** sẽ tự động gán vào view.
- 📖 Lưu trữ theo kiểu string thì lấy và dùng trực tiếp ở view

```
ViewData["Name"] = "Test";
```

 - Truy xuất ở View: `@ViewData["Message"]`

Thêm view

📖 Lưu trữ theo kiểu object thì phải ép kiểu sang kiểu dữ liệu xác định

```
ViewData["Product"] = new Product()
{
    ProductID = 1,
    Name = "ABC",
    Brand = "XYZ",
    Price = 1000
};
```

- Truy xuất ở View phải chuyển sang kiểu Product.

```
@{ var product = ViewData["Product"] as Product; }
@product.ProductID<br>
@product.Name<br>
@product.Brand<br>
@product.Price<br>
```

Thêm view

- Chỉnh sửa phương thức Welcome trong controller

```
public IActionResult Welcome(string name, int numTimes = 1)
{
    ViewData["Name"] = name;
    ViewData["NumTimes"] = numTimes;
    return View();
}
```

- View Welcome.cshtml

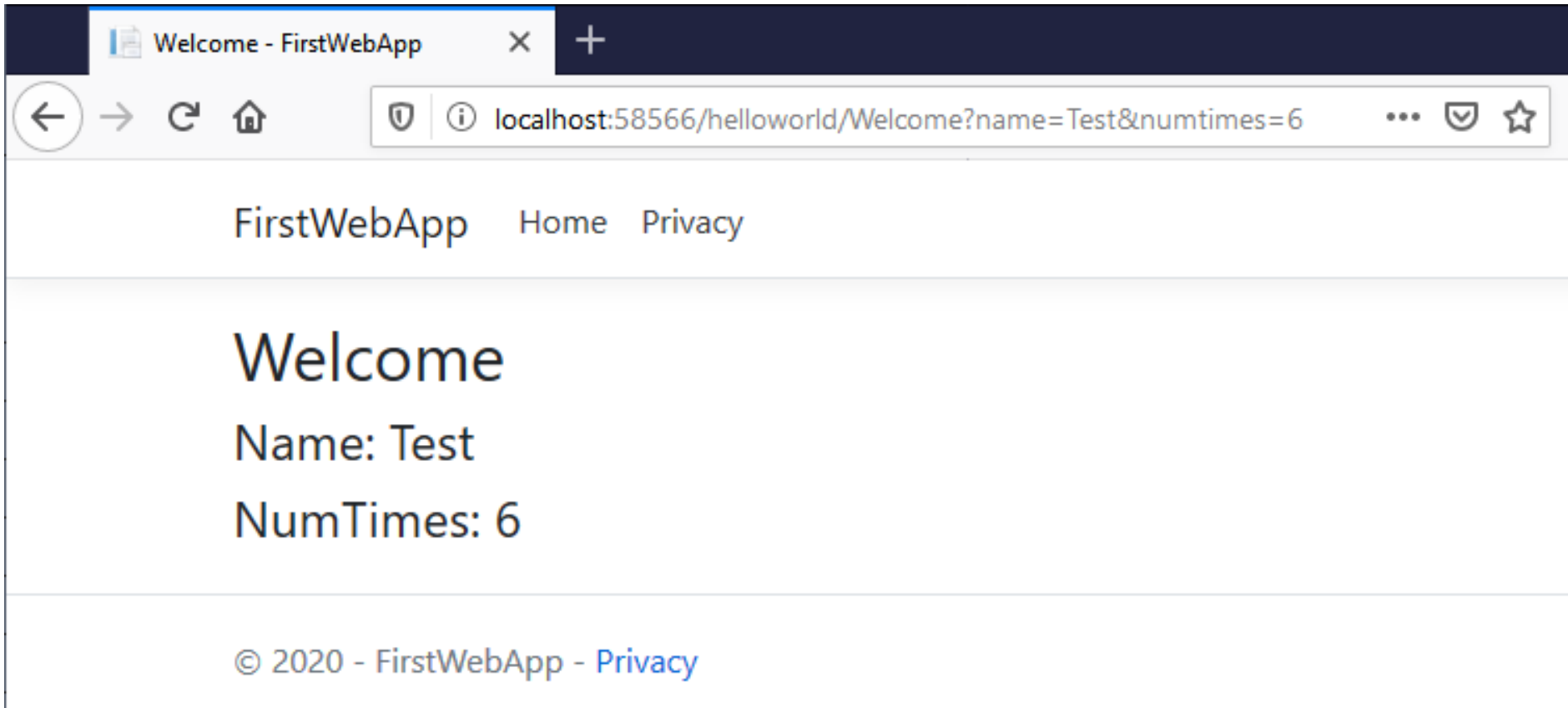
```
@{
    ViewData["Title"] = "Welcome";
}
<h2>Welcome</h2>

<h4>Name: @ViewData["Name"]</h4>
<h4>NumTimes: @ViewData["NumTimes"]</h4>
```


Thêm view



Kết quả:



Razor Page: Layout

- Trang web thường chia làm năm phần: header, footer, navigation menu, sidebar và content. Trừ content, các phần còn lại thông thường được thống nhất chung trong nhiều trang được gọi là layout.
- Razor page cung cấp cơ chế Layout để thực hiện việc này.
- Trong ASP.NET Core MVC là file **_Layout.cshtml** trong thư mục View > Shared.
- Phương thức **@RenderBody()** là phương thức để xuất nội dung của bất kỳ page nào sử dụng layout.
- Sử dụng:

```
@{  
    Layout = "_Layout";  
}
```

Razor Page: Layout

_Layout.cshtml

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <title>@ViewData["Title"] - FirstWebApp</title>
  <link rel="stylesheet" href="~/lib/bootstrap/dist/css/bootstrap.min.css" />
  <link rel="stylesheet" href="~/css/site.css" />
</head>
<body>
  <header>
    <nav class="navbar navbar-expand-sm navbar-toggleable-sm navbar-light bg-white border-bottom box-shadow mb-3">
      <div class="container">
        <a class="navbar-brand" asp-area="" asp-controller="Home" asp-action="Index">FirstWebApp</a>
        <button class="navbar-toggler" type="button" data-toggle="collapse" data-target=".navbar-collapse" aria-controls="navbarSupportedContent"
          aria-expanded="false" aria-label="Toggle navigation">
          <span class="navbar-toggler-icon"></span>
        </button>
        <div class="navbar-collapse collapse d-sm-inline-flex flex-sm-row-reverse">
          <ul class="navbar-nav flex-grow-1">
            <li class="nav-item">
              <a class="nav-link text-dark" asp-area="" asp-controller="Home" asp-action="Index">Home</a>
            </li>
            <li class="nav-item">
              <a class="nav-link text-dark" asp-area="" asp-controller="Home" asp-action="Privacy">Privacy</a>
            </li>
          </ul>
        </div>
      </div>
    </nav>
  </header>
  <div class="container">
    <main role="main" class="pb-3">
      @RenderBody()
    </main>
  </div>

  <footer class="border-top footer text-muted">
    <div class="container">
      &copy; 2020 - FirstWebApp - <a asp-area="" asp-controller="Home" asp-action="Privacy">Privacy</a>
    </div>
  </footer>
  <script src="~/lib/jquery/dist/jquery.min.js"></script>
  <script src="~/lib/bootstrap/dist/js/bootstrap.bundle.min.js"></script>
  <script src="~/js/site.js" asp-append-version="true"></script>
  @RenderSection("Scripts", required: false)
</body>
</html>
```

Razor Page: ViewStart

- 📖 Sử dụng Layout thì đầu mỗi trang đều phải khai báo layout cần sử dụng.
- 📖 Sử dụng ViewStart để đặt layout mặc định cho nhiều trang
- 📖 Trong ASP.NET Core MVC là file **_ViewStart.cshtml** trong thư mục View.
- 📖 _ViewStart.cshtml có tác dụng trong thư mục chứa nó và các thư mục con. Nếu thư mục con cũng có file _ViewStart thì các file trong thư mục con sẽ chịu tác động của file _ViewStart này.

```
_ViewStart.cshtml  ➦ X
1      @{
2          Layout = "_Layout";
3      }
```

Razor View Engine

- 📖 View Engine: kết hợp ngôn ngữ lập trình với ngôn ngữ HTML để xuất ra mã HTML:
 - Hoạt động như một chương trình dịch (compiler hoặc interpreter).
 - Khi Controller gọi Action Method và kiểu trả về là một Action Result; cụ thể là ViewResult.
 - ViewResult được cung cấp bởi View Engine sẽ sinh ra code HTML.
- 📖 Razor View Engine: là View Engine mặc định của ASP.NET Core; lấy mã Razor trong file view chuyển thành HTML response.

Razor

- ❏ Razor là loại ngôn ngữ/cú pháp đánh dấu (markup/language syntax) sử dụng Razor View Engine của ASP.NET Core
- ❏ Cú pháp Razor kết hợp C# và HTML với nhiệm vụ xác định (đánh dấu) code C# và code HTML trong file để view engine xử lý.
- ❏ Tạo ra các trang web động bằng HTML.
- ❏ Gần tương tự với ngôn ngữ ASP hoặc PHP.
- ❏ Là ngôn ngữ cho các framework trong ASP.NET Core: Razor Pages, MVC và Blazor.
- ❏ File Razor có đuôi **.cshtml**.
- ❏ Khi duyệt file .cshtml, mã C# sẽ được thực thi để tạo dữ liệu HTML và gửi về client.

Cú pháp Razor

- Trong file .cshtml (razor file/page) tồn tại song song hai ngôn ngữ C# và HTML. Vì vậy, cú pháp Razor có nhiệm vụ xác định cụ thể code C# và HTML để View Engine dịch và xử lý.
- Razor sử dụng ký tự @ để chuyển cách viết từ HTML sang C#. Có 2 cách khai báo:
 - Razor expression*: bắt đầu bằng @ và theo sau là code C#, chèn chung với code HTML

```
<h4>@DateTime.Now.Year</h4>
```
 - Khối lệnh Razor*: bắt đầu bằng @ và nằm trong cặp dấu {}; có thể được dùng để thao tác Model, khai báo biến, đặt thuộc tính của View, không nên sử dụng cho việc xử lý logic.

```
@{  
    var message = "Welcome";  
    var weekDay = DateTime.Now.DayOfWeek;  
}
```

Một số lưu ý của Razor

- 📖 Ngôn ngữ mặc định trong file cshtml là HTML.
- 📖 Không có đánh dấu đặc biệt thì Razor View Engine sẽ xem đây hoàn toàn là HTML và gửi khối HTML về client
- 📖 Ký tự @ để Razor xác định đây là code C# và sẽ chuyển sang chế độ dịch C#.
- 📖 Kết quả biên dịch cuối cùng của file cshtml là mã HTML.
- 📖 Viết 2 lần ký tự @ (@@ - @ escape) để biểu diễn cho ký tự @ trong Razor.
- 📖 Razor tự phân biệt được @ trong email.
- 📖 Cách ghi chú thích: vùng code C# chú thích kiểu C#, vùng code HTML chú thích kiểu HTML.
- 📖 Ký tự @ đi cùng một số từ đặc biệt: @page, @model, @using (gọi là directive) được Razor sử dụng riêng

Biểu thức Razor

- ❏ Biểu thức Razor (Razor expression): bắt đầu bằng @ và theo sau là biểu thức của C#, chèn chung với code HTML.
- ❏ Được dùng để chèn giá trị (tính bằng C#) vào vị trí tương ứng với HTML.
- ❏ VD: `<h4>@message</h4>`
- ❏ Biểu thức C# (expression) là những lệnh có trả về kết quả: phương thức, kết quả thực thi các phép toán, giá trị của biến; phương thức không trả về kết quả (câu lệnh – statement) không được dùng làm biểu thức Razor.
- ❏ Có 2 dạng biểu thức Razor:
 - Biểu thức ẩn (Implicit expression)
 - Biểu thức rõ (Explicit expression)

Biểu thức Razor



Biểu thức ẩn (Implicit expression):

- Không cho phép chứa dấu cách (khoảng trống) hoặc ký tự đặc biệt (nhầm lẫn với code HTML xung quanh). VD: không thể chứa kiểu generics.
- Không thể là các phép toán thông thường (+, -, *, /)
- Là dạng đơn giản hóa, tiện lợi khi tích hợp biểu thức C# với HTML
- VD:

```
<p>Hello @name</p>
```

```
<p>@DateTime.Now.DayOfWeek</p>
```

```
<p>Name : @cust.name</p>
```

```
<p>Address : @cust.address</p>
```

Biểu thức Razor



Biểu thức rõ (Explicit expression):

- Đặt trong cặp dấu ().
- Là dạng đầy đủ khi tích hợp biểu thức C# và HTML
- VD:

`<p>ISBN: @(No)</p>`

`<p>1 + 1: @(1 + 1)</p>`

`<p>Diện tích = @(Math.PI * r * r)</p>`

Khối lệnh Razor

 Khối lệnh (khối code – Razor code block): là những câu lệnh C# nằm trong cặp dấu {} và bắt đầu bằng ký tự @.

@{

```
var message = "Welcome";  
int Cong(int a, int b)  
{  
    return a + b;  
}
```

}

 Đặc điểm:

- Được dùng để khai báo, khởi tạo biến và khai báo hàm cục bộ.
- Các biến và hàm tạo trong khối lệnh có thể được dùng làm biểu thức Razor hoặc trong khối lệnh khác trong cùng trang.
- Không giới hạn khối lệnh và vị trí viết trên một trang.
- Hàm có thể được gọi trong khối lệnh khác trước khi được khai báo.
- Biến phải được khai báo trước khi sử dụng.
- Chú thích khối lệnh dùng cặp: @*.....*@

Khối lệnh Razor

- Trong khối lệnh Razor có thể có code HTML. Razor sẽ tự động chuyển đổi từ C# sang HTML.

```
@{  
    void WelcomeMessage(string name)  
    {  
        <div class="text-center">  
            <h1 class="display-4">Hello @name. Welcome to Razor</h1>  
        </div>  
    }  
}
```

- Chuyển đổi chủ động: trong trường hợp không muốn sử dụng thẻ HTML trong khối lệnh Razor.
 - Do không sử dụng code HTML trong hàm, nên Razor sẽ không xuất ra HTML được.
 - Dùng “@:” để xuất giá trị trên một dòng.

```
void WelcomeMessage(string name)  
{  
    @:Hello @name.  
    @:Welcome to Razor  
}
```

Khối lệnh Razor

- Dùng thẻ `<text></text>` để bao quanh khối giá trị cần xuất trên nhiều dòng.

```
void WelcomeMessage(string name)
{
    <text>Hello @name.
    Welcome to Razor</text>
}
```

Cấu trúc điều khiển trong Razor

- 📖 Là những đoạn khối lệnh dùng để điều chỉnh việc phát sinh mã HTML như phân nhánh, lặp....
- 📖 Trong khối lệnh (C# là ngôn ngữ mặc định) thì viết câu lệnh theo cấu trúc điều khiển C# bình thường.
- 📖 Ngoài khối lệnh (HTML là ngôn ngữ mặc định) thì sử dụng kiểu viết markup của Razor.
- 📖 Các markup của Razor:
 - Cấu trúc điều kiện: @if - else if – else, @switch
 - Cấu trúc lặp: @for, @foreach, @while và @do...while
 - Cấu trúc @using
 - Cấu trúc @try – catch - finally

Cấu trúc điều kiện

📖 Cấu trúc điều kiện: **@if - else if – else:**

- Cấu trúc @if - else if – else là một khối lệnh.
- Chỉ cần viết @ ở trước if.
- Điều kiện logic viết theo ngôn ngữ C#

📖 VD:

```
@{ var today = DateTime.Now.DayOfWeek; }  
@if (today == DayOfWeek.Saturday || today == DayOfWeek.Sunday)  
{  
    <strong>Hôm nay cuối tuần, quấy thôi!!!</strong>  
}  
else  
{  
    <strong>Hôm nay ngày thường, đi chơi thôi!!!</strong>  
}
```


Cấu trúc điều kiện

Cấu trúc điều kiện: @switch

- Đặt ký tự @ vào trước cấu trúc switch của C#
- Mỗi case trong switch là một khối lệnh, có thể kết hợp C# và HTML

VD:

```
@switch (DateTime.Now.DayOfWeek)
{
    case DayOfWeek.Saturday:
        <strong>Hôm nay Thứ 7, ngủ thôi!!!</strong>
        break;
    case DayOfWeek.Sunday:
        <strong>Hôm nay Chủ Nhật, quấy thôi!!!</strong>
        break;
    default:
        <strong>Hôm nay ngày thường, đi chơi thôi!!!</strong>
        break;
}
```

Cấu trúc lặp

Cấu trúc lặp: @for, @foreach

- Đặt ký tự @ vào trước cấu trúc for hoặc foreach của C#
- Bên trong for là khối lệnh, có thể kết hợp C# và HTML

VD:

```
@for (int i = 0; i < 5; i++)  
{  
    <span> @i </span>  
}
```

```
@{  
    var custList = new []{  
        "ABC", "XYZ"  
    };  
}
```

```
@foreach (var cust in custList)  
{  
    <h4>cust</h4>  
}
```

Cấu trúc lặp

Cấu trúc lặp: @while, @do...while

- Đặt ký tự @ vào trước cấu trúc while hoặc do...while của C#
- Bên trong for là khối lệnh, có thể kết hợp C# và HTML

VD:

```
@{ var i = 0; }  
@while (i < 5){  
    <p>@i</p>  
    i++;  
}
```

```
@{ var i = 0; }  
@do  
{  
    var cust = custList[i++];  
    <h4>@cust</h4>  
}  
while (i < custList.Length);
```


Bắt và xử lý ngoại lệ

Cấu trúc: @try-catch-finally

- Đặt ký tự @ vào trước cấu trúc try-catch-finally của C#
- Mỗi nhánh là một code block.

VD:

```
@try
{
    throw new InvalidOperationException("Lỗi.");
}
catch (Exception ex)
{
    <p>The exception message: @ex.Message</p>
}
finally
{
    <p>The finally statement.</p>
}
```



Q & A

Giảng viên: Tạ Việt Phương
E-mail: phuongtv@uit.edu.vn