

Bài giảng tích hợp:

ASP NET CORE

Faculty of IT

Email: smdat@hueic.edu.vn

ASP.NET CORE

Chương 1. Giới thiệu chung về ASP.NET Core

Chương 2. ASP.NET Core Razor Pages

Chương 3. ASP.NET CORE MVC

Chương 2. ASP.NET Core Razor Pages

- 2.1. Razor Pages: giới thiệu, cài đặt, project, page, URL
- 2.2. Layout, ViewStart, section trong ASP.Net Core
- 2.3. Routing trong Razor Pages – ánh xạ URL sang page
- 2.4. Cú pháp Razor cơ bản, biểu thức và khối code
- 2.5. Các cấu trúc điều khiển của Razor, directive, ViewImports
- 2.6. Model class trong Razor Pages
- 2.7. Handler và Action Results trong Razor Pages
- 2.8. Query string, route data, route template – xử lý truy vấn GET
- 2.9. HTML Form trong Razor Pages – xử lý truy vấn POST
- 2.10. Model binding cơ bản trong Razor Pages
- 2.11. Thực hành 1 – Xây dựng ứng dụng quản lý sách
- 2.12. Thực hành 2 – CRUD trong Razor Pages

2.1. Razor Pages: giới thiệu, cài đặt, project, page, URL

• Razor Pages là gì?

- là một trong các **framework** dành cho xây dựng ứng dụng web bên trên **ASP.NET Core**.
- **Razor Pages** cho phép trộn **HTML** và **C#** (gọi là cú pháp **Razor**) vào cùng một file (có đuôi **cshtml**) để dễ dàng tạo ra **HTML** theo logic của chương trình. Nhờ vậy dữ liệu **HTML** tạo ra “**động**” chứ không cố định như ở các trang web “**tĩnh**” thiết kế sẵn.
- **Razor pages** được xây dựng dựa trên tư tưởng “**Page centric**” – lấy “**trang**” (**page**) làm trung tâm của việc tổ chức file và mã nguồn của dự án. Khi trình duyệt truy xuất trang (thực chất là file **cshtml**), mã **C#** tương ứng trên trang sẽ thực thi để tạo ra dữ liệu **HTML** trả về cho trình duyệt.

2.1. Razor Pages: giới thiệu, cài đặt, project, page, URL

• Thực hành 1: Cấu hình dự án Razor Pages

- **Bước 1:** Tạo một project trống
- **Bước 2:** Cấu hình sử dụng service

• Ứng dụng Razor Pages yêu cầu sử dụng *service* riêng bằng cách gọi phương thức *AddRazorPages* trong *ConfigureServices*. Mở file *Startup.cs*, tìm phương thức *ConfigureServices* và viết code như sau:

```
8. public void ConfigureServices(IServiceCollection services) {  
9.     services.AddRazorPages();  
10. }
```

• **Bước 3:** Cấu hình sử dụng **middleware**

• Ứng dụng Razor Pages yêu cầu sử dụng *middleware Routing* và *MapRazorPages endpoint*. Mở file *Startup.cs*, tìm phương thức *Configure* và điều chỉnh code như sau:

```
12. public void Configure(IApplicationBuilder app, IWebHostEnvironment env) {  
13.     if (env.IsDevelopment()) {  
14.         app.UseDeveloperExceptionPage();  
15.     }  
16.  
17.     app.UseRouting();  
18.  
19.     app.UseEndpoints(endpoints => {  
20.         endpoints.MapRazorPages();  
21.     });  
22. }
```

2.1. Razor Pages: giới thiệu, cài đặt, project, page, URL

- **Thực hành 1: Cấu hình dự án Razor Pages**
 - **Bước 4:** Tạo page `cshtml`
 - Tạo thư mục **Pages** trực thuộc **project**. Trong thư mục này tạo tiếp file văn bản *About.cshtml* như sau:
 - **Bước 5:** Viết code cho *About.cshtml*

2.1. Razor Pages: giới thiệu, cài đặt, project, page, URL

```
1. @page
2. @{
3.     Layout = null;
4.     ViewData["Title"] = "About";
5. }
6.
7. <!DOCTYPE html>
8.
9. <html>
10.    <head>
11.        <meta name="viewport" content="width=device-width" />
12.        <title>@ViewData["Title"] - Tự học ICT</title>
13.        <style>
14.            .header {
15.                color: cadetblue;
16.            }
17.
18.            .footer {
19.                text-align: center;
20.                font-size: small;
21.            }
22.        </style>
23.    </head>
24.    <body>
25.        <h1 class="header">Razor Pages tutorial</h1>
26.        <hr />
27.
28.        <h2>@SayHello("Covid")</h2>
29.        <p>It's @DateTime.Now.ToString("yyyy-MM-dd")</p>
30.
31.        <p>ASP.NET Core Razor Pages is a page-focused framework for building dynamic, data-d
32.
33.        <hr />
34.        <h2 class="footer">@tuhocict.com - @DateTime.Now.Year</h2>
35.    </body>
36. </html>
37.
38. @{
39.     string SayHello(string name) {
40.         return $"Hello, {name}. Welcome to Razor Pages!";
41.     }
42. }
```

2.1. Razor Pages: giới thiệu, cài đặt, project, page, URL

- Thực hành 1: Cấu hình dự án Razor Pages
 - Bước 6: Chạy thử ứng dụng



Razor Pages tutorial

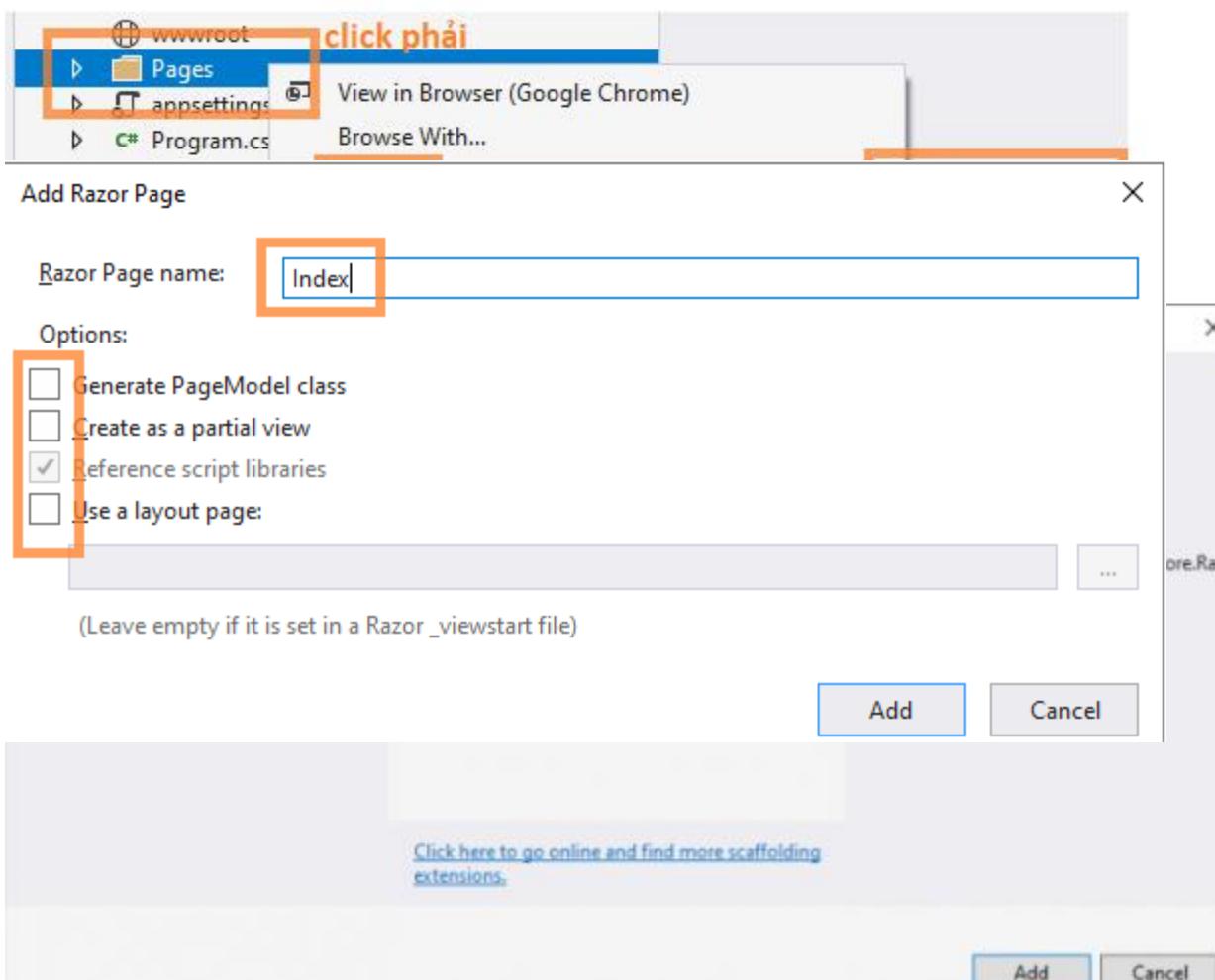
Hello, Covid. Welcome to Razor Pages!

It's Thursday, April 9, 2020

ASP.NET Core Razor Pages is a page-focused framework for building dynamic, data-driven web sites with clean separation of concerns. Based on the latest version of ASP.NET from Microsoft - ASP.NET Core, Razor Pages supports cross platform development and can be deployed to Windows, Unix and Mac operating systems.

2.1. Razor Pages: giới thiệu, cài đặt, project, page, URL

- Thực hành 2: Tạo page mới và trang mặc định
 - Bước 1: Thêm trang *Index.cshtml* vào thư mục Pages



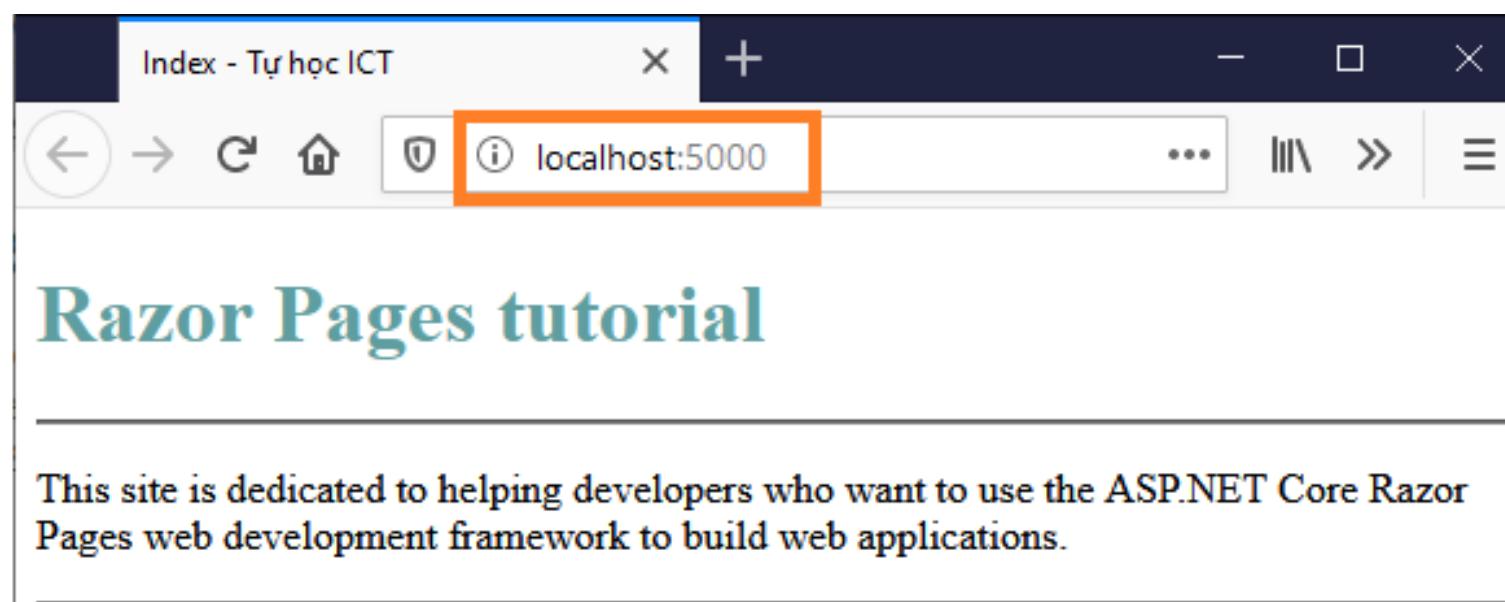
2.1. Razor Pages: giới thiệu, cài đặt, project, page, URL

- Thực hành 2: Tạo page mới và trang mặc định
 - Bước 2: Điều chỉnh nội dung của *Index.cshtml* như sau:

```
1. @page
2. @{
3.     Layout = null;
4.     ViewData["Title"] = "Index";
5. }
6.
7. <!DOCTYPE html>
8.
9. <html>
10. <head>
11.     <meta name="viewport" content="width=device-width" />
12.     <title>@ViewData["Title"] - Tự học ICT</title>
13.     <style>
14.         .header {
15.             color: cadetblue;
16.         }
17.
18.         .footer {
19.             text-align: center;
20.             font-size: small;
21.         }
22.     </style>
23. </head>
24. <body>
25.     <h1 class="header">Razor Pages tutorial</h1>
26.
27.     <hr />
28.     <p>This site is dedicated to helping developers who want to use the ASP.NET
29.
30.     <hr />
31.     <h2 class="footer">@tuhocict.com - @DateTime.Now.Year</h2>
32. </body>
33. </html>
```

2.1. Razor Pages: giới thiệu, cài đặt, project, page, URL

- **Thực hành 2: Tạo page mới và trang mặc định**
 - **Bước 3:** Chạy lại chương trình mà không chỉ định rõ trang cần tải bạn sẽ thấy trang *Index.cshtml* đã được tải vào trình duyệt.



2.1. Razor Pages: giới thiệu, cài đặt, project, page, URL

• Thực hành 3: Cấu trúc thư mục và URL

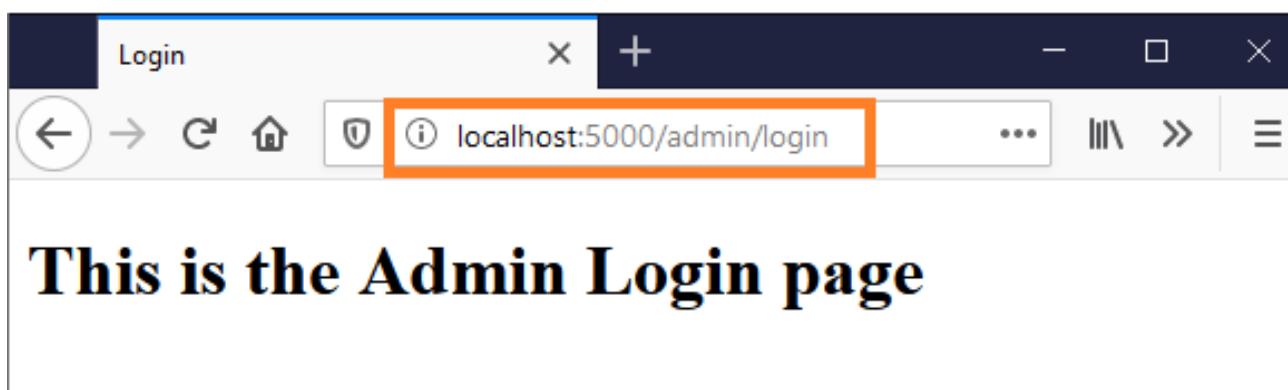
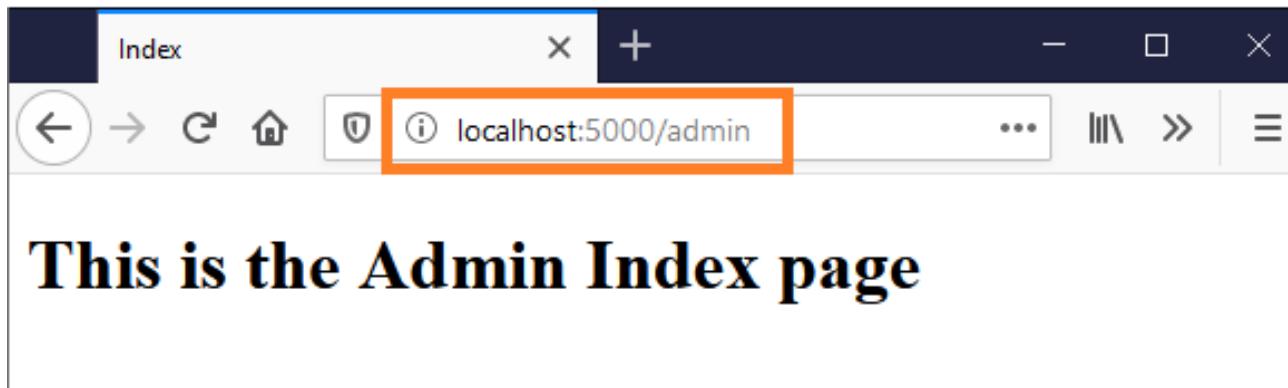
- **Bước 1:** Tạo thư mục con *Admin* của *Pages*
- **Bước 2:** Tạo thêm hai file *Index.cshtml* và *Login.cshtml* trong thư mục *Admin*
- **Bước 3:** Lần lượt điều chỉnh nội dung của *Login.cshtml* và *Index.cshtml* như sau:

```
1. @page
2. @{
3.     Layout = null;
4. }
5.
6. <!DOCTYPE html>
7.
8. <html>
9. <head>
10.    <meta name="viewport" content="width=device-width" />
11.    <title>Login</title>
12. </head>
13. <body>
14.    <h1>This is the Admin Login page</h1>
15. </body>
16. </html>
```

```
1. @page
2. @{
3.     Layout = null;
4. }
5.
6. <!DOCTYPE html>
7.
8. <html>
9. <head>
10.    <meta name="viewport" content="width=device-width" />
11.    <title>Index</title>
12. </head>
13. <body>
14.    <h1>This is the Admin Index page</h1>
15. </body>
16. </html>
```

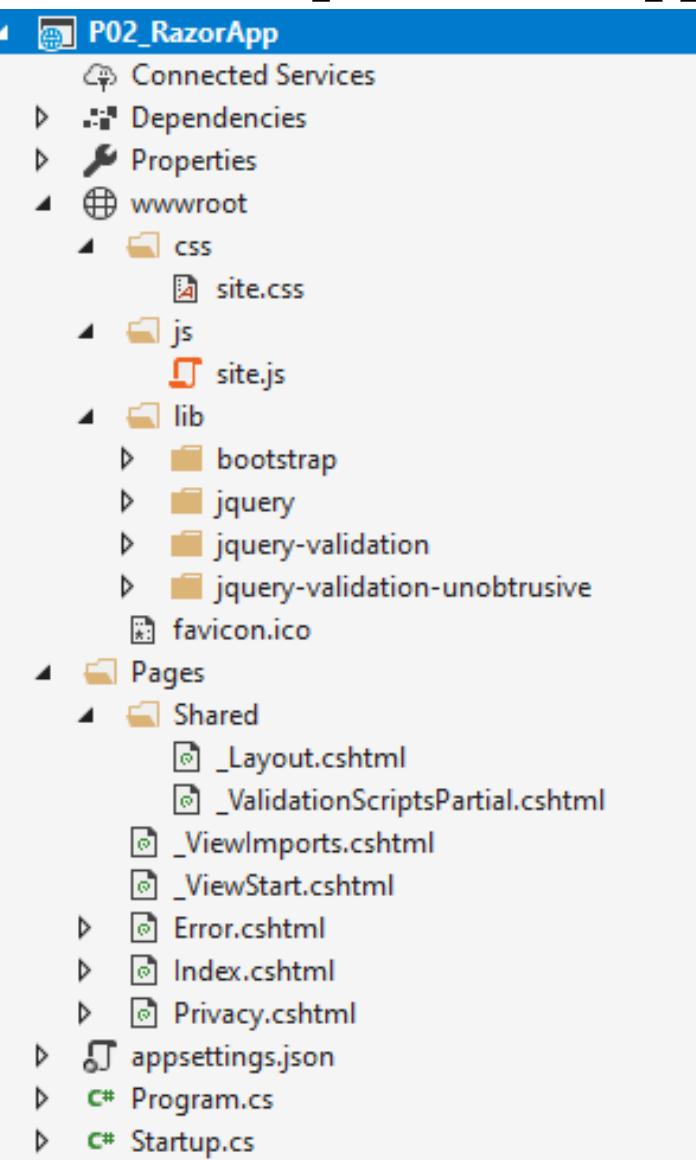
2.1. Razor Pages: giới thiệu, cài đặt, project, page, URL

- **Thực hành 3: Cấu trúc thư mục và URL**
 - **Bước 4:** Chạy chương trình với URL như sau:



2.1. Razor Pages: giới thiệu, cài đặt, project, page, URL

- **Thực hành 4: Sử dụng mẫu project Web Application**
 - **Bước 1:** Thêm project Asp.net core vào solution
 - **Bước 2:** Lựa chọn template Web Application



2.1. Razor Pages: giới thiệu, cài đặt, project, page, URL

2.2. Layout, ViewStart, section trong ASP.NET Core

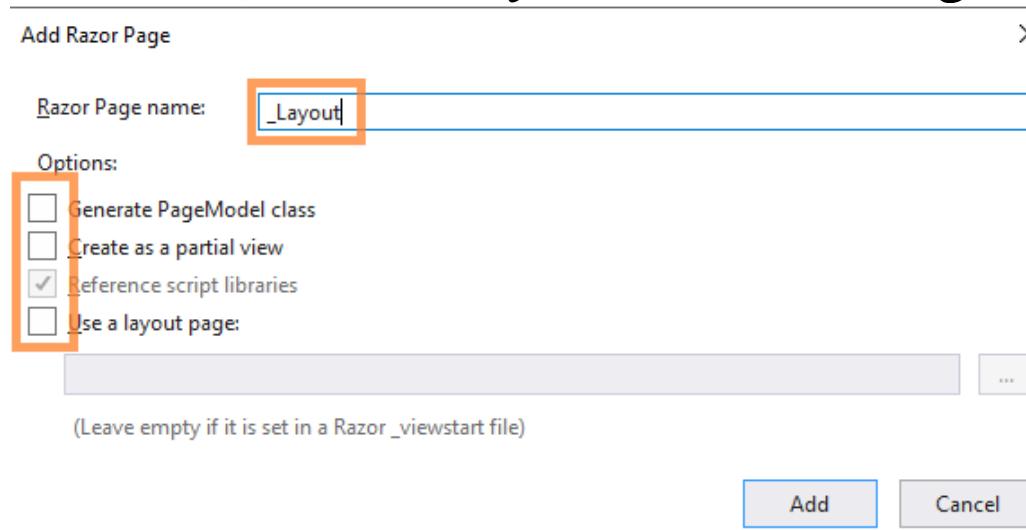
• Khái niệm layout trong ASP.NET Core

- Cấu trúc trang thường được chia thành các phần **header**, **footer**, **navigation menu** và **phần nội dung chính**. Đa số có thêm **sidebar** (phải | trái | hai bên). Cấu trúc chung thống nhất của các page trong cùng một site được gọi là **layout**
- Trong bài học trước: hai page **About.cshtml** và **Index.cshtml** chỉ khác biệt nhau về một phần nội dung hiển thị. Những phần khác giống hệt nhau. Tuy nhiên chúng ta đang phải lặp lại code trong hai page này.
- Từ đây phát sinh vấn đề phải tạo ra một cái khung chung thống nhất mà tất cả các page của một site đều sử dụng để tránh việc phải lặp đi lặp lại các đoạn code **HTML**, **CSS** và **JavaScript**.
- **Razor page** cung cấp một cơ chế để thực hiện: **Layout**.

2.1. Razor Pages: giới thiệu, cài đặt, project, page, URL

2.2. Layout, ViewStart, section trong ASP.NET Core

- **Thực hành 1: Sử dụng layout trong Razor Pages**
 - **Bước 1: Tạo file *_Layout.cshtml* trong thư mục Pages**



2.1. Razor Pages: giới thiệu, cài đặt, project, page, URL

2.2. Layout, ViewStart, section trong ASP.NET Core

- Thực hành 1: Sử dụng layout trong Razor Pages
 - Bước 2: Nhập nội dung cho file _Layout.cshtml như sau:

```
1. <!DOCTYPE html>
2.
3. <html>
4.   <head>
5.     <meta name="viewport" content="width=device-width" />
6.     <title>@ViewData["Title"] - Tự học ICT</title>
7.     <style>
8.       .header {
9.         color: cadetblue;
10.      }
11.
12.      .footer {
13.        text-align: center;
14.        font-size: small;
15.      }
16.    </style>
17.  </head>
18.  <body>
19.    <h1 class="header">Razor Pages tutorial</h1>
20.    <hr />
21.
22.    @RenderBody()
23.
24.    <hr />
25.    <h2 class="footer">@tuhocict.com - @DateTime.Now.Year</h2>
26.  </body>
27. </html>
```

2.1. Razor Pages: giới thiệu, cài đặt, project, page, URL

2.2. Layout, ViewStart, section trong ASP.NET Core

- **Thực hành 1: Sử dụng layout trong Razor Pages**
 - **Bước 3:** Thay đổi nội dung các page để sử dụng layout vừa tạo

```
Index.cshtml About.cshtml Admin/Index.cshtml Admin/Login.cshtml
1. @page
2.
3.     Layout = "_Layout";
4.     ViewData["Title"] = "Index";
5.
6.
7. <p>This site is dedicated to helping developers who want to use the ASP.NET Co
```

```
Index.cshtml About.cshtml Admin/Index.cshtml Admin/Login.cshtml
1. @page
2.
3.     Layout = "_Layout";
4.     ViewData["Title"] = "About";
5.
6.
7. <h2>@SayHello("Covid")</h2>
8.
9. <p>It's @DateTime.Now.ToString("yyyy-MM-dd")</p>
10.
11. <p>ASP.NET Core Razor Pages is a page-focused framework for building dynamic,
```

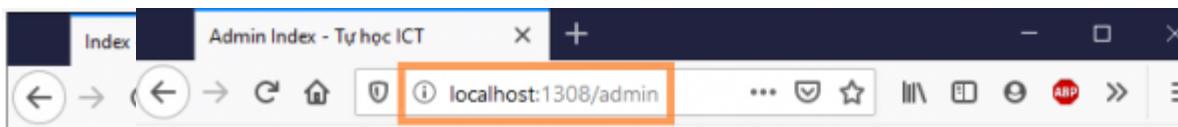
```
Index.cshtml About.cshtml Admin/Index.cshtml Admin/Login.cshtml
1. @page
2.
3.     Layout = "AdminLayout";
4.
```

```
Index.cshtml About.cshtml Admin/Index.cshtml Admin/Login.cshtml
1. @page
2.
3.     Layout = "_Layout";
4.     ViewData["Title"] = "Admin login";
5.
6.
7. <h1>This is the Admin Login page</h1>
```

2.1. Razor Pages: giới thiệu, cài đặt, project, page, URL

2.2. Layout, ViewStart, section trong ASP.NET Core

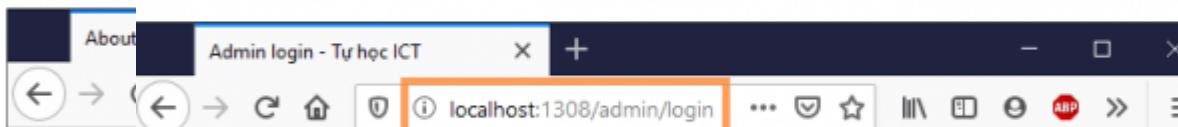
- **Thực hành 1: Sử dụng layout trong Razor Pages**
 - **Bước 4:** Chạy ứng dụng và thử truy cập vào các URL bạn thu được kết quả như sau:



Razor Pages tutorial

This site is developed by developer.com
This is the Admin Index page

©tuhocict.com - 2020



Razor Pages tutorial

Hello, C#
This is the Admin Login page

It's Thursday

ASP.NET Core
©tuhocict.com - 2020

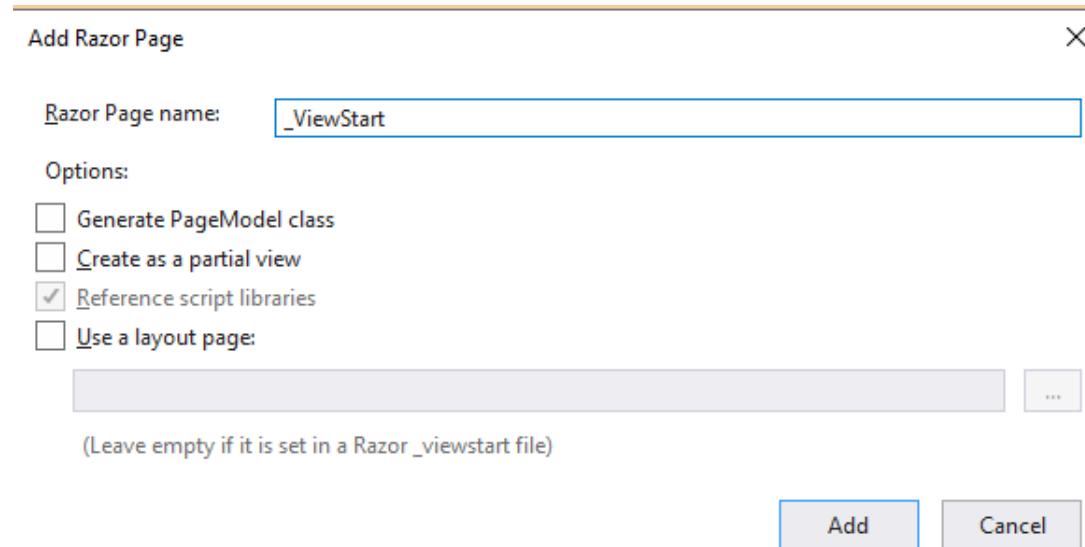
with clean separation of concerns. Based on the latest version of ASP.NET Core from Microsoft - ASP.NET Core, Razor Pages supports cross platform development and can be deployed to Windows, Unix and Mac operating systems.

2.1. Razor Pages: giới thiệu, cài đặt, project, page, URL

2.2. Layout, ViewStart, section trong ASP.NET Core

- **Thực hành 2: Tự động chọn layout trong Razor: file _ViewStart.cshtml**

- **Bước 1: Tạo file _ViewStart.cshtml trong thư mục Pages:**



- **Bước 2: Viết code cho _ViewStart như sau:**

```
1. @{
2.     Layout = "_Layout";
3. }
```

2.1. Razor Pages: giới thiệu, cài đặt, project, page, URL

2.2. Layout, ViewStart, section trong ASP.NET Core

- **Thực hành 2: Tự động chọn layout trong Razor: file _ViewStart.cshtml**

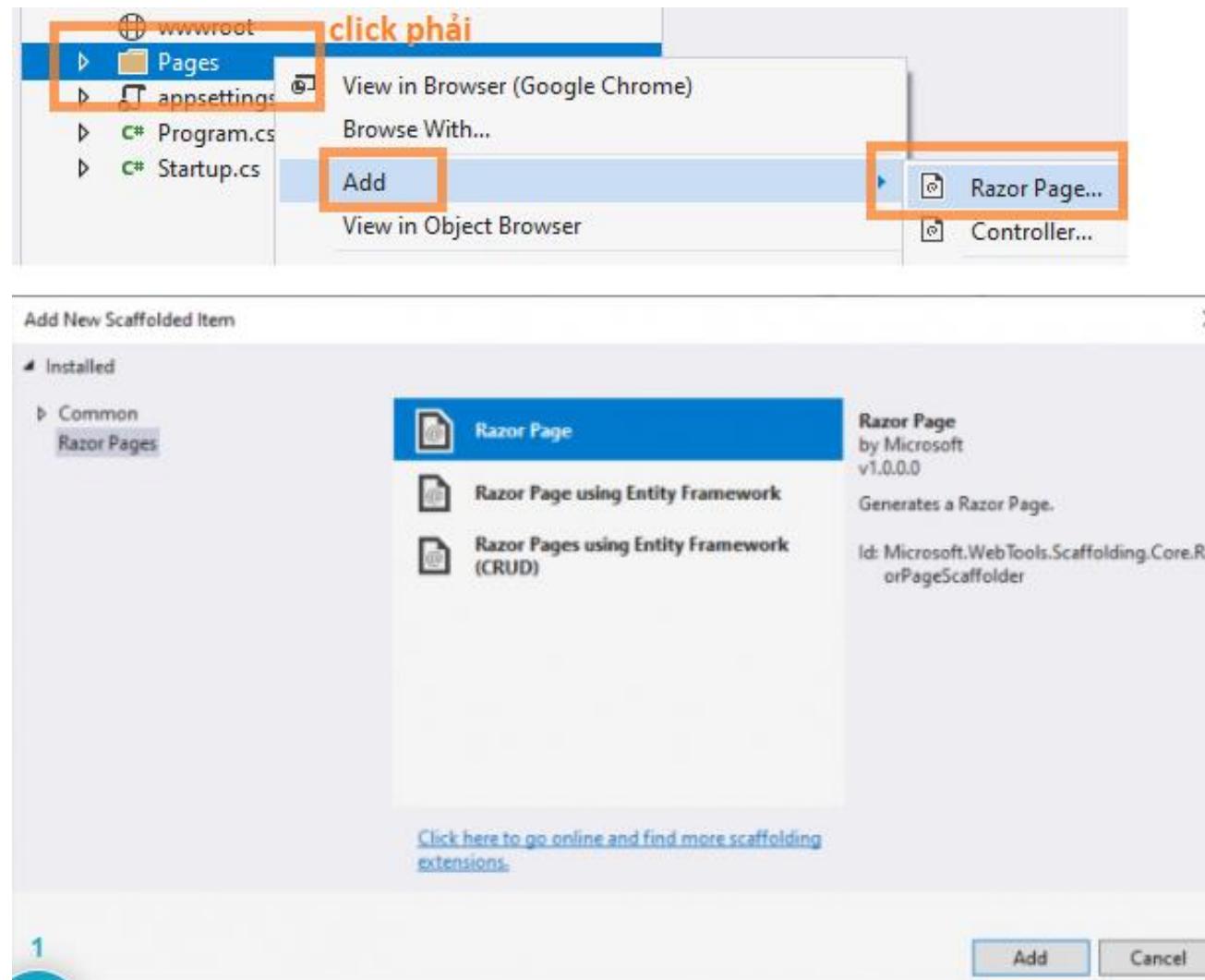
- **Bước 3:** Xóa bỏ lệnh *Layout = "_Layout"* hoặc *Layout = null* trong tất cả các page.
- **Bước 4:** Chạy thử chương trình, bạn sẽ thu được cùng một kết quả như phần **thực hành 1**. Tất cả các file đều sử dụng chung *_Layout.cshtml*.
- Như vậy, nếu giờ đây bạn muốn thử nghiệm một layout mới thì chỉ cần thay thế nó trong **_ViewStart** chứ không cần thay trong từng page. **_ViewStart** là cơ chế chạy code chung trước khi xử lý page.

2.1. Razor Pages: giới thiệu, cài đặt, project, page, URL

2.2. Layout, ViewStart, section trong ASP.NET Core

• Thực hành 3: Tạo Razor page với layout bằng Visual studio

• Bước 1: Mở hộp thoại tạo Add Razor Page

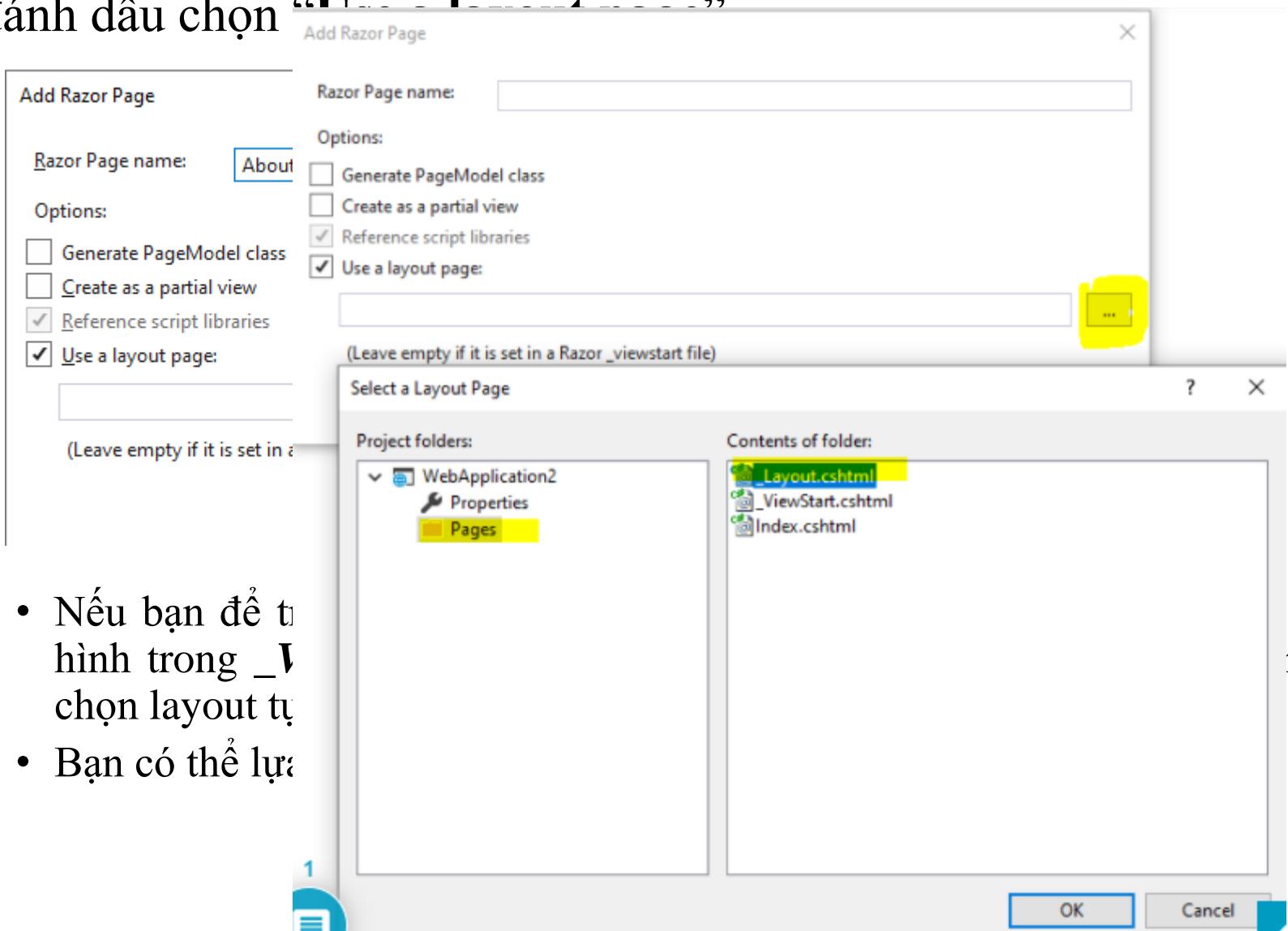


2.1. Razor Pages: giới thiệu, cài đặt, project, page, URL

2.2. Layout, ViewStart, section trong ASP.NET Core

• Thực hành 3: Tạo Razor page với layout bằng Visual studio

- Bước 2: Trong hộp thoại Add Razor Page chọn tên cho page và đánh dấu chọn “ Use a layout page”



- Nếu bạn để _V hình trong _V chọn layout tự
- Bạn có thể lựa

cấu
n hình

2.1. Razor Pages: giới thiệu, cài đặt, project, page, URL

2.2. Layout, ViewStart, section trong ASP.NET Core

• Sử dụng section trong layout

- **ASP.NET Core Razor Pages** sử dụng cơ chế **section** để hiển thị nội dung ở nhiều vị trí khác nhau trong **layout**. Cơ chế **section** cho phép tạo ra nhiều vị trí để chèn nội dung trên layout. “**Nội dung**” ở đây có thể hiểu là **HTML, CSS hay JavaScript**.
- Để sử dụng **section** cần có sự phối hợp giữa layout và trang nội dung.
- Ở **layout**, bạn sử dụng lệnh sau ở vị trí bạn muốn chèn nội dung vào:

```
1. @RenderSection("Tên_section", true|false)
```

- Ở **trang nội dung**, bạn khai báo **section** với cấu trúc sau:

```
1. @section Tên_section {  
2.     <!-- nội dung viết ở đây -->  
3. }
```

- Trong đó **Tên_section** ở nơi khai báo (trên trang nội dung) và nơi sử dụng (trên layout) phải trùng khớp nhau.
- Tham số thứ hai của **RenderSection** là một biến kiểu **bool**. Nếu có giá trị **true** thì bất kỳ trang nội dung nào sử dụng **layout** này đều phải khai báo **section** có tên tương ứng. Nếu có giá trị **false** thì trang nội dung **không bắt buộc** phải khai báo **section**.
- Một trong những trường hợp phải sử dụng **section** là khi cần render khối code **JavaScript**.

2.1. Razor Pages: giới thiệu, cài đặt, project, page, URL

2.2. Layout, ViewStart, section trong ASP.NET Core

• Sử dụng section trong layout

- Nếu bạn sử dụng **template WebApplication** khi tạo dự án, **Razor Pages** tạo sẵn cho bạn một file **layout**. Hãy mở file này ra và tìm đến vị trí ngay trước thẻ kết thúc **</body>**, bạn sẽ thấy một ví dụ về cách sử dụng **section**:

```
1. @RenderSection("Scripts", false)
```

- Nếu ở một trang nội dung nào đó bạn cần sử dụng **JavaScript**, bạn chỉ cần định nghĩa một **Scripts section** riêng:

```
1. @section Scripts {
2.     <script>
3.         // mã javascript viết ở đây
4.     </script>
5. }
```

- Khi này, khối **javascript** của bạn sẽ được chèn vào đúng vị trí **@RenderSection("Scripts", false)** trên **layout**, ngay trước khi kết thúc **</body>** – đây cũng là vị trí khuyến nghị để chèn **javascript** cho trang.

2.1. Razor Pages: giới thiệu, cài đặt, project, page, URL

2.2. Layout, ViewStart, section trong ASP.NET Core

2.3. Routing trong Razor Pages – ánh xạ URL sang page

- **Routing** là một khái niệm rất quan trọng trong các **web framework** nói chung. **Routing** có nhiệm vụ ánh xạ (**map**) truy vấn của người dùng (dưới dạng **URL** – **Uniform Resource Locator**) sang trang web/phương thức trên **server**.
- Trong **Razor Pages**, **routing** là cơ chế ánh xạ **URL** sang trang **cshtml**. Đồng thời **routing** trong **Razor Pages** còn có thêm những nhiệm vụ quan trọng khác nữa.

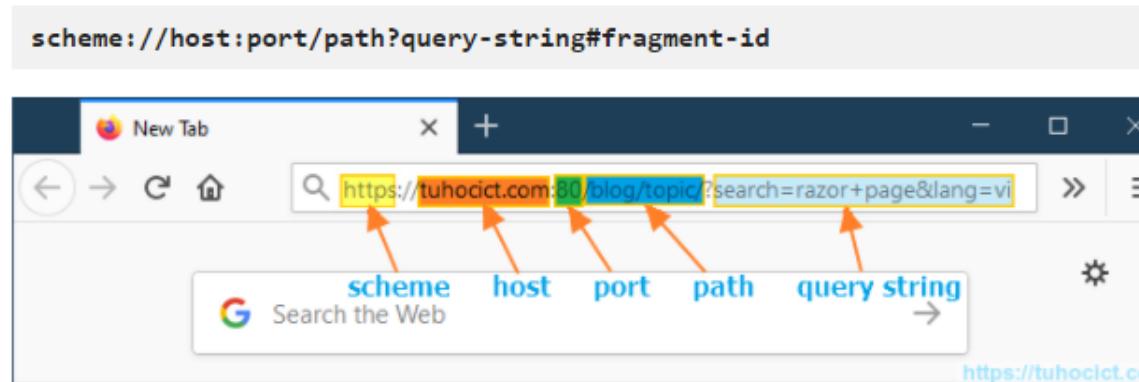
2.1. Razor Pages: giới thiệu, cài đặt, project, page, URL

2.2. Layout, ViewStart, section trong ASP.NET Core

2.3. Routing trong Razor Pages – ánh xạ URL sang page

• Cấu trúc URL

- là viết tắt của *Uniform Resource Locator*, là địa chỉ của một tài nguyên trên Internet.
- Cấu trúc tổng quát nhất của một URL như sau:



- **Scheme** thường gặp là **http**, **https** hoặc một số giao thức khác (như **ftp**).
- Trong cấu trúc trên, phần **port** ít gặp hơn. **Port** mặc định mà web server chiếm là **80**. Các trình duyệt đều tự động thêm port **80** vào khi cần. Nếu sử dụng port khác **80**, bạn phải tự mình chỉ định. Ví dụ một số site quản lý có thể dùng port **8080** thay cho **80**.
- Phần **path** có thể hiểu tương tự như đường dẫn tới một file (trên ổ cứng). Các **web framework** thường sẽ định nghĩa khuôn mẫu cho phần **path** trong một cơ chế gọi là **routing**
- **Query string** chứa các cặp khóa =giá trị. Các cặp phân tách bởi ký tự **&**. **Query string** phân tách với path bởi ký tự **?**.
- **Fragment** (hoặc **anchor**) thường chỉ gặp trong các tài nguyên nội dung (như **blog post**) dùng để đánh dấu các tiêu đề

2.1. Razor Pages: giới thiệu, cài đặt, project, page, URL

2.2. Layout, ViewStart, section trong ASP.NET Core

2.3. Routing trong Razor Pages – ánh xạ URL sang page

• **Routing là gì?**

- Khi bạn cần tải một nội dung gì đó, bạn nhập **URL** tương ứng vào thanh địa chỉ trình duyệt. **URL** thường được hiểu là “*địa chỉ*” của một *tài nguyên* nào đó trên **Internet**.
- “*Tài nguyên*” thường được hiểu là các file trên mạng. Do đó, thông thường mỗi **URL** tương ứng với một file trên server, ví dụ file ảnh, file video, file audio.
- Một ứng dụng web cần: (1) định nghĩa tập **URL** hợp lệ; (2) kiểm tra tính hợp lệ của một **URL** nó nhận được; (3) xác định xem **URL** (hợp lệ) tương ứng với tài nguyên nào.
- Cơ chế thực hiện nhóm nhiệm vụ đó trong các ứng dụng web thường được gọi là **routing**
- Để thực hiện routing, các **framework** thường định nghĩa các **route template** – khuôn mẫu để tạo ra tập hợp các **Url** hợp lệ cho mỗi tài nguyên, đồng thời cũng dùng để kiểm tra tính hợp lệ của **Url** nhận được từ client. Mỗi cặp **Url** hợp lệ và tài nguyên tương ứng của nó tạo thành một **route**
- Như thế, khi hiểu **route template**, có thể xác định được **URL** hợp lệ cho tài nguyên, đồng thời từ **URL** có thể xác định được tài nguyên tương ứng (nếu có) trên server.

2.1. Razor Pages: giới thiệu, cài đặt, project, page, URL

2.2. Layout, ViewStart, section trong ASP.NET Core

2.3. Routing trong Razor Pages – ánh xạ URL sang page

• Routing mặc định trong Razor Pages

- Cơ chế routing mặc định **Razor Pages** sử dụng khuôn mẫu là đường dẫn tương đối từ thư mục **Pages** đến các file *cshtml* (bỏ qua phần mở rộng *cshtml*)
- Thư mục **Pages** được gọi là **thư mục gốc (root folder)**
- **Ví dụ:** nếu trong thư mục **Pages** có 3 file *Index.cshtml*, *Privacy.cshtml*, *Error.cshtml*, **Razor Pages** sẽ tạo ra một tập hợp 4 **route** sau:
 - / < - > file Pages\Index.cshtml
 - /index < - > file Pages\Index.cshtml
 - /error < - > file Pages\Error.cshtml
 - /privacy < - > file Pages\Privacy.cshtml
- Tức là phần **path** của **Url** sẽ tương ứng với đường dẫn tới file *cshtml* (tính từ thư mục **Pages**). Nói chung, **route template** mặc định của mỗi trang đều rất đơn giản.

2.1. Razor Pages: giới thiệu, cài đặt, project, page, URL

2.2. Layout, ViewStart, section trong ASP.NET Core

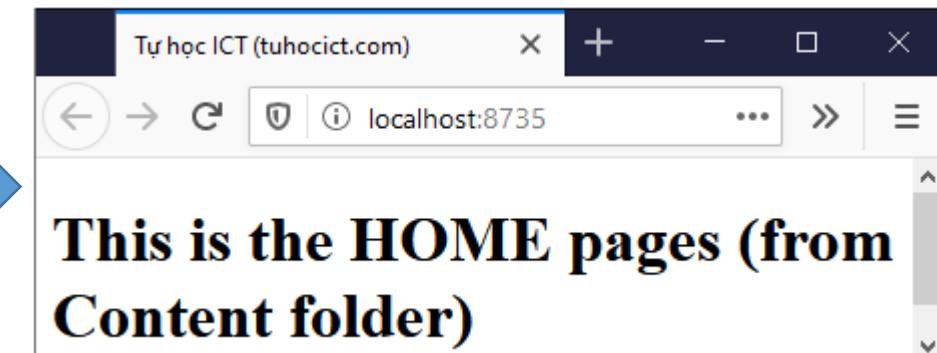
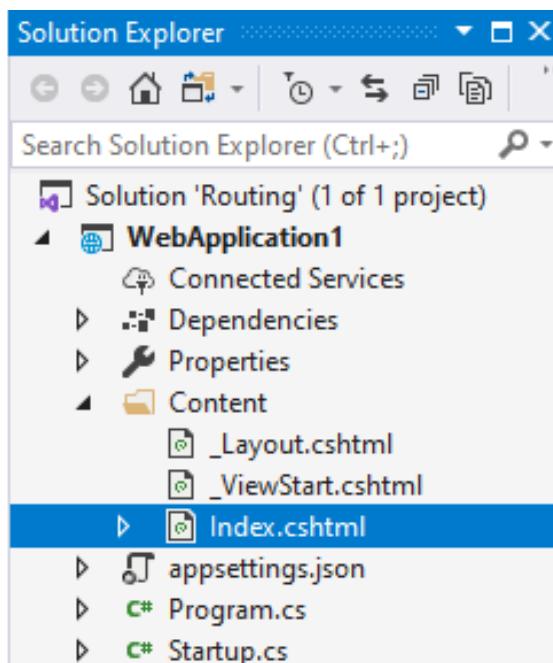
2.3. Routing trong Razor Pages – ánh xạ URL sang page

• Thay root folder

- Mặc định **Razor Pages** sử dụng **Pages** làm **root folder**, nghĩa là URL được ánh xạ sang các page nằm trong thư mục **Pages**. Ví dụ, để **Content** trở thành **root folder** trong phương thức **ConfigureServices** (lớp **Startup**) như sau:

```
1. public void ConfigureServices(IServiceCollection services) {  
2.     services  
3.         .AddRazorPages()  
4.         .AddRazorPagesOptions(options => {  
5.             options.RootDirectory = "/Content";  
6.         });  
7. }
```

- Khi này nếu thêm page **Index.cshtml** vào thư mục **Content**



2.1. Razor Pages: giới thiệu, cài đặt, project, page, URL

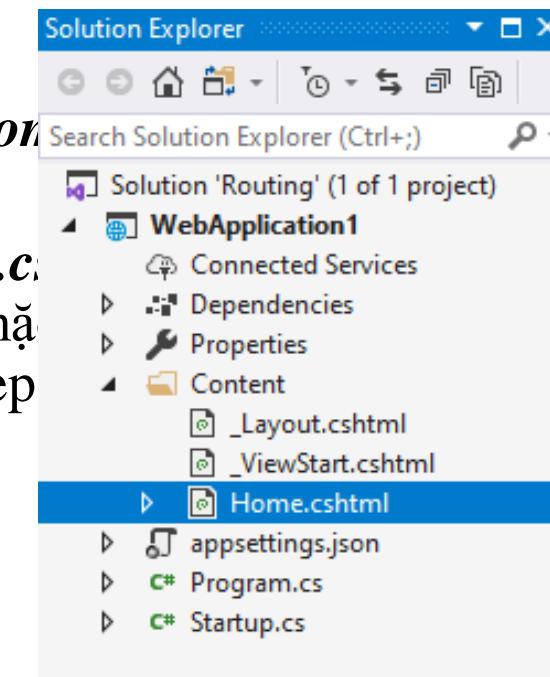
2.2. Layout, ViewStart, section trong ASP.NET Core

2.3. Routing trong Razor Pages – ánh xạ URL sang page

• Thay đổi homepage

- **Homepage** (trang chủ) là page tương ứng với URL /, là trang mặc định nếu người dùng không nhập gì thêm vào sau **hostname** trong URL.
- **Razor page** sẽ ánh xạ sang trang chủ mặc định là <root folder>|**Index.cshtml**.
- Thay đổi trong **ConfigureServices** như sau:

```
1.     public void ConfigureServices(IServiceCollection services) {
2.         services
3.             .AddRazorPages()
4.             .AddRazorPagesOptions(options => {
5.                 options.RootDirectory = "/Content";
6.                 options.Conventions.AddPageRoute("/Home", "/");
7.             });
8.     }
```



- Lệnh **options.Conventions.AddPageRoute('/Home', '/')** tương ứng với file <root folder>\Home.cshtml
- Lưu ý rằng, nếu trong **root folder** có file **Index.cshtml**, sẽ gây lỗi vì **Razor Pages** luôn coi **Index** làm page mặc định. Nếu nằm trong <root folder> thì nó luôn làm homepage và sẽ tương ứng với 2 page, và điều này sẽ gây lỗi.

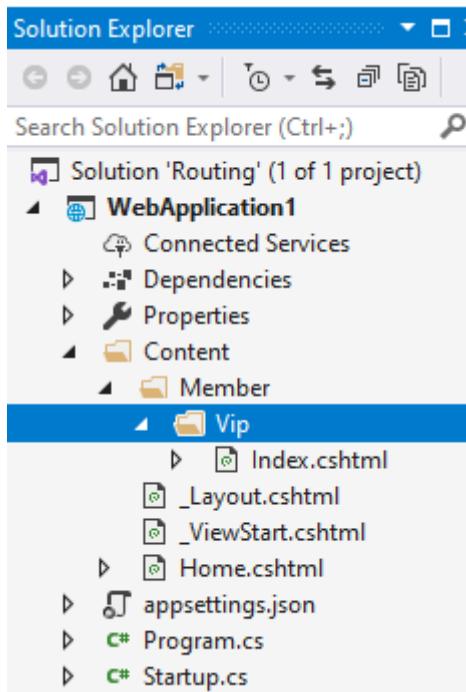
2.1. Razor Pages: giới thiệu, cài đặt, project, page, URL

2.2. Layout, ViewStart, section trong ASP.NET Core

2.3. Routing trong Razor Pages – ánh xạ URL sang page

• Ghi đè route mặc định

- Giả sử trong <root folder> bạn tạo thư mục con Member, trong Member tạo tiếp thư mục Vip, trong Vip tạo file **Index.cshtml**:



- Khi này URL mặc định cho **Index.cshtml** là **/member/vip** hoặc **/member/vip/index**
- Giờ muốn ghi đè route mặc định này, ví dụ, thay url tương ứng với file thành **/blog**. Bạn có hai cách thực hiện:

2.1. Razor Pages: giới thiệu, cài đặt, project, page, URL

2.2. Layout, ViewStart, section trong ASP.NET Core

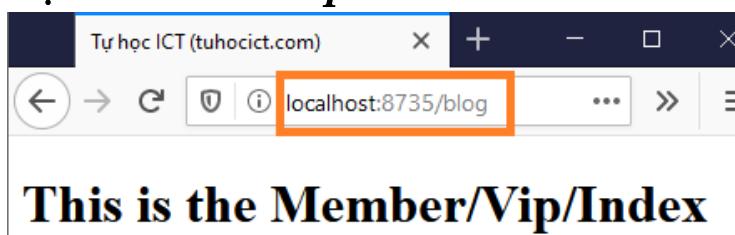
2.3. Routing trong Razor Pages – ánh xạ URL sang page

• Ghi đè route mặc định

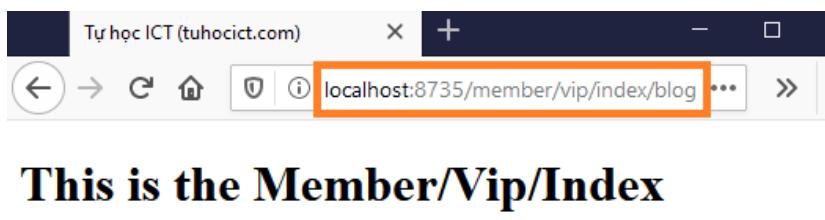
- Option 1: Mở file */Member/Vip/Index.cshtml* và thay đổi *@page directive* như sau:

```
1. @page "/blog"
2. @model WebApplication1.Content.Member.Vip.IndexModel
3. @{
4. }
```

- Lưu ý trong directive *@page "/blog"* phải có ký tự "/" hoặc "~/blog" để báo cho **Razor Pages** biết rằng bạn đang muốn ghi đè **url** trong route mặc định. Trong trường hợp này **Url** trong **route** tới page sẽ là **/blog**, thay vì url mặc định **/member/vip/index** hoặc **/member/vip/**



- **Chú ý:** Nếu thiếu ký tự "/" **Razor Pages** sẽ hiểu rằng đây là một **đoạn (segment)** của một **url** chứ không phải là **url** hoàn chỉnh. Cùng ví dụ trên, nếu bạn viết *@page "blog"* thì **url** tương ứng với page sẽ là **/member/vip/index/blog** hoặc **/member/vip/blog**. Nghĩa là **blog** khi đó trở thành **segment** cuối cùng của **url** mặc định tương ứng:



2.1. Razor Pages: giới thiệu, cài đặt, project, page, URL

2.2. Layout, ViewStart, section trong ASP.NET Core

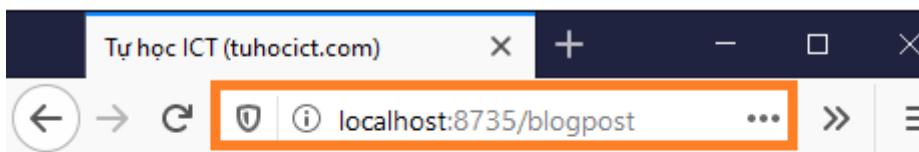
2.3. Routing trong Razor Pages – ánh xạ URL sang page

• Ghi đè route mặc định

- Option 2: Sử dụng phương thức **AddPageRoute** trong phương thức **ConfigureServices** lớp **Startup**:

```
1. public void ConfigureServices(IServiceCollection services) {
2.     services
3.     .AddRazorPages()
4.     .AddRazorPagesOptions(options => {
5.         options.RootDirectory = "/Content";
6.         options.Conventions.AddPageRoute("/Home", "/");
7.         options.Conventions.AddPageRoute("/Member/Vip/Index", "/blogpost");
8.     });
9. }
```

- Phương thức **AddPageRoute** nhận hai tham số: tham số thứ nhất là đường dẫn tương đối từ **root folder** tới **page** (bỏ đuôi *cshtml*); tham số thứ hai là **url** cần ánh xạ tới page này.
- Trong lệnh trên bạn cấu hình một route mới ánh xạ url */blogpost* sang page */Member/Vip/Index*
- Bằng cách này bạn có thể định nghĩa nhiều route tới cùng một page.



This is the Member/Vip/Index

2.1. Razor Pages: giới thiệu, cài đặt, project, page, URL

2.2. Layout, ViewStart, section trong ASP.NET Core

2.3. Routing trong Razor Pages – ánh xạ URL sang page

2.4. Cú pháp Razor cơ bản, biểu thức và khối code

- **Razor** là loại cú pháp kết hợp C# và HTML để sinh ra HTML động trong ứng dụng **ASP.NET Core**. **Razor** được sử dụng trong **Razor Pages**, **MVC** và **Blazor**. Đây là loại cú pháp rất quan trọng mà bạn bắt buộc phải biết khi học **ASP.NET Core**.

• **Razor là gì?**

- **Razor** là tên gọi của **view engine** sử dụng bởi **ASP.NET Core**, đồng thời cũng là tên gọi của loại ngôn ngữ đánh dấu sử dụng trong **view engine** này.

• **Razor View Engine**

- **Razor** là tên gọi của **view engine** – cơ chế sinh ra mã **HTML** (từ nhiều nguồn gốc khác nhau) để trả lại cho trình duyệt – sử dụng trong **ASP.NET Core**. **Razor view engine** hoạt động dựa trên khả năng đọc loại mã kết hợp giữa C# và HTML.

• **Razor syntax/language**

- Tên gọi **Razor** cũng được dùng để chỉ loại cú pháp/ngôn ngữ đánh dấu (**markup syntax/language**) sử dụng trong **Razor View Engine** của **ASP.NET Core**. Cú pháp **Razor** kết hợp C# và HTML với nhiệm vụ “**đánh dấu**” xem đâu là C#, đâu là HTML để **view engine** xử lý cho phù hợp.
- **Razor** là loại “**ngôn ngữ**” sử dụng trong các framework của **ASP.NET Core** như Razor Pages, MVC, Blazor
- File **Razor** có phần mở rộng là **cshtml**

2.1. Razor Pages: giới thiệu, cài đặt, project, page, URL

2.2. Layout, ViewStart, section trong ASP.NET Core

2.3. Routing trong Razor Pages – ánh xạ URL sang page

2.4. Cú pháp Razor cơ bản, biểu thức và khối code

• Giới thiệu chung về cú pháp Razor

- Cú pháp **Razor** có nhiệm vụ phân định xem đâu là C# và đâu là **HTML** để **View Engine** có thể dịch và xử lý chính xác.

```
1. @page
2.
3. @{
4.     // đây là một code block
5.     // Tiêu đề trang
6.     ViewData["Title"] = "Razor syntax tutorial";
7.
8.
9.     var Name = "nCov"; // khai báo biến
10.
11.    string SayHello(string name) => $"Hello in code block, {name}"; // khai báo phương t
12.
13.    var countries = new[] { "USA", "Russia", "France", "GB" }; // khai báo mảng
14.
15.    void RenderName(string name) { // đây là một phương thức có cách chuyển đổi C# -> HTML
16.        <p>Name: <strong>@name</strong></p>
17.    }
18.
19.    <!-- Đây là thẻ h1 chứa biểu thức razor -->
20.    <h1>@SayHello(Name)</h1>
21.
22.    <!-- Đây là thẻ html <p> thông thường -->
23.    <p>Razor is a markup syntax for embedding server-based code into webpages. The Razor sy
24.
25.    <!-- Thẻ ul chứa cấu trúc điều khiển foreach sinh ra các thẻ li tương ứng -->
26.    <ul>
27.        @foreach (var c in countries) {
28.            <li>@c</li>
29.        }
    </ul>
```

2.1. Razor Pages: giới thiệu, cài đặt, project, page, URL

2.2. Layout, ViewStart, section trong ASP.NET Core

2.3. Routing trong Razor Pages – ánh xạ URL sang page

2.4. Cú pháp Razor cơ bản, biểu thức và khối code

• Giới thiệu chung về cú pháp Razor

Nhìn chung bạn cần nhớ các nguyên tắc cơ bản sau đây:

- (1) **Ngôn ngữ mặc định trong file cshtml là HTML:** Nghĩa là nếu không có đánh dấu gì đặc biệt thì Razor (view engine) sẽ hiểu nội dung bạn trình bày là ngôn ngữ HTML, và sẽ chuyển thẳng khỏi HTML này thành kết quả đầu ra.
- (2) **Ký tự @ yêu cầu Razor chuyển từ HTML sang C#:** Nghĩa là, nếu nhìn thấy ký tự @, Razor sẽ hiểu là bạn đang muốn viết code C# và chuyển sang chế độ dịch C#.
- (3) **Kết quả dịch mã cuối cùng của file Razor (cshtml) là mã HTML.** Tất cả những gì bạn viết trên page đều hướng tới sinh ra HTML (để trả lại cho trình duyệt).

2.1. Razor Pages: giới thiệu, cài đặt, project, page, URL

2.2. Layout, ViewStart, section trong ASP.NET Core

2.3. Routing trong Razor Pages – ánh xạ URL sang page

2.4. Cú pháp Razor cơ bản, biểu thức và khối code

• Giới thiệu chung về cú pháp Razor

Ngoài ra bạn cũng ghi nhớ một số vấn đề sau:

- (1) @ **escape**: Viết hai lần ký tự @ (tức là @@) để diễn đạt cho chính chữ @ trong mã Razor (chữ không phải chuyển đổi sang C#);
- (2) Razor có khả năng tự phân biệt ký tự @ nằm trong địa chỉ email;
- (3) Cách ghi **chú thích phụ thuộc vào ngôn ngữ**: nếu trong vùng code C# thì ghi chú thích kiểu C#, nếu trong vùng mã HTML thì sử dụng cách ghi chú thích của HTML
- (4) Ký tự @ khi đi cùng một số từ đặc biệt (được Razor sử dụng cho mục đích riêng) gọi là các **directive**, ví dụ @page, @model, @using.

2.1. Razor Pages: giới thiệu, cài đặt, project, page, URL

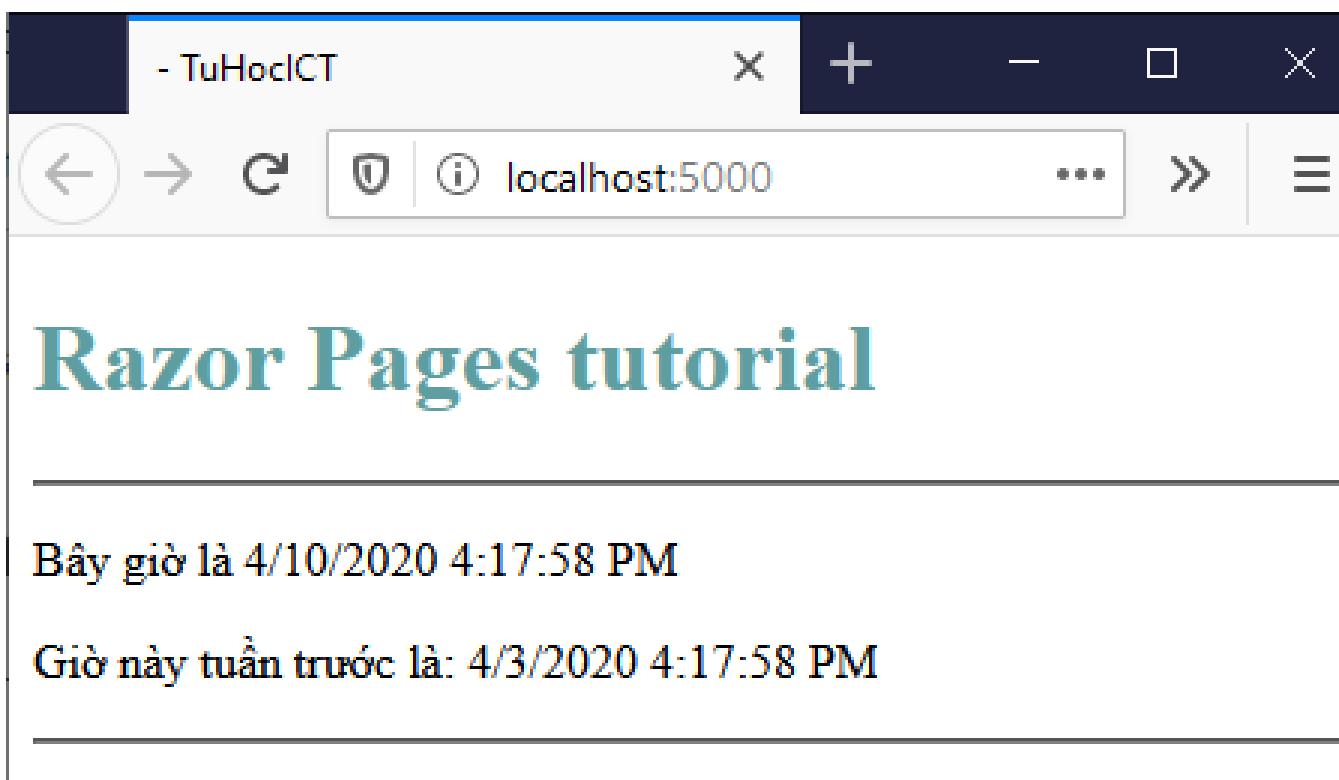
2.2. Layout, ViewStart, section trong ASP.NET Core

2.3. Routing trong Razor Pages – ánh xạ URL sang page

2.4. Cú pháp Razor cơ bản, biểu thức và khối code

- **Biểu thức Razor:** gồm có **biểu thức** và **khối lệnh**
 - Khái niệm biểu thức Razor
 - là *biểu thức của C#* được ghép trực tiếp cùng mã **HTML** thông qua ký tự @. Biểu thức **Razor** có tác dụng chèn giá trị (tính bằng C#) vào vị trí tương ứng của **HTML**.
 - Ví dụ:

```
1. <p>Bây giờ là @DateTime.Now</p>
2. <p>Giờ này tuần trước là: @ (DateTime.Now - TimeSpan.FromDays(7))</p>
```



2.1. Razor Pages: giới thiệu, cài đặt, project, page, URL

2.2. Layout, ViewStart, section trong ASP.NET Core

2.3. Routing trong Razor Pages – ánh xạ URL sang page

2.4. Cú pháp Razor cơ bản, biểu thức và khối code

- **Biểu thức Razor:** gồm có **biểu thức** và **khối lệnh**
 - **Biểu thức (expression)** trong C# là những lệnh có trả về kết quả. Đó có thể là lời gọi **phương thức** (method call) hoặc kết quả thực hiện các **phép toán** (operator). Những phương thức không trả về kết quả (còn gọi là các statement – lệnh) không thể sử dụng làm biểu thức **Razor**.
 - giá trị được tính ra bởi biểu thức C# và chèn vào vị trí tương ứng của mã **HTML**
 - **Biểu thức rõ (explicit expression)** và **biểu thức ẩn (implicit expression)**
 - Trong ví dụ trên, `@DateTime.Now` là **biểu thức ẩn**, `@(DateTime.Now - TimeSpan.FromDays(7))` là **biểu thức rõ**.
 - **Biểu thức ẩn (implicit expression)** không được phép chứa khoảng trống hoặc những ký tự đặc biệt gây nhầm lẫn với code **HTML** xung quanh.
 - **Ví dụ** biểu thức ẩn không thể chứa **generics** (vì cú pháp generics sử dụng cặp dấu <> với tham số kiểu, dẫn đến nhầm lẫn với thẻ **HTML**). **Biểu thức ẩn** cũng không thể là các phép toán thông thường (+, -, *, /).
 - Nếu một biểu thức không thể biểu diễn ở **dạng ẩn** thì bạn bắt buộc phải dùng **biểu thức rõ**: biểu thức đặt trong cặp dấu ()

2.1. Razor Pages: giới thiệu, cài đặt, project, page, URL

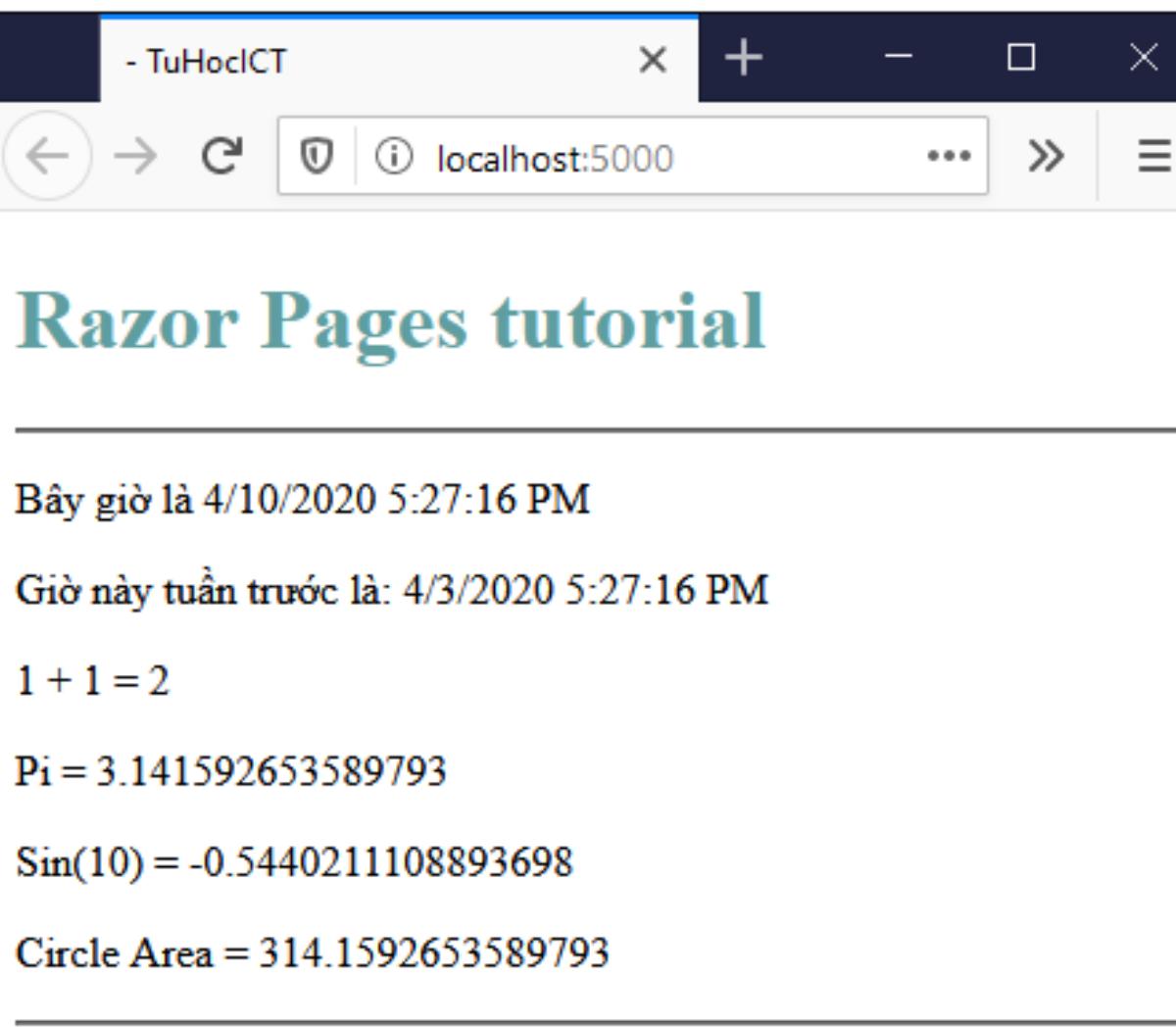
2.2. Layout, ViewStart, section trong ASP.NET Core

2.3. Routing trong Razor Pages – ánh xạ URL sang page

2.4. Cú pháp Razor cơ bản, biểu thức và khối code

- Biểu thức Razor: gồm có biểu thức và khối lệnh

```
1. <p>1 + 1 = @(1 + 1)</p> <!-- explicit -->
2. <p>Pi = @Math.PI</p> <!-- implicit -->
3. <p>Sin(10) = @Math.Sin(10)</p> <!-- implicit -->
4. <p>Circle Area = @(Math.PI * 10 * 10)</p> <!-- explicit -->
```



2.1. Razor Pages: giới thiệu, cài đặt, project, page, URL

2.2. Layout, ViewStart, section trong ASP.NET Core

2.3. Routing trong Razor Pages – ánh xạ URL sang page

2.4. Cú pháp Razor cơ bản, biểu thức và khối code

- **Biểu thức Razor:** gồm có **biểu thức** và **khối lệnh**
 - **Khối code Razor:** là một khái niệm cơ bản và rất quan trọng của Razor. **Khối code Razor (Razor code block)** là tất cả những code nằm trong một vùng `@{ ... }`. Khối code là một vùng trong **file/page Razor** trong đó ngôn ngữ mặc định là **C#**.

```
1. @page
2. @{
3.     // đây là một khối code
4.     // khai báo và khởi tạo biến
5.     var str = "Đây là một biến khai báo và khởi tạo trong code block";
6.
7.     string SayHello(string name) {
8.         return $"Hello, {name}";
9.     }
10.
11.    var sol = Solve(1, -2, 1);
12.
13.    <h3>@SayHello("Covid")</h3>
14.
15.    <p>@str</p>
16.
17.    @{
18.        // đây là một khối code khác
19.        str = "Cập nhật giá trị của biến trong một code block khác";
20.    }
21.
22.    <p>@str</p>
23.
24.    <p>Phương trình  $x^2 + 2x + 1 = 0$  có nghiệm là @Solve</p>
25.
26.    <p>Phương trình  $x^2 - 2x + 1 = 0$  có nghiệm là @sol</p>
27.
28.    @{
29.        // đây cũng là một khối code
30.        (double, double) Solve(double a, double b, double c) {
31.            var d = b * b - 4 * a * c;
32.            return
33.                ((-b + Math.Sqrt(d)) / (2 * a),
34.                 (-b - Math.Sqrt(d)) / (2 * a));
35.        }
36.    }
}
```



2.1. Razor Pages: giới thiệu, cài đặt, project, page, URL

2.2. Layout, ViewStart, section trong ASP.NET Core

2.3. Routing trong Razor Pages – ánh xạ URL sang page

2.4. Cú pháp Razor cơ bản, biểu thức và khối code

- **Biểu thức Razor:** gồm có **biểu thức** và **khối lệnh**
- **Những đặc điểm của code block**

- (1) **Code block** là nơi bạn dùng để khai báo/khởi tạo biến và khai báo hàm cục bộ (**local function**). Trong ví dụ trên bạn đã khai báo và khởi tạo biến *str*, hàm *SayHello*, hàm *Solve*, biến *sol*
- (2) Các biến và hàm tạo ra trong **code block** có thể được sử dụng làm biểu thức **Razor** cũng như được sử dụng trong **code block** khác trên page đó.
- (3) Không giới hạn số lượng và vị trí viết **code block** trên một page.
- (4) Vị trí khai báo hàm không quan trọng, nghĩa là bạn có thể gọi hàm ở **code block** trước nơi hàm đó khai báo. Như trong ví dụ trên, Lời gọi **hàm Solve** được viết trước khi hàm này được khai báo.
- (5) Biến chỉ có thể sử dụng sau vị trí khai báo. Đây là điều khác biệt với hàm. Nếu bạn sử dụng biến trước khi khai báo sẽ bị lỗi.

2.1. Razor Pages: giới thiệu, cài đặt, project, page, URL

2.2. Layout, ViewStart, section trong ASP.NET Core

2.3. Routing trong Razor Pages – ánh xạ URL sang page

2.4. Cú pháp Razor cơ bản, biểu thức và khối code

- **Biểu thức Razor:** gồm có **biểu thức** và **khối lệnh**
- **Chuyển đổi ngôn ngữ trong code block**

- Trong file **Razor**, ngôn ngữ mặc định là **HTML**. Riêng trong **code block**, ngôn ngữ mặc định lại là **C#**. **Razor** cho phép chuyển đổi ngôn ngữ trong **code block** từ **C#** sang **HTML**. Đây là đặc điểm rất mạnh nhưng cũng rối rắm của **Razor**.
- Tự động chuyển đổi ngôn ngữ trong code block

```
1. @page
2. @{
3.     void SayHello(string name) {
4.         <p>
5.             <strong style="color:royalblue;">Hello, @name!</strong><br />
6.             <span>Welcome to Razor tutorial!</span>
7.         </p>
8.     }
9.
10.    // Lời gọi hàm bình thường
11.    SayHello("H1N1");
12.
13.    // Dưới đây là đoạn HTML kết hợp với lời gọi hàm
14.    <div style="background: #b6ff00;">@{SayHello("Covid")}; </div>
15.
16.    // Viết code C# bình thường
17. }
18.
19. <div style="background: #ffd800;">
20.     @{ SayHello("Sars"); }
21. </div>
```

```
1. // Dưới đây là đoạn HTML kết hợp với lời gọi hàm
2. <div style="background: #b6ff00;">@{SayHello("Covid")}; </div>
```

2.1. Razor Pages: giới thiệu, cài đặt, project, page, URL

2.2. Layout, ViewStart, section trong ASP.NET Core

2.3. Routing trong Razor Pages – ánh xạ URL sang page

2.4. Cú pháp Razor cơ bản, biểu thức và khối code

- **Biểu thức Razor:** gồm có **biểu thức** và **khối lệnh**
 - **Chuyển đổi ngôn ngữ trong code block**

• **Chuyển đổi ngôn ngữ chủ động:** Trong một số trường hợp cơ chế chuyển đổi ngôn ngữ tự động có thể hoạt động không đúng ý bạn. Khi đó bạn có thể lựa chọn sử dụng cơ chế chuyển đổi ngôn ngữ chủ động của **Razor**. Đây là tình huống khi bạn **không muốn sử dụng thẻ HTML**

```
1. void SayHello(string name) {  
2.     <text>Name: @name</text>  
3. }
```

- **Hàm mẫu (template function/method):** là loại hàm cục bộ đặc biệt có khả năng xuất ra (**render**) **HTML** ở vị trí gọi hàm. Đặc điểm này giúp phân biệt hàm mẫu với hàm cục bộ thông thường (không render HTML).
 - Thực chất hàm mẫu là cách vận dụng tính năng chuyển đổi ngôn ngữ và xuất **HTML** của **Razor** trong **code block**.

```
1. <div style="background: #b6ff00;">@{SayHello("Covid"); }</div>  
2.  
3. <div style="background: #ffd800;">  
4.     @{ SayHello("Sars"); }  
5. </div>
```

- 2.1. Razor Pages: giới thiệu, cài đặt, project, page, URL
- 2.2. Layout, ViewStart, section trong ASP.NET Core
- 2.3. Routing trong Razor Pages – ánh xạ URL sang page
- 2.4. Cú pháp Razor cơ bản, biểu thức và khối code
- 2.5. Các cấu trúc điều khiển của Razor, directive, ViewImports

• Cấu trúc điều khiển của Razor

- Các cấu trúc điều khiển của Razor là những **code block** có nhiệm vụ điều chỉnh quá trình sinh mã HTML như phân nhánh, lặp, v.v., tương tự như vai trò của cấu trúc điều khiển trong C#.
- Các cấu trúc điều khiển giúp đơn giản hóa việc sinh mã HTML dựa theo các logic phức tạp.
- Trước hết cần lưu ý rằng, **bên trong code block** (nơi ngôn ngữ mặc định là C#), sử dụng các cấu trúc điều khiển của C# như bình thường.
- Tuy nhiên, ở **ngoài code block** (nơi ngôn ngữ mặc định là HTML), không thể trực tiếp sử dụng cách viết cấu trúc điều khiển của C# được. Khi đó, cần sử dụng lối viết **markup** của Razor dành cho cấu trúc điều khiển.
- Các **cấu trúc điều khiển** của Razor là dạng mở rộng của **code block**. Vì vậy những đặc điểm của **code block** (như viết code tự do, chuyển đổi ngôn ngữ) đều áp dụng cho các cấu trúc điều khiển. Cũng do vậy, sẽ thường xuyên gặp lối viết trộn lẫn lộn C# và **HTML**.
- Có thể sử dụng **hàm mầu** trong **code block** để tạo ra các khối **code HTML** phức tạp theo logic
- Khả năng sử dụng các **cấu trúc điều khiển** sẽ giúp bạn đơn giản hóa hơn nữa việc sinh ra mã **HTML** theo các logic phức tạp.

2.1. Razor Pages: giới thiệu, cài đặt, project, page, URL

2.2. Layout, ViewStart, section trong ASP.NET Core

2.3. Routing trong Razor Pages – ánh xạ URL sang page

2.4. Cú pháp Razor cơ bản, biểu thức và khối code

2.5. Các cấu trúc điều khiển của Razor, directive, ViewImports

- **Razor định nghĩa các markup dành cho:**

- (1) Các cấu trúc điều kiện, bao gồm cấu trúc **@if-else if-else** và **@switch**;
- (2) Các cấu trúc lặp, bao gồm **@for**, **@foreach**, **@while**, và **@do while**;
- (3) Cấu trúc **@using**;
- (4) Cấu trúc bắt lỗi **@try-catch-finally**.

2.1. Razor Pages: giới thiệu, cài đặt, project, page, URL

2.2. Layout, ViewStart, section trong ASP.NET Core

2.3. Routing trong Razor Pages – ánh xạ URL sang page

2.4. Cú pháp Razor cơ bản, biểu thức và khối code

2.5. Các cấu trúc điều khiển của Razor, directive, ViewImports

• Các cấu trúc điều kiện

• Cấu trúc @if-else if-else

```
1. @page
2. <p>@DateTime.Today</p>
3. @if(DateTime.Today.DayOfWeek == DayOfWeek.Monday) {
4.     var today = DateTime.Today;
5.     if(today.Day == 1) {
6.         <i>Hôm nay là ngày đầu tháng</i>
7.     }
8.     <strong>Chúc bạn một tuần làm việc mới hiệu quả</strong>
9. }
10. else if(DateTime.Now.DayOfWeek == DayOfWeek.Friday) {
11.     <strong>Chúc bạn ngày cuối tuần vui vẻ</strong>
12. }
13. else{
14.     <strong>Chúc bạn một ngày làm việc hiệu quả</strong>
15. }
```

2.1. Razor Pages: giới thiệu, cài đặt, project, page, URL

2.2. Layout, ViewStart, section trong ASP.NET Core

2.3. Routing trong Razor Pages – ánh xạ URL sang page

2.4. Cú pháp Razor cơ bản, biểu thức và khối code

2.5. Các cấu trúc điều khiển của Razor, directive, ViewImports

• Các cấu trúc điều kiện

• Cấu trúc @switch

```
1. @page
2. @switch (DateTime.Today.DayOfWeek) {
3.     case DayOfWeek.Monday:
4.         var today = DateTime.Today;
5.         if (today.Day == 1) {
6.             <i>Hôm nay là ngày đầu tháng</i>
7.         }
8.         <strong>Chúc bạn một tuần làm việc mới hiệu quả</strong>
9.         break;
10.    case DayOfWeek.Friday:
11.        <strong>Chúc bạn ngày cuối tuần vui vẻ</strong>
12.        break;
13.    default:
14.        <strong>Chúc bạn một ngày làm việc hiệu quả</strong>
15.        break;
16. }
```

2.1. Razor Pages: giới thiệu, cài đặt, project, page, URL

2.2. Layout, ViewStart, section trong ASP.NET Core

2.3. Routing trong Razor Pages – ánh xạ URL sang page

2.4. Cú pháp Razor cơ bản, biểu thức và khối code

2.5. Các cấu trúc điều khiển của Razor, directive, ViewImports

• Các cấu trúc điều kiện

• Các cấu trúc lặp

```
1. @page
2.
3. @{
4.     var countries = new[] {
5.         "China",
6.         "India",
7.         "United States",
8.         "Indonesia",
9.         "Pakistan",
10.        "Brazil",
11.        "Nigeria",
12.        "Bangladesh",
13.        "Russia",
14.        "Mexico" };
15. }
16.
17. <div style="background-color:cadetblue;padding:10px;">
18.     <strong>Các quốc gia đông dân nhất thế giới:</strong>
19.     <ol>
20.         @for (var i = 0; i < countries.Length; i++) {
21.             var country = countries[i];
22.             if (country == "China" || country == "India") {
23.                 <li>@country (trên 1 tỷ dân)</li>
24.             }
25.             else {
26.                 <li>@country</li>
27.             }
28.         }
29.     </ol>
30. </div>
```

2.1. Razor Pages: giới thiệu, cài đặt, project, page, URL

2.2. Layout, ViewStart, section trong ASP.NET Core

2.3. Routing trong Razor Pages – ánh xạ URL sang page

2.4. Cú pháp Razor cơ bản, biểu thức và khối code

2.5. Các cấu trúc điều khiển của Razor, directive, ViewImports

• Các cấu trúc điều kiện

• Cấu trúc @foreach

```
1.  @page
2.
3.  @{
4.      var countries = new[] {
5.          "China",
6.          "India",
7.          "United States",
8.          "Indonesia",
9.          "Pakistan",
10.         "Brazil",
11.         "Nigeria",
12.         "Bangladesh",
13.         "Russia",
14.         "Mexico" };
15.     }
16.
17. <div style="background-color:cadetblue;padding:10px;">
18.     <strong>Các quốc gia đông dân nhất thế giới:</strong>
19.     <ol>
20.         @foreach(var country in countries) {
21.             if (country == "China" || country == "India") {
22.                 <li>@country (trên 1 tỷ dân)</li>
23.             }
24.             else {
25.                 <li>@country</li>
26.             }
27.         }
28.     </ol>
29. </div>
```

2.1. Razor Pages: giới thiệu, cài đặt, project, page, URL

2.2. Layout, ViewStart, section trong ASP.NET Core

2.3. Routing trong Razor Pages – ánh xạ URL sang page

2.4. Cú pháp Razor cơ bản, biểu thức và khối code

2.5. Các cấu trúc điều khiển của Razor, directive, ViewImports

- Các cấu trúc điều kiện
 - Cấu trúc @while

```
1.  @page
2.
3.  @{
4.      var countries = new[] {
5.          "China",
6.          "India",
7.          "United States",
8.          "Indonesia",
9.          "Pakistan",
10.         "Brazil",
11.         "Nigeria",
12.         "Bangladesh",
13.         "Russia",
14.         "Mexico" };
15.     }
16.
17. <div style="background-color:cadetblue;padding:10px;">
18.     <strong>Các quốc gia đông dân nhất thế giới:</strong>
19.     <ol>
20.         @{
21.             var i = 0;
22.             while (i < countries.Length) {
23.                 var country = countries[i++];
24.                 if (country == "China" || country == "India") {
25.                     <li>@country (trên 1 tỷ dân)</li>
26.                 }
27.                 else {
28.                     <li>@country</li>
29.                 }
30.             }
31.         </ol>
32.     </div>
```

2.1. Razor Pages: giới thiệu, cài đặt, project, page, URL

2.2. Layout, ViewStart, section trong ASP.NET Core

2.3. Routing trong Razor Pages – ánh xạ URL sang page

2.4. Cú pháp Razor cơ bản, biểu thức và khối code

2.5. Các cấu trúc điều khiển của Razor, directive, ViewImports

• Các cấu trúc điều kiện

• Cấu trúc @do-while

```
1.  @page
2.
3.
4.  @{
5.      var countries = new[] {
6.          "China",
7.          "India",
8.          "United States",
9.          "Indonesia",
10.         "Pakistan",
11.         "Brazil",
12.         "Nigeria",
13.         "Bangladesh",
14.         "Russia",
15.         "Mexico" };
16.
17. <div style="background-color:cadetblue;padding:10px;">
18.     <strong>Các quốc gia đông dân nhất thế giới:</strong>
19.     <ol>
20.         @ { var i = 0; }
21.         @do {
22.             var country = countries[i++];
23.             if (country == "China" || country == "India") {
24.                 <li>@country (trên 1 tỷ dân)</li>
25.             }
26.             else {
27.                 <li>@country</li>
28.             }
29.         } while (i < countries.Length);
30.     </ol>
31. </div>
```

2.1. Razor Pages: giới thiệu, cài đặt, project, page, URL

2.2. Layout, ViewStart, section trong ASP.NET Core

2.3. Routing trong Razor Pages – ánh xạ URL sang page

2.4. Cú pháp Razor cơ bản, biểu thức và khối code

2.5. Các cấu trúc điều khiển của Razor, directive, ViewImports

- **Bắt và xử lý ngoại lệ**

- Razor cũng hỗ trợ cách viết cấu trúc @try-catch-finally như sau:

```
1. @try {
2.     throw new InvalidOperationException("You did something invalid.");
3. }
4. catch (Exception ex) {
5.     <p>The exception message: @ex.Message</p>
6. }
7. finally {
8.     <p>The finally statement.</p>
9. }
```

- 2.1. Razor Pages: giới thiệu, cài đặt, project, page, URL
- 2.2. Layout, ViewStart, section trong ASP.NET Core
- 2.3. Routing trong Razor Pages – ánh xạ URL sang page
- 2.4. Cú pháp Razor cơ bản, biểu thức và khối code
- 2.5. Các cấu trúc điều khiển của Razor, directive, ViewImports

• Khái niệm directive trong Razor

- Từ trước đến giờ, đều gặp `@page` ở đầu trang. `@page` được gọi là **directive** trong **Razor**.
- **Directive** là những từ đặc biệt được **Razor** lựa chọn để điều khiển những tính năng nhất định. Có thể hình dung **directive** như những từ khóa hoặc điều khiển riêng của **Razor**.
- **Directive** thường được đặt ở đầu page và chỉ có tác dụng trên page đó.

- 2.1. Razor Pages: giới thiệu, cài đặt, project, page, URL
- 2.2. Layout, ViewStart, section trong ASP.NET Core
- 2.3. Routing trong Razor Pages – ánh xạ URL sang page
- 2.4. Cú pháp Razor cơ bản, biểu thức và khối code
- 2.5. Các cấu trúc điều khiển của Razor, directive, ViewImports

• Khái niệm directive trong Razor

- Thường xuyên gặp các directive sau đây:

- **@page** – bắt buộc phải nằm ở dòng đầu tiên của một **content page**. Không có bất kỳ ký tự nào được phép đứng trước **@page**, nếu không **Razor** sẽ báo lỗi. **@page** báo cho **Razor** rằng đây là một **content page** và cần xử lý. Directive **@page** giúp phân biệt một trang **Razor** thông thường với **view component** hay **partial page**.
- **@model** – chỉ định **model class** cho page. Sau **@model** có thể là tên đầy đủ của **class**, hoặc là tên ngắn gọn nếu như có thêm directive **@using** trước đó.
- **@using** – tương tự như **using** của C# dùng để chỉ định **namespace** của các class sẽ được sử dụng trong page. Ví dụ muốn sử dụng class **File** (trong **System.IO**), bạn cần gọi **@using System.IO**.
- **@namespace** – tương tự như **namespace** của C#, dùng để chỉ định không gian tên chứa page class nếu không muốn sử dụng không gian tên mặc định theo quy ước của **Razor Pages**.
- **@functions** – chỉ định **functions block**

- 2.1. Razor Pages: giới thiệu, cài đặt, project, page, URL
- 2.2. Layout, ViewStart, section trong ASP.NET Core
- 2.3. Routing trong Razor Pages – ánh xạ URL sang page
- 2.4. Cú pháp Razor cơ bản, biểu thức và khối code
- 2.5. Các cấu trúc điều khiển của Razor, directive, ViewImports

• Directive và ViewImports

- Thông thường **directive** được đặt ở đầu của mỗi trang và chỉ có tác dụng trên trang đó. Tuy nhiên trong một số trường hợp bạn cần directive tự động có tác dụng trên tất cả các trang của site.
- **Ví dụ**, bạn cần chỉ định một **namespace** (sử dụng directive `@using`) mà tất cả các page đều cần truy cập, hoặc bạn cần thiết lập **namespace** của tất cả các page class (sử dụng directive `@namespace`)
- **Razor** cung cấp một cơ chế để thiết lập **directive** có tác dụng toàn cục như vậy, gọi là **import**.

2.1. Razor Pages: giới thiệu, cài đặt, project, page, URL

2.2. Layout, ViewStart, section trong ASP.NET Core

2.3. Routing trong Razor Pages – ánh xạ URL sang page

2.4. Cú pháp Razor cơ bản, biểu thức và khối code

2.5. Các cấu trúc điều khiển của Razor, directive, ViewImports

• Ví dụ Directive và ViewImports

- **Bước 1:** Tạo một page mới trong thư mục **Pages** và đặt tên là **_ViewImports.cshtml**
- **Bước 2:** Nhập nội dung như sau cho **_ViewImports.cshtml**:

```
1. @using WebApplication
2. @namespace WebApplication.Pages
3. @addTagHelper *, Microsoft.AspNetCore.Mvc.TagHelpers
```

- Từ giờ trở đi, tất cả các page trong thư mục **Pages** đều có **namespace** mặc định là **WebApplication.Pages**, đồng thời có thể truy xuất tất cả các class trong namespace **WebApplication** qua tên ngắn gọn.
- **Lưu ý:**
 - (1) Tương tự như **_ViewStart.cshtml**, **_ViewImports.cshtml** cũng có tác dụng trong thư mục chứa nó và tất cả các thư mục con của nó, với điều kiện trong thư mục con không có **_ViewStarts** của riêng mình.
 - (2) **_ViewImports** trong thư con sẽ đè lên **_ViewImports** của thư mục cha.

2.1. Razor Pages: giới thiệu, cài đặt, project, page, URL

2.2. Layout, ViewStart, section trong ASP.NET Core

2.3. Routing trong Razor Pages – ánh xạ URL sang page

2.4. Cú pháp Razor cơ bản, biểu thức và khối code

2.5. Các cấu trúc điều khiển của Razor, directive, ViewImports

2.6. Model class trong Razor Pages

- **Model class** là loại class đặc biệt có nhiệm vụ cung cấp dữ liệu và khả năng xử lý cho trang trong **Razor Pages**. **Model class** cùng với cú pháp **Razor** tạo ra sức mạnh cho **Razor Pages**. Sự kết hợp này giúp biến mỗi trang **Razor** trở thành một ứng dụng thực sự.
- Trong hai bài học trước bạn đã học cách thức viết code trên trang **Razor**. Lối viết code như vậy được gọi là mô hình **Single Page**. Tuy nhiên, nếu bạn tiếp tục sử dụng cách thức viết code như vậy, chương trình bạn viết ra sẽ có rất nhiều hạn chế.
- **Razor Pages** hỗ trợ hai cách viết code để xử lý logic của một trang: *functions block*, và *model class*. Trong đó, **model class** là cách viết “chính thống” nhất và được khuyến khích sử dụng trong mọi trường hợp.

2.1. Razor Pages: giới thiệu, cài đặt, project, page, URL

2.2. Layout, ViewStart, section trong ASP.NET Core

2.3. Routing trong Razor Pages – ánh xạ URL sang page

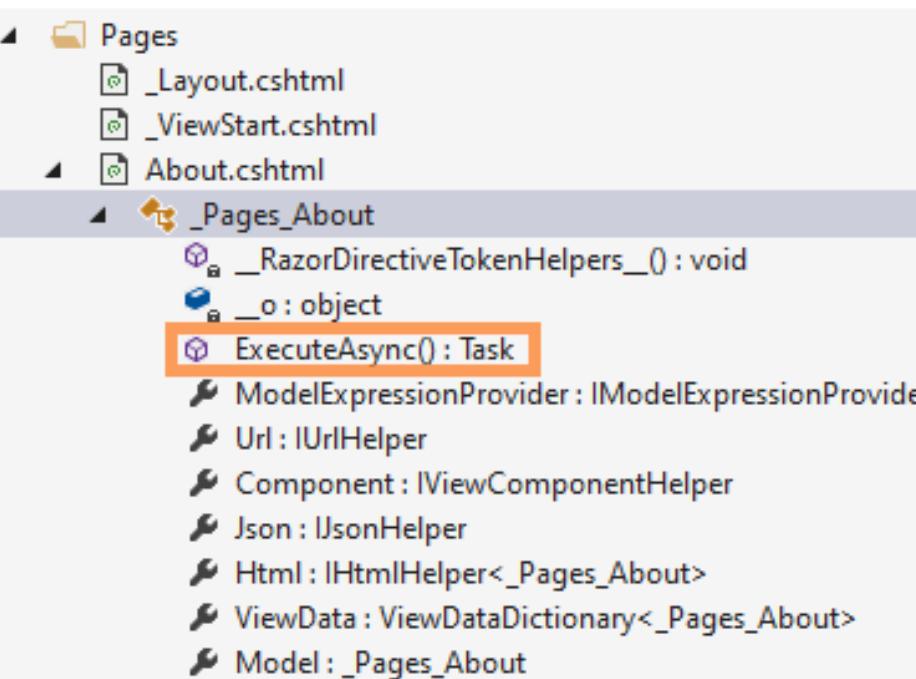
2.4. Cú pháp Razor cơ bản, biểu thức và khối code

2.5. Các cấu trúc điều khiển của Razor, directive, ViewImports

2.6. Model class trong Razor Pages

- **Cách thức hoạt động của trang Razor, mô hình Single Page**

- Trong hai bài học về cú pháp Razor chúng ta đã nói rằng không thể khai báo kiểu dữ liệu mới (*class*, *struct*, *enum*, v.v.) bên trong **code block**.
- Điều này liên quan đến cơ chế hoạt động của trang **Razor**.
- Khi tạo một page, giả sử đặt tên là *About.cshtml* bên trong thư mục **Pages**, **Razor** sẽ tự động tạo một class có tên là *_Pages_About*



- Tất cả những gì bạn tự viết trên page, bao gồm các biểu thức và khối code Razor, thực tế đều nằm trong một phương thức **ExecuteAsync()**

2.1. Razor Pages: giới thiệu, cài đặt, project, page, URL

2.2. Layout, ViewStart, section trong ASP.NET Core

2.3. Routing trong Razor Pages – ánh xạ URL sang page

2.4. Cú pháp Razor cơ bản, biểu thức và khối code

2.5. Các cấu trúc điều khiển của Razor, directive, ViewImports

2.6. Model class trong Razor Pages

- **Cách thức hoạt động của trang Razor, mô hình Single Page**
 - Điều này cũng giải thích cho mọi vấn đề bạn có lẽ đã gặp phải:
 - (1) có thể khai báo hàm cục bộ (không phải là phương thức), vì C# hỗ trợ hàm cục bộ (hàm nằm trong phương thức);
 - (2) không thể khai báo kiểu dữ liệu (C# không cho khai báo kiểu bên trong phương thức);
 - (3) code block lại có thể render HTML;
 - (4) không thể khai báo property (vì property không thể nằm trong method); v.v..
 - Đây cũng là cách thức viết code đơn giản nhất của page trong **Razor** và thường được gọi là mô hình *single page*. Mô hình này đã được chúng ta sử dụng từ đầu đến giờ
 - Tuy nhiên đây lại là mô hình không khuyến khích sử dụng khi phát triển ứng dụng **Razor Pages**. Nó có rất nhiều nhược điểm khi xây dựng các app lớn cũng như bị hạn chế về tính năng.
 - **Razor** cung cấp thêm hai mô hình hoạt động mạnh mẽ hơn: mô hình *functions block* và mô hình *page model class*

2.1. Razor Pages: giới thiệu, cài đặt, project, page, URL

2.2. Layout, ViewStart, section trong ASP.NET Core

2.3. Routing trong Razor Pages – ánh xạ URL sang page

2.4. Cú pháp Razor cơ bản, biểu thức và khối code

2.5. Các cấu trúc điều khiển của Razor, directive, ViewImports

2.6. Model class trong Razor Pages

• Functions block

- **Thực hành 1:** Mô hình này được gọi theo tên của một loại code block mới trong file Razor.
 - **Bước 1:** Tạo project Asp.net Core trống đặt tên là **WebApplication** và cấu hình để project này trở thành **Razor Pages**
 - **Bước 2:** Cấu hình layout và viewstart
 - **Bước 3:** Tạo page **Index.cshtml** trong **Pages** sử dụng layout và viewstart

```
1. @page
2.
3. @{
4.     void SayHello(string name) {
5.         <p class="block">Hello, @name! I'm in the code block</p>
6.     }
7. }
8.
9. @functions
10. {
11.     void S
12.     <p
13.     }
14.
15. <!-- #####
16. <style>
17.     .block { color: dodgerblue; }
18.     .function { color: yellowgreen; }
19.     .class { color: blueviolet; }
20.
21. </style>
22.
23. <!-- ##### -->
24.
25. @{
26.     SayHello("Covid");
    this.SayHello("Covid");
}
```

2.1. Razor Pages: giới thiệu, cài đặt, project, page, URL

2.2. Layout, ViewStart, section trong ASP.NET Core

2.3. Routing trong Razor Pages – ánh xạ URL sang page

2.4. Cú pháp Razor cơ bản, biểu thức và khối code

2.5. Các cấu trúc điều khiển của Razor, directive, ViewImports

2.6. Model class trong Razor Pages

• Model class

- Mặc dù **functions block** rất mạnh nhưng nó cũng có nhiều nhược điểm khi phát triển các ứng dụng lớn với nhiều page phức tạp:
 - Dễ thấy nhất là khi lượng code tăng lên, kích thước của page sẽ phình ra rất khó quản lý
 - Thứ hai là mô hình này trộn lẫn lộn cả logic-dữ liệu-giao diện vào cùng một khối – vốn là điều tối kỵ trong làm phần mềm.
 - Thứ ba, do trộn lẫn lộn các thành phần, các page này rất khó test.
- Vì vậy **Razor** đưa ra mô hình thứ ba –**model class**

2.1. Razor Pages: giới thiệu, cài đặt, project, page, URL

2.2. Layout, ViewStart, section trong ASP.NET Core

2.3. Routing trong Razor Pages – ánh xạ URL sang page

2.4. Cú pháp Razor cơ bản, biểu thức và khối code

2.5. Các cấu trúc điều khiển của Razor, directive, ViewImports

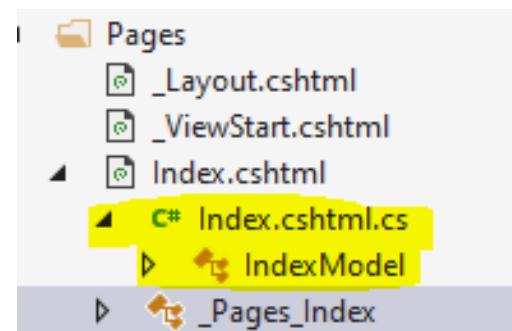
2.6. Model class trong Razor Pages

• Model class

• Thực hành 2: tiếp tục sử dụng project từ phần thực hành 1-2

- Bước 1: Tạo file **Index.cshtml.cs** trong thư mục **Pages**
- Bước 2: Viết code cho file **Index.cshtml.cs** như sau:

```
1.     using Microsoft.AspNetCore.Mvc.RazorPages;
2.
3.     namespace WebApplication.Pages {
4.         public class IndexModel : PageModel {
5.             public int Age { get; set; }
6.             public string FirstName { get; set; }
7.             public string LastName { get; set; }
8.             public string FullName => $"{FirstName} {LastName}";
9.
10.            public string SayGoodbye() => $"Hello, {FullName}! I'm in the model class";
11.
12.            public void OnGet() {
13.                FirstName = "Donald";
14.                LastName = "Trump";
15.            }
16.        }
17.    }
```



2.1. Razor Pages: giới thiệu, cài đặt, project, page, URL

2.2. Layout, ViewStart, section trong ASP.NET Core

2.3. Routing trong Razor Pages – ánh xạ URL sang page

2.4. Cú pháp Razor cơ bản, biểu thức và khối code

2.5. Các cấu trúc điều khiển của Razor, directive, ViewImports

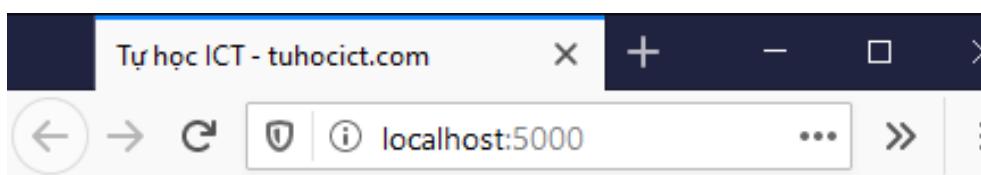
2.6. Model class trong Razor Pages

• Model class

• Thực hành 2: tiếp tục sử dụng project từ phần thực hành 1-2

• Bước 3: Xóa hết code cũ (nếu có) của *Index.cshtml* và viết lại code như sau:

```
1. @page
2.
3. @model WebApplication.Pages.IndexModel
4.
5. <div>
6.     <h2>@Model.SayGoodbye()</h2>
7.     <div>
8.         <div><label>First name:</label>@Model.FirstName</div>
9.         <div><label>Last name:</label>@Model.LastName</div>
10.    </div>
11. </div>
```



Razor Pages tutorial

Hello, Donald Trump! I'm in the model class

First name:Donald
Last name:Trump

2.1. Razor Pages: giới thiệu, cài đặt, project, page, URL

2.2. Layout, ViewStart, section trong ASP.NET Core

2.3. Routing trong Razor Pages – ánh xạ URL sang page

2.4. Cú pháp Razor cơ bản, biểu thức và khối code

2.5. Các cấu trúc điều khiển của Razor, directive, ViewImports

2.6. Model class trong Razor Pages

• Khái niệm model class

- Trong phần thực hành trên chúng ta đã thực hiện viết code theo mô hình **page model class**
- Lớp **IndexModel** mà bạn đã viết trong file **Index.cshtml.cs** được gọi là **model class** của page **Index.cshtml**.
- **Model class** cho phép bạn tách rời toàn bộ data logic của page ra một class riêng. Trên page giờ chỉ còn lại **UI logic**. Code trở nên sáng sủa, dễ đọc, dễ quản lý hơn rất nhiều.

2.1. Razor Pages: giới thiệu, cài đặt, project, page, URL

2.2. Layout, ViewStart, section trong ASP.NET Core

2.3. Routing trong Razor Pages – ánh xạ URL sang page

2.4. Cú pháp Razor cơ bản, biểu thức và khối code

2.5. Các cấu trúc điều khiển của Razor, directive, ViewImports

2.6. Model class trong Razor Pages

• Sử dụng model class

- **Ưu điểm của model class:** Mục đích chính của **model class** là tạo ra sự phân chia rõ ràng giữa giao diện và xử lý logic/dữ liệu của trang. Sự phân chia này đem lại một loạt ưu điểm:
 - (1) làm giảm sự phức tạp của giao diện và dễ bảo trì: giờ đây giao diện chỉ chịu trách nhiệm hiển thị thông tin. Các logic trên giao diện là logic phục vụ hiển thị thông tin. Những xử lý khác như dữ liệu được thực hiện ở **model class**. Giao diện trở nên đơn giản và dễ bảo trì.
 - (2) giúp thực hiện unit test: model class là một class C# bình thường được xây dựng dựa trên cơ chế Dependency Injection, rất tiện lợi cho test tự động.
 - (3) giúp phát triển song song trong team: do logic và UI được tách rời, team có thể dễ dàng phân chia công việc và đồng thời thực hiện.
 - (4) tạo ra các khối code nhỏ dễ dàng tái sử dụng.

2.1. Razor Pages: giới thiệu, cài đặt, project, page, URL

2.2. Layout, ViewStart, section trong ASP.NET Core

2.3. Routing trong Razor Pages – ánh xạ URL sang page

2.4. Cú pháp Razor cơ bản, biểu thức và khối code

2.5. Các cấu trúc điều khiển của Razor, directive, ViewImports

2.6. Model class trong Razor Pages

• Sử dụng model class

- **Đặc điểm của model class:** Khi sử dụng page model class cần lưu ý những điểm sau:

- (1) Model class phải kế thừa từ lớp **PageModel** (*Microsoft.AspNetCore.Mvc.RazorPages.PagesModel*). Ngoài ra không có giới hạn gì khác.
- (2) Để chỉ định một class trở thành model class cho một page, bạn phải sử dụng directive **@model <tên_class>**

```
1. @model WebApplication.Pages.IndexModel
```

- (3) Một khi đã chỉ định model class, bạn có thể truy xuất object của nó thông qua property **Model** của *page class* ở bất kỳ vị trí nào trong page.

```
1. <div>
2.   <h2>@Model.SayGoodbye()</h2>
3.   <div>
4.     <div><label>First name:</label>@Model.FirstName</div>
5.     <div><label>Last name:</label>@Model.LastName</div>
6.   </div>
7. </div>
```

2.1. Razor Pages: giới thiệu, cài đặt, project, page, URL

2.2. Layout, ViewStart, section trong ASP.NET Core

2.3. Routing trong Razor Pages – ánh xạ URL sang page

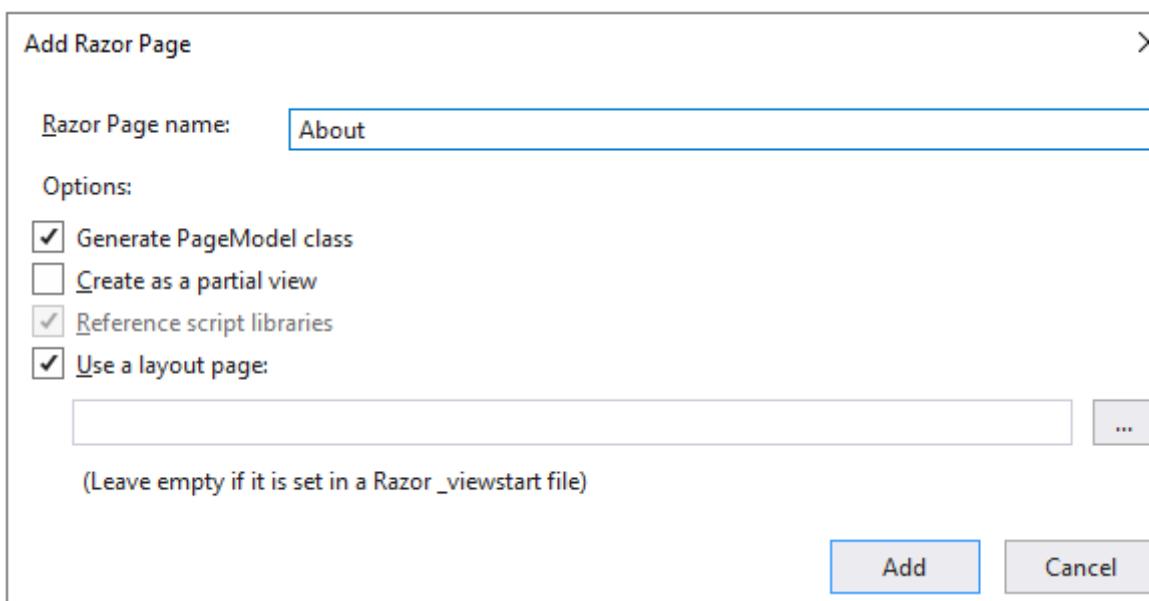
2.4. Cú pháp Razor cơ bản, biểu thức và khối code

2.5. Các cấu trúc điều khiển của Razor, directive, ViewImports

2.6. Model class trong Razor Pages

• Một số lưu ý khác

- (1) Mặc dù không có quy định bắt buộc nhưng tên model class quy ước đặt theo tên page + Model. Ví dụ, model class của trang **Index** sẽ là **IndexModel**.
- (2) Tương tự, model class thường đặt trong file cùng tên với page nhưng có đuôi cs. Ví dụ, page Index nằm trong file Index.cshtml thì model class sẽ để trong file Index.cshtml.cs. Cách đặt tên này giúp Visual Studio hiển thị ghép các file vào cùng một node (gọi là file nesting). File nesting giúp quản lý file dễ dàng hơn.
- (3) Visual Studio hỗ trợ tạo page với model class trong hộp thoại Add Razor Page như sau:



2.1. Razor Pages: giới thiệu, cài đặt, project, page, URL

2.2. Layout, ViewStart, section trong ASP.NET Core

2.3. Routing trong Razor Pages – ánh xạ URL sang page

2.4. Cú pháp Razor cơ bản, biểu thức và khối code

2.5. Các cấu trúc điều khiển của Razor, directive, ViewImports

2.6. Model class trong Razor Pages

• ViewData và model class

• ViewData object

- **ViewData** là một object thuộc kiểu **Dictionary** (chính xác hơn, là kiểu từ điển đặc biệt **ViewDataDictionary**) được cơ chế **Razor Pages** tạo sẵn và có tác dụng trên toàn bộ ứng dụng. Nghĩa là ở bất kỳ đâu trong ứng dụng bạn đều có thể truy xuất được object này.
- **ViewData** có thể lưu trữ object thuộc bất kỳ kiểu dữ liệu nào nhưng khóa của nó phải thuộc kiểu **string** (để có thể truy xuất từ các file **cshtml**).
- Dưới đây là một ví dụ về lưu trữ các cặp khóa/giá trị với ViewData:

```
1. // Index.cshtml.cs
2. public class IndexModel : PageModel {
3.     public void OnGet() {
4.         ViewData["MyNumber"] = 2020;
5.         ViewData["MyString"] = "Hello World, Razor Pages!";
6.         ViewData["MyObject"] = new Book {
7.             Title = "ASP.NET Core for Dummy",
8.             Publisher = "Tự học ICT Press",
9.             Author = "Mai Chi"
10.        };
11.    }
12. }
```

```
1. @page
2. @model IndexModel
3. @{
4.     // phải cast kiểu object sang kiểu cụ thể
5.     var book = (Book) ViewData["MyObject"];
6. }
7.
8. <h2>@ViewData["MyString"]</h2>
9. <p>The answer to everything is @ViewData["MyNumber"]</p>
10.
11. <h2>@book.Title</h2>
12. <p>@book.Author</p>
13. <p>@book.Publisher</p>
```

2.1. Razor Pages: giới thiệu, cài đặt, project, page, URL

2.2. Layout, ViewStart, section trong ASP.NET Core

2.3. Routing trong Razor Pages – ánh xạ URL sang page

2.4. Cú pháp Razor cơ bản, biểu thức và khối code

2.5. Các cấu trúc điều khiển của Razor, directive, ViewImports

2.6. Model class trong Razor Pages

• ViewData và model class

• ViewData object

- Có thể truy xuất **ViewData** trong page Index.cshtml như sau:

```
1. @page
2. @model IndexModel
3. @{
4.     // phải cast kiểu object sang kiểu cụ thể
5.     var book = (Book) ViewData["MyObject"];
6. }
7.
8. <h2>@ViewData["MyString"]</h2>
9. <p>The answer to everything is @ViewData["MyNumber"]</p>
10.
11. <h2>@book.Title</h2>
12. <p>@book.Author</p>
13. <p>@book.Publisher</p>
```

2.1. Razor Pages: giới thiệu, cài đặt, project, page, URL

2.2. Layout, ViewStart, section trong ASP.NET Core

2.3. Routing trong Razor Pages – ánh xạ URL sang page

2.4. Cú pháp Razor cơ bản, biểu thức và khối code

2.5. Các cấu trúc điều khiển của Razor, directive, ViewImports

2.6. Model class trong Razor Pages

• ViewData và model class

• ViewData attribute

- Khi sử dụng mô hình model class bạn còn có một cách khác nữa để làm việc với ViewData. Hãy cùng xem ví dụ nhỏ sau:

```
1. public class IndexModel : PageModel
2. {
3.     [ViewData]
4.     public string Message { get; set; }
5.     public void OnGet()
6.     {
7.         Message = "Hello World";
8.     }
9. }
```

- Chú ý attribute **[ViewData]** phía trước khai báo property **Message**
- Giờ đây có thể sử dụng **ViewData["Message"]** ở bất kỳ đâu.

```
1. @page
2. @model IndexModel
3.
4. <h2>@Model.Message</h2>
5. <h2>@ViewData["Message"]</h2>
```

- Cách sử dụng này có tên gọi **ViewData attribute**

- 2.1. Razor Pages: giới thiệu, cài đặt, project, page, URL
- 2.2. Layout, ViewStart, section trong ASP.NET Core
- 2.3. Routing trong Razor Pages – ánh xạ URL sang page
- 2.4. Cú pháp Razor cơ bản, biểu thức và khối code
- 2.5. Các cấu trúc điều khiển của Razor, directive, ViewImports
- 2.6. Model class trong Razor Pages
- 2.7. Handler và Action Results trong Razor Pages

- **Handler** – phương thức xử lý truy vấn – trong **Razor Pages** là những phương thức được tự động chạy khi có truy vấn (tới từ trình duyệt hoặc chương trình client khác).
- **Handler** là thành phần không thể thiếu nếu bạn cần xử lý các yêu cầu từ người dùng. Đây cũng là cách **Razor Pages** tạo ra cái tương tự như mô hình sự kiện giúp đơn giản hóa xử lý tương tác với người dùng.

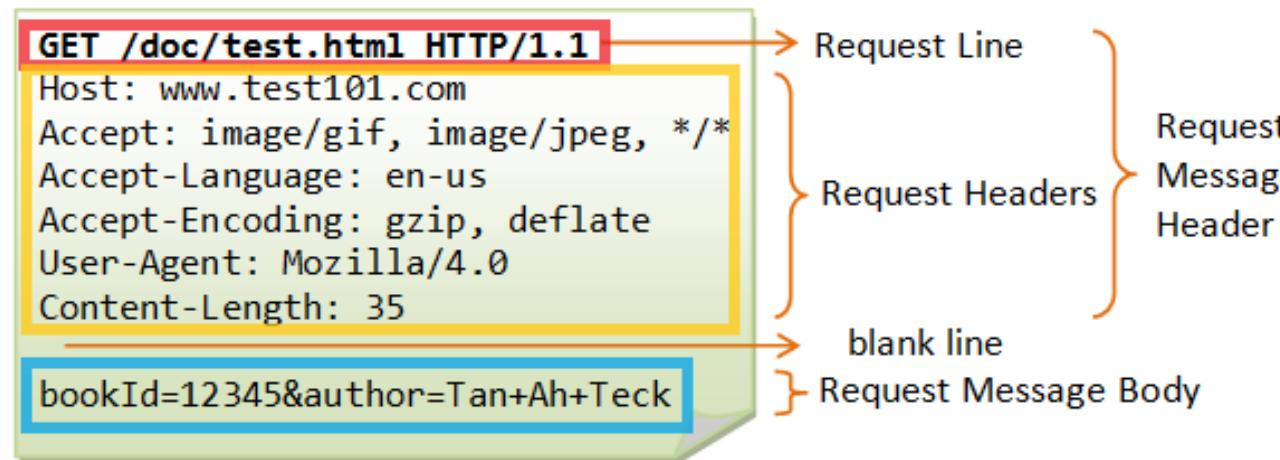
- 2.1. Razor Pages: giới thiệu, cài đặt, project, page, URL
- 2.2. Layout, ViewStart, section trong ASP.NET Core
- 2.3. Routing trong Razor Pages – ánh xạ URL sang page
- 2.4. Cú pháp Razor cơ bản, biểu thức và khối code
- 2.5. Các cấu trúc điều khiển của Razor, directive, ViewImports
- 2.6. Model class trong Razor Pages
- 2.7. Handler và Action Results trong Razor Pages

• Truy vấn HTTP

- **HTTP** là một giao thức dạng văn bản, nghĩa là dữ liệu của nó thuần túy là chuỗi văn bản (rất dài) được định dạng theo quy định thống nhất. Chuỗi trình duyệt gửi server gọi là **HTTP Request** (truy vấn); Chuỗi server gửi lại trình duyệt gọi là **HTTP Response** (phản hồi)
- Chuỗi truy vấn được trình duyệt tạo ra mỗi khi nhập **URL** vào thanh địa chỉ của trình duyệt, hoặc khi click vào nút **Submit** trên form. Ngoài ra chuỗi truy vấn cũng có thể được tạo ra tự động bởi chương trình.

- 2.1. Razor Pages: giới thiệu, cài đặt, project, page, URL
- 2.2. Layout, ViewStart, section trong ASP.NET Core
- 2.3. Routing trong Razor Pages – ánh xạ URL sang page
- 2.4. Cú pháp Razor cơ bản, biểu thức và khối code
- 2.5. Các cấu trúc điều khiển của Razor, directive, ViewImports
- 2.6. Model class trong Razor Pages
- 2.7. Handler và Action Results trong Razor Pages

• Truy vấn HTTP



- Dòng trên cùng (đánh dấu đỏ) gọi là **dòng truy vấn** (**request line**). Trong dòng này có hai thông tin quan trọng là **phương thức** (**verb**) và **URL** (**Uniform Resource Locator**).
- Phương thức thường gặp bao gồm **Get**, **Post**, **Put**, **Delete**, **Patch** (và một số khác nữa) chỉ định loại hành động. **Url** chỉ định đối tượng chịu tác động của phương thức. **Verb** và **Url** là hai thông tin giúp server hiểu client muốn cái gì.
- Trong các truy vấn, **Get** và **Post** là hai loại **verb** được sử dụng rộng rãi nhất. **Put**, **Delete** và **Patch** thường gặp trong các **Restful API**. Truy vấn **GET** tạo ra khi nhập **URL** vào thanh địa chỉ trình duyệt. Truy vấn **POST** tạo ra khi bạn **submit form**. Các truy vấn **Put**, **Delete**, **Patch** thường tạo ra bằng chương trình.
- Mỗi loại truy vấn sẽ đi kèm với dữ liệu và yêu cầu xử lý riêng.

- 2.1. Razor Pages: giới thiệu, cài đặt, project, page, URL
- 2.2. Layout, ViewStart, section trong ASP.NET Core
- 2.3. Routing trong Razor Pages – ánh xạ URL sang page
- 2.4. Cú pháp Razor cơ bản, biểu thức và khối code
- 2.5. Các cấu trúc điều khiển của Razor, directive, ViewImports
- 2.6. Model class trong Razor Pages
- 2.7. Handler và Action Results trong Razor Pages

• Handler mặc định trong Razor Pages

- **Khái niệm Handler:** Trong Razor Pages, mặc định mỗi khi truy vấn tới, Razor Pages căn cứ vào loại truy vấn (verb) để tìm kiếm phương thức có tên tương ứng và tự thực thi:
 - **OnGet()** – đối với truy vấn GET;
 - **OnPost()** – đối với truy vấn POST;
 - **OnPut()** – đối với truy vấn PUT;
 -
- Các phương thức này được gọi chung là *phương thức xử lý truy vấn*, hoặc **handler**
- Có thể hình dung **handler** là công cụ giúp bạn can thiệp vào việc xử lý page **của Razor Pages**.

2.1. Razor Pages: giới thiệu, cài đặt, project, page, URL

2.2. Layout, ViewStart, section trong ASP.NET Core

2.3. Routing trong Razor Pages – ánh xạ URL sang page

2.4. Cú pháp Razor cơ bản, biểu thức và khối code

2.5. Các cấu trúc điều khiển của Razor, directive, ViewImports

2.6. Model class trong Razor Pages

2.7. Handler và Action Results trong Razor Pages

• Thực hành 1

- **Bước 1:** Tạo một project Web Application
- **Bước 2:** Viết code cho *Index.cshtml* như sau:

```
1. @page
2. @addTagHelper *, Microsoft.AspNetCore.Mvc.TagHelpers
3.
4. <h3>@Message</h3>
5. <form method="post">
6.     <p><button class="btn btn-primary" type="submit">Send Post Request</button></p>
7. </form>
8. <p><a href="/" class="btn btn-primary">Send GET request</a></p>
9.
0. @functions{
1.     public string Message { get; set; }
2.     public void OnGet() {
3.         Message = "This is a GET request";
4.     }
5.     public void OnPost() {
6.         Message = "This is a POST request";
7.     }
8. }
```

- Trong đoạn mã trên, có thẻ **form** – loại thẻ html cho phép gửi dữ liệu về server. **Form** trong ví dụ trên chỉ chứa duy nhất một **nút bấm (button)** với nhiệm vụ gửi truy vấn post về server.

2.1. Razor Pages: giới thiệu, cài đặt, project, page, URL

2.2. Layout, ViewStart, section trong ASP.NET Core

2.3. Routing trong Razor Pages – ánh xạ URL sang page

2.4. Cú pháp Razor cơ bản, biểu thức và khối code

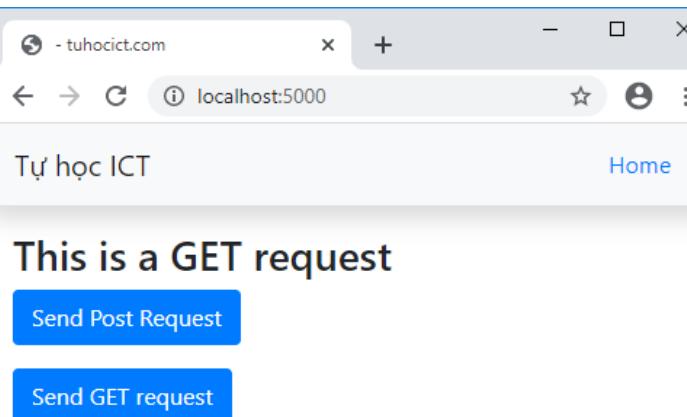
2.5. Các cấu trúc điều khiển của Razor, directive, ViewImports

2.6. Model class trong Razor Pages

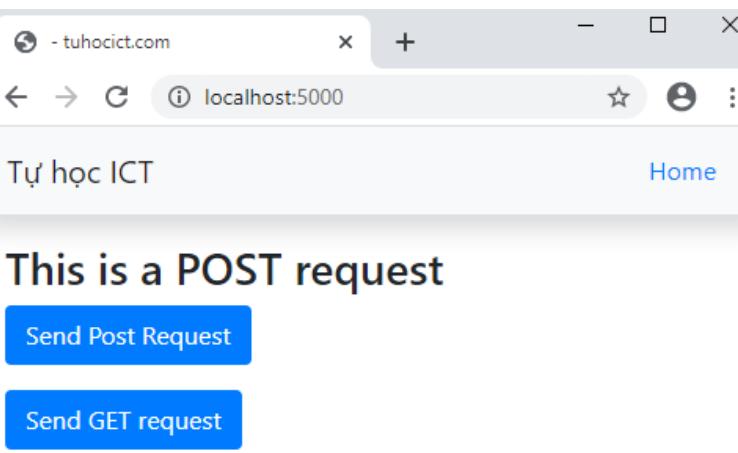
2.7. Handler và Action Results trong Razor Pages

• Thực hành 1

- Khi site được tải lần thứ nhất, phương thức **OnGet()** được thực thi nên dòng thông báo (từ property Message) có giá trị “This is a GET request”:



- Nếu bạn bấm nút “Send Post Request”, phương thức **OnPost()** được thực thi và gán giá trị mới cho property Message:



- Bấm nút Send GET request sẽ lại gửi truy vấn GET.

- 2.1. Razor Pages: giới thiệu, cài đặt, project, page, URL
- 2.2. Layout, ViewStart, section trong ASP.NET Core
- 2.3. Routing trong Razor Pages – ánh xạ URL sang page
- 2.4. Cú pháp Razor cơ bản, biểu thức và khối code
- 2.5. Các cấu trúc điều khiển của Razor, directive, ViewImports
- 2.6. Model class trong Razor Pages
- 2.7. Handler và Action Results trong Razor Pages

- Thực hành 2: Làm lại ví dụ trên nhưng giờ sử dụng model class:

```
Index.cshtml.cs Index.cshtml
1. using Microsoft.AspNetCore.Mvc.RazorPages;
2.
3. namespace WebApplication1.Pages {
4.     public class IndexModel : PageModel {
5.         public string Message { get; set; }
6.         public void OnGet() {
7.             Message = "This is a GET request";
8.         }
9.         public void OnPost() {
10.             Message = "This is a POST request";
11.         }
12.     }
13. }
```

```
Index.cshtml.cs Index.cshtml
1. @page
2. @model WebApplication1.Pages.IndexModel
3. @addTagHelper *, Microsoft.AspNetCore.Mvc.TagHelpers
4.
5. <h3>@Model.Message</h3>
6. <form method="post">
7.     <p><button class="btn btn-primary" type="submit">Send Post Request</button></p>
8. </form>
9. <p><a href="/" class="btn btn-primary">Send GET request</a></p>
```

- Sẽ thu được cùng kết quả như khi sử dụng functions block.

- 2.1. Razor Pages: giới thiệu, cài đặt, project, page, URL
- 2.2. Layout, ViewStart, section trong ASP.NET Core
- 2.3. Routing trong Razor Pages – ánh xạ URL sang page
- 2.4. Cú pháp Razor cơ bản, biểu thức và khối code
- 2.5. Các cấu trúc điều khiển của Razor, directive, ViewImports
- 2.6. Model class trong Razor Pages
- 2.7. Handler và Action Results trong Razor Pages

• Một số lưu ý:

- (1) Phương thức handler có thể viết trong **functions block** hoặc **model class**. Nhìn chung khi dùng **handler** bạn nên viết trong **model class**. Đây là mô hình được khuyến nghị trong **Razor Pages**.
- (2) Phương thức handler phải có mức truy cập là **public**. Nên đặc biệt lưu ý điều này nếu gặp tình huống phương thức không được kích hoạt như dự định.
- (3) Phương thức handler có thể trả về kết quả thuộc bất kỳ kiểu dữ liệu nào. Thông thường kiểu trả về là **void** (hoặc **Task** nếu bạn sử dụng bản **Async**). **ActionResult** và **IActionResult** cũng là kiểu trả về thường gặp.
- (4) **Razor Pages** phân biệt phương thức **handler** căn cứ vào **tên phương thức** chứ không quan tâm đến danh sách tham số.
 - **Ví dụ**, khi bạn viết hai phương thức **OnGet** và **OnGet(string) str** trong cùng **model class**, compiler sẽ đồng ý (vẫn dịch thành công) nhưng khi chạy Razor sẽ báo lỗi.
 - Cơ chế **handler** mặc định cho phép bạn phản ứng với **từng loại truy vấn**. Nếu cần phản ứng với **từng truy vấn** cụ thể, bạn cần đến một cơ chế khác: **named handler**.

- 2.1. Razor Pages: giới thiệu, cài đặt, project, page, URL
- 2.2. Layout, ViewStart, section trong ASP.NET Core
- 2.3. Routing trong Razor Pages – ánh xạ URL sang page
- 2.4. Cú pháp Razor cơ bản, biểu thức và khôi code
- 2.5. Các cấu trúc điều khiển của Razor, directive, ViewImports
- 2.6. Model class trong Razor Pages
- 2.7. Handler và Action Results trong Razor Pages

- **Named handler – định danh phương thức xử lý truy vấn:**
 - Nếu sử dụng **handler** mặc định, trên mỗi page bạn chỉ có thể viết một **handler** cho mỗi **loại truy vấn**. Nó có nghĩa là **Razor Pages** đối xử như nhau với tất cả truy vấn **Get**, không quan tâm đến các thông tin khác trong truy vấn. Tình hình xảy ra tương tự với các truy vấn Post. Thông thường yêu cầu này đáp ứng tốt cho đa số site.
 - Tuy nhiên, trong một số trường hợp bạn muốn có **nhiều handler** cho **cùng một verb**
 - Ví dụ, mặc dù cùng là truy vấn **GET**, nhưng nếu **GET** đến từ nút bấm **A** sẽ được đối xử khác với **GET** đến từ nút bấm **B**.
 - Hoặc lấy ví dụ khác, nếu trên page bạn có 3 **form** cho 3 mục đích khác nhau, bạn sẽ cần đến 3 **handler** để tránh trộn lẫn chức năng. Khi đó bạn sẽ cần đến nhiều **handler** cho cùng một **verb** trên cùng một **model class**.
 - **Razor Pages** hỗ trợ khả năng này thông qua một tính năng có tên gọi là **named handler – handler có định danh**.

2.1. Razor Pages: giới thiệu, cài đặt, project, page, URL

2.2. Layout, ViewStart, section trong ASP.NET Core

2.3. Routing trong Razor Pages – ánh xạ URL sang page

2.4. Cú pháp Razor cơ bản, biểu thức và khối code

2.5. Các cấu trúc điều khiển của Razor, directive, ViewImports

2.6. Model class trong Razor Pages

2.7. Handler và Action Results trong Razor Pages

• Thực hành 3:

- Bước 1: Tạo thêm page *Multiform.cshtml* với *model class*.
- Bước 2: Viết code cho *Multiform* như sau:

The screenshot shows a dual-pane code editor. The left pane displays the code for *Multiform.cshtml.cs*, and the right pane displays the code for *Multiform.cshtml*. Both panes have line numbers from 1 to 21.

```
using Microsoft.AspNetCore.Mvc;
namespace WebApplication1.Pages
public class MultiformModel
{
    public string Message {
        get; set;
    }
    public void OnGet() {
    }
    public void OnPost() {
        Message = "Form Post";
    }
    public void OnPostDelete() {
        Message = "Delete han";
    }
    public void OnPostEdit() {
        Message = "Edit han";
    }
    public void OnPostView() {
        Message = "View han";
    }
}
```

```
@page
@addTagHelper *, Microsoft.AspNetCore.Mvc.TagHelpers
@model WebApplication1.Pages.MultiformModel
 @{
     ViewData["Title"] = "Multiform";
 }

<div class="row">
    <div class="col-sm">
        <form asp-page-handler="edit" method="post">
            <button class="btn btn-outline-primary">Edit</button>
        </form>
    </div>
    <div class="col-sm">
        <form asp-page-handler="delete" method="post">
            <button class="btn btn-outline-success">Delete</button>
        </form>
    </div>
    <div class="col-sm">
        <form asp-page-handler="view" method="post">
            <button class="btn btn-outline-dark">View</button>
        </form>
    </div>
</div>
<hr />
<h3 class="clearfix">@Model.Message</h3>
```

2.1. Razor Pages: giới thiệu, cài đặt, project, page, URL

2.2. Layout, ViewStart, section trong ASP.NET Core

2.3. Routing trong Razor Pages – ánh xạ URL sang page

2.4. Cú pháp Razor cơ bản, biểu thức và khối code

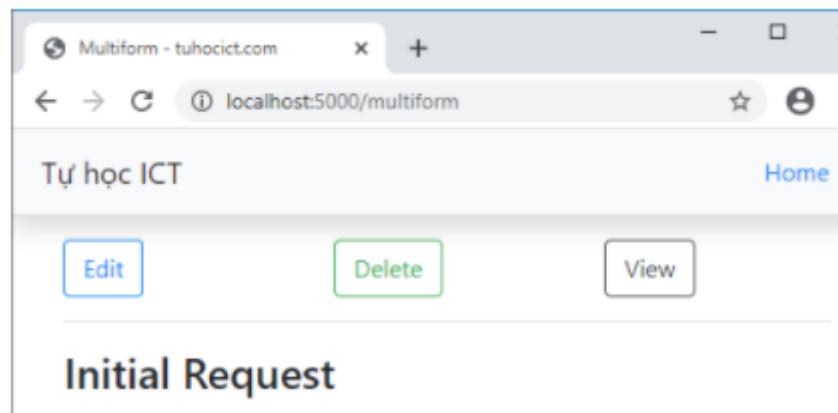
2.5. Các cấu trúc điều khiển của Razor, directive, ViewImports

2.6. Model class trong Razor Pages

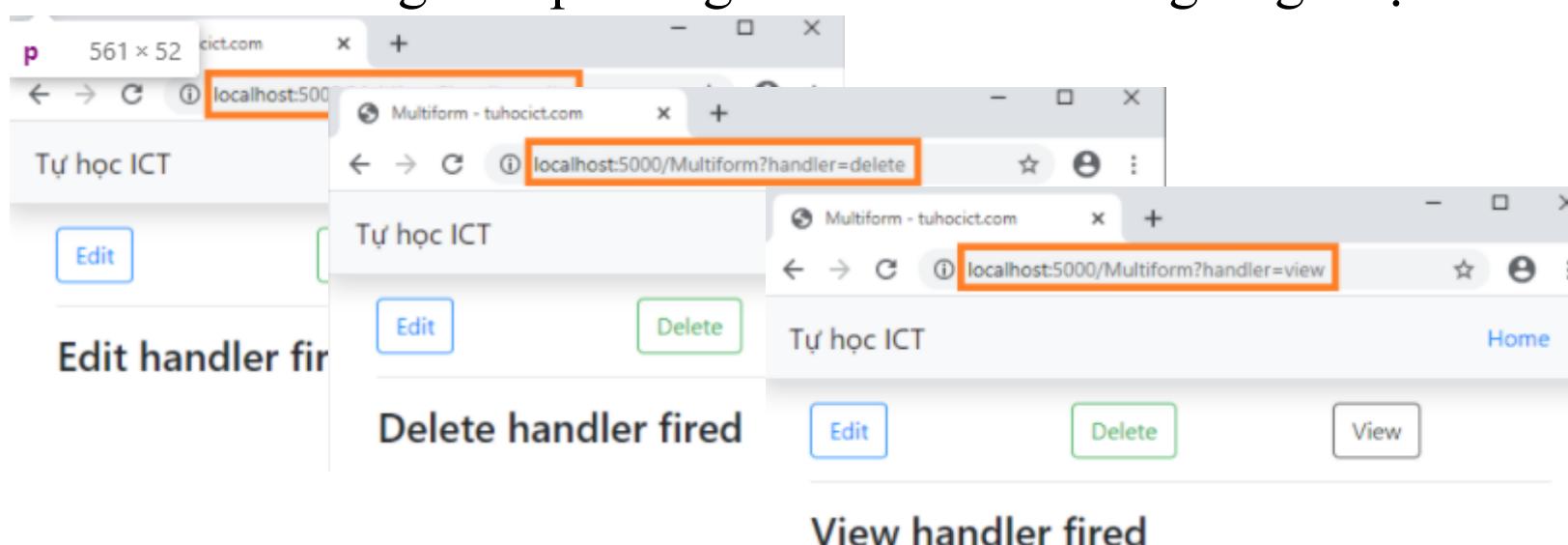
2.7. Handler và Action Results trong Razor Pages

• Thực hành 3:

- Bạn thu được kết quả từ lần chạy đầu tiên như sau:



- Click vào các nút, từng form sẽ lần lượt gửi truy vấn post riêng về server. Đồng thời phương thức handler tương ứng được kích hoạt.



2.1. Razor Pages: giới thiệu, cài đặt, project, page, URL

2.2. Layout, ViewStart, section trong ASP.NET Core

2.3. Routing trong Razor Pages – ánh xạ URL sang page

2.4. Cú pháp Razor cơ bản, biểu thức và khối code

2.5. Các cấu trúc điều khiển của Razor, directive, ViewImports

2.6. Model class trong Razor Pages

2.7. Handler và Action Results trong Razor Pages

• Thực hành 3:

• Cơ chế named handler

- Trong file *Multiform.cshtml* để ý một **attribute** lạ trong thẻ form: *asp-page-handler*
- Đây không phải là **attribute** tiêu chuẩn của html form mà là một **tag helper** của Razor Pages. *asp-page-handler* giúp chỉ định phần tên của handler sẽ được kích hoạt để xử lý truy vấn post tương ứng:
 - OnPostDelete** xử lý truy vấn post đến từ form có *asp-page-handler="delete"*;
 - OnPostEdit** xử lý truy vấn post đến từ form có *asp-page-handler = "edit"*;
 - OnPostView** xử lý truy vấn post đến từ form có *asp-page-handler="view"*.
- Nếu mở cửa sổ developer của trình duyệt (F12) thì sẽ thấy kết quả như sau:

```
▼<div class="row">
  ▼<div class="col-sm">
    ►<form method="post" action="/Multiform?handler=edit">...</form>
  </div>
  ▼<div class="col-sm">
    ►<form method="post" action="/Multiform?handler=delete">...</form>
  </div>
  ▼<div class="col-sm">
    ►<form method="post" action="/Multiform?handler=view">...</form>
  </div>
</div>
```

- Đây là cách *asp-page-handler* chuyển thành **HTML**.
- Hãy để ý đến chuỗi truy vấn trong **Url(?handler=...)**. Đây là cách **Razor Pages** phân biệt giữa nhiều handler có tên khác nhau.

2.1. Razor Pages: giới thiệu, cài đặt, project, page, URL

2.2. Layout, ViewStart, section trong ASP.NET Core

2.3. Routing trong Razor Pages – ánh xạ URL sang page

2.4. Cú pháp Razor cơ bản, biểu thức và khối code

2.5. Các cấu trúc điều khiển của Razor, directive, ViewImports

2.6. Model class trong Razor Pages

2.7. Handler và Action Results trong Razor Pages

• Thực hành 3:

• Cơ chế named handler

- Nếu không muốn sử dụng chuỗi truy vấn trong Url, bạn có thể sử dụng directive `@page "{handler?}"` trong file *Multiform.cshtml*. Khi này Url sẽ chuyển thành:

① <localhost:5000/Multiform/edit>

① <localhost:5000/Multiform/delete>

① <localhost:5000/Multiform/view>

- 2.1. Razor Pages: giới thiệu, cài đặt, project, page, URL
- 2.2. Layout, ViewStart, section trong ASP.NET Core
- 2.3. Routing trong Razor Pages – ánh xạ URL sang page
- 2.4. Cú pháp Razor cơ bản, biểu thức và khối code
- 2.5. Các cấu trúc điều khiển của Razor, directive, ViewImports
- 2.6. Model class trong Razor Pages
- 2.7. Handler và Action Results trong Razor Pages

• Tham số của handler

- Các truy vấn tới thông thường đều chứa tham số. Các tham số này có thể diễn đạt bằng những cách khác nhau tùy thuộc vào loại truy vấn.
- Đối với truy vấn **GET** thường có hai cách cung cấp tham số:
 - thông qua chuỗi truy vấn (query string) của URL;
 - thông qua route data (là các segment của chính URL).
- Đối với truy vấn **POST**, tham số thường được chứa trong phần thân của truy vấn **HTTP**.
- Khi xử lý truy vấn, **handler** cũng có thể (và cần phải) truy xuất được các tham số này.
- **Razor Pages** cho phép các **handler** có tham số tùy ý, cả kiểu phức tạp (class) lẫn các kiểu cơ sở (string, int, bool, v.v.).
- Khi truy vấn tới, **Razor Pages** sẽ tự động truyền tham số của truy vấn sang tham số của **handler** nếu chúng trùng tên nhau.
- Đối với tham số **handler** có kiểu phức tạp, **Razor Pages** cũng hỗ trợ tự động hóa việc chuyển đổi này, giúp người lập trình tiện lợi hơn khi đọc dữ liệu người dùng.

- 2.1. Razor Pages: giới thiệu, cài đặt, project, page, URL
- 2.2. Layout, ViewStart, section trong ASP.NET Core
- 2.3. Routing trong Razor Pages – ánh xạ URL sang page
- 2.4. Cú pháp Razor cơ bản, biểu thức và khối code
- 2.5. Các cấu trúc điều khiển của Razor, directive, ViewImports
- 2.6. Model class trong Razor Pages
- 2.7. Handler và Action Results trong Razor Pages

• Action Result và handler

- Đôi khi muốn sau khi thực hiện xong handler thì sẽ chuyển hướng sang một trang khác chứ không tiếp tục tải trang hiện tại nữa. Bạn cũng có thể muốn trả lại mã lỗi cho trình duyệt, giả sử, khi người dùng không có quyền truy xuất tài nguyên. Bạn cũng có thể cần phải trả lại kết quả là một chuỗi html sinh ra từ **partial page** cho truy vấn **AJAX**.
- Những tình huống tương tự dẫn tới nhu cầu tạo ra các kiểu trả về đặc biệt cho **handler**. Trong **Asp.net Core**, các kiểu trả về đặc biệt như thế được gọi chung là **action result**.
- Mọi **action result** trong Asp.net Core đều được xây dựng từ lớp abstract **ActionResult** hoặc thực thi giao diện **IActionResult**.
- **Ví dụ**, nếu muốn điều hướng sang một trang khác khi kết thúc thực thi handler, bạn thực hiện như sau:

```
public IActionResult OnGet() {  
    // code xử lý khác  
  
    return new RedirectToPageResult("Index");  
}
```

- 2.1. Razor Pages: giới thiệu, cài đặt, project, page, URL
- 2.2. Layout, ViewStart, section trong ASP.NET Core
- 2.3. Routing trong Razor Pages – ánh xạ URL sang page
- 2.4. Cú pháp Razor cơ bản, biểu thức và khối code
- 2.5. Các cấu trúc điều khiển của Razor, directive, ViewImports
- 2.6. Model class trong Razor Pages
- 2.7. Handler và Action Results trong Razor Pages

- **Mô hình sự kiện trên Razor Pages**

- Từ cơ chế **named handler**, có thể rút ra một ứng dụng rất mạnh của nó: tạo nhiều nút bấm/link, mỗi nút bấm/link sẽ được xử lý bằng một phương thức riêng.
- Điều này giúp viết ứng dụng **Razor Pages** đơn giản hơn và gần gũi với mô hình **xử lý sự kiện** trong Winforms hay WPF.

2.1. Razor Pages: giới thiệu, cài đặt, project, page, URL

2.2. Layout, ViewStart, section trong ASP.NET Core

2.3. Routing trong Razor Pages – ánh xạ URL sang page

2.4. Cú pháp Razor cơ bản, biểu thức và khối code

2.5. Các cấu trúc điều khiển của Razor, directive, ViewImports

2.6. Model class trong Razor Pages

2.7. Handler và Action Results trong Razor Pages

• Thực hành 4:

- **Bước 1:** Tạo thêm page Multiaction
- **Bước 2:** Viết code cho model class trong file *Multiaction.cshtml.cs*:

```
1.  using Microsoft.AspNetCore.Mvc.RazorPages;  
2.  
3.  namespace WebApplication1.Pages {  
4.      public class MultiactionModel : PageModel {  
5.          public string Message { get; set; } = "Default GET method";  
6.          public void OnPostRegister() => Message = "Post Register fired!";  
7.          public void OnPostRequestinfo() => Message = "Post Request Info fired!";  
8.          public void OnGetClear() => Message = "Get Clear fired!";  
9.          public void OnGetReset() => Message = "Get Reset fired!";  
10.     }  
11. }
```

- **Bước 3:** Viết code cho trang *Multiaction.cshtml* như sau:

```
1.  @page "{handler?}"  
2.  @addTagHelper *, Microsoft.AspNetCore.Mvc.TagHelpers  
3.  @model WebApplication1.Pages.MultiactionModel  
4.  {@  
5.      ViewData["Title"] = "Multiaction";  
6.  }  
7.  
8.  <form method="post">  
9.      <button asp-page-handler="Register">Register Now</button>  
10.     <button asp-page-handler="Requestinfo">Request Info</button>  
11.  </form>  
12.  <a class="btn btn-danger" asp-page-handler="clear">Clear All</a>  
13.  <a class="btn btn-info" asp-page-handler="reset">Reset All</a>  
14.  <h3>@Model.Message</h3>
```

- 2.1. Razor Pages: giới thiệu, cài đặt, project, page, URL
- 2.2. Layout, ViewStart, section trong ASP.NET Core
- 2.3. Routing trong Razor Pages – ánh xạ URL sang page
- 2.4. Cú pháp Razor cơ bản, biểu thức và khôi code
- 2.5. Các cấu trúc điều khiển của Razor, directive, ViewImports
- 2.6. Model class trong Razor Pages
- 2.7. Handler và Action Results trong Razor Pages

• Thực hành 4:

- Hãy để ý đến cách chúng ta sử dụng tag **helper *asp-page-handler*** trong thẻ **button** hoặc **a**, thay vì dùng trong thẻ form như ở bài thực hành 3. Tuy vậy nó cũng có cùng một tác dụng chỉ định:
 - *OnPostRegister()* xử lý phương thức **Post** khi click nút có **asp-page-handler = "Register"**
 - *OnPostRequestinfo()* xử lý phương thức **Post** khi click nút có **asp-page-handler = "Requestinfo"**
 - *OnGetClear()* xử lý phương thức **Get** khi click link có **asp-page-handler = "Clear"**
 - *OnGetReset()* xử lý phương thức **Get** khi click link có **asp-page-handler = "Reset"**
- Directive **@page "{handler?}"** cũng giúp chúng ta có url “đẹp” (không có chuỗi truy vấn).

Chương 2. ASP.NET Core Razor Pages

2.1. Razor Pages: giới thiệu, cài đặt, project, page, URL

2.2. Layout, ViewStart, section trong ASP.NET Core

2.3. Routing trong Razor Pages – ánh xạ URL sang page

2.4. Cú pháp Razor cơ bản, biểu thức và khối code

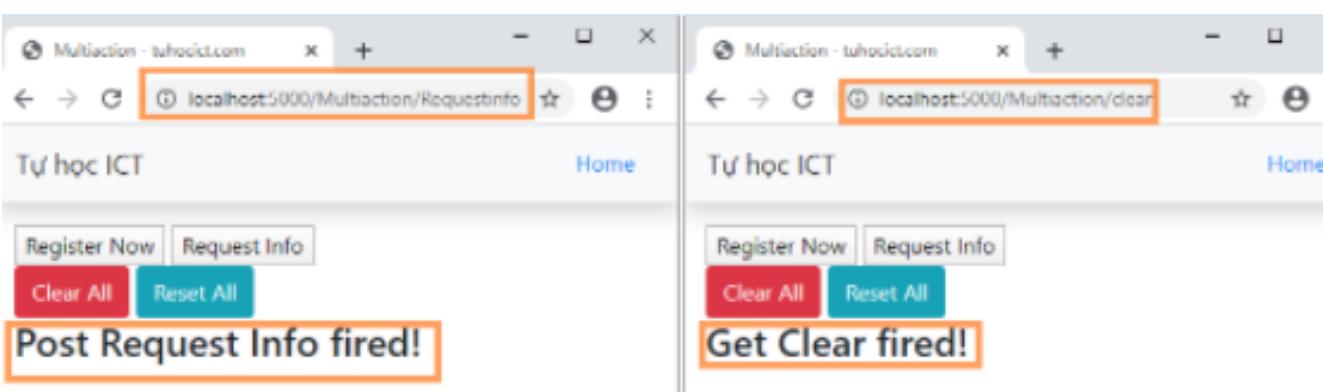
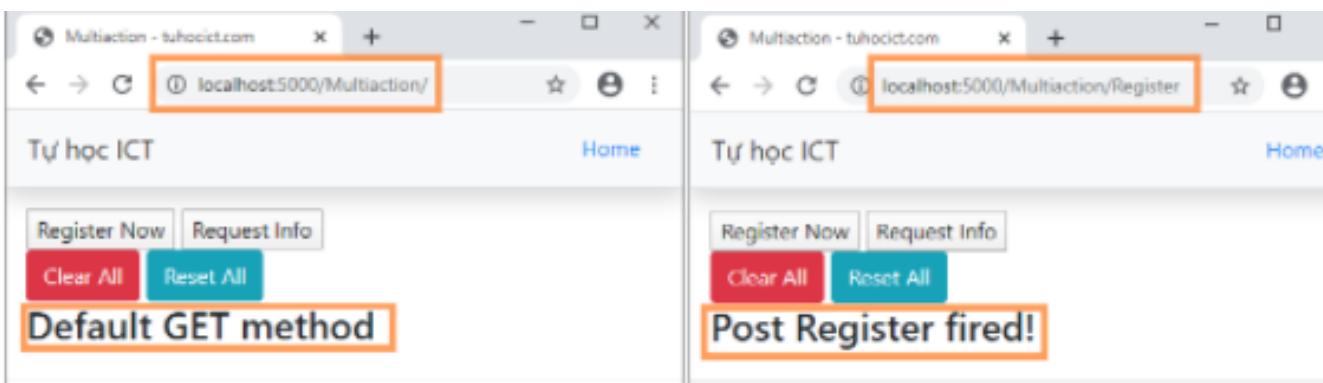
2.5. Các cấu trúc điều khiển của Razor, directive, ViewImports

2.6. Model class trong Razor Pages

2.7. Handler và Action Results trong Razor Pages

• Thực hành 4:

- Khi chạy chương trình bạn sẽ thu được kết quả như sau (chú ý đến url) khi lần lượt click vào các nút:



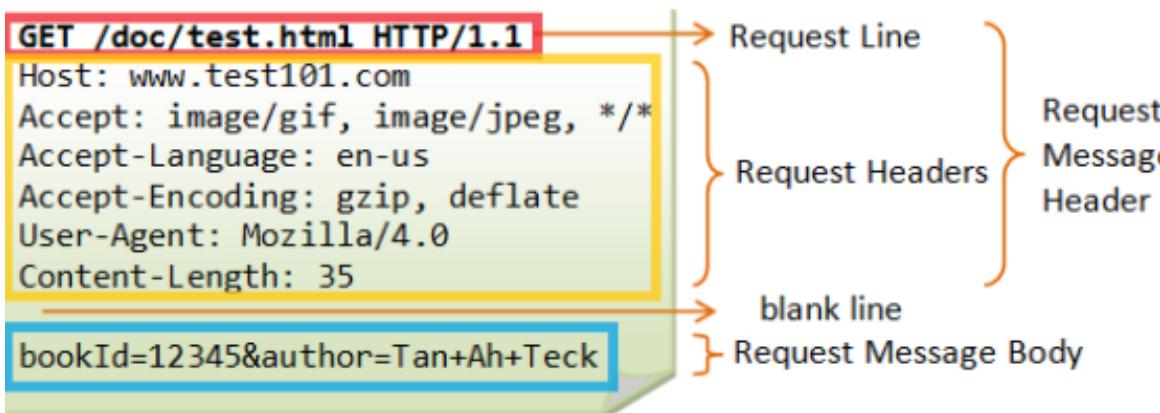
2.8. Query string, route data, route template – xử lý truy vấn GET

- **Truyền dữ liệu về server qua HTTP:**
 - Một yêu cầu phổ biến là gửi dữ liệu từ trình duyệt cho chương trình trên máy chủ qua **HTTP**. Lấy ví dụ, khi nhập dữ liệu vào một form và nhấn nút **Submit**. Dữ liệu này sẽ được đóng gói vào truy vấn **HTTP** và gửi về cho **server**.
 - Trình duyệt có ba cách để gửi dữ liệu về **server** qua truy vấn **HTTP**:
 - (1) Tạo **chuỗi truy vấn** (query string) và ghép vào **Url**.
 - **Ví dụ:** *https://tuhocict.com?s=razor+pages* là một **Url** chứa chuỗi truy vấn *s=razor+pages*. Chuỗi truy vấn và Url phân tách bởi ký tự ? (dấu chấm hỏi). Chuỗi truy vấn tạo ra từ các cặp **<tham số>=<giá trị>**. Các cặp này phân tách bởi ký tự &
 - Phương pháp này thường dùng với phương thức **GET**.
 - (2) Sử dụng **route data**: ghép trực tiếp tham số vào **Url** để trở thành một **segment** của Url.
 - **Ví dụ**, trong url *https://tuhocict.com/topic/razor/*, segment */razor/* thực tế là một tham số cung cấp cho page chuyên hiển thị danh sách bài viết theo chủ đề.
 - Phương pháp này được sử dụng nếu không có nhiều tham số phức tạp. Ngoài ra, phương pháp này cũng tạo ra các Url “thân thiện” với máy tìm kiếm (như Google, Bing, Yandex).

2.8. Query string, route data, route template – xử lý truy vấn GET

• Truyền dữ liệu về server qua HTTP:

- Một yêu cầu phổ biến là gửi dữ liệu từ trình duyệt cho chương trên máy chủ qua **HTTP**. Lấy ví dụ, khi nhập dữ liệu vào một form và ấn nút **Submit**. Dữ liệu này sẽ được đóng gói vào truy vấn **HTTP** và gửi về cho **server**.
- Trình duyệt có ba cách để gửi dữ liệu về **server** qua truy vấn **HTTP**:
 - (3) Gửi dữ liệu qua thân truy vấn **HTTP**.
 - Dữ liệu được tạo ra giống hệt như chuỗi truy vấn trong phương pháp 1) nhưng được ghép vào phần thân (body) của truy vấn **HTTP** (phần đánh dấu màu xanh).



- Phương pháp này thường dùng với phương thức **POST** để truyền dữ liệu lớn (như file).
- Khi dữ liệu tới chương trình, bạn cần lấy dữ liệu ra để xử lý và phản ứng lại cho phù hợp. Thao tác này gọi chung là **xử lý truy vấn**.

2.8. Query string, route data, route template – xử lý truy vấn GET

- Đọc dữ liệu từ query string:

- Sử dụng tham số của handler.

- Trước hết hãy chuẩn bị một dự án mới theo mẫu **Web Application project** và tạo thêm trang **Post** trong thư mục **Pages**
- Viết code cho **Post.cshtml** như sau:

```
1.  using Microsoft.AspNetCore.Mvc;
2.  using Microsoft.AspNetCore.Mvc.RazorPages;
3.
4.  namespace WebApplication1.Pages {
5.      public class PostModel : PageModel {
6.          [ViewData]
7.          public string Title { get; set; }
8.
9.          public string Text { get; set; }
10.
11.         public void OnGet(string title) {
12.             //var title = RouteData.Values["title"].ToString();
13.             Title = title
14.                 .Replace('-', ' ')
15.                 .ToUpper();
16.             Text = "Lorem ipsum dolor sit amet, consectetur adipiscing elit, se
17.         }
18.     }
19. }
```

- Viết code cho **Post.cshtml** như sau:

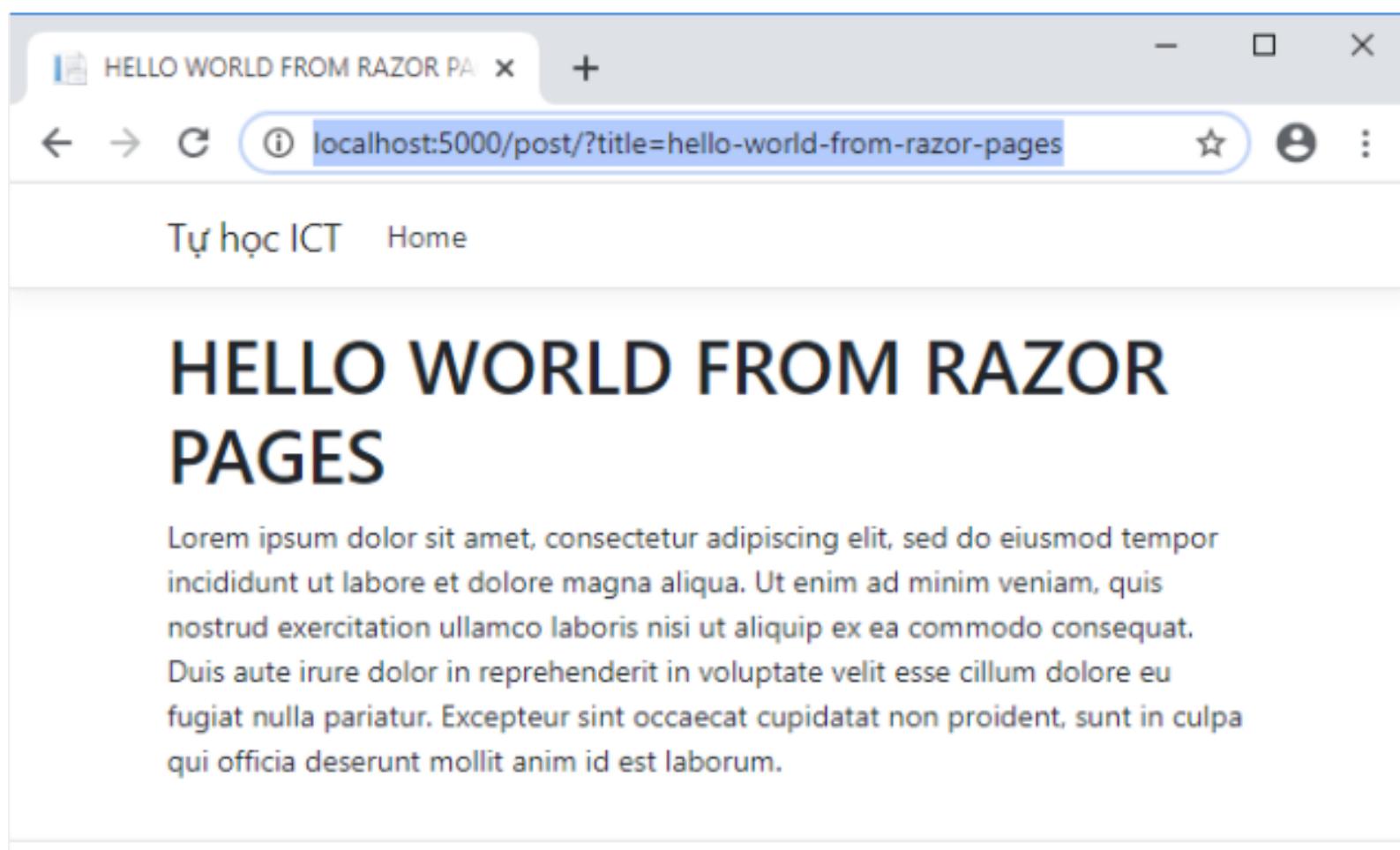
```
1.  @page
2.
3.  @model WebApplication1.Pages.PostModel
4.
5.  <h1>@Model.Title</h1>
6.  <p>@Model.Text</p>
```

2.8. Query string, route data, route template – xử lý truy vấn GET

- Đọc dữ liệu từ query string:

- Sử dụng tham số của handler.

- Bạn sẽ thu được kết quả như sau:



2.8. Query string, route data, route template – xử lý truy vấn GET

- Đọc dữ liệu từ query string:

- Sử dụng object Request.

- Phương pháp thứ hai giúp truy xuất tham số từ chuỗi truy vấn GET là sử dụng object **Request** như sau:

```
1. var title = Request.Query["title"].ToString();
```

- Request** là một property của lớp **PageModel**, do đó bất kỳ model class nào (do kế thừa từ **PageModel**) đều có thể truy xuất. **Object** này chứa các thông tin về truy vấn. **Query** là một property của object này chứa các cặp khóa/giá trị tương ứng tách ra từ chuỗi truy vấn
- Khi sử dụng object **Request** bạn có thể viết lại phương thức OnGet như sau:

```
1. public void OnGet() {
2.     var title = Request.Query["title"].ToString();
3.     Title = title
4.         .Replace('-', ' ')
5.         .ToUpper();
6.     Text = "Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum."
7. }
```

2.8. Query string, route data, route template – xử lý truy vấn GET

- **Route template:**

- **Route template** là khuôn mẫu để định nghĩa và kiểm tra các Url hợp lệ của page.
- Điều này có nghĩa là, khi bạn đưa ra một route template, bạn đang định nghĩa một tập Url hợp lệ cho page. Khi một Url tới từ trình duyệt, route template lại được sử dụng để kiểm tra xem Url này có hợp lệ hay không.
- Trước đây khi học về cơ chế **routing trong Razor Pages**, đã biết rằng cơ chế này sử dụng **route template** mặc định là đường dẫn tới file **cshtml**. Nghĩa là mặc định **Url** hợp lệ tương tự như đường dẫn tương đối tới file **cshtml** tính từ thư mục **Pages**.
- Cũng đã biết rằng **Razor Pages** cho phép ghi đè khuôn mẫu mặc định. Một trong những phương pháp ghi đè là sử dụng directive **@page**

2.8. Query string, route data, route template – xử lý truy vấn GET

Directive	Url
@page "/article"	http://localhost:5000/article
@page "article"	http://localhost:5000/post/article
@page "{title}"	http://localhost:5000/post/anything-can-be-passed-here
@page "{title?}"	http://localhost:5000/post/anything-can-be-passed-here http://localhost:5000/post/
@page "/article/{title}"	http://localhost:5000/article/anything-can-be-passed-here
@page "{year}/{month}/{day}/{title}"	http://localhost:5000/post/post/2020/04/01/hello-world
@page "{title=first post}"	Giống trường hợp 4
@page "{id:int}"	http://localhost:5000/post/10
@page "{title:minlength(2)}"	http://localhost:5000/post/anything-can-be-passed-here

• Route template:

- Với trang *Post.cshtml* ở trên, lần lượt thay đổi @page directive và thử dùng Url tương ứng:

Trường hợp 5 tạo một url có định /article. Bởi thế nếu thay đổi mã định /article thành /post có thể sẽ không tạo ra được URL này với nhiều segment như /article/post/post/article/id. Không có giới hạn số lượng từm giới hạn cho không bị giới hạn giới hạn gì khi tạo url tĩnh. Một template chỉ có thể tạo ra template hợp **2**, @page trở thành **segment** chèn {year}/{month}/{day}/{title}. Trong template này lẻ túc/post/article bất kỳ url nào có 4 segment đứng sau /post. Ví dụ /post/2020/04/01/hello-world (trường hợp 6).

Trường hợp 3 và **4** chấp nhận bất kỳ chuỗi nào để tạo thành **segment cuối của url mặc định** /post/{title}. Trong các trường hợp này, đây là tình huống cung cấp giá trị mặc định cho **route template** tạo ra một **tập hợp** (group) **route data** nếu **segment** cuối cùng bị để trống. (thay cho 1 url như trong trường hợp 1 và 2).

Trường hợp 5 và **6** tạo ra một URL dài (route data) hoặc dài (tham số) nhưng sẽ bị lược bỏ nếu đã thêm phần (segment) của url thúc **đặt giới hạn** trong **route template**.

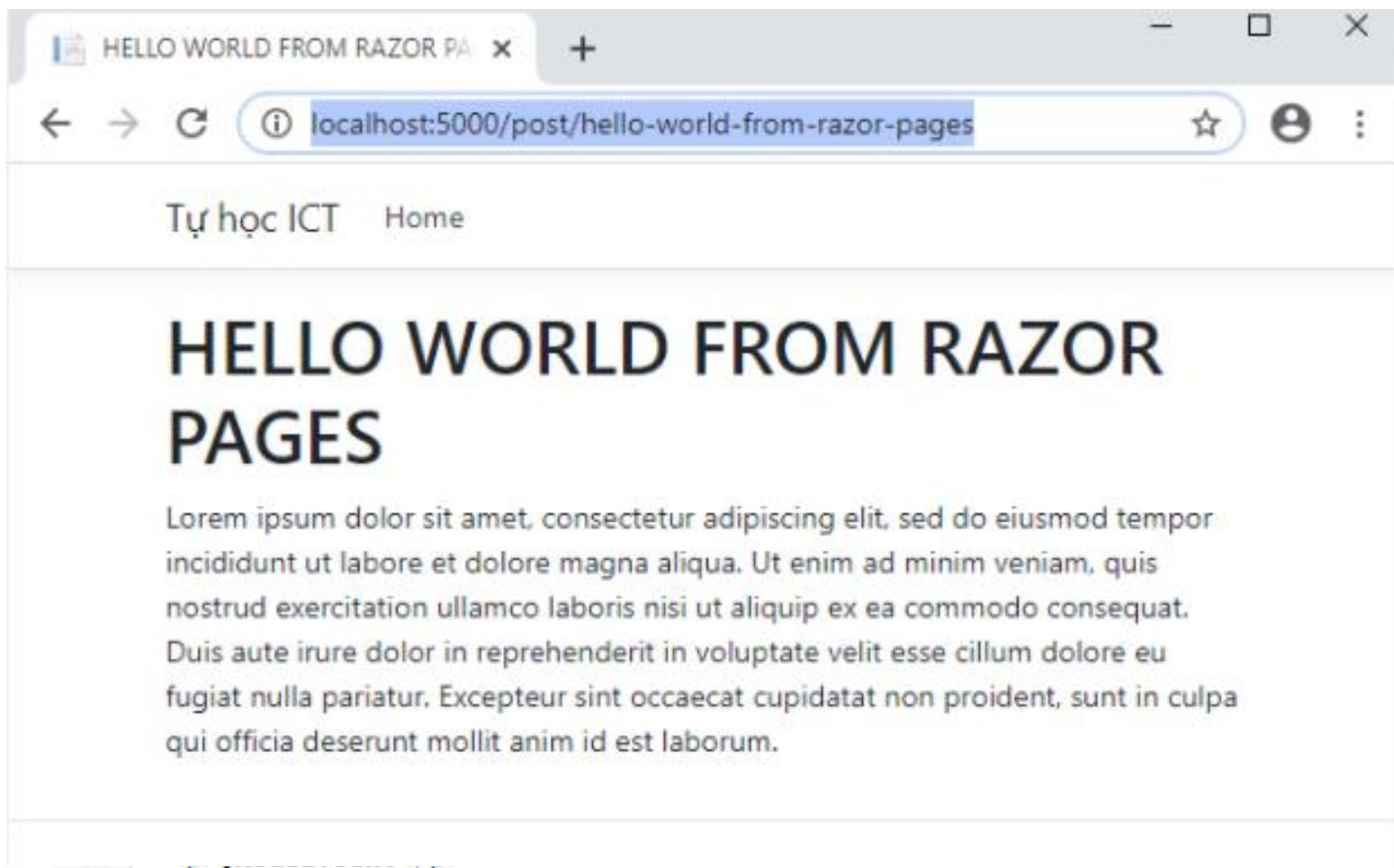
2.8. Query string, route data, route template – xử lý truy vấn GET

- Đọc dữ liệu từ route data:

- Truy xuất qua tham số của handler:

- Trong các template có sử dụng **placeholder**, tên của placeholder được xem là một **tên biến** và giá trị của **segment** tương ứng trong **url** chính là giá trị của biến. Loại template này cho phép tạo ra các url với khả năng mang tham số.

- Ví dụ:



2.8. Query string, route data, route template – xử lý truy vấn GET

- Đọc dữ liệu từ route data:

- Truy xuất qua object RouteData:

- Một khi thiết lập **route template** sử dụng **placeholder**, **Razor Pages** đều phải phân tích **Url** tới xem có phù hợp hay không. Nếu một **Url** hợp lệ, các thành phần của nó được phân tách và lưu trữ trong một object có tên là **RouteData**.
- Có thể truy xuất giá trị tham số sử dụng tên của placeholder như sau:

```
1. var title = RouteData.Values["title"].ToString();
```

```
1. public void OnGet() {
2.     var title = RouteData.Values["title"].ToString();
3.     Title = title
4.         .Replace('-', ' ')
5.         .ToUpper();
6.     Text = "Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do e:
```

2.8. Query string, route data, route template – xử lý truy vấn GET

- **Thực hành:** viết ứng dụng giải phương trình bậc 2 sử dụng Razor Pages:

- **Bước 1.** Tạo page mới *Equation.cshtml*.
- **Bước 2.** Viết code cho model class **EquationModel** trong *Equation.cshtml.cs* như sau:

```
1.  using System;
2.
3.  using Microsoft.AspNetCore.Mvc.RazorPages;
4.
5.  namespace WebApplication1.Pages {
6.      public class EquationModel : PageModel {
7.          public (double x1, double x2) Solutions { get; private set; }
8.          public (double a, double b, double c) Coefficients { get; private set; }
9.
10.         public void OnGet(double a, double b, double c) {
11.             Coefficients = (a, b, c);
12.             var d = b * b - 4 * a * c;
13.             Solutions =
14.                 ((-b + Math.Sqrt(d)) / (2 * a),
15.                  (-b - Math.Sqrt(d)) / (2 * a));
16.         }
17.     }
18. }
```

2.8. Query string, route data, route template – xử lý truy vấn GET

- **Thực hành:** viết ứng dụng giải phương trình bậc 2 sử dụng Razor Pages:

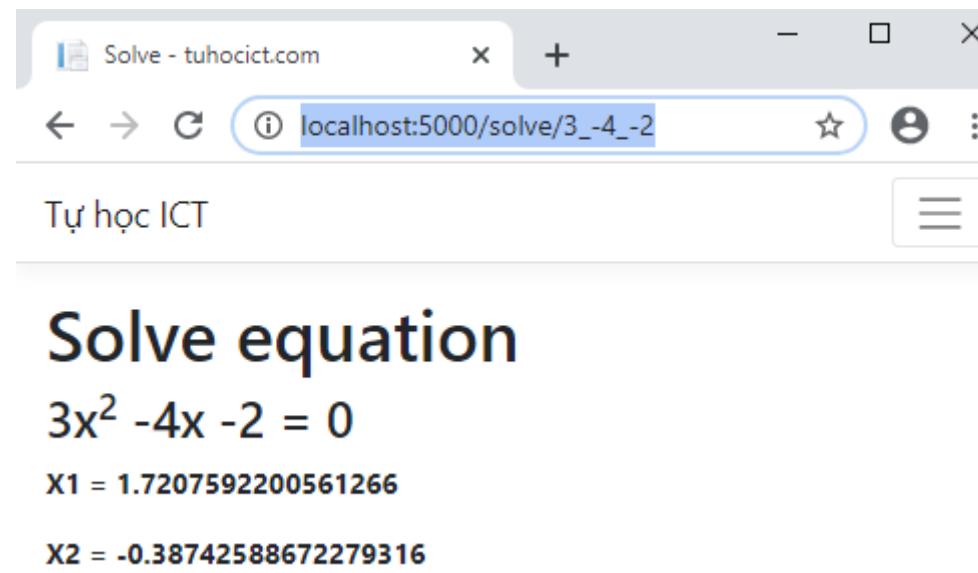
- **Bước 3.** Viết code cho page *Equation.cshtml* như sau:

```
1. @page "/solve/{a:double}_{b:double}_{c:double}"
2. @model WebApplication1.Pages.EquationModel
3.
4.     ViewData["Title"] = "Solve";
5.     var a = Model.Coefficients.a;
6.     var b = Model.Coefficients.b;
7.     var c = Model.Coefficients.c;
8.
9.
10.    <h1>Solve equation</h1>
11.    <h3>@(a)x2 @(b<0?"":+") @(b)x @(c<0?"":+") @c = 0</h3>
12.    <p><strong>X1 = @Model.Solutions.x1</strong></p>
13.    <p><strong>X2 = @Model.Solutions.x2</strong></p>
```

2.8. Query string, route data, route template – xử lý truy vấn GET

- **Thực hành:** viết ứng dụng giải phương trình bậc 2 sử dụng Razor Pages:

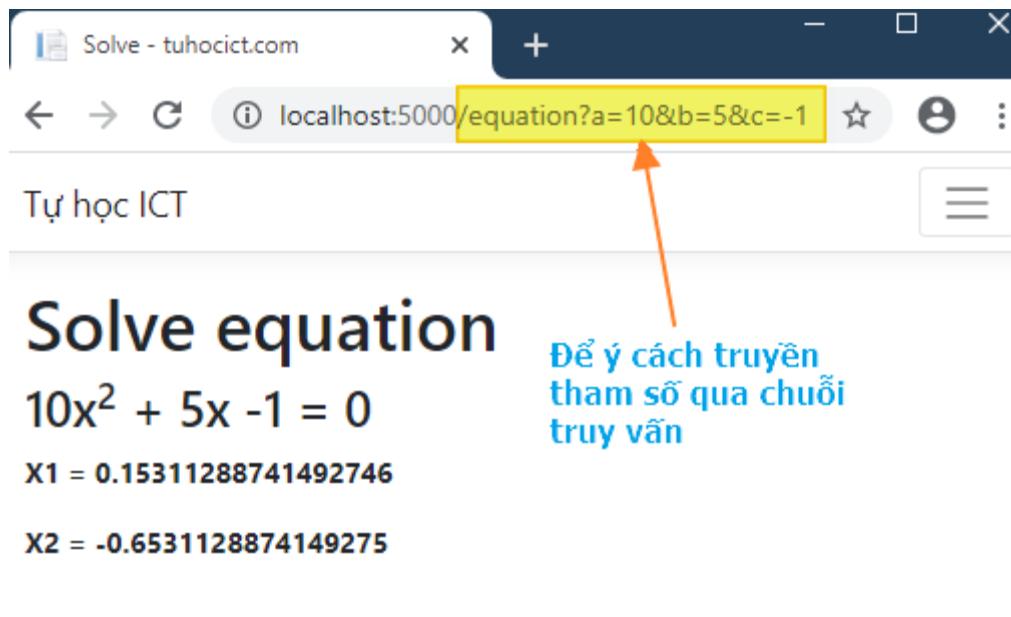
- Chạy thử với Url /solve/3_-4_-2 bạn thu được kết quả như sau:



2.8. Query string, route data, route template – xử lý truy vấn GET

- **Thực hành:** viết ứng dụng giải phương trình bậc 2 sử dụng Razor Pages:

- Nếu bỏ route template sau `@page` directive, bạn có thể gọi phương thức `OnGet` như sau:



- Khi này `Equation.cshtml` tương ứng với Url (mặc định) là `/equation`. Bạn truyền tham số cho `OnGet` thông qua chuỗi truy vấn `?a=10&b=5&c=-1`

2.8. Query string, route data, route template –
xử lý truy vấn GET

2.9. HTML Form trong
Razor Pages – xử lý
truy vấn POST

- Trong các ứng dụng web, **form** được sử dụng để **thu thập dữ liệu** (từ người dùng) và gửi trở lại **server** (để xử lý). Tạo và xử lý form cũng là yêu cầu chính của mọi ứng dụng web.
- Thực hành:**
 - Bước 1.** Tạo project mới theo mẫu Web Application
 - Bước 2.** Viết code cho **model class** trong file **Index.cshtml.cs** như sau:

```
1.  using Microsoft.AspNetCore.Mvc.RazorPages;
2.
3.  namespace WebApplication1.Pages {
4.      public class IndexModel : PageModel {
5.          public string Title { get; private set; }
6.          public string Text { get; private set; }
7.          public void OnPost(string title, string text) {
8.              Title = title;
9.              Text = text;
10.         }
11.     }
12. }
```

2.8. Query string, route data, route template – xử lý truy vấn GET

2.9. HTML Form trong Razor Pages – xử lý truy vấn POST

- Thực hành:
 - Bước 3. Viết code razor cho page *Index.cshtml* như sau::

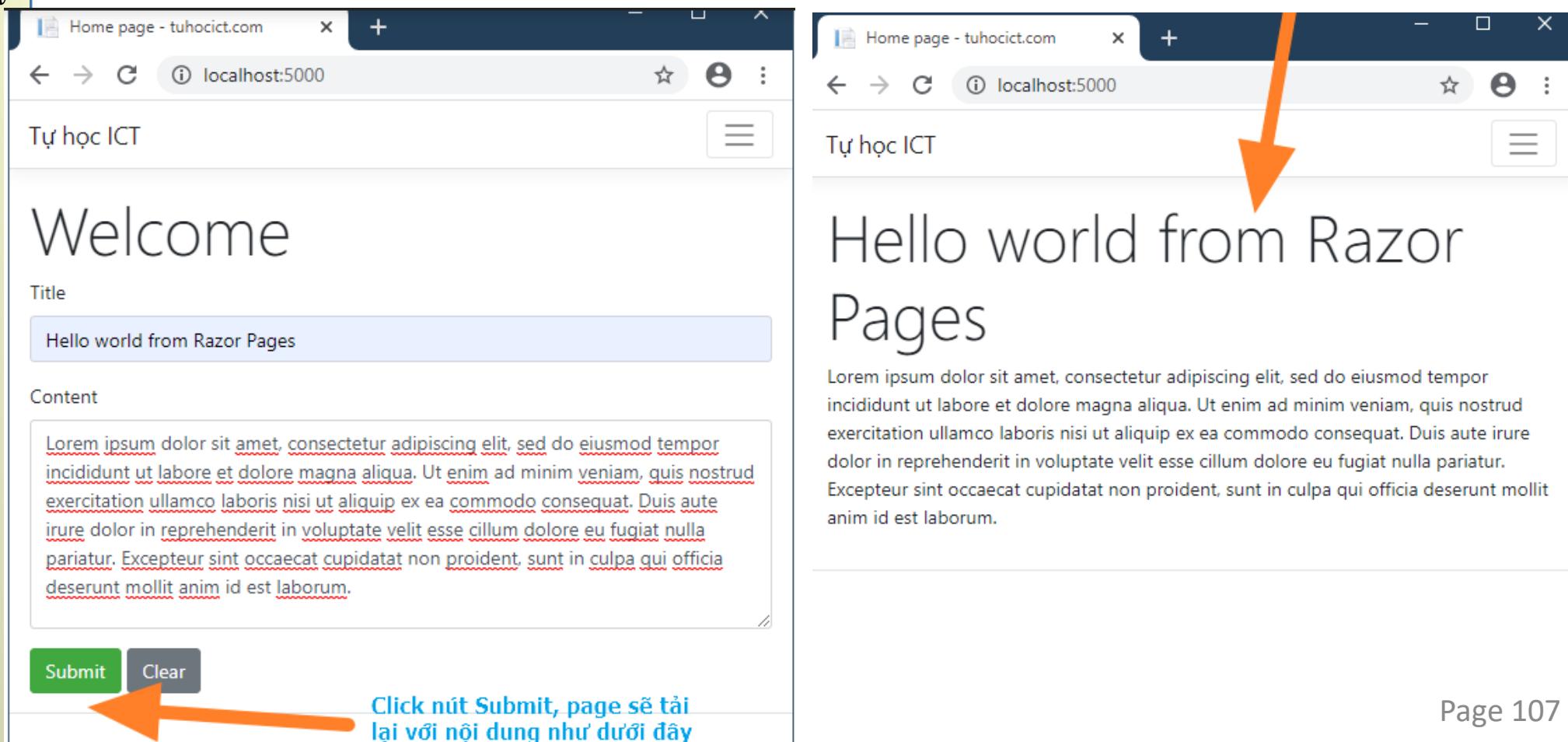
```
1. @page
2. @model IndexModel
3. @{
4.     ViewData["Title"] = "Home page";
5. }
6.
7. @if (Request.Method.ToUpper() == "GET") {
8.     <h1 class="display-4">Welcome</h1>
9.     <form method="post">
10.         <div class="form-group">
11.             <label for="title">Title</label>
12.             <input type="text" class="form-control" id="title" name="title" />
13.         </div>
14.         <div class="form-group">
15.             <label for="text">Content</label>
16.             <textarea class="form-control" id="text" name="text"></textarea>
17.         </div>
18.         <div>
19.             <button class="btn btn-success" type="submit">Submit</button>
20.             <button class="btn btn-secondary" type="reset">Clear</button>
21.         </div>
22.     </form>
23. }
24. else if (Request.Method.ToUpper() == "POST") {
25.     <h1 class="display-4">@Model.Title</h1>
26.     <p>@Model.Text</p>
27. }
```

2.8. Query string, route data, route template – xử lý truy vấn GET

2.9. HTML Form trong Razor Pages – xử lý truy vấn POST

• Thực hành:

- Tùy thuộc vào truy vấn page sẽ hiển thị các nội dung khác nhau. Trong lần tải page đầu tiên (luôn là truy vấn GET) sẽ hiển thị form. Khi bấm nút Submit để gửi truy vấn POST về server, page sẽ tải lại nhưng giờ sẽ chỉ hiển thị lại những nội dung người dùng đã nhập. Khi tải lại trang sau truy vấn POST sẽ không hiển thị form.



Click nút Submit, page sẽ tải
lại với nội dung như dưới đây

2.8. Query string, route
data, route template –
xử lý truy vấn GET

2.9. HTML Form trong
Razor Pages – xử lý
truy vấn POST

• Thực hành:

- Trong ví dụ trên bạn đã thấy, dữ liệu của điều khiển **Input** với thuộc tính **name = "title"** sẽ được truyền vào tham số **title** của **OnPost** – phương thức xử lý truy vấn **POST**. Tương tự như vậy với **TextArea** **name='text'**.
- Razor Pages** cũng cho phép bạn truy xuất dữ liệu trong truy vấn **POST** sử dụng đặc tính **Request.Form** như sau:

```
1. public void OnPost() {  
2.     Title = Request.Form["title"];  
3.     Text = Request.Form["text"];  
4. }
```

- Qua đây bạn cũng thấy việc xử lý dữ liệu từ **form** (và truy vấn **POST**) không có gì khác biệt với xử lý dữ liệu đến từ **URL** (qua truy vấn **GET**)

2.8. Query string, route data, route template – xử lý truy vấn GET

2.9. HTML Form trong Razor Pages – xử lý truy vấn POST

• Form và các điều khiển:

• Form

- Ngôn ngữ HTML sử dụng thẻ `<form></form>` để định nghĩa một form – nơi chứa các điều khiển.
- **Điều khiển**, cũng được gọi là phần tử của form, là những công cụ chuyên dụng cho mục đích nhập dữ liệu, như hộp văn bản, nút, hộp chọn, v.v..
- Tự bản thân thẻ `<form>` không hiển thị gì trên trang. Nó chỉ đóng vai trò nhóm các phần tử nhập dữ liệu lại và chỉ định cách truyền dữ liệu từ các điều khiển về server.
- Một thẻ form thường có dạng như sau:

```
1.   <form action="/index" method="POST">
2.       <!-- các phần tử (điều khiển) -->
3.   </form>
```

2.8. Query string, route data, route template – xử lý truy vấn GET

2.9. HTML Form trong Razor Pages – xử lý truy vấn POST

• Form và các điều khiển:

• Các phần tử của form

- Trên form bạn có thể đặt các điều khiển để người dùng nhập dữ liệu.
- Các phần tử của form, cũng gọi là các điều khiển, là những đối tượng đồ họa giúp người dùng nhập dữ liệu.
- Có nhiều loại phần tử khác nhau dành cho từng mục đích nhập liệu. Rất dễ hình dung các điều khiển trên **HTML** form cũng giống như các điều khiển trên giao diện đồ họa của ứng dụng desktop. Chúng cũng có cùng hình thức, cùng tên gọi và mục đích.
- Khi làm việc với các điều khiển cần lưu ý:
 - (1) Để có thể thu thập được dữ liệu vào đóng gói vào truy vấn HTTP, trình duyệt yêu cầu mỗi điều khiển phải được đặt tên bằng cách thiết lập giá trị cho thuộc tính **name**
 - (2) Giá trị của thuộc tính **name** cũng đồng thời là khóa để truy cập giá trị trong code C# qua object **Request.Form**, hoặc cũng chính là tên tham số của **OnPost** – phương thức xử lý truy vấn **POST** từ form.
 - (3) Mỗi form phải có một nút **Submit** `<input type='submit'>` hoặc `<button type='submit'>`) để kết thúc quá trình nhập dữ liệu và gửi về server.

2.8. Query string, route
data, route template –
xử lý truy vấn GET

2.9. HTML Form trong
Razor Pages – xử lý
truy vấn POST

- Đóng gói dữ liệu form vào thân truy vấn POST:

- Khi bạn ấn nút **Submit**, dữ liệu từ các điều khiển trên form được thu thập và đóng gói vào truy vấn **HTTP** để gửi lại **server**. Việc “đóng gói” dữ liệu phụ thuộc vào thiết lập **enctype**, và cũng phụ thuộc vào loại dữ liệu cần gửi đi.
- Có hai phương pháp thường gặp để đóng gói dữ liệu vào truy vấn **HTTP**: **multipart/form-data** và **x-www-form-urlencoded**. Cả hai phương pháp đều có thể đóng gói và gửi về server dữ liệu ở dạng văn bản và nhị phân.
- Phương pháp **x-www-form-urlencoded** sẽ lấy tên các điều khiển trên form và giá trị tương ứng ghép lại với nhau thành một chuỗi tương tự như chuỗi truy vấn của URL.
- Phương pháp multipart/form-data sẽ chọn một cụm ký tự (không thể xuất hiện trong dữ liệu) làm *chuỗi đánh dấu (boundary)*. Vai trò của chuỗi đánh dấu là phân tách dữ liệu ra từng *khoi* (chunk).

2.8. Query string, route data, route template – xử lý truy vấn GET

2.9. HTML Form trong Razor Pages – xử lý truy vấn POST

• Đóng gói dữ liệu form vào thân truy vấn POST:

- Ví dụ, nếu trên form bạn có hai điều khiển như sau:

```
1. <input type="text" name="title" />  
2. <textarea name="text"></textarea>
```

- Nếu người dùng nhập “*Hello world*” vào **textbox**, và “*From Razor Pages*” vào **textarea**. Phương pháp **x-www-form-urlencoded** sẽ đóng gói dữ liệu vào truy vấn **HTTP** như sau:

```
1. POST / HTTP/1.1  
2. Host: localhost:5000  
3. Content-Type: application/x-www-form-urlencoded  
4.  
5. title=Hello world&text=From Razor Pages
```

- Phương pháp **multipart/form-data** sẽ đóng gói dữ liệu như sau:

```
1. POST / HTTP/1.1  
2. Host: localhost:5000  
3. Content-Type: multipart/form-data; boundary=----WebKitFormBoundary7MA4YWxkTrZu0gW  
4.  
5. ----WebKitFormBoundary7MA4YWxkTrZu0gW  
6. Content-Disposition: form-data; name="title"  
7.  
8. Hello world  
9. ----WebKitFormBoundary7MA4YWxkTrZu0gW  
10. Content-Disposition: form-data; name="text"  
11.  
12. From Razor Pages  
13. ----WebKitFormBoundary7MA4YWxkTrZu0gW
```

- Ở đây —WebKitFormBoundary7MA4YWxkTrZu0gW được chọn làm chuỗi đánh dấu.

2.8. Query string, route
data, route template –
xử lý truy vấn GET

2.9. HTML Form trong
Razor Pages – xử lý
truy vấn POST

- Đóng gói dữ liệu form vào thân truy vấn POST:

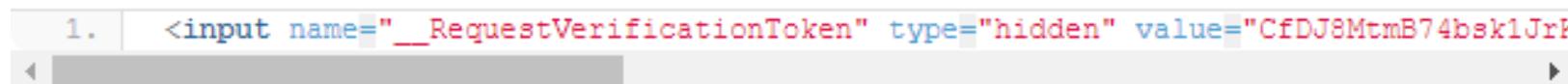
- Phương pháp **multipart/form-data** thường dùng khi cần tải file lên server, trong khi **x-www-form-urlencoded** được sử dụng trong những trường hợp khác.
- Như bạn nhìn thấy, trong cả hai phương pháp đóng gói, **HTTP** đều sử dụng đến giá trị và tên của điều khiển. Do vậy, khi tạo form, bạn đừng quên thuộc tính **name**.

2.8. Query string, route data, route template – xử lý truy vấn GET

2.9. HTML Form trong Razor Pages – xử lý truy vấn POST

• Bảo vệ form khỏi tấn công CSRF:

- **CSRF (Cross-Site Request Forgery)**, tạm dịch là giả mạo yêu cầu liên trang, là một dạng tấn công rất phổ biến đối với các ứng dụng web.
- Razor Pages cung cấp sẵn một cơ chế bảo vệ khỏi tấn công **CSRF** có tên gọi là **xác minh truy vấn (Request Verification)**. Cơ chế này được sử dụng mặc định mỗi khi bạn xây dựng form.
- Với ứng dụng đơn giản đã xây dựng ở trên, nếu bạn mở **Development Tool (F12)**, tab **Elements** để xem mã **HTML** nhận được, bạn sẽ nhìn thấy một điều khiển lạ thuộc loại hidden như sau:



1. <input name="__RequestVerificationToken" type="hidden" value="CfDJ8MtmB74bsk1JrK"/>

- Mỗi lần bạn tải lại page, giá trị cũng biến đổi. Giá trị này chỉ có ý nghĩa một lần duy nhất.
- Khi bạn ấn nút **submit**, trường **__RequestVerificationToken** và giá trị của nó cũng được đóng gói vào truy vấn **POST** gửi ngược trở lại server.

2.8. Query string, route
data, route template –
xử lý truy vấn GET

2.9. HTML Form trong
Razor Pages – xử lý
truy vấn POST

• **Bảo vệ form khỏi tấn công CSRF:**

- Khi chuyển sang tab Application, mục Storage ->Cookies, bạn sẽ nhìn thấy site đã thả lên máy bạn một **cookie** có tên **.AspNetCore.Antiforgery.UfgPPsOClIs** (giá trị dưới đây chỉ mang tính minh họa).
- Tổ hợp giá trị **cookie** và chuỗi xác minh truy vấn được **ASP.NET Core** kiểm tra mỗi khi nhận truy vấn **POST** để đảm bảo rằng truy vấn đến từ đúng form “xin”. Nếu không có giá trị hoặc xác minh không đạt, ASP.NET Core sẽ trả lại phản hồi với **mã 400 – Bad Request**, và truy vấn **POST** không thể thực hiện được.
- Cơ chế này mặc dù được sử dụng tự động, nghĩa là **ASP.NET Core** luôn luôn kiểm tra cookie và chuỗi xác minh. Tuy nhiên chuỗi xác minh trong trường **_RequestVerificationToken** lại chỉ được sinh ra nếu bạn sử dụng tag **helper** với **directive**
- Khi sử dụng template **Web Application**, **directive** này được bật sẵn trong file **_ViewImports.cshtml**. Nếu bạn sử dụng template **Empty**, bạn phải tự mình thêm **directive** trên vào page hoặc vào **_ViewImports**.

2.8. Query string, route data, route template – xử lý truy vấn GET

2.9. HTML Form trong Razor Pages – xử lý truy vấn POST

2.10. Xử lý form control trong Razor Pages

• File upload:

- Ví dụ:

- Chuẩn bị: Tạo project mới theo mẫu **Web Application**
- Bước 1: Tạo thư mục **upload** trực thuộc project.
- Bước 2. Tạo page **Upload** sử dụng model class.
- Bước 3. Viết code như sau cho **Upload.cshtml.cs**:

```
using System;
using System.IO;

using Microsoft.AspNetCore.Hosting;
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc.RazorPages;

namespace WebApplication1.Pages {
    public class UploadModel : PageModel {
        private readonly IWebHostEnvironment _environment;
        public UploadModel(IWebHostEnvironment environment) => _environment = environment;

        public bool Success { get; private set; } = true;

        public void OnPost(IFormFile file) {
            try {
                var f = Path.Combine(_environment.ContentRootPath, "upload", file.FileName);
                using var fs = new FileStream(f, FileMode.Create);
                file.CopyTo(fs);
                ViewData["file"] = file.FileName;
            }
            catch (Exception) {
                Success = false;
            }
        }
    }
}
```

2.8. Query string, route data, route template – xử lý truy vấn GET

2.9. HTML Form trong Razor Pages – xử lý truy vấn POST

2.10. Xử lý form control trong Razor Pages

• File upload:

- Ví dụ:

- **Bước 4:** Viết mã razor cho *Upload.cshtml* như sau:

```
@page
@model WebApplication1.Pages.UploadModel
 @{
    ViewData["Title"] = "Upload";
}

<h1>File Upload</h1>

<form method="post" enctype="multipart/form-data">
    <input type="file" name="file" />
    <input type="submit" value="Submit now" class="btn btn-info" />
</form>

@if (Request.Method == "POST") {
    @if (Model.Success) {
        <h3 class="text-success">File '@ViewData["file"]' uploaded successfully!</h3>
    }
    else {
        <h2 class="text-danger">Failed to upload file '@ViewData["file"]'!</h2>
    }
}
```

- Lưu ý form phải thiết lập *enctype="multipart/form-data"* mới có thể upload file.

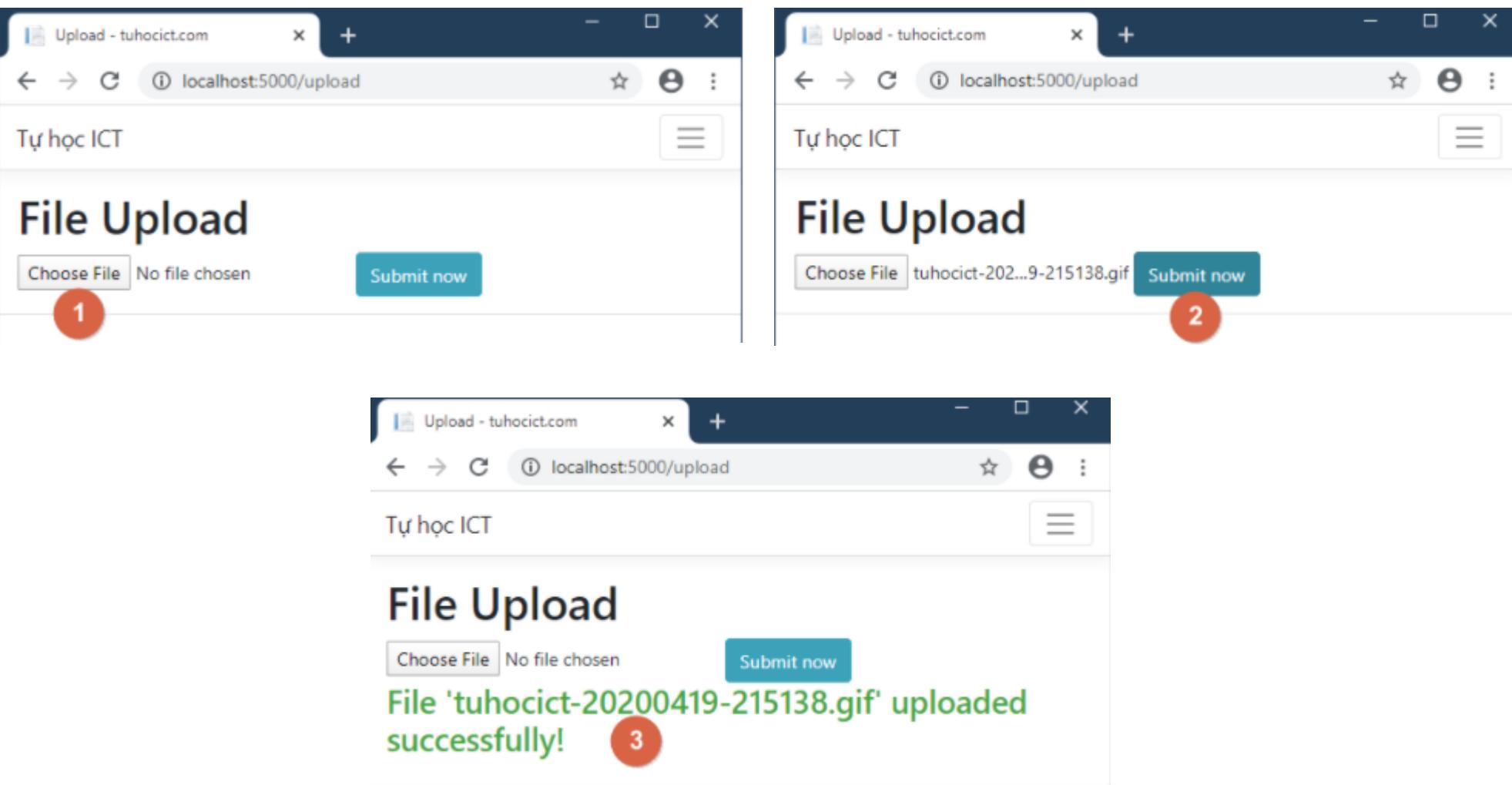
2.8. Query string, route
data, route template –
xử lý truy vấn GET

2.9. HTML Form trong
Razor Pages – xử lý
truy vấn POST

2.10. Xử lý form
control trong Razor
Pages

- **File upload:**

- Ví dụ:



2.8. Query string, route data, route template – xử lý truy vấn GET

2.9. HTML Form trong Razor Pages – xử lý truy vấn POST

2.10. Xử lý form control trong Razor Pages

• Làm việc với checkbox:

- Checkbox (hộp chọn) cho phép người dùng click chọn đồng thời nhiều mục. HTML sử dụng thẻ input để tạo checkbox cơ bản như sau:

```
1. <input type="checkbox" id="idEnglish" name="languages" value="English">  
2. <label for="idEnglish">English</label>
```

- Thẻ trên sẽ tạo ra checkbox như sau: 

Lưu ý:

- (1) Nhiều input cùng thuộc một nhóm phải có cùng giá trị name;
- (2) Giá trị của thuộc tính value sẽ được gửi về server nếu checkbox được chọn chứ không hiển thị cạnh checkbox;
- (3) Thẻ `<label>` hiển thị văn bản bên cạnh checkbox (trong khi value sẽ không hiển thị);
- (4) Checkbox sẽ được chọn mặc định nếu có mặt thuộc tính checked (không quan tâm đến giá trị).

2.8. Query string, route
data, route template –
xử lý truy vấn GET

2.9. HTML Form trong
Razor Pages – xử lý
truy vấn POST

2.10. Xử lý form
control trong Razor
Pages

- **Làm việc với checkbox:**

- **Ví dụ:**

- **Bước 1.** Tạo trang Checkbox.cshtml.
 - **Bước 2:** Viết code cho *Checkbox.cshtml.cs* (model class) như sau:

```
1.  using System.Collections.Generic;
2.  using Microsoft.AspNetCore.Mvc.RazorPages;
3.
4.  namespace WebApplication1.Pages {
5.      public class CheckboxModel : PageModel {
6.          public List<string> Languages { get; private set; }
7.
8.          public void OnPost(List<string> languages) => Languages = languages;
9.      }
10. }
```

2.8. Query string, route data, route template – xử lý truy vấn GET

2.9. HTML Form trong Razor Pages – xử lý truy vấn POST

2.10. Xử lý form control trong Razor Pages

- **Bước 3.** Viết code cho *Checkbox.cshtml* như sau:

```
@page
@model WebApplication1.Pages.CheckboxModel
 @{
    ViewData["Title"] = "Checkbox";
}

@if (Request.Method.ToUpper() == "GET") {
<h3>Which languages can you speak?</h3>
<form method="post">
    <!-- Default unchecked -->
    <div class="custom-control custom-checkbox">
        <input type="checkbox" class="custom-control-input" id="Arabic" name="languages" value="Arabic">
        <label class="custom-control-label" for="Arabic">Arabic</label>
    </div>
    <div class="custom-control custom-checkbox">
        <input type="checkbox" class="custom-control-input" id="Chinese" name="languages" value="Chinese">
        <label class="custom-control-label" for="Chinese">Chinese</label>
    </div>
    <!-- Default checked -->
    <div class="custom-control custom-checkbox">
        <input type="checkbox" class="custom-control-input" id="English" name="languages" value="English" checked="checked">
        <label class="custom-control-label" for="English">English</label>
    </div>
    <div class="custom-control custom-checkbox">
        <input type="checkbox" class="custom-control-input" id="French" name="languages" value="French">
        <label class="custom-control-label" for="French">French</label>
    </div>
    <div class="custom-control custom-checkbox">
        <input type="checkbox" class="custom-control-input" id="Russian" name="languages" value="Russian">
        <label class="custom-control-label" for="Russian">Russian</label>
    </div>
    <div class="custom-control custom-checkbox">
        <input type="checkbox" class="custom-control-input" id="Spanish" name="languages" value="Spanish">
        <label class="custom-control-label" for="Spanish">Spanish</label>
    </div>
    <div>
        <button class="btn btn-success" type="submit">Submit</button>
        <button class="btn btn-secondary" type="reset">Clear</button>
    </div>
</form>
}
else if(Request.Method.ToUpper() == "POST") {
    <h3 class="alert-success">Great! You can speak: @foreach (var l in Model.Languages) { <b>@l</b>
}
```

2.8. Query string, route data, route template – xử lý truy vấn GET

2.9. HTML Form trong Razor Pages – xử lý truy vấn POST

2.10. Xử lý form control trong Razor Pages

A screenshot of a web browser window titled "Checkbox - tuhocict.com". The address bar shows "localhost:5000/checkbox". The page content includes the text "Tự học ICT" and "Which languages can you speak?". Below this is a list of language options with checkboxes:

- Arabic
- Chinese
- English
- French
- Russian
- Spanish

At the bottom of the form are two buttons: a green "Submit" button and a grey "Clear" button.

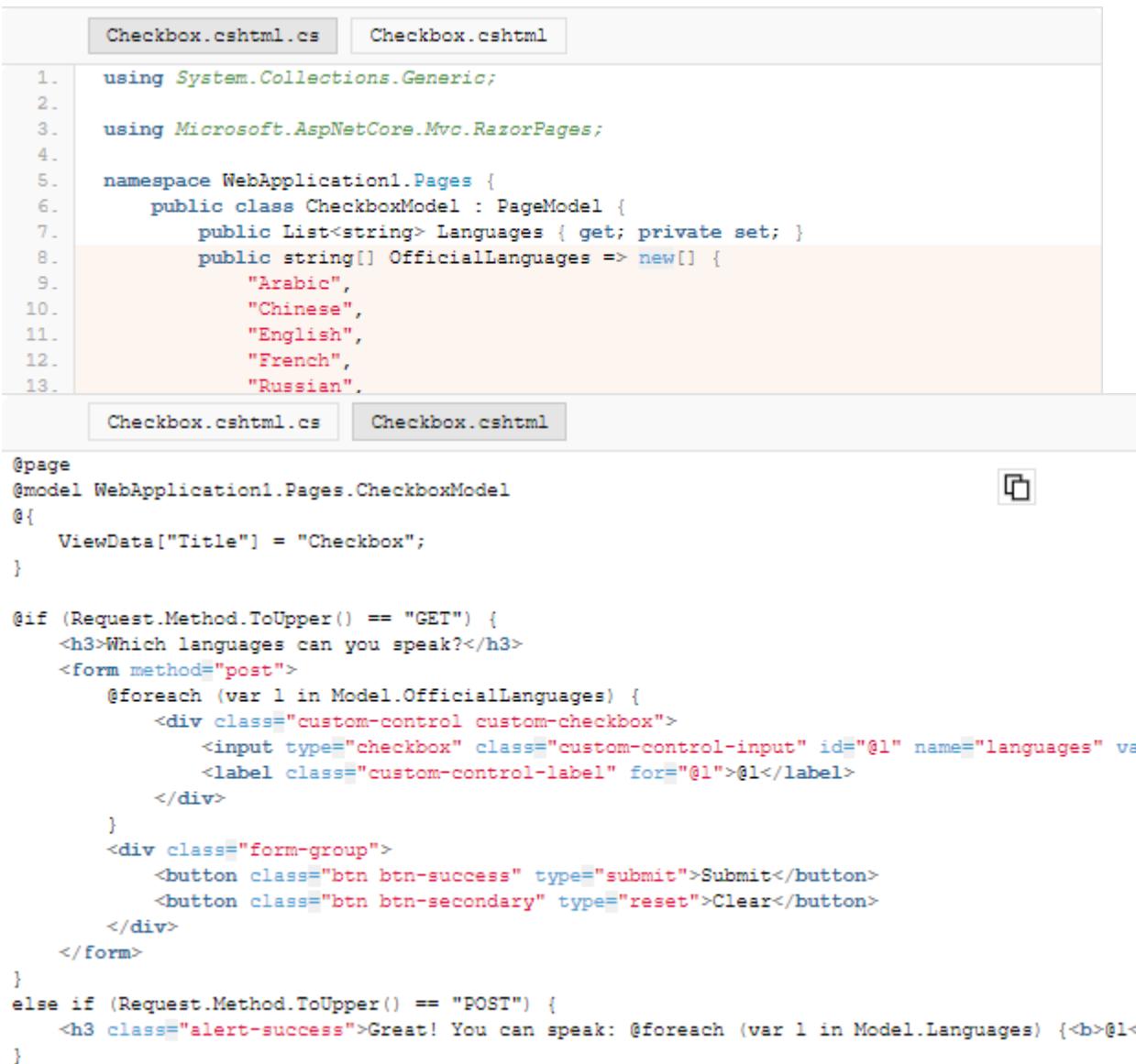
A screenshot of a web browser window titled "Checkbox - tuhocict.com". The address bar shows "localhost:5000/checkbox". The page content includes the text "Tự học ICT" and a green success message box containing the text "Great! You can speak: English Russian".

2.8. Query string, route data, route template – xử lý truy vấn GET

2.9. HTML Form trong Razor Pages – xử lý truy vấn POST

2.10. Xử lý form control trong Razor Pages

- Để ý rằng code ở trên có nhiều chỗ lặp lại mà bạn có thể sử dụng Razor code thay thế.
- Do checkbox thường là dạng danh sách, bạn có thể sử dụng vòng lặp để xuất ra HTML. Bạn có thể cải tiến ví dụ trên sử dụng cú pháp Razor cho ngắn gọn như sau:



The screenshot shows a Visual Studio code editor with two tabs: `Checkbox.cshtml.cs` and `Checkbox.cshtml`. The `Checkbox.cshtml.cs` tab contains C# code for a `CheckboxModel` class. The `Checkbox.cshtml` tab contains the corresponding Razor view code.

```
using System.Collections.Generic;
using Microsoft.AspNetCore.Mvc.RazorPages;

namespace WebApplication1.Pages {
    public class CheckboxModel : PageModel {
        public List<string> Languages { get; private set; }
        public string[] OfficialLanguages => new[] {
            "Arabic",
            "Chinese",
            "English",
            "French",
            "Russian".
        }
    }
}

@page
@model WebApplication1.Pages.CheckboxModel
 @{
    ViewData["Title"] = "Checkbox";
}

@if (Request.Method.ToUpper() == "GET") {
    <h3>Which languages can you speak?</h3>
    <form method="post">
        @foreach (var l in Model.OfficialLanguages) {
            <div class="custom-control custom-checkbox">
                <input type="checkbox" class="custom-control-input" id="@l" name="languages" value="1" checked="checked" />
                <label class="custom-control-label" for="@l">@l</label>
            </div>
        }
        <div class="form-group">
            <button class="btn btn-success" type="submit">Submit</button>
            <button class="btn btn-secondary" type="reset">Clear</button>
        </div>
    </form>
}
else if (Request.Method.ToUpper() == "POST") {
    <h3 class="alert-success">Great! You can speak: @foreach (var l in Model.Languages) {<b>@l</b> }</h3>
}
```

2.8. Query string, route data, route template – xử lý truy vấn GET

2.9. HTML Form trong Razor Pages – xử lý truy vấn POST

2.10. Xử lý form control trong Razor Pages

• Sử dụng radio button

- **Radio button**, còn gọi là nút đài, có hình thức gần tương tự như **checkbox**, nhưng chỉ cho phép chọn một phương án duy nhất.
- Trong **Razor Pages**, nút đài có cách xuất (ra html) tương tự như **checkbox** nhưng có cách *đọc dữ liệu từ form* giống như **textbox** hay **textarea** bạn đã biết trong bài học trước.
- Radio button được tạo ra trên form với thẻ **input**, tương tự như **checkbox**, nhưng thuộc tính **type='radio'**:

2.8. Query string, route
data, route template –
xử lý truy vấn GET

2.9. HTML Form trong
Razor Pages – xử lý
truy vấn POST

2.10. Xử lý form
control trong Razor
Pages

• Sử dụng radio button

• Ví dụ:

- **Bước 1.** Thêm trang **RadioButton** vào dự án.
- **Bước 2.** Viết code cho model class trong file **RadioButton.cshtml.cs** như sau:

```
1.  using Microsoft.AspNetCore.Mvc.RazorPages;
2.
3.  namespace WebApplication1.Pages {
4.      public class RadioButtonModel : PageModel {
5.          public string Language { get; private set; }
6.          public string[] OfficialLanguages => new[] {
7.              "Arabic",
8.              "Chinese",
9.              "English",
10.             "French",
11.             "Russian",
12.             "Spanish"
13.         };
14.
15.         public void OnPost(string language) => Language = language;
16.
17.         //public void OnPost() => Language = Request.Form["language"];
18.     }
19. }
```

2.8. Query string, route
data, route template –
xử lý truy vấn GET

2.9. HTML Form trong
Razor Pages – xử lý
truy vấn POST

2.10. Xử lý form
control trong Razor
Pages

• Sử dụng radio button

• Ví dụ:

- Bước 3. Viết code razor cho **RadioButton.cshtml** như sau:

```
@page
@model WebApplication1.Pages.RadioButtonModel
 @{
    ViewData["Title"] = "Radio Button";
}

@if (Request.Method.ToUpper() == "GET") {
    <h3>What is your native language?</h3>

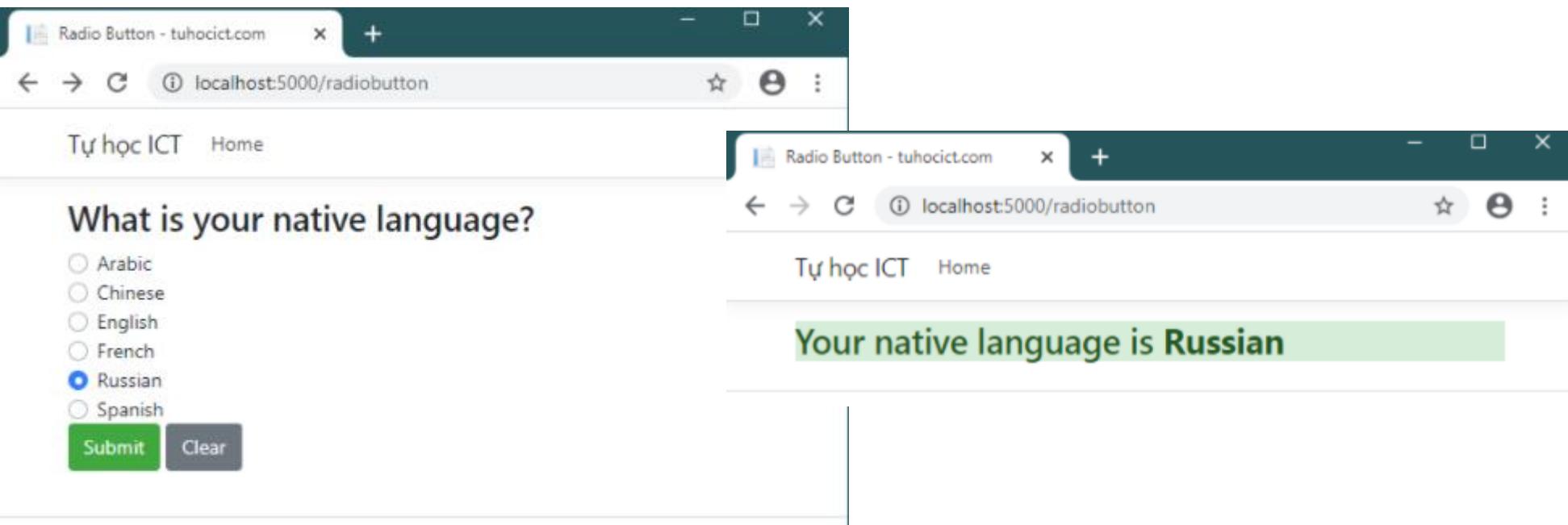
    <form method="post">
        @foreach (var l in Model.OfficialLanguages) {
            <div class="custom-control custom-radio">
                <input type="radio" class="custom-control-input" id="@l" name="language" value="@l">
                <label class="custom-control-label" for="@l">@l</label>
            </div>
        }
        <div class="form-group">
            <button class="btn btn-success" type="submit">Submit</button>
            <button class="btn btn-secondary" type="reset">Clear</button>
        </div>
    </form>
}
else if (Request.Method.ToUpper() == "POST") {
    <h3 class="alert-success">Your native language is <b>@Model.Language</b></h3>
}
```

2.8. Query string, route data, route template – xử lý truy vấn GET

2.9. HTML Form trong Razor Pages – xử lý truy vấn POST

2.10. Xử lý form control trong Razor Pages

- Sử dụng radio button
 - Ví dụ:



Lưu ý khi sử dụng nút đài:

- (1) Các input cùng thuộc một nhóm phải có cùng giá trị name;
- (2) Giá trị của thuộc tính value sẽ được gửi về server nếu nút đài được chọn;
- (3) Thẻ <label> hiển thị văn bản bên cạnh nút (value thì không hiển thị);
- (4) Nút sẽ được chọn mặc định nếu có mặt thuộc tính checked (không quan tâm đến giá trị).

2.8. Query string, route data, route template – xử lý truy vấn GET

2.9. HTML Form trong Razor Pages – xử lý truy vấn POST

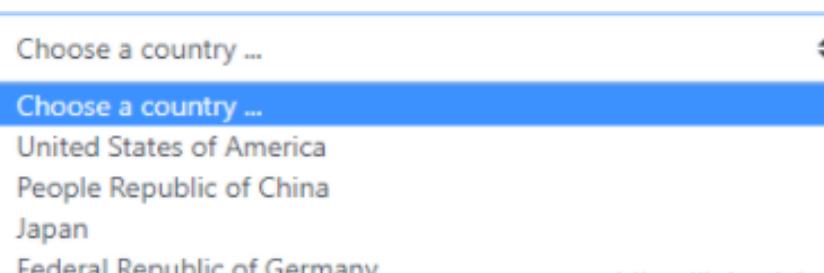
2.10. Xử lý form control trong Razor Pages

• Sử dụng Combo Box

- Combo Box cho phép bạn chọn một giá trị từ một danh sách. Trong HTML, **combo box** được tạo ra bằng thẻ `<select></select>`. Mỗi phương án chọn được tạo bằng thẻ `<option />` nằm bên trong `<select></select>`

```
1. <select class="browser-default custom-select" name="country">
2.   <option value="Not specified" selected>Choose a country ...</option>
3.   <option value="US">United States of America</option>
4.   <option value="China">People Republic of China</option>
5.   <option value="Japan">Japan</option>
6.   <option value="Germany">Federal Republic of Germany</option>
7. </select>
```

- Sẽ tạo ra combo box



- Thuộc tính **selected** quyết định xem option nào sẽ được chọn ngay từ đầu (giá trị mặc định).
- Khi bạn chọn một phần tử và **submit** form, dữ liệu từ thuộc tính **value** của **option** tương ứng sẽ trả về **server**. Trong ví dụ trên, giả sử bạn chọn mục “United States of America” thì giá trị trả về là US.

2.8. Query string, route
data, route template –
xử lý truy vấn GET

2.9. HTML Form trong
Razor Pages – xử lý
truy vấn POST

2.10. Xử lý form
control trong Razor
Pages

• Sử dụng Combo Box

• Ví dụ:

- **Bước 1.** Tạo trang mới **ComboBox**.
- **Bước 2.** Viết code cho model class trong **ComboBox.cshtml.cs** như sau:

```
1.  using Microsoft.AspNetCore.Mvc.RazorPages;
2.
3.  namespace WebApplication1.Pages {
4.      public class ComboBoxModel : PageModel {
5.          public string Country { get; private set; }
6.          public void OnPost(string country) => Country = country;
7.          //public void OnPost() => Country = Request.Form["country"];
8.      }
9.  }
```

• Bước 3:

```
1.  @page
2.  @model WebApplication1.Pages.ComboBoxModel
3.  @{
4.      ViewData["Title"] = "ComboBox";
5.      var method = Request.Method.ToUpper();
6.  }
7.
8.  @if (method == "GET") {
9.      <form method="post" class="border border-light p-5">
10.         <h3>What is your country of birth?</h3>
11.         <div>
12.             <select class="browser-default custom-select" name="country">
13.                 <option value="Not specified" selected>Choose a country ...</option>
14.                 <option value="US">United States of America</option>
15.                 <option value="China">People Republic of China</option>
16.                 <option value="Japan">Japan</option>
17.                 <option value="Germany">Federal Republic of Germany</option>
18.             </select>
19.         </div>
20.         <button class="btn btn-success btn-block my-2" type="submit">Submit</button>
21.     </form>
22. }
23. else if (method == "POST") {
24.     <h3>You were born in @Model.Country</h3>
25. }
```

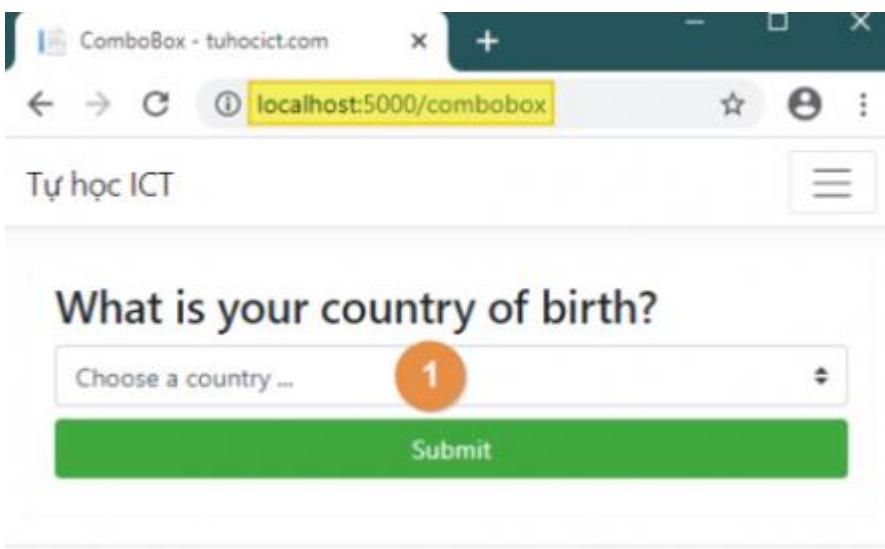
2.8. Query string, route
data, route template –
xử lý truy vấn GET

2.9. HTML Form trong
Razor Pages – xử lý
truy vấn POST

2.10. Xử lý form
control trong Razor
Pages

• Sử dụng Combo Box

• Ví dụ:



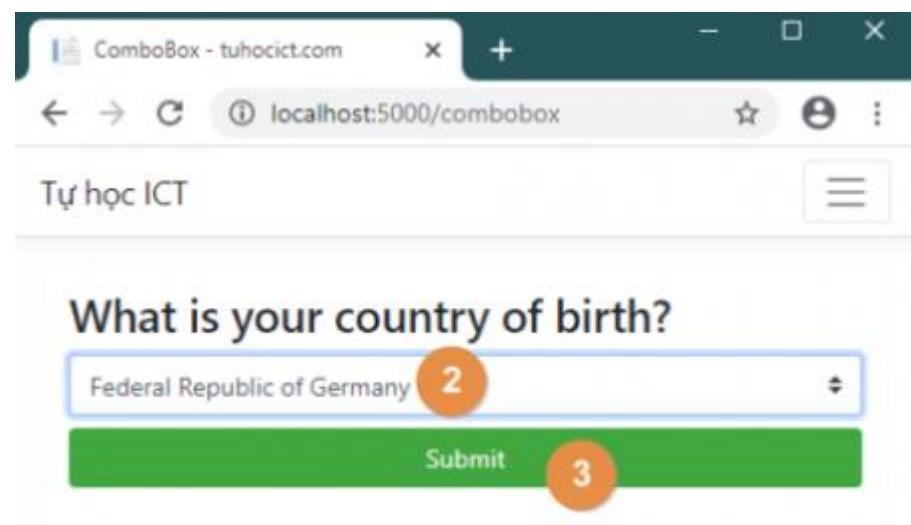
localhost:5000/combobox

Tự học ICT

What is your country of birth?

Choose a country ... **1**

Submit



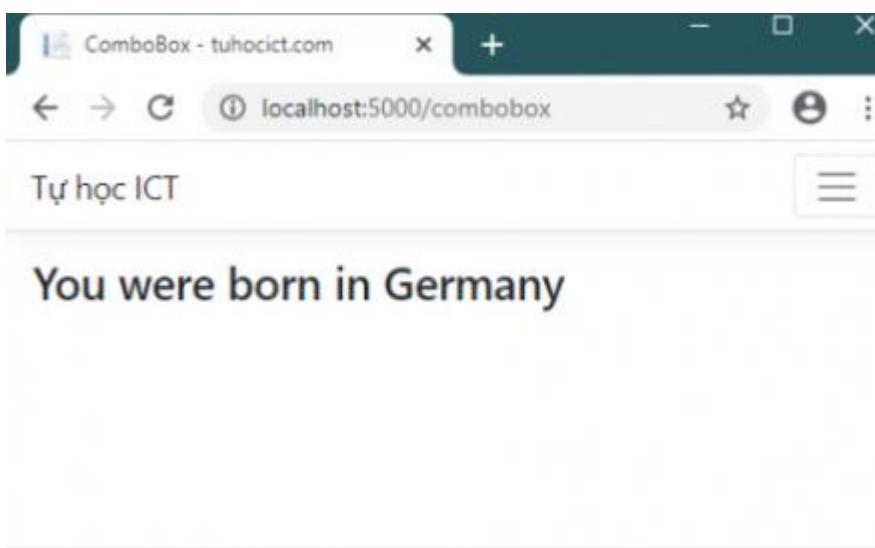
localhost:5000/combobox

Tự học ICT

What is your country of birth?

Federal Republic of Germany **2**

Submit **3**



localhost:5000/combobox

Tự học ICT

You were born in Germany

2.8. Query string, route data, route template – xử lý truy vấn GET

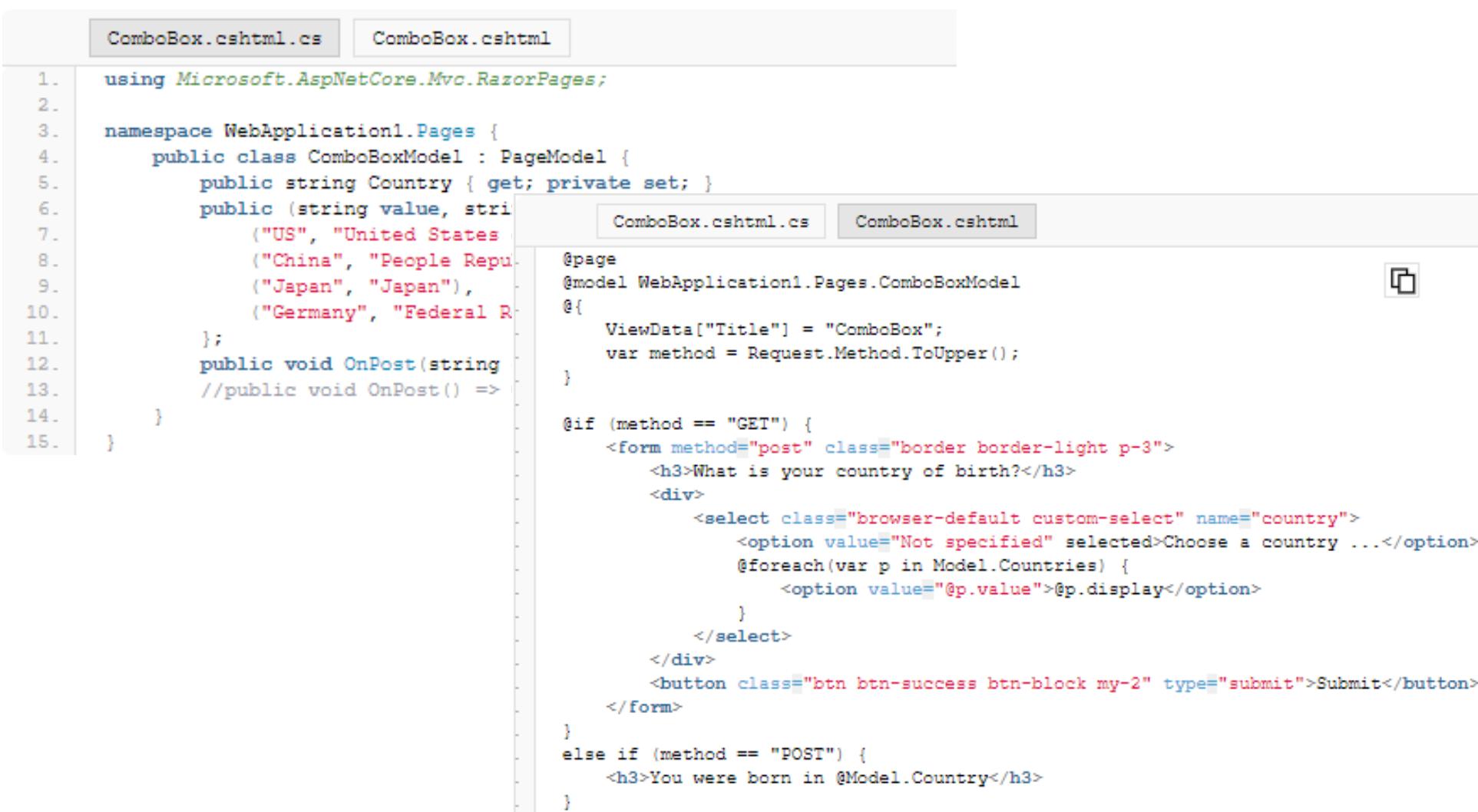
2.9. HTML Form trong Razor Pages – xử lý truy vấn POST

2.10. Xử lý form control trong Razor Pages

• Sử dụng Combo Box

• Ví dụ:

- Nếu ưa thích xử lý code thay cho viết thẻ HTML, bạn có thể sử dụng phương án sau:



The screenshot shows a code editor with two tabs: `ComboBox.cshtml.cs` and `ComboBox.cshtml`. The `ComboBox.cshtml.cs` tab contains C# code for a Razor Page model. The `ComboBox.cshtml` tab contains the corresponding Razor page markup.

```
ComboBox.cshtml.cs
```

```
ComboBox.cshtml
```

```
1. using Microsoft.AspNetCore.Mvc.RazorPages;
2.
3. namespace WebApplication1.Pages {
4.     public class ComboBoxModel : PageModel {
5.         public string Country { get; private set; }
6.         public (string value, string display)[] Countries {
7.             get {
8.                 return new[]
9.                     {
10.                         ("US", "United States"),
11.                         ("China", "People Repu."),
12.                         ("Japan", "Japan"),
13.                         ("Germany", "Federal R")
14.                     };
15.             }
16.         }
17.         public void OnPost(string country) {
18.             //public void OnPost() =>
19.         }
20.     }
21. }
```

```
ComboBox.cshtml
```

```
@page
@model WebApplication1.Pages.ComboBoxModel
 @{
    ViewData["Title"] = "ComboBox";
    var method = Request.Method.ToUpper();
}

@if (method == "GET") {
    <form method="post" class="border border-light p-3">
        <h3>What is your country of birth?</h3>
        <div>
            <select class="browser-default custom-select" name="country">
                <option value="Not specified" selected>Choose a country ...</option>
                @foreach(var p in Model.Countries) {
                    <option value="@p.value">@p.display</option>
                }
            </select>
        </div>
        <button class="btn btn-success btn-block my-2" type="submit">Submit</button>
    </form>
}
else if (method == "POST") {
    <h3>You were born in @Model.Country</h3>
}
```

2.8. Query string, route data, route template – xử lý truy vấn GET

2.9. HTML Form trong Razor Pages – xử lý truy vấn POST

2.10. Xử lý form control trong Razor Pages

• Sử dụng list box

- List box là một dạng khác của danh sách chọn, trong đó bạn có thể chọn nhiều mục từ một danh sách.
- Trong HTML, list box được tạo ra giống hệt như combo box, khác biệt duy nhất là trong thẻ `<select>` cần có thêm thuộc tính **multiple** (không cần giá trị):

```
1. <select name="countries" multiple>
2.   <option value="US">United States of America</option>
3.   <option value="China">People Republic of China</option>
4.   <option value="Japan">Japan</option>
5.   <option value="Germany">Federal Republic of Germany</option>
6. </select>
```



- Tuy cách tạo ra **listbox** tương tự như **combobox** nhưng cách xử lý **listbox** tại server lại giống hệt như đối với **checkbox** (do cùng trả về nhiều giá trị qua cùng một tên)

2.8. Query string, route data, route template – xử lý truy vấn GET

2.9. HTML Form trong Razor Pages – xử lý truy vấn POST

2.10. Xử lý form control trong Razor Pages

• Sử dụng list box

• Ví dụ:

- **Bước 1:** Tạo trang mới ListBox
- **Bước 2:** Viết code cho model class trong *ListBox.cshtml.cs* như sau:

```
1.  using System.Collections.Generic;
2.  using System.Linq;
3.  using Microsoft.AspNetCore.Mvc.RazorPages;
4.
5.  namespace WebApplication1.Pages {
6.      public class ListBoxModel : PageModel {
7.          public (string value, string display)[] Countries => new[] {
8.              ("US", "United States of America"),
9.              ("China", "People Republic of China"),
10.             ("Japan", "Japan"),
11.             ("Germany", "Federal Republic of Germany"),
12.         };
13.         public string[] VisitedCountries { get; private set; }
14.
15.         public void OnPost() {
16.             var countries = Request.Form["countries"];
17.             VisitedCountries = countries.ToArray();
18.         }
19.     }
20. }
```

2.8. Query string, route data, route template – xử lý truy vấn GET

2.9. HTML Form trong Razor Pages – xử lý truy vấn POST

2.10. Xử lý form control trong Razor Pages

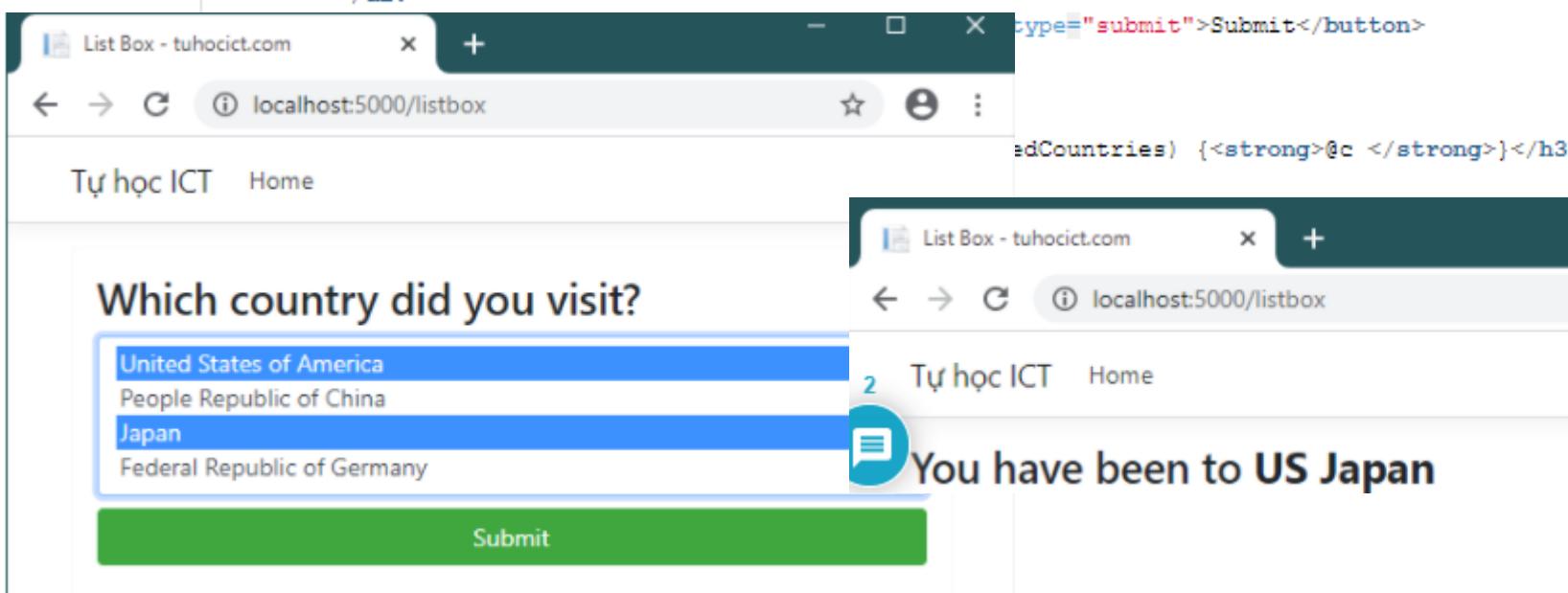
• Sử dụng list box

• Ví dụ:

- Bước 3: Viết mã razor cho ListBox.cshtml như sau:

```
@page
@model WebApplication1.Pages.ListBoxModel
 @{
    ViewData["Title"] = "List Box";
    var method = Request.Method.ToUpper();
}

@if (method == "GET") {
    <form method="post" class="border border-light p-3">
        <h3>Which country did you visit?</h3>
        <div>
            <select class="browser-default custom-select" name="countries" multiple>
                @foreach (var p in Model.Countries) {
                    <option value="@p.value">@p.display</option>
                }
            </select>
        </div>
    </form>
}
```



2.8. Query string, route data, route template – xử lý truy vấn GET

2.9. HTML Form trong Razor Pages – xử lý truy vấn POST

2.10. Xử lý form control trong Razor Pages

2.11. Model binding cơ bản trong Razor Pages

• Model binding

- là một loại kỹ thuật đặc trưng và rất quan trọng trong **Razor Pages** giúp bạn dễ dàng làm việc với dữ liệu người dùng cung cấp qua truy vấn.
- Nhờ **Model binding** bạn không cần phải trích dữ liệu từ truy vấn một cách thủ công. Thay vào đó, **model binding** sẽ trích dữ liệu tự động giúp quá trình đọc dữ liệu từ truy vấn nhanh và an toàn hơn.
- Khi học về cách *truy xuất dữ liệu từ URL* hoặc *truy xuất dữ liệu form*, bạn đã biết rằng **Razor Pages** phân tách các thành phần của truy vấn **HTTP** và đưa vào object **Request**. Bạn có thể truy xuất dữ liệu của truy vấn thông qua property **Query** hoặc **Form**.
- Ví dụ, nếu page nhận truy vấn GET với **URL** như sau:

```
/?username=Donald%20Trump&email=trump@gmail.com
```

- Bạn có thể truy xuất giá trị của username và email từ server code như sau:

```
var username = Request.Query["username"];  
var email = Request.Query["email"];
```

2.8. Query string, route data, route template – xử lý truy vấn GET

2.9. HTML Form trong Razor Pages – xử lý truy vấn POST

2.10. Xử lý form control trong Razor Pages

2.11. Model binding cơ bản trong Razor Pages

• Model binding

- Hoặc nếu bạn sử dụng form:

```
<form method="post">  
    <input name="username" type="text">  
    <input name="email" type="text">  
    <button type="submit">POST Subscribe</button>  
</form>
```

- Bạn sẽ truy xuất giá trị của các textbox username và email như sau:

```
var username = Request.Form["username"];  
var email = Request.Form["email"];
```

- Đây là cách thức truy xuất dữ liệu từ truy vấn cơ bản nhất và thường gặp trong các **web framework**.
- Phương pháp này đơn giản nhưng chỉ phù hợp với chuỗi truy vấn hoặc form đơn giản. Khi phải xử lý những form phức tạp với hàng chục thông tin, việc truy xuất dữ liệu như trên có thể tạo ra hàng loạt vấn đề.
- Vấn đề dễ thấy nhất là cơ chế nhắc code của **Intellisense** không hoạt động được khiến bạn phải ghi nhớ các tên gọi. Từ đây dẫn tới khả năng viết lầm. Ví dụ khi truy xuất email, bạn dễ dàng viết **Request.Form[“emial”]** thay cho **Request.Form[“email”]**. Những lỗi như vậy rất khó theo dõi.
- Để giải quyết những vấn tương tự, **Razor Pages** đưa ra cơ chế **model binding**.

2.8. Query string, route data, route template – xử lý truy vấn GET

2.9. HTML Form trong Razor Pages – xử lý truy vấn POST

2.10. Xử lý form control trong Razor Pages

2.11. Model binding cơ bản trong Razor Pages

- **Model binding**

- **Model binding** là cơ chế tự động phân tách giá trị từ truy vấn HTTP và ánh xạ chúng vào tham số của phương thức xử lý truy vấn (**handler method**) hoặc vào thuộc tính của *model class* tương ứng.
- Cơ chế **model binding** giải phóng lập trình viên khỏi việc trích xuất dữ liệu thủ công khỏi truy vấn rồi tự gán cho từng biến cục bộ. Qua đó, **model binding** giúp tránh lỗi và tăng hiệu quả công việc.

2.8. Query string, route data, route template – xử lý truy vấn GET

2.9. HTML Form trong Razor Pages – xử lý truy vấn POST

2.10. Xử lý form control trong Razor Pages

2.11. Model binding cơ bản trong Razor Pages

• Parameter và Property binding

- **Parameter binding** là ánh xạ dữ liệu của truy vấn vào tham số của phương thức xử lý (**handler**). Trong **parameter binding**, đặt tham số vào phương thức xử lý truy vấn. Tên của tham số cần trùng với tên của điều khiển (**form**)/tên tham số của chuỗi truy vấn/tên placeholder của **route**.
- **Property binding** là ánh xạ dữ liệu của truy vấn vào **property** của *model class*. Trong **property binding**, khai báo thêm các **property** trong *model class* sao cho tên **property** trùng với tên điều khiển/tham số placeholder, đồng thời đặt thêm **attribute [BindProperty]** phía trước.
- Khi này cơ chế **model binding** sẽ tự trích dữ liệu từ truy vấn, chuyển đổi thành kiểu phù hợp và truyền vào tham số và thuộc tính tương ứng.
- Sự tương quan giữa **tên tham số/property** và **tên điều khiển/tham số/placeholder** không phân biệt chữ hoa/thường. Nghĩa là bạn có thể đặt tên viết thường cho điều khiển nhưng vẫn có thể dùng tên viết hoa chữ cái đầu cho **property**. Như vậy bạn vẫn giữ được quy ước đặt tên ở **HTML** và **C#**.
- Do việc chuyển đổi kiểu là tự động, cần đặt kiểu tham số/ property **phù hợp** với dữ liệu chờ đợi. **Ví dụ**, nếu trên form bạn đặt input để người dùng nhập số thì cần đặt kiểu cho tham số / property tương ứng là một trong các kiểu số.
- Trong một số trường hợp bạn phải dùng kiểu đặc biệt, thay cho các kiểu cơ sở quen thuộc. **Ví dụ**, khi tải file, kiểu của tham số / property tương ứng với file tải lên là **IFormFile**.

2.8. Query string, route data, route template – xử lý truy vấn GET

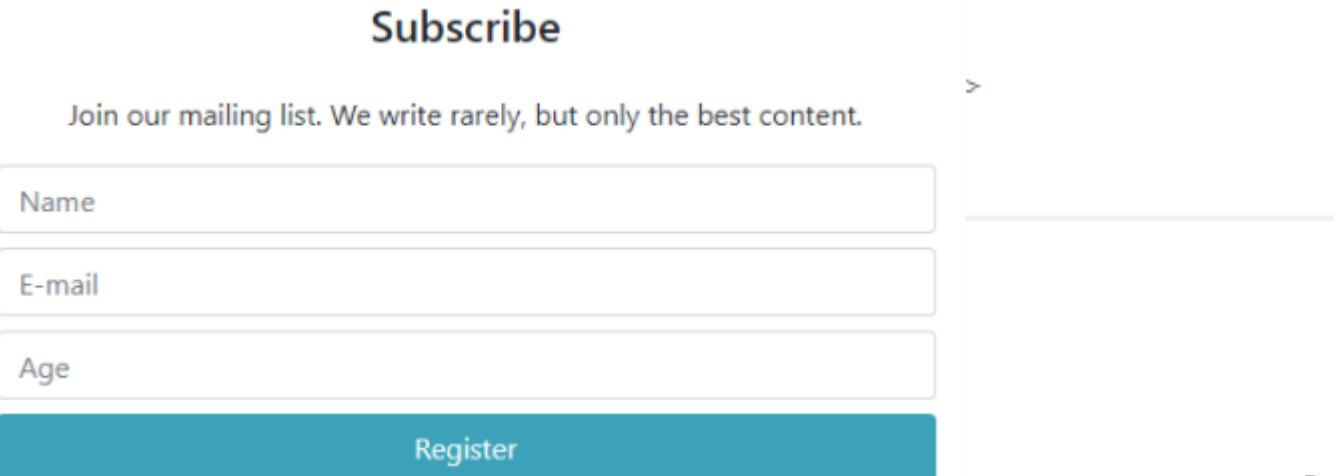
2.9. HTML Form trong Razor Pages – xử lý truy vấn POST

2.10. Xử lý form control trong Razor Pages

2.11. Model binding cơ bản trong Razor Pages

- **Sử dụng Model Binding với form:** Tạo page mới đặt tên là FormBinding. Viết code cho Razor view như sau:

```
1. @page
2. @model ModelBinding.Pages.FormBindingModel
3.
4. @{
5.     ViewData["Title"] = "FormBinding";
6.     var method = Request.Method.ToUpper();
7.
8.     @if (method == "GET") {
9.         <form class="center border border-light p-3 w-50" method="post">
10.             <div class="text-center">
11.                 <p class="h4 mb-4">Subscribe</p>
12.                 <p>Join our mailing list. We write rarely, but only the best content.</p>
13.             </div>
14.
15.             <input name="fullname" type="text" placeholder="Name" class="form-control mb-2">
16.             <input name="email" type="email" placeholder="E-mail" class="form-control mb-2">
17.             <input name="age" type="number" placeholder="Age" class="form-control mb-2">
18.             <button type="submit" class="btn btn-info btn-block">Register</button>
19.         </form>
20.     }
21.     else if
22.     <div>
23.         <div>
24.             <div>
25.             }>
```



2.8. Query string, route
data, route template –
xử lý truy vấn GET

2.9. HTML Form trong
Razor Pages – xử lý
truy vấn POST

2.10. Xử lý form
control trong Razor
Pages

2.11. Model binding cơ
bản trong Razor Pages

• Sử dụng Model Binding với form:

- **Parameter binding:** Nếu sử dụng parameter binding, model class sẽ có dạng như sau:

```
using Microsoft.AspNetCore.Mvc.RazorPages;

namespace ModelBinding.Pages {
    public class FormBindingModel : PageModel {
        public string Message { get; private set; }

        public void OnPost(string fullName, string eMail, int age) => Message = age < 18 ?
            $"Sorry, {fullName}. You cannot subscribe for our site." :
            $"Hello, {fullName}. Thank you for your subscription. We will send email to the
        }
    }
}
```

- Tham số của **OnGet** lần lượt là **fullName**, **eMail** và **age**. Riêng **age** có kiểu là **int**

Hãy lưu ý rằng,

- (1) tên tham số của **OnPost** và tên điều khiển tương ứng có sự khác biệt về chữ hoa/thường. Razor Pages vẫn có khả năng ánh xạ chính xác sang tham số tương ứng của **OnGet**;
- (2) mặc dù trong truy vấn HTTP, mọi thứ đều là văn bản (từ góc nhìn của C#) nhưng Razor Pages vẫn chuyển đổi được giá trị từ điều khiển **age** (form) về giá trị số nguyên cho tham số **age** của **OnPost**.

2.8. Query string, route data, route template – xử lý truy vấn GET

2.9. HTML Form trong Razor Pages – xử lý truy vấn POST

2.10. Xử lý form control trong Razor Pages

2.11. Model binding cơ bản trong Razor Pages

• Sử dụng Model Binding với form:

- **Property binding:** Thay thế code của FormBindingModel như sau:

```
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.RazorPages;

namespace ModelBinding.Pages {
    public class FormBindingModel : PageModel {
        [BindProperty]
        public int Age { get; set; }
        [BindProperty]
        public string FullName { get; set; }
        [BindProperty]
        public string Email { get; set; }

        public string Message { get; private set; }

        public void OnPost() => Message = Age < 18 ?
            $"Sorry, {FullName}. You cannot subscribe for our site." :
            $"Hello, {FullName}. Thank you for your subscription. We will send email to the
        }
    }
}
```

- Trong đoạn code trên, bỏ tham số trong **OnPost**. Thay vào đó, khai báo thêm 3 **property**: **Age**, **FullName** và **Email**. Các **property** này đều được đánh dấu bởi attribute **[BindProperty]**. Bạn có thể thấy tên **property** không hoàn toàn trùng khớp (hoa/thường) với tên điều khiển.
- Trong trường hợp này **Razor Pages** sẽ gắn dữ liệu từ truy vấn vào các **property** có tên tương ứng. Bạn có thể truy xuất các **property** ở bất kỳ đâu trong **model class** cũng như trên **Razor view**.

2.8. Query string, route
data, route template –
xử lý truy vấn GET

2.9. HTML Form trong
Razor Pages – xử lý
truy vấn POST

2.10. Xử lý form
control trong Razor
Pages

2.11. Model binding cơ
bản trong Razor Pages

• Sử dụng Model Binding với form:

• Property binding cho toàn thể model class:

- Nếu bạn có nhiều property hơn nữa, việc đánh dấu **[BindProperty]** cho từng property làm code nhìn rối hơn nhiều. Bắt đầu từ **ASP.NET Core 2.1** bạn có thể sử dụng attribute **[BindProperties]** cho model class như sau:

```
namespace ModelBinding.Pages {  
    [BindProperties]  
    public class FormBindingModel : PageModel {  
        public int Age { get; set; }  
        public string FullName { get; set; }  
        public string Email { get; set; }  
  
        public string Message { get; private set; }  
  
        public void OnPost() => Message = Age < 18 ?  
            $"Sorry, {FullName}. You cannot subscribe for our site."  
            $"Hello, {FullName}. Thank you for your subscription. We will send email to the :  
    }  
}
```

- [BindProperties]** tự động áp dụng property binding cho bất kỳ **public property** nào của class. Theo đó, **Age**, **FullName**, **Email** đều tham gia vào **property binding**.
- Tuy nhiên cách này có một vấn đề nhỏ: **Message** cũng tham gia vào **property binding** vì nó cũng là một **public property**. Vì vậy, khi sử dụng cần lưu ý đến đặc điểm của class để hạn chế những property không cần thiết tham gia vào quá trình binding.

2.8. Query string, route data, route template – xử lý truy vấn GET

2.9. HTML Form trong Razor Pages – xử lý truy vấn POST

2.10. Xử lý form control trong Razor Pages

2.11. Model binding cơ bản trong Razor Pages

• Sử dụng Model Binding với chuỗi truy vấn

- Giả sử chúng ta muốn xử lý chuỗi truy vấn có dạng:
?fullname=xyz&email=abc@gmail.com&age=18.
- Tạo page mới **UrlBinding** và viết code cho **Razor view** như sau:

```
1. @page
2. @model ModelBinding.Pages.UrlBindingModel
3. @{
4.     ViewData["Title"] = "UrlBinding";
5. }
6.
7. <strong>@Model.Message</strong>
```

• Sử dụng parameter binding

- Nếu sử dụng **parameter binding**, code của *model class* như sau:

```
using Microsoft.AspNetCore.Mvc.RazorPages;

namespace ModelBinding.Pages {
    public class UrlBindingModel : PageModel {
        public string Message { get; set; }
        public void OnGet(string fullName, string eMail, int age) => Message = age < 18 ?
            $"Sorry, {fullName}. You cannot subscribe for our site." :
            $"Hello, {fullName}. Thank you for your subscription. We will send email to the address {eMail}."
    }
}
```

2.8. Query string, route
data, route template –
xử lý truy vấn GET

2.9. HTML Form trong
Razor Pages – xử lý
truy vấn POST

2.10. Xử lý form
control trong Razor
Pages

2.11. Model binding cơ
bản trong Razor Pages

- Sử dụng Model Binding với chuỗi truy vấn
 - Sử dụng property binding

- Mặc định **property binding** chỉ hỗ trợ truy vấn **POST**. Nếu muốn sử dụng với truy vấn **GET** để trích dữ liệu từ chuỗi truy vấn vào **property**, cần điều chỉnh attribute **[BindProperty]** thành **[BindProperty(SupportsGet = true)]**

```
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.RazorPages;

namespace ModelBinding.Pages {
    public class UrlBindingModel : PageModel {
        [BindProperty(SupportsGet = true)]
        public int Age { get; set; }

        [BindProperty(SupportsGet = true)]
        public string FullName { get; set; }

        [BindProperty(SupportsGet = true)]
        public string Email { get; set; }

        public string Message { get; private set; }

        public void OnGet() => Message = Age < 18 ?
            $"Sorry, {FullName}. You cannot subscribe for our site." :
            $"Hello, {FullName}. Thank you for your subscription. We will send email to the a
        }
    }
}
```

- Tương tự, bạn cũng có thể sử dụng attribute **[BindProperties(SupportsGet = true)]** cho *model class*.

2.8. Query string, route
data, route template –
xử lý truy vấn GET

2.9. HTML Form trong
Razor Pages – xử lý
truy vấn POST

2.10. Xử lý form
control trong Razor
Pages

2.11. Model binding cơ
bản trong Razor Pages

• Sử dụng Model Binding với route data

- **Route data** là loại dữ liệu được đặt trong phần *path của URL* thay vì trong query string. Như đã học, **Razor Pages** cho phép **định nghĩa route** (mẫu URL) của mỗi page sử dụng cụm đánh dấu (**placeholder**). Cụm đánh dấu được sử dụng như một biến trong quá trình trích xuất dữ liệu từ **URL**.
- Tạo page mới **RouteBinding**. Viết code như sau cho **Razor view**:

```
1. @page "/register/{fullname?}/{email?}/{age:int?}"
2. @model ModelBinding.Pages.RouteBindingModel
3. @{
4.     ViewData["Title"] = "RouteBinding";
5. }
6. <a href="/register/Donald&20Trump/trump@gmail.com/18">Get binding</a>
7. <strong>@Model.Message</strong>
```

2.8. Query string, route
data, route template –
xử lý truy vấn GET

2.9. HTML Form trong
Razor Pages – xử lý
truy vấn POST

2.10. Xử lý form
control trong Razor
Pages

2.11. Model binding cơ
bản trong Razor Pages

• Sử dụng Model Binding với route data

- Lúc này bạn có thể sử dụng **parameter binding** để trích xuất dữ liệu từ Url path như sau:

```
using Microsoft.AspNetCore.Mvc.RazorPages;

namespace ModelBinding.Pages {
    public class RouteBindingModel : PageModel {
        public string Message { get; private set; }
        public void OnGet(string FullName, string Email, int Age) => Message = Age < 18 ?
            $"Sorry, {FullName}. You cannot subscribe for our site." :
            $"Hello, {FullName}. Thank you for your subscription. We will send email to the {Email}."
    }
}
```

- Do đây cũng là một truy vấn GET, bạn có thể tiếp tục sử dụng kỹ thuật **property binding** cho route data, giống hệt như đôi với query string:

```
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.RazorPages;

namespace ModelBinding.Pages {
    // [BindProperties(SupportsGet = true)]
    public class RouteBindingModel : PageModel {

        [BindProperty(SupportsGet = true)]
        public int Age { get; set; }

        [BindProperty(SupportsGet = true)]
        public string FullName { get; set; }

        [BindProperty(SupportsGet = true)]
        public string Email { get; set; }

        public string Message { get; private set; }

        public void OnGet() => Message = Age < 18 ?
            $"Sorry, {FullName}. You cannot subscribe for our site." :
            $"Hello, {FullName}. Thank you for your subscription. We will send email to the {Email}."
    }
}
```

2.8. Query string, route data, route template – xử lý truy vấn GET

2.9. HTML Form trong Razor Pages – xử lý truy vấn POST

2.10. Xử lý form control trong Razor Pages

2.11. Model binding cơ bản trong Razor Pages

• Sử dụng Model Binding với object

- Từ đầu đến giờ chúng ta đều chỉ vận dụng model binding với các kiểu dữ liệu đơn giản như string, int. Nhìn chung, model binding hoạt động tương tự nhau với hầu hết các kiểu dữ liệu cơ sở của C#.
- Tuy nhiên, trong các dự án, chắc chắn sẽ phải xây dựng các *domain class* riêng với rất nhiều **property**. Với kỹ thuật đã học, có thể khai báo các **property** của *domain class* ngay trong *model class* để sử dụng **property binding**.
- Tuy nhiên, việc trộn lẫn *domain class* với *model class* là rất khó chấp nhận vì hai loại class này có mục đích khác biệt.
- **Razor Pages** cho phép thực hiện **property binding** với cả các class phức tạp do mình tự xây dựng.

2.8. Query string, route data, route template – xử lý truy vấn GET

2.9. HTML Form trong Razor Pages – xử lý truy vấn POST

2.10. Xử lý form control trong Razor Pages

2.11. Model binding cơ bản trong Razor Pages

• Sử dụng Model Binding với object

- Sử dụng **property binding**: Hãy cùng thực hiện ví dụ sau:

- **Bước 1.** Tạo thư mục **Models** trực thuộc dự án.
- **Bước 2.** Tạo class **Subscription** trong thư mục **Models** và viết code như sau:

```
1.  namespace ModelBinding.Models {
2.      public class Subscription {
3.          public int Id { get; set; }
4.          public string FullName { get; set; }
5.          public string Email { get; set; }
6.          public int Age { get; set; }
7.      }
8.  }
```

- **Bước 3.** Tạo trang mới trong thư mục **Pages** đặt tên là **ObjectBinding**.
- **Bước 4.** Viết code như sau cho *model class* (file **ObjectBinding.cshtml.cs**):

```
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.RazorPages;

using ModelBinding.Models;

namespace ModelBinding.Pages {
    public class ObjectBindingModel : PageModel {
        public string Message { get; private set; }
        private string ResponseMessage(Subscription subscription) => Message = subscription.Age
            $"Sorry, {subscription.FullName}. You cannot subscribe for our site." :
            $"Hello, {subscription.FullName}. Thank you for your subscription. We will send email

        [BindProperty]
        public Subscription Subscription { get; set; }
        public void OnPost() => Message = ResponseMessage(Subscription);

        //public void OnPost(Subscription subscription) => Message = ResponseMessage(subscription)
    }
}
```

2.8. Query string, route
data, route template –
xử lý truy vấn GET

2.9. HTML Form trong
Razor Pages – xử lý
truy vấn POST

2.10. Xử lý form
control trong Razor
Pages

2.11. Model binding cơ
bản trong Razor Pages

• Sử dụng Model Binding với object

- Sử dụng **property binding**: Hãy cùng thực hiện ví dụ sau:

- **Bước 5.** Viết code như sau cho file *ObjectBinding.cshtml*:

```
@page
@model ModelBinding.Pages.ObjectBindingModel
{
    ViewData["Title"] = "ObjectBinding";
    var method = Request.Method.ToUpper();
}

@if (method == "GET") {
    <form class="center border border-light p-3 w-50" method="post">
        <div class="text-center">
            <p class="h4 mb-4">Subscribe</p>
            <p>Join our mailing list. We write rarely, but only the best content.</p>
        </div>

        <input name="Subscription.Fullname" type="text" placeholder="Name" class="form-control mb-2" />
        <input name="Subscription.Email" type="email" placeholder="E-mail" class="form-control mb-2" />
        <input name="Subscription.Age" type="number" placeholder="Age" class="form-control mb-2" />
        <button type="submit" class="btn btn-info btn-block">Register</button>
    </form>
}
else if (method == "POST") {
    <div class="center text-center border border-success p-3 w-50">
        <p class="h5">@Model.Message</p>
    </div>
}
```

- Chạy chương trình và thu được kết quả như các ví dụ khác.
- Cơ chế **model binding** của **Razor Pages** có khả năng tự trích dữ liệu từ form và dùng chúng để khởi tạo object.

2.8. Query string, route data, route template – xử lý truy vấn GET

2.9. HTML Form trong Razor Pages – xử lý truy vấn POST

2.10. Xử lý form control trong Razor Pages

2.11. Model binding cơ bản trong Razor Pages

• Sử dụng Model Binding với object

• Sử dụng parameter binding:

- Nếu không muốn sử dụng **property binding**, bạn cũng có thể áp dụng **parameter binding** cho phương thức **OnPost** như sau:

```
using Microsoft.AspNetCore.Mvc.RazorPages;  
  
using ModelBinding.Models;  
  
namespace ModelBinding.Pages {  
    public class ObjectBindingModel : PageModel {  
        public string Message { get; private set; }  
        private string ResponseMessage(Subscription subscription) => Message = subscription.Age  
            $"Sorry, {subscription.FullName}. You cannot subscribe for our site."  
            $"Hello, {subscription.FullName}. Thank you for your subscription. We will send email  
  
        public void OnPost(Subscription subscription) => Message = ResponseMessage(subscription)  
    }  
}
```

- Trong trường hợp này, biến `subscription` thuộc kiểu **Subscription** được sử dụng làm tham số của **OnPost**. Cơ chế **model binding** của Razor Pages có khả năng tự trích dữ liệu từ form và dùng chúng để khởi tạo object.

2.8. Query string, route data, route template – xử lý truy vấn GET

2.9. HTML Form trong Razor Pages – xử lý truy vấn POST

2.10. Xử lý form control trong Razor Pages

2.11. Model binding cơ bản trong Razor Pages

• Sử dụng Model Binding với object

• Object binding với truy vấn GET:

- Mặc định việc **binding** với object được sử dụng với truy vấn **POST**. Tuy nhiên có thể sử dụng kỹ thuật này với truy vấn **GET**. Tuy nhiên cần điều chỉnh một chút.
- Nếu sử dụng **parameter binding**, bạn cần thêm attribute **[FromQuery]** vào trước tham số của **handler**. Attribute này báo cho Razor Pages biết cần trích dữ liệu từ chuỗi truy vấn.

```
1. public void OnGet([FromQuery] Subscription subscription) {  
2.     Message = ResponseMessage(subscription);  
3. }
```

- Phương thức **OnGet** như trên có thể trích dữ liệu từ **query string** và khởi tạo object **subscription** dựa trên dữ liệu thu được. Có thể nhập các truy vấn với **query string** như bình thường:

```
?fullname=xyz&email=abc@gmail.com&age=18
```

- Sở dĩ phải có attribute **[FromQuery]** là vì Razor Pages mặc định hiểu rằng **parameter binding** cho object sẽ luôn lấy dữ liệu từ form. Tức là, nếu không chỉ định rõ nguồn thì Razor Pages sẽ luôn hiểu object tham số của handler sẽ được áp dụng attribute **[FromForm]**. Do vậy, khi đọc dữ liệu từ form, không cần viết **[FromForm]** trước tham số nữa.

2.8. Query string, route data, route template – xử lý truy vấn GET

2.9. HTML Form trong Razor Pages – xử lý truy vấn POST

2.10. Xử lý form control trong Razor Pages

2.11. Model binding cơ bản trong Razor Pages

• Sử dụng Model Binding với object

• Object binding với route data:

- Object binding có thể hoạt động cả với route data.
- Giả sử định nghĩa route cho trang bằng directive `@page "/register/{fullname?}/{email?}/{age:int?}"`
- Nếu sử dụng **parameter binding**, bạn cần chỉ định attribute `[FromRoute]` trước tên tham số để báo cho Razor Pages biết rằng cần trích dữ liệu từ route data để khởi tạo object tương ứng của tham số.

```
1. public void OnGet([FromRoute] Subscription subscription) {  
2.     Message = ResponseMessage(subscription);  
3. }
```

- Nếu sử dụng **property binding**, bạn áp dụng `[BindProperty(SupportsGet=true)]` giống hệt như khi sử dụng chuỗi truy vấn:

```
1. [BindProperty(SupportsGet=true)]  
2. public Subscription Subscription {get; set;}
```

Thank you!