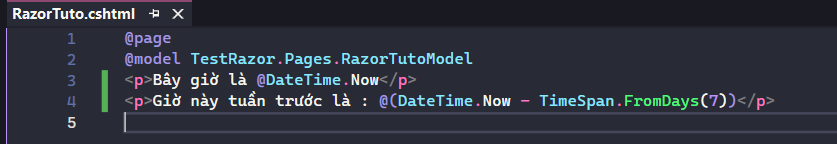
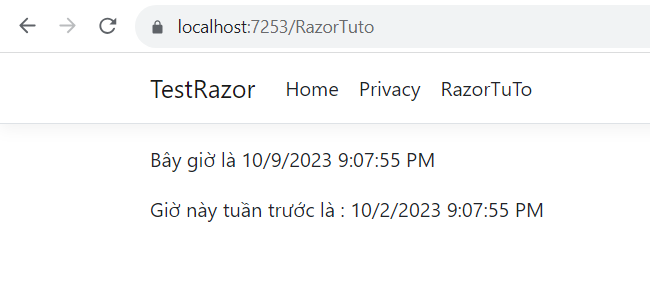
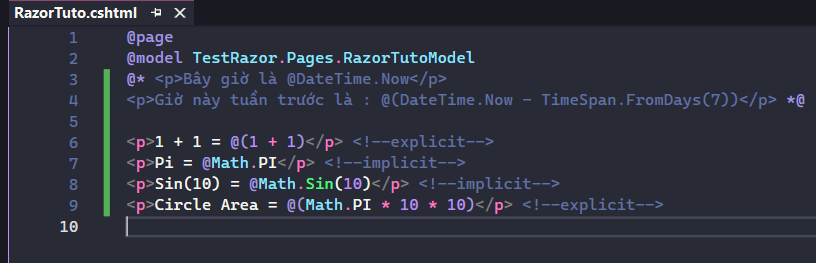
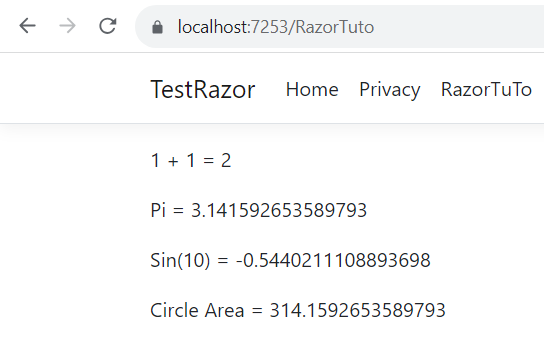
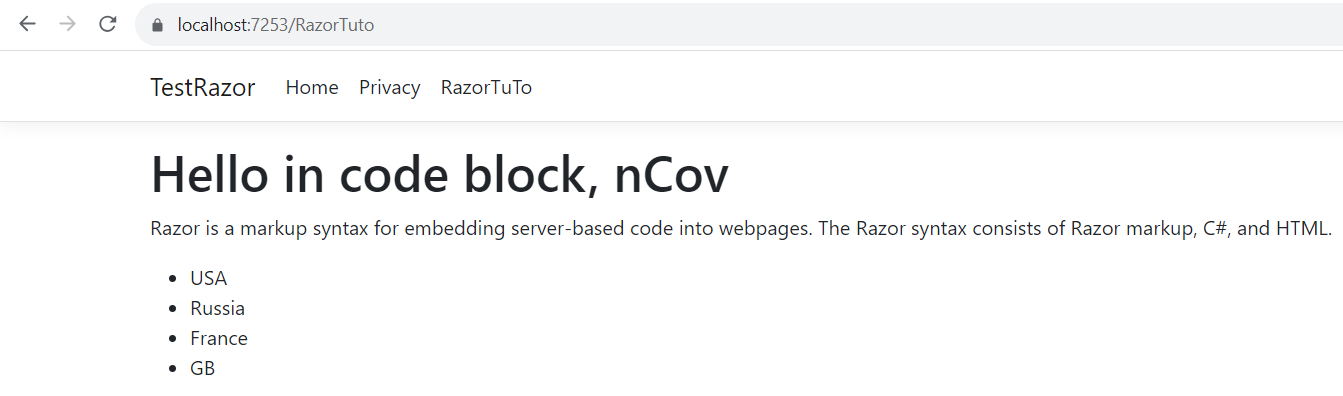
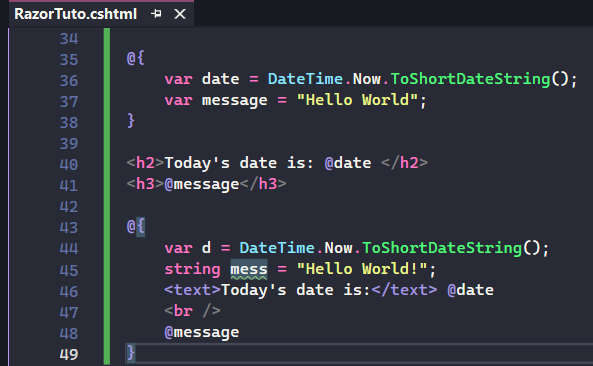
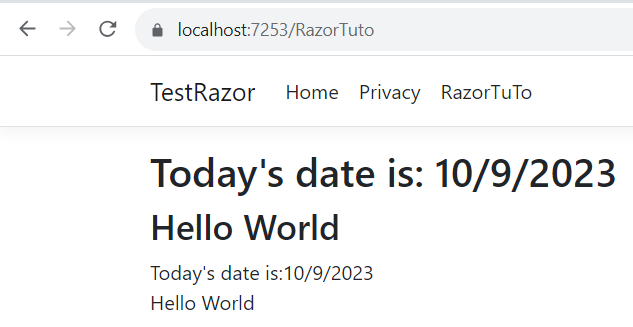
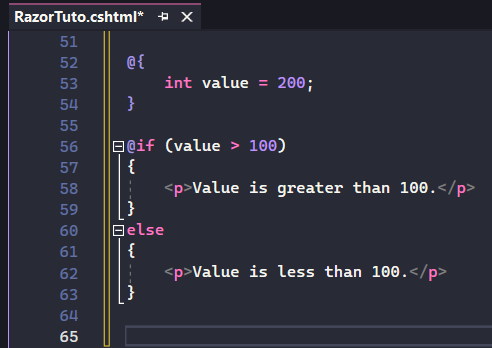
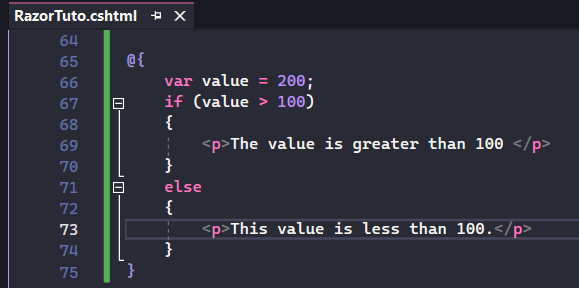
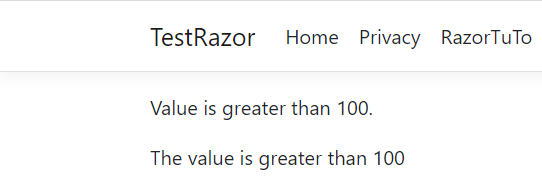
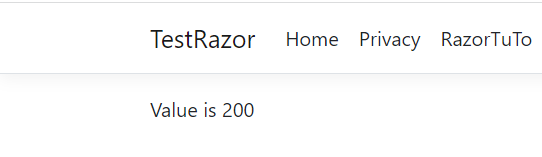
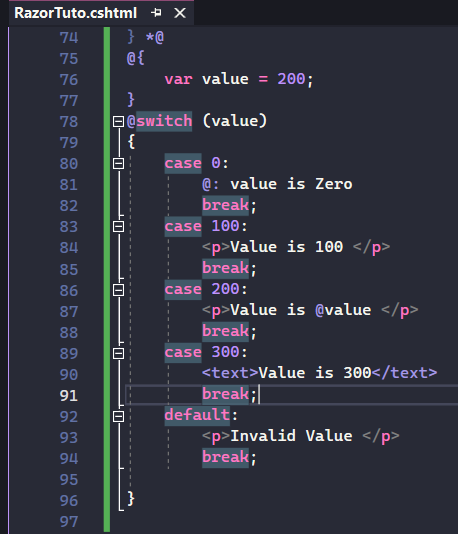
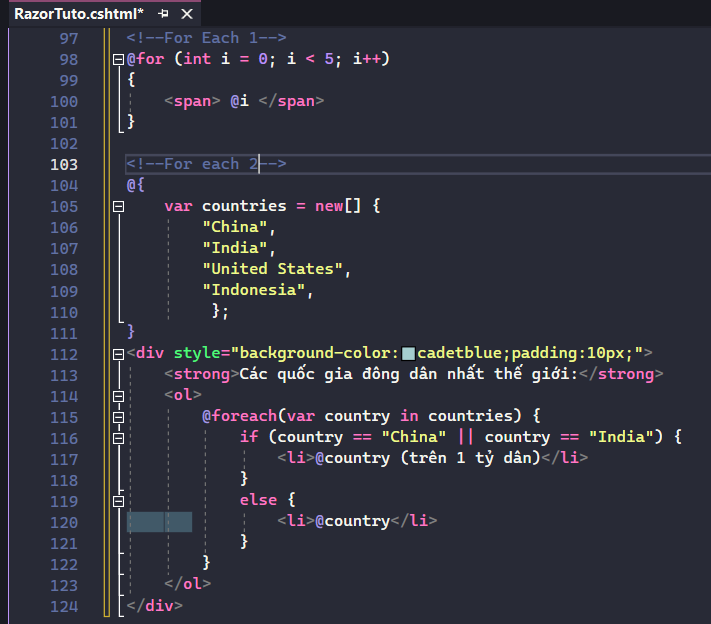
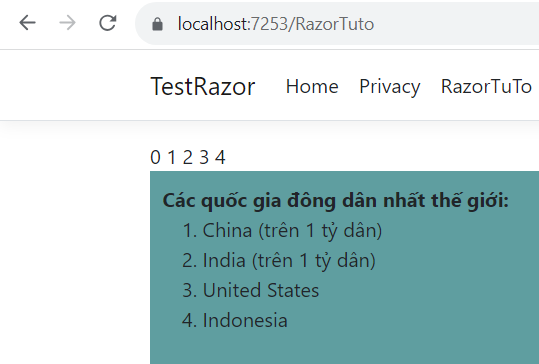
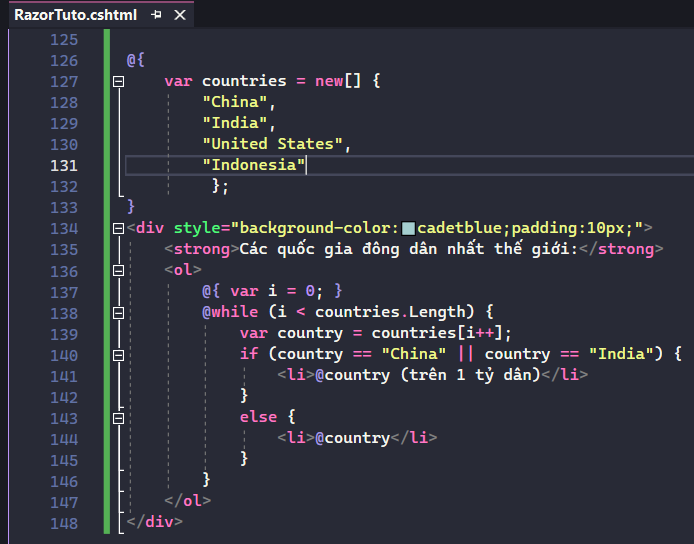
1. **Razor View Engine**

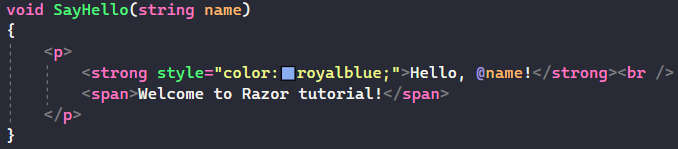
* Razor là loại cú pháp kết hợp C# và HTML để sinh ra HTML động trong ứng dụng ASP.NET Core.
* Razor View Engine là View Engine mặc định của ASP.NET Core. Nó lấy mã Razor trong file View và chuyển sang HTML response. Người ta tạo ra View Engine với cách thức hoạt động như một compiler để chuyển loại ngôn ngữ lai này thành HTML
* Directive tương tự như C# vậy. Nó cho phép chúng ta import namespace. Thường thấy :
* @page – bắt buộc nằm ở dòng đầu tiên của 1 content page nếu không Razor sẽ báo lỗi. Directive @page giúp phân biệt 1 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

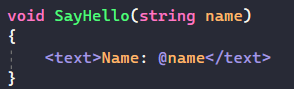
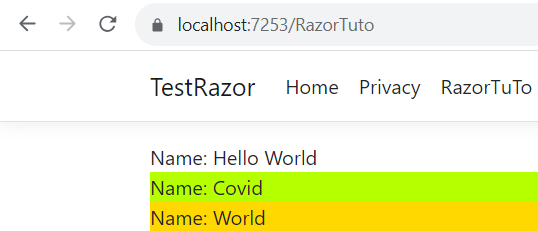
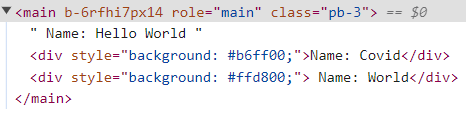
1. **Razor syntax fundamentals**

* Trong file cshtml (cũng được gọi là razor file/page), cú pháp Razor có nhiệm vụ xác định xem đâu là C# và đâu là HTML để View Engine có thể dịch và xử lý chính xác.
* Razor sử dụng ký tự @ để chuyển HTML sang C#.  Có 2 cách để khai báo :
* Sử dụng Razor Expressions : 2 loại implicit (ngầm định) và explicit (tường minh)
* Biểu thức Razor là biểu thức C# được ghép trực tiếp cùng HTML thông qua ký tự @. Biểu thức Razor có tác dụng chèn giá trị vào vị trí tương ứng của HTML. Ví dụ :
* Biểu thức ngầm định (implicit): không được phép chứa khoảng trống vì nó dùng để kết thúc biểu thức hoặc những ký tự đặc biệt gây nhầm lẫn với HTML xung quanh. Example : biểu thức ngầm định không thể chứa generics( vì generics sử dụng cặp dấu <> với tham số kiểu, dẫn đến nhầm lẫn với HTML) và cũng không thể là các phép toán thông thường (+, -, \*, /)
* Biểu thức tường minh (explicit) : Nếu 1 biểu thức không thể biểu diễn ở dạng implicit thì sử dụng explicit với biểu thức đặt trong cặp dấu ()
* Sử dụng khối code Razor
*  Khối code Razor là tất cả code nằm trong một vùng @{…}. Khối code là 1 vùng trong file/page Razor trong đó ngôn ngữ mặc định là C#. Ví dụ :
* Trên ví dụ trên, đã khai báo và khởi tạo biến “str”, func “SayHello”, func “Solve”, biến "sol”
* Không giới hạn số lượng và vị trí viết code block trên 1 page
* Vị trí khai báo hàm không quan trọng, nghĩa là có thể gọi func ở code block trước nơi hàm đó được khai báo. Ví dụ trên, lời gọi hàm func “Solve” được viết trước khi hàm được khai báo
* Biến chỉ có thể sử dụng sau vị trí khai báo != func. Nếu sử dụng biến trước khi khai báo sẽ lỗi. Ví dụ trên, biến “str” được khai báo trước khi được sử dụng “<p>@str</p>”
* Một số lệnh khác
* Đa dòng lệnh : Đối với nhiều dòng bạn cần phải có thẻ <text></text>. Ví dụ dưới đây đều cho ra kết quả giống nhau
* If else

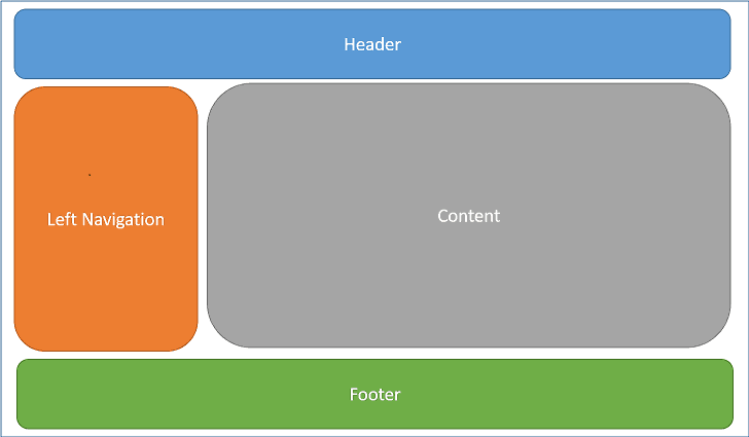
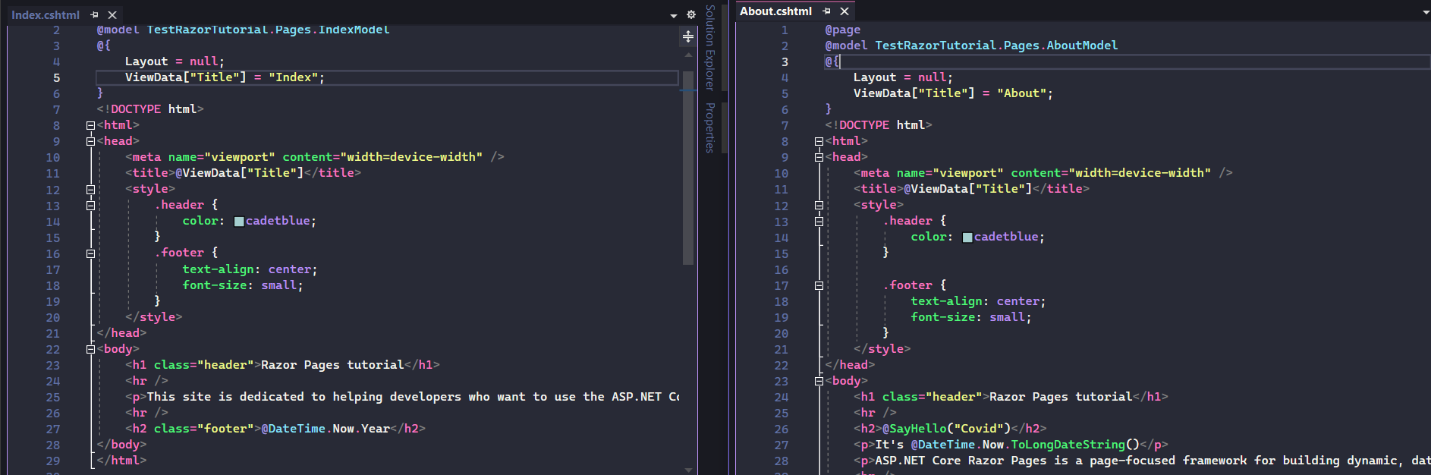
||

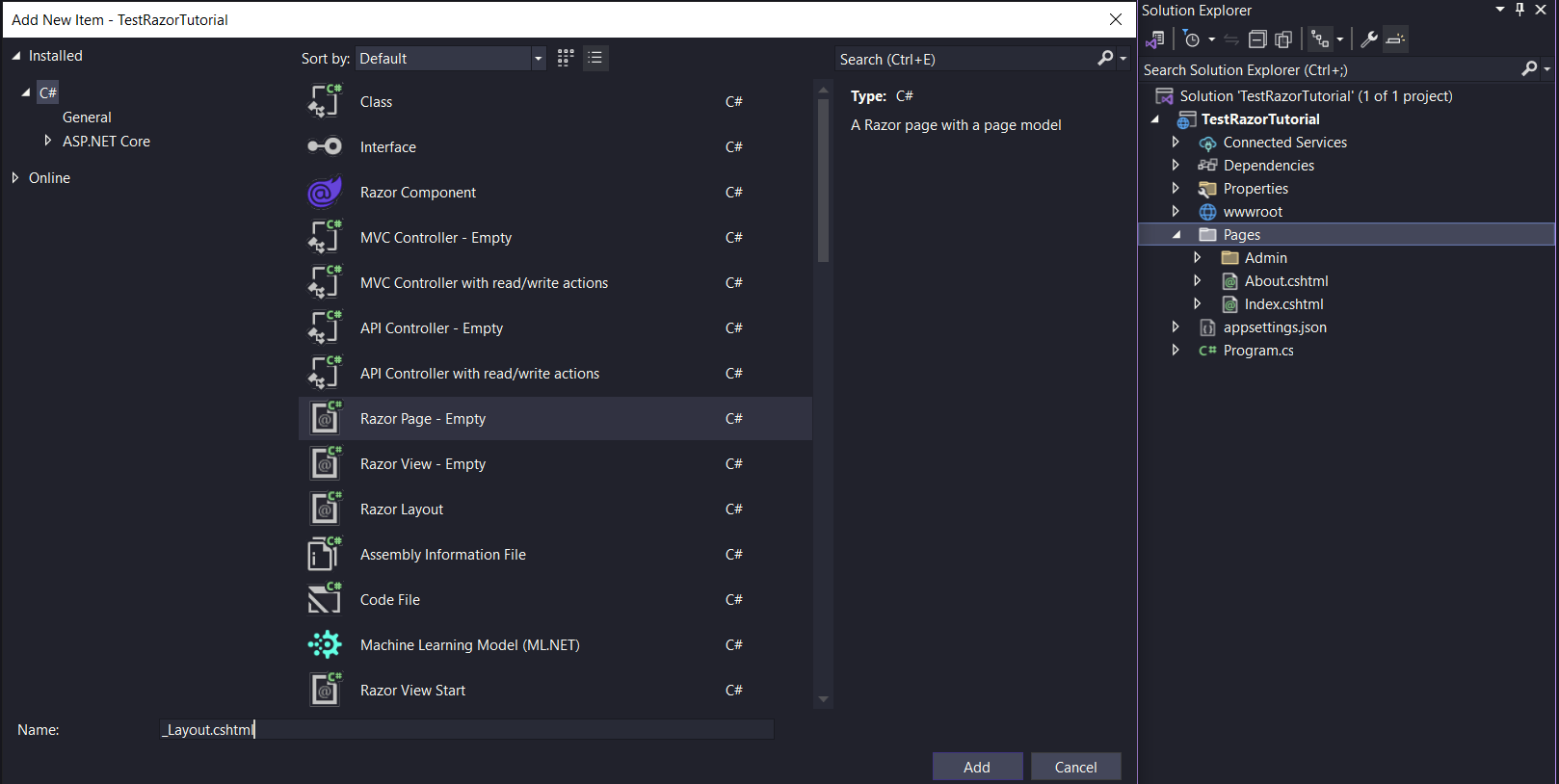
* Switch case
* For each
* While
* Tự động chuyển đổi ngôn ngữ trong code block:

1. Trong func “SayHello” chứa HTML, Razor tự động chuyển đổi từ C# sang HTML nếu nó thấy thẻ HTML trong thân. Cơ chế này gọi là Implicit Transition (chuyển đổi ngôn ngữ tự động).
2. Cơ chế chuyển đổi ngôn ngữ tự động hoạt động trong cả thân hàm và code block. Một khi đã chuyển sang HTML, Razor sẽ chỉ chuyển đổi trở lại C# sau khi cặp thẻ đóng. Trong khối HTML phải chuyển đổi từ HTML sang C#.

* Chủ động chuyển đổi ngôn ngữ : Trong một số trường hợp cơ chế chuyển đổi ngôn ngữ tự động có thể hoạt động không đúng ý. Khi đó, có thể lựa chọn sử dụng cơ chế chuyển đổi ngôn ngữ chủ động của Razor. Ví dụ :
* Do không dùng thẻ HTML, Razor không hiểu đang muốn xuất HTML.
* F12 trình duyệt sẽ thấy : Rõ ràng kết quả xuất ra HTML của hàm không sử dụng thẻ nào. Nghĩa là, <text></text> chỉ có tác dụng thông báo cho Razor để chuyển sang chế độ render HTML chứ không phải là một thẻ HTML.

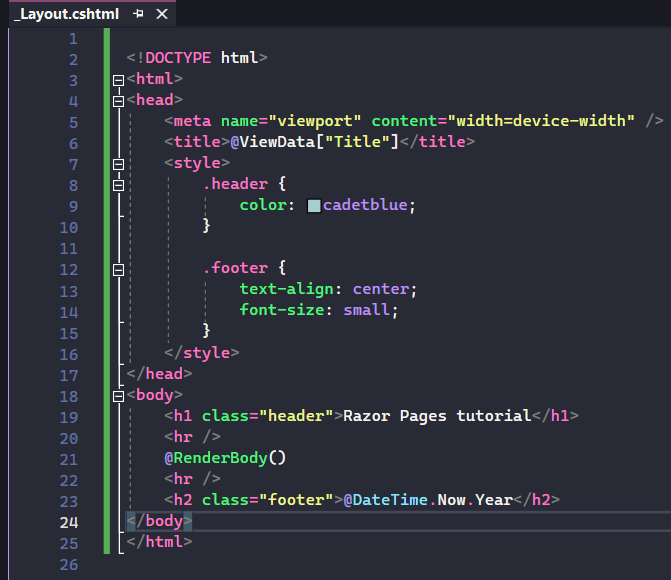
1. **Layout view Razor (Master Pages)**

* Thiết kế của hầu hết các website bao gồm một menu, một phần header, footer và phần thanh sidebar. Như bạn thấy đi từ một trang này đến trang kia trong website thì chỉ có nội dung ở giữa là thay đổi. Nhưng menu, header, footer hay màu sắc toàn cục vẫn giữ nguyên. Điều này giúp website có cái nhìn thống nhất.
* Tạo ra 2 razor page để thấy so sánh sự != . Nhận thấy hình dưới, hai page Index.cshtml và About.cshtml chỉ khác biệt về một phần nội dung hiển thị, những phần còn lại giống nhau. Tuy nhiên chúng ta đang phải lặp lại đoạn code ở 2 page. Từ đây, Razor cung cấp cơ chế \_Layout để tạo ra khung thống nhất mà tất cả các page của 1 site đều sử dụng để tránh việc lặp lại đoạn HTML,CSS,JavaScript
* Ví dụ :

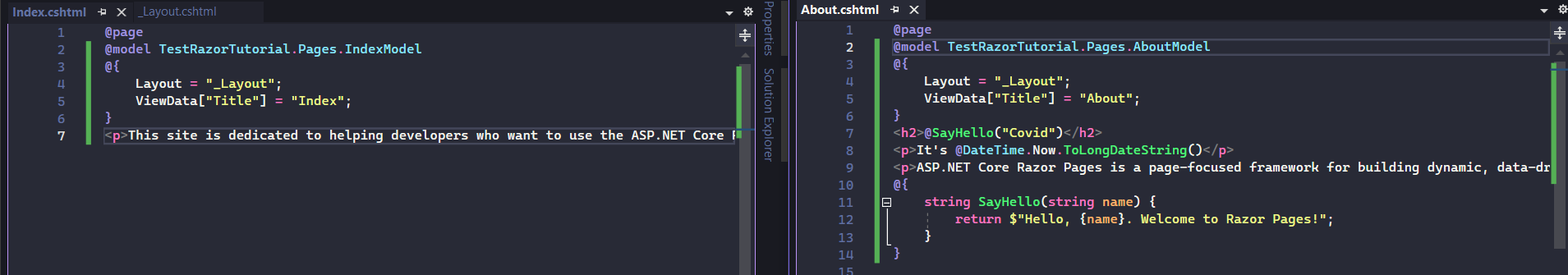
1. Tạo file \_Layout.cshtml trong thư mục Pages

* Dấu gạch chân trước tên gọi của file thường dùng cho 1 số loại file đặc biệt trong Razor Pages để phân biệt nó với trang nội dung khác.
* Các trang đặc biệt như \_Layout, \_ViewImports, \_ViewStart đều không chứa directive ở đầu.

1. Code cho file \_Layout.cshtml

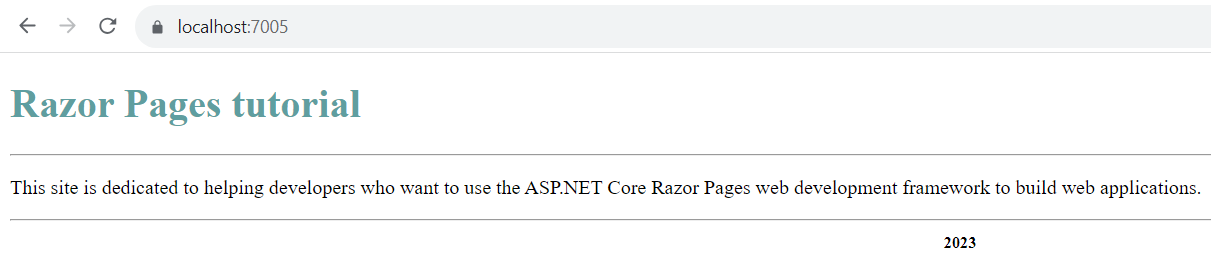
* Nội dung của \_Layout.cshtml thực chất chính là nội dung chung của Index.cshtml và About.cshtml. Có lời gọi phương thức @RenderBody().
* RenderBody() là phương thức đặc biệt trong Razor Pages có nhiệm vụ xuất ra trang nội dung ở vị trí tương ứng trong trang layout.
* Nói cách khác, nếu hiểu layout là một cái khung, các file còn lại chỉ cung cấp nội dung riêng. Thì nội dung riêng của từng file sẽ xuất hiện ở đúng vị trí gọi phương thức RenderBody().
* Nếu thiếu lời gọi RenderBody(), trang layout sẽ trở thành vô dụng.

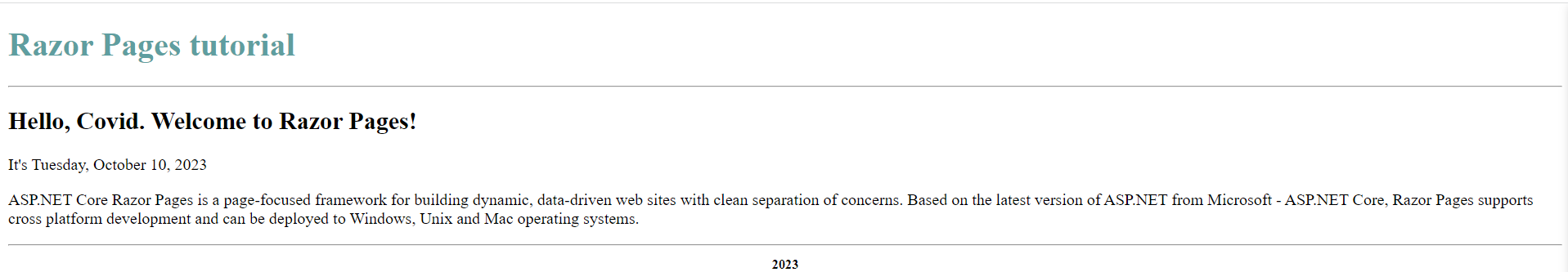
1. Thay đổi nội dung các page để sử dụng layout vừa tạo



* Nhận thấy sự != , mỗi page đều chỉ chứa nội dung riêng biệt của mình. Những phần chung đã được file \_Layout.cshtml sử dụng.
* Lệnh Layout :
* Để chỉ định file layout để hiển thị nội dung của page, gọi lệnh Layout = "\_Layout" trong khối code C#. Chỉ cần cung cấp tên file layout đặt trong dấu ngoặc kép
* Nếu page nào đó không muốn sử dụng layout thì chỉ cần viết Layout = null
* Nếu muốn sử dụng một layout khác, có thể thay thế tên layout vào
* ViewData object :
* ViewData là một object thuộc kiểu Dictionary được Razor Pages tạo sẵn và có thể truy cập từ bất kỳ đâu trong ứng dụng. Nhiệm vụ của nó là lưu trữ các cặp khóa / giá trị để sử dụng xuyên suốt trong tất cả các page của site.
* Trong trường hợp này, ViewData được dùng để trao đổi giá trị dòng title của từng page với layout: Giá trị dòng title được gán ở từng page và hiển thị trên layout.

1. Run và thử truy cập các URL



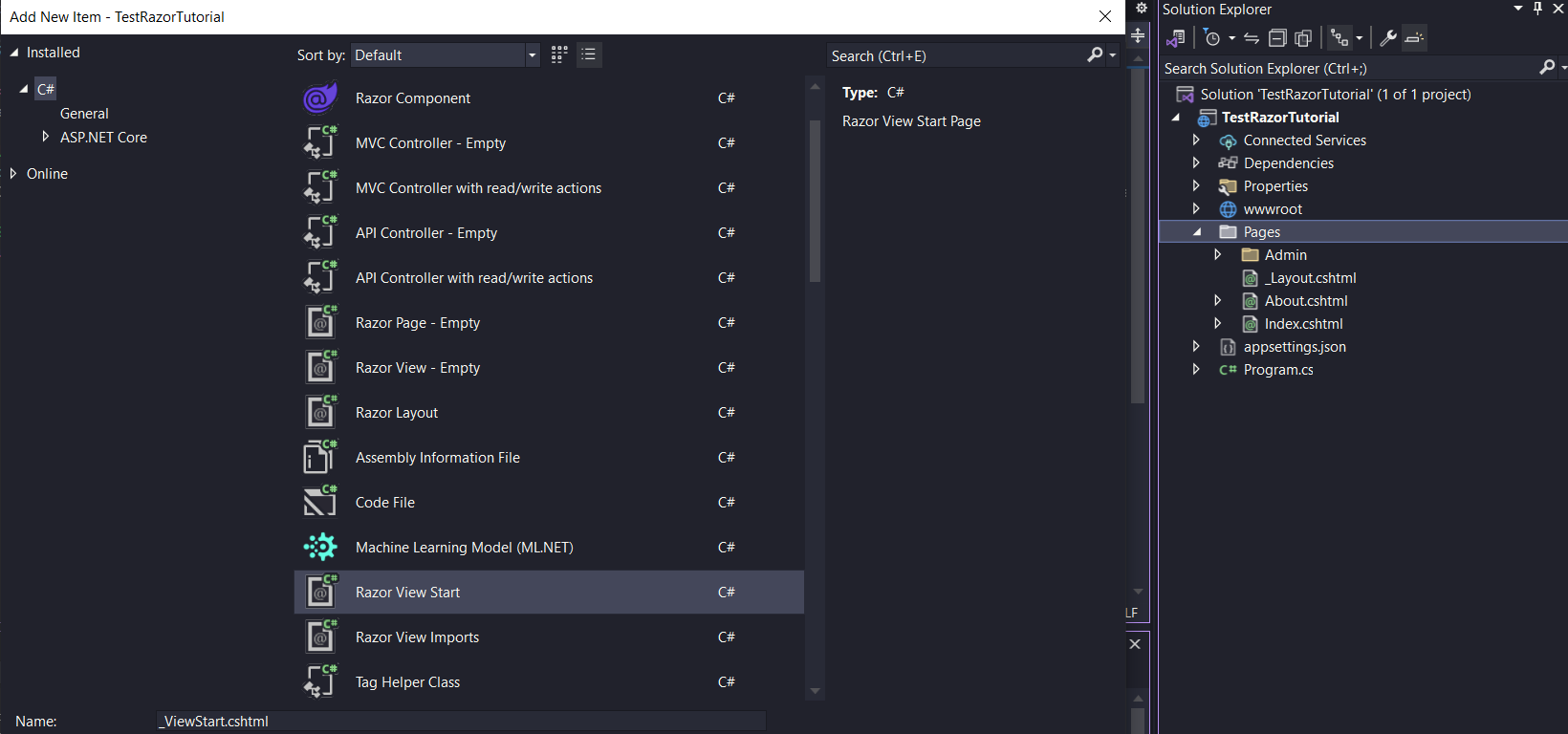


* Tất cả các page bây giờ đã có cùng layout

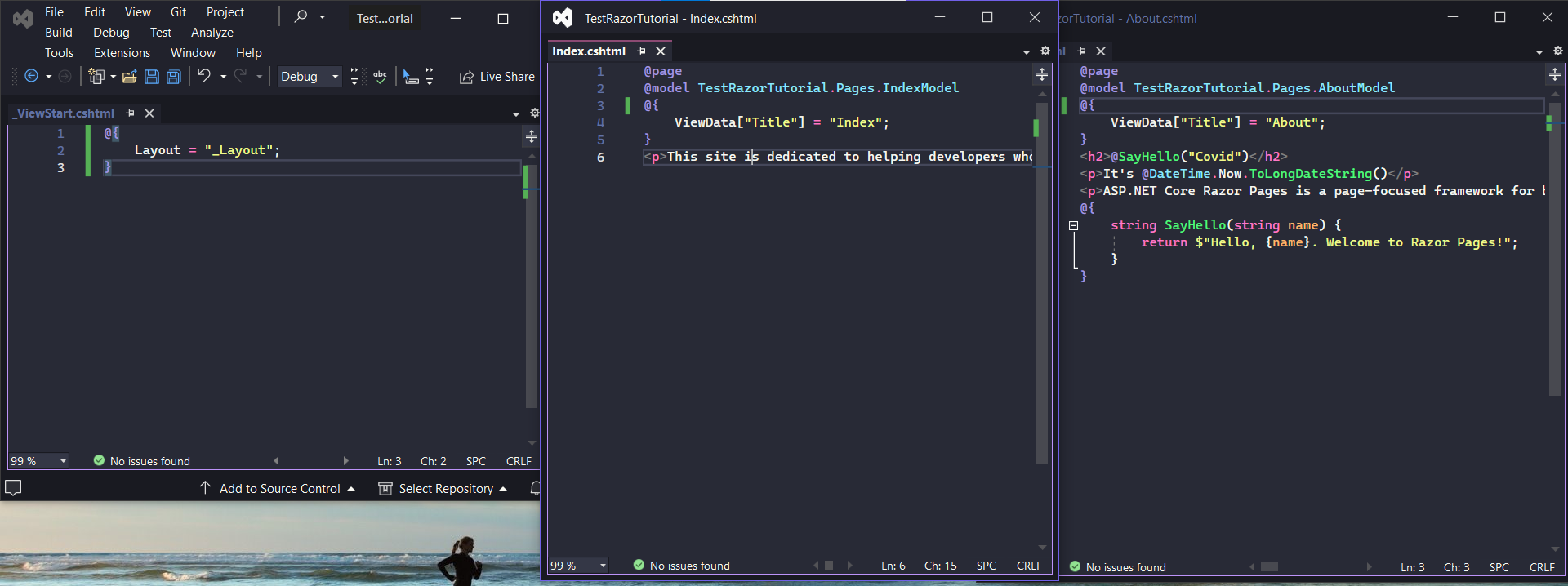
1. **Significance of \_ ViewStart.cshtml**

* Mỗi page đều phải chỉ định rõ file layout sử dụng và nếu có hàng trăm site và page và nếu chỉnh sửa thì mất thời gian. Trong tình huống này Razor Pages cung cấp công cụ : ViewStart (là 1 file đặc biệt chứa những đoạn Razor được thực hiện trước khi xử lý mỗi Page)
* Ví dụ :

1. Tạo file \_ViewStart.cshtml

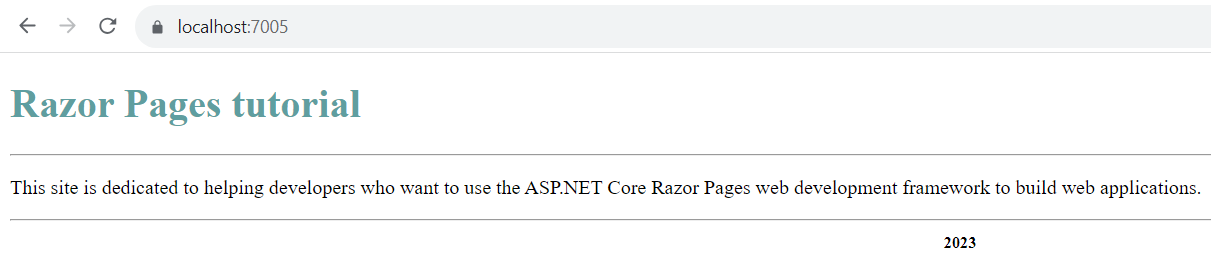


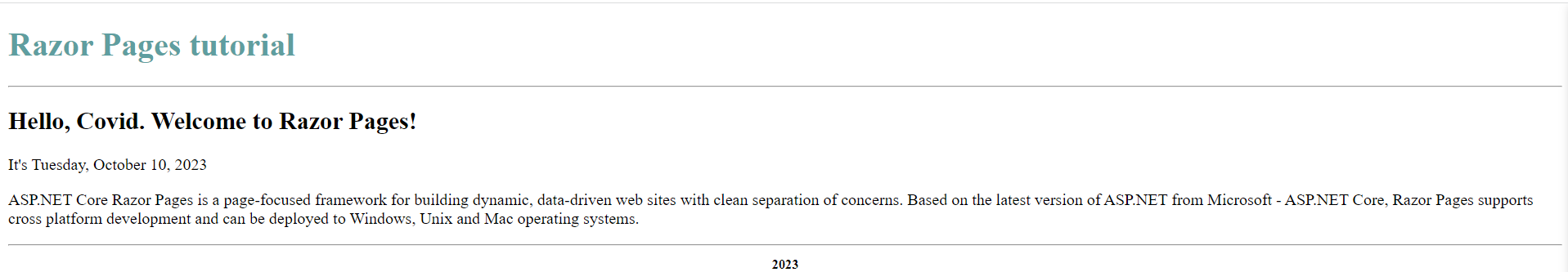
1. Viết code và xoá bỏ lệnh chỉ định sử dụng \_Layout trong tất cả các page

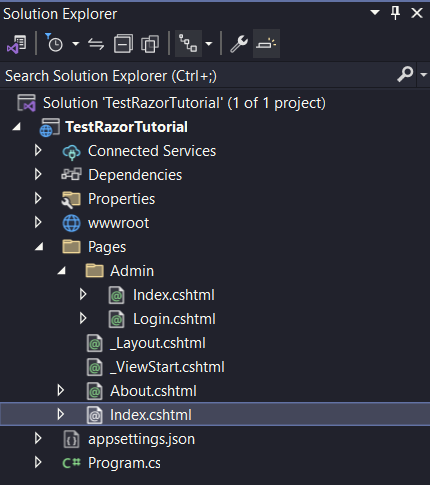


1. Chạy thử

Kết quả thu được như lần chạy \_Layout.cshtml

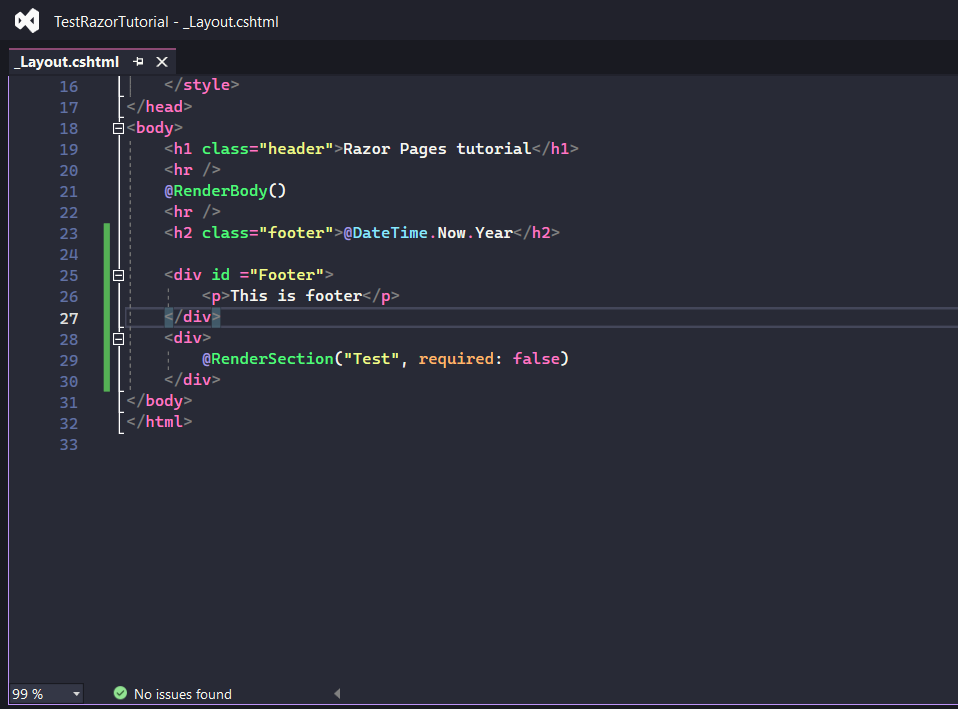




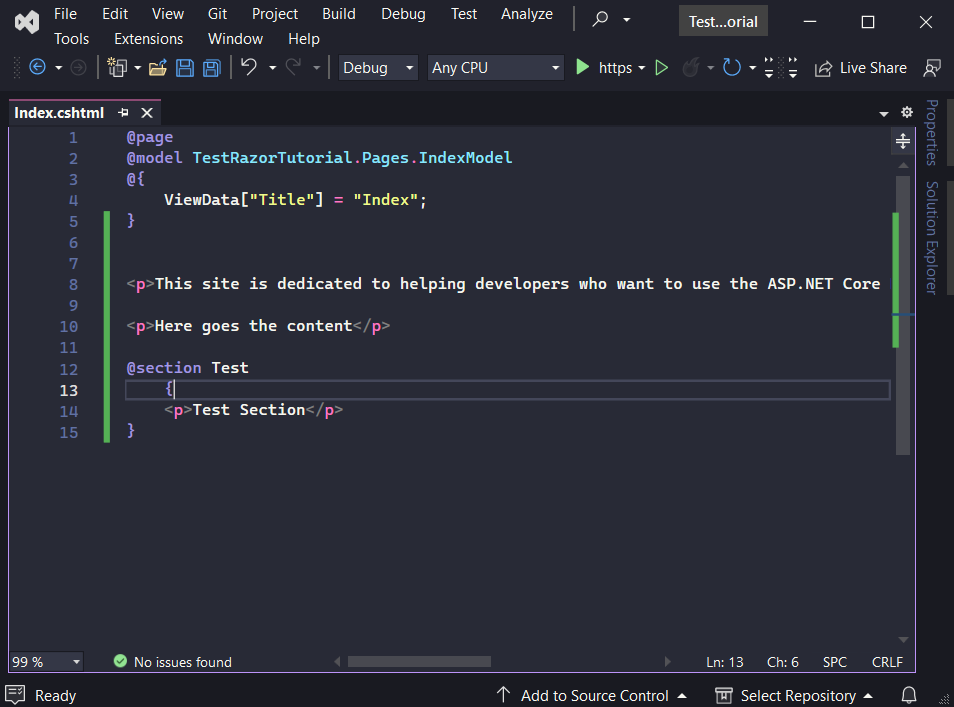


* \_ViewStart có tác dụng trong thư mục chứa nó và tất cả các thư mục con. Nếu để \_ViewStart trong Pages, nó có tác dụng trên toàn bộ site.
* Nếu đặt \_ViewStart trong một thư mục con của Pages, ví dụ trong thư mục Admin, \_ViewStart chỉ có tác dụng trong thư mục con này (và các thư mục con của thư mục đó). Khi này các page ở bên ngoài thư mục Admin sẽ không nhận layout tự động (do \_ViewStart này không có tác dụng bên ngoài Admin).
* \_ViewStart trong thư mục con sẽ có tác dụng đè lên \_ViewStart ở thư mục cha. Ví dụ, nếu thư mục Pages có thư mục con Admin và cả hai thư mục này đều có file \_ViewStart thì các page trong Admin (và các thư mục con của Admin) sẽ chịu tác dụng của \_ViewStart đặt trong Admin.

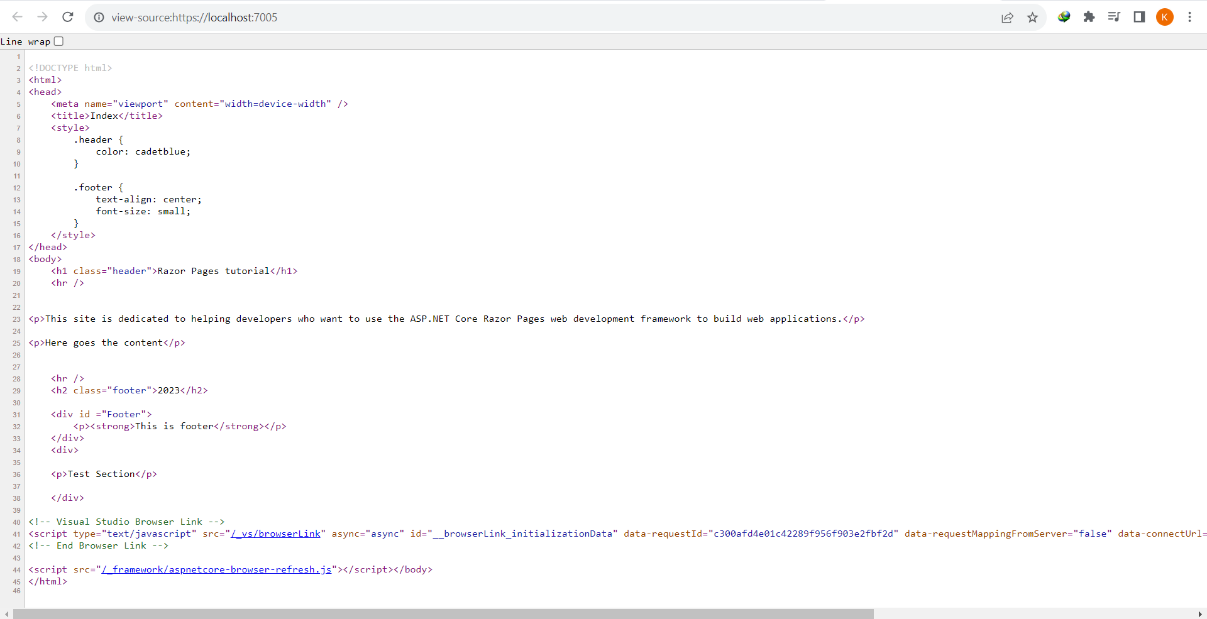
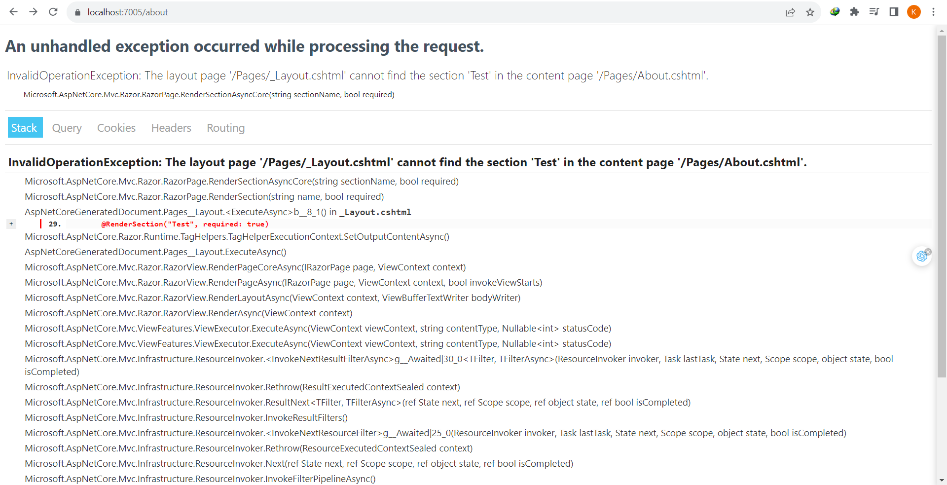
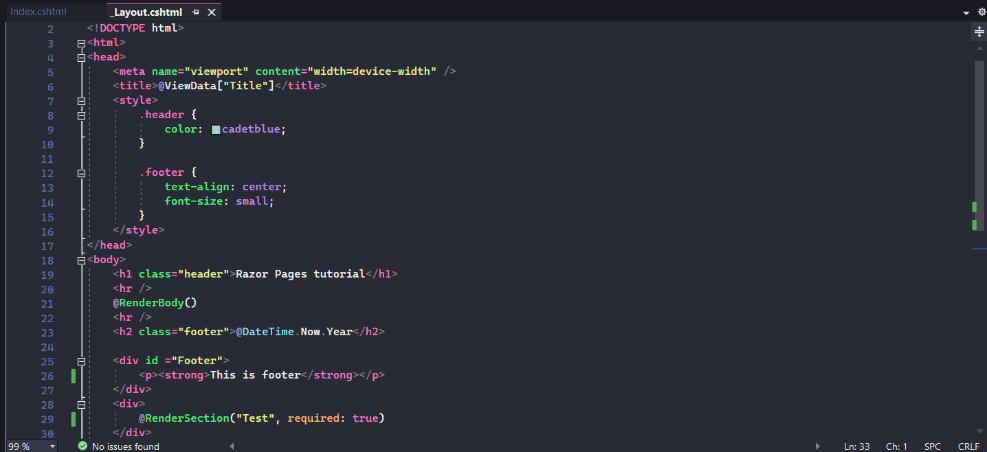
1. **Working with Sections**

* Khi sử dụng layout, dễ dàng đưa content của 1 page vào layout nơi đặt @RenderBody(). Nhưng yêu cầu phải render nội dung ở nhiều nơi khác nhau trong layout, chứ không phải chỉ render trang nội dung duy nhất ở trong layout nữa.
* 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 một cái tên và một thuộc tính chỉ ra xem nó có bắt buộc phải có trong các view con không. Phương thức **RenderSection(string name, bool required:false)** định nghĩa tên **section**.
* Ví dụ :
* Thêm section vào layout

Chúng ta định nghĩa một section tên **"Test"** với tham số required là false tức là không yêu cầu bắt buộc phải có section này trong view con.

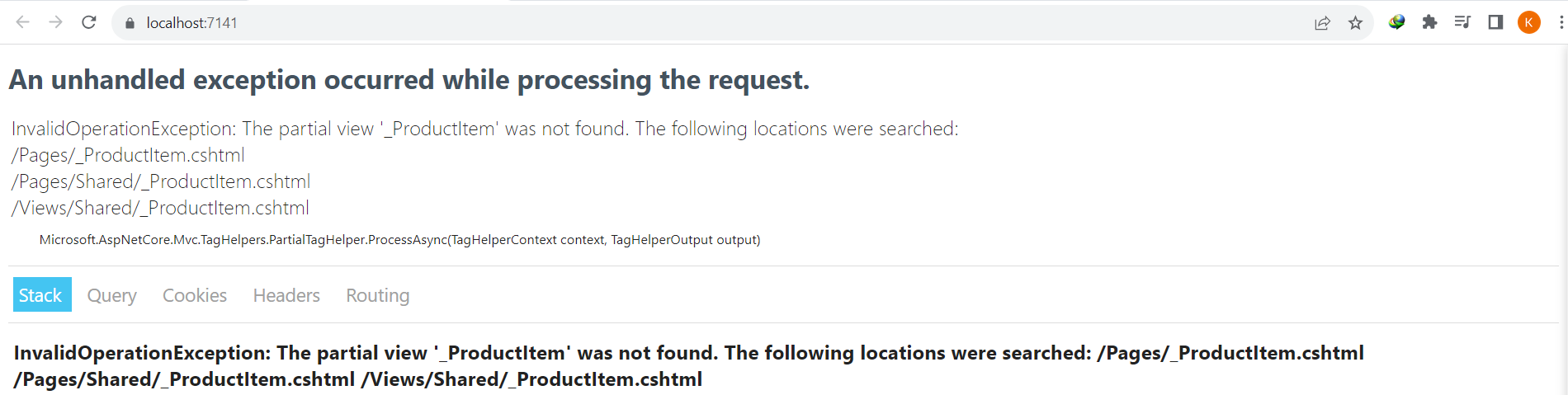
* ****Định nghĩa section vào view

Các section được định nghĩa với một khối lệnh Razor là **@section** bằng tên nó đi kèm. Chạy ứng dụng và kiểm tra xem các section đó có hiển thị sau content.

* Chạy thử
* Nếu tạo section với required = true và vào đường dẫn con /about thì sao ?
* Giờ hãy chạy ứng dụng và vào URL /about sau đó bạn nhìn thấy lỗi sau:
* InvalidOperationException: The layout page '/Pages/\_Layout.cshtml' cannot find the section 'Test' in the content page '/Pages/About.cshtml'.
* Điều này xảy ra vì chúng ta không định nghĩa section **"Test"** trong view about

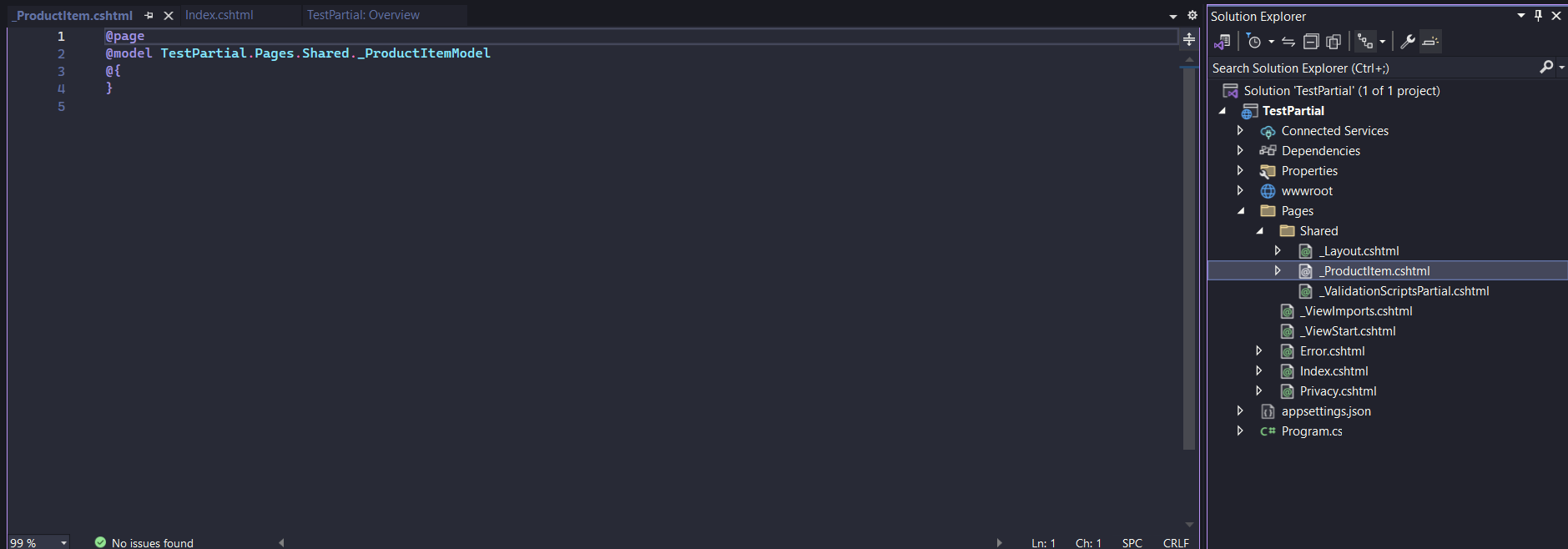
1. **Working with Partial Views**

* Trong các View khái niệm Partial Page hoặc Partial View được sử dụng khi chia nhỏ các trang Razor Pages hay các View ra thành từng phần nhỏ
* Lợi ích của việc chia nhỏ là :
* Một View (Razor Page) phức tạp, khó bảo trì được chia thành nhiều phần nhỏ, dễ kiểm soát
* Sử dụng lại được những phần chung cho các trang
* Partial Page chính là các file .cshtml - chỉ có có điều loại file này không được routing tới (không được gọi trực tiếp), nên tên của nó sẽ bắt đầu bằng ký tự \_
* Cách 1 : Sử dụng TagHelper <partial/> chèn vào partial view name.Nếu không tìm thấy trên hệ thống sẽ bị báo lỗi

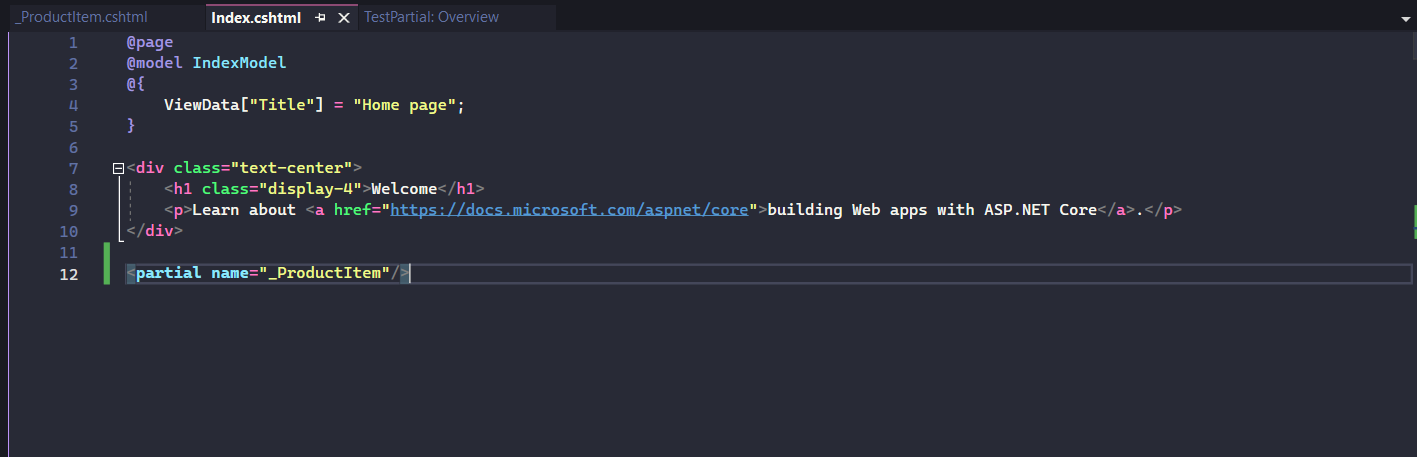


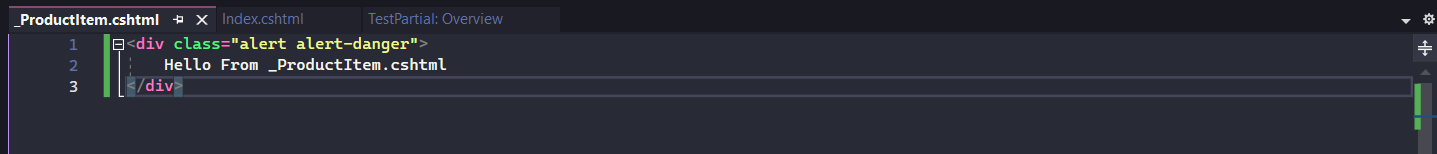
* Ví dụ trên báo lỗi hệ thống không tìm thấy partial nào tên “\_ProductItem”.
* Sửa lỗi trên:

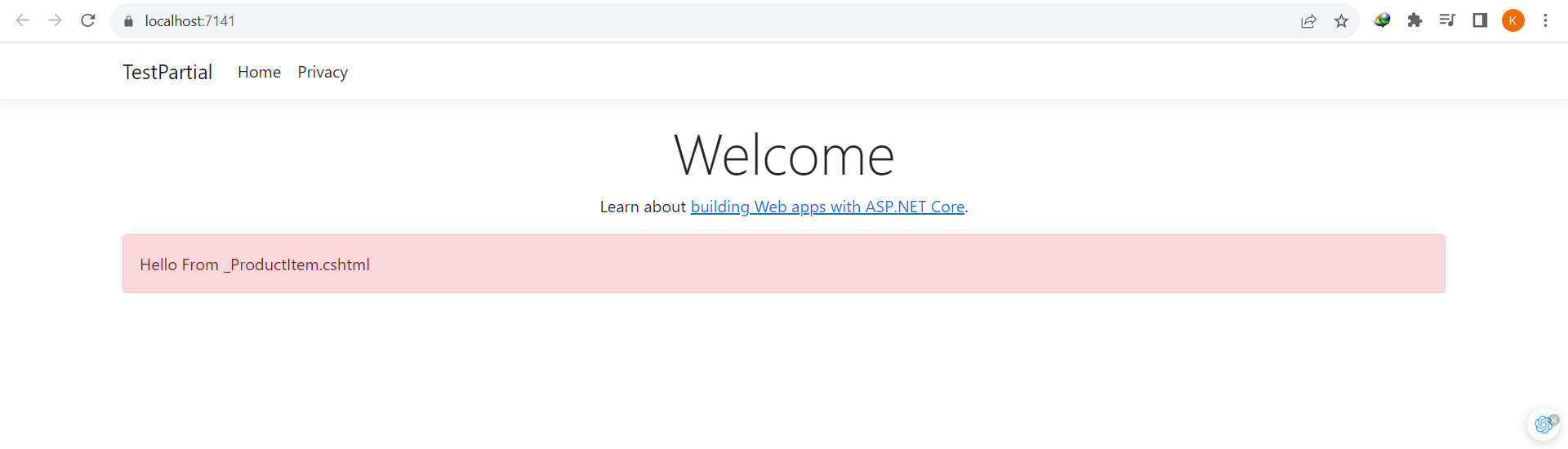
1. Tạo file \_ProductItem.cshtml, trong đường dẫn Pages/Shared/\_ProductItem.cshtml



1. Viết code và chạy thử

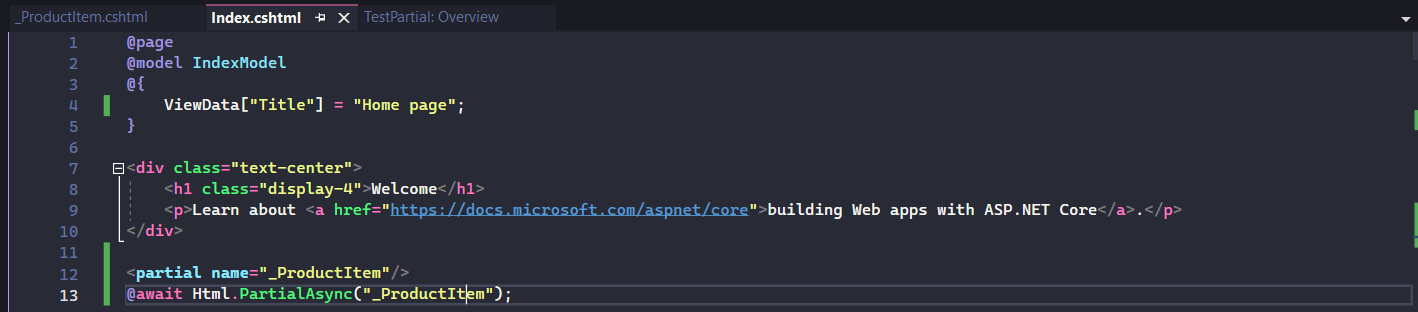
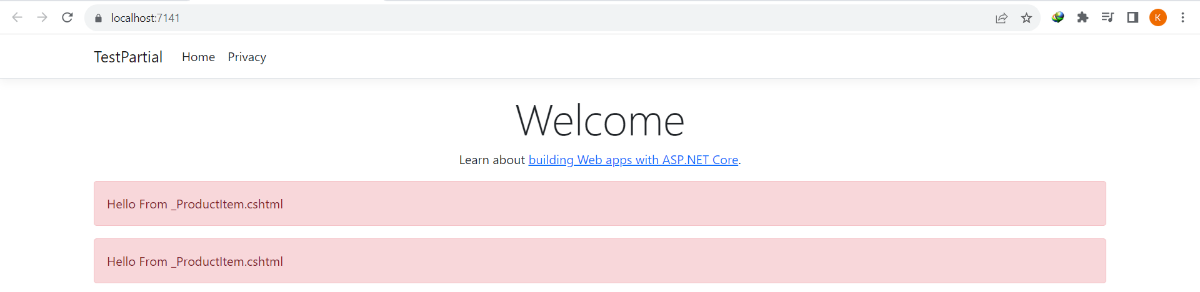






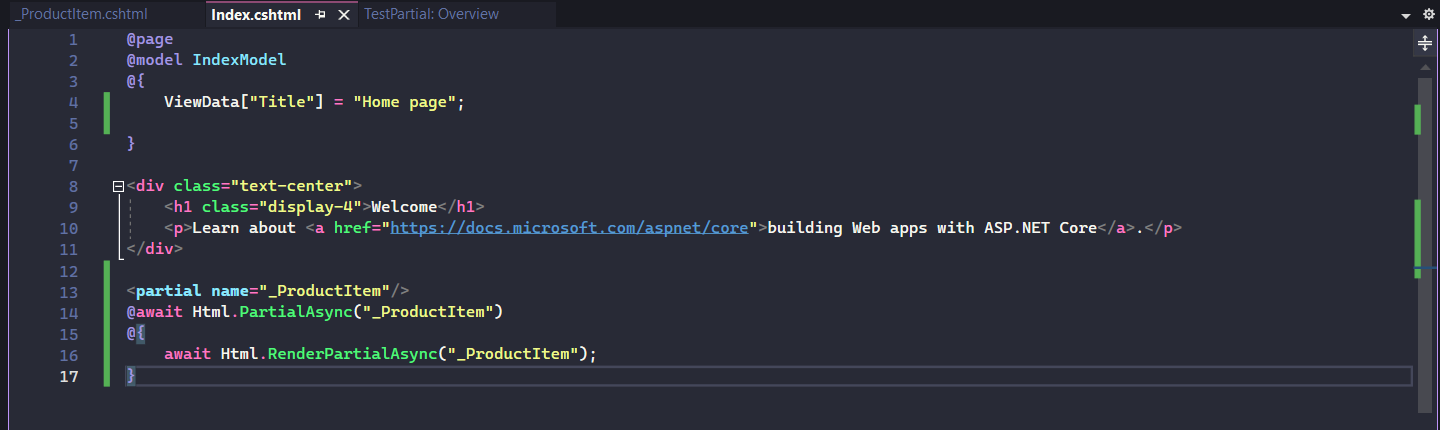
* Cách 2 : Để chèn partial view, sử dụng HTMLHelper Html.PartialAsync(“\_”)

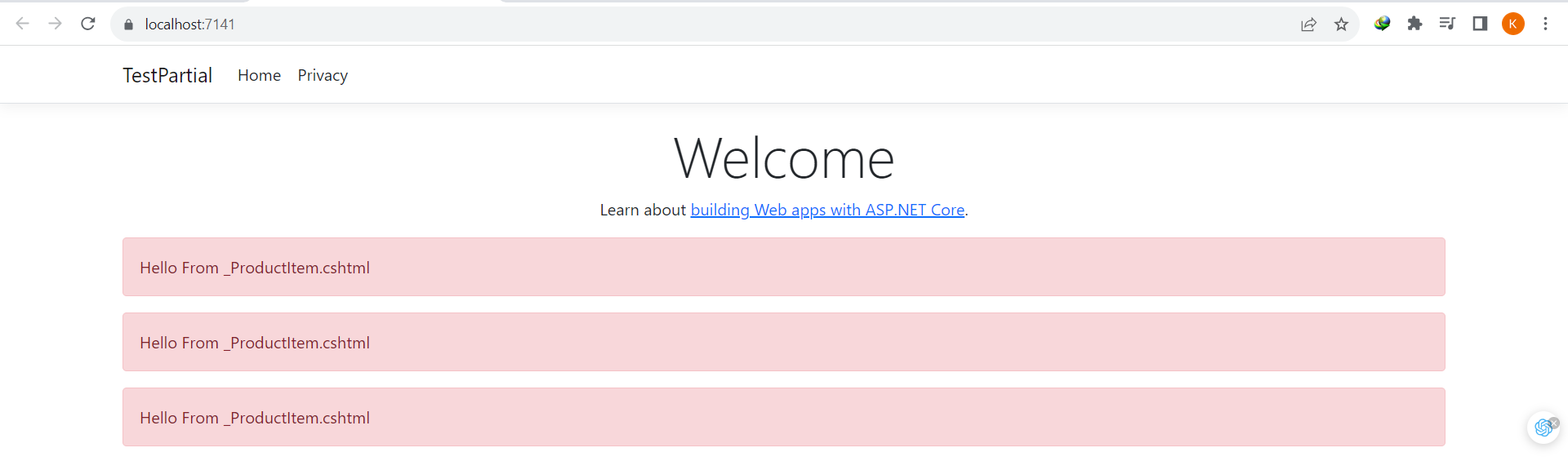
1. Viết code và chạy thử



* Cách 3 : Để chèn partial view, sử dụng HTMLHelper Html.RenderPartial(“\_”)

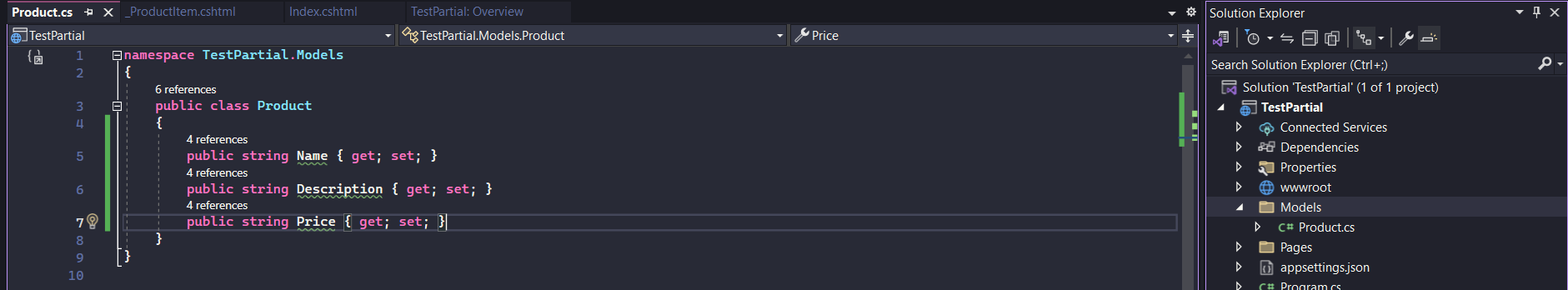
1. Viết code và chạy thử



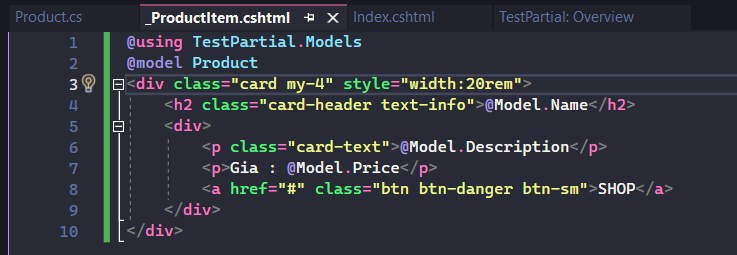


* Lưu ý sự khác biệt giữa Partial và RenderPartial là ở chỗ:
* Partial trả về chuỗi HTML, do đó nó được gọi như một biểu thức Razor (Razor Expressions) @Html.Partial("\_Header");
* RenderPartial có kiểu trả về là void nên nó phải được gọi trong khối code Razor (Razor code block) @{ Html.RenderPartial("\_Header"); }.
* Partial có phương án bất đồng bộ tương ứng là PartialAsync. RenderPartial có phương án bất đồng bộ tương ứng là RenderPartialAsync.
* Trong tất cả các phương án trên, RenderPartialAsync và RenderPartial viết trực tiếp kết quả vào chuỗi phản hồi. Do vậy nó có hiệu suất cao nhất. Tuy nhiên sự khác biệt về hiệu suất thực tế là không đáng kể.
* Partial Pages và dữ liệu :
* Partial page cũng có thể có model class của riêng nó.
* Dùng directive @model để chỉ định model class đứng sau partial page.
* Loại partial page có model class cũng được gọi là strongly typed partial page.
* Model class cung cấp dữ liệu và khả năng xử lý logic cho partial page.
* Ví dụ :

1. Tạo folder Models có đối tượng Product.cs



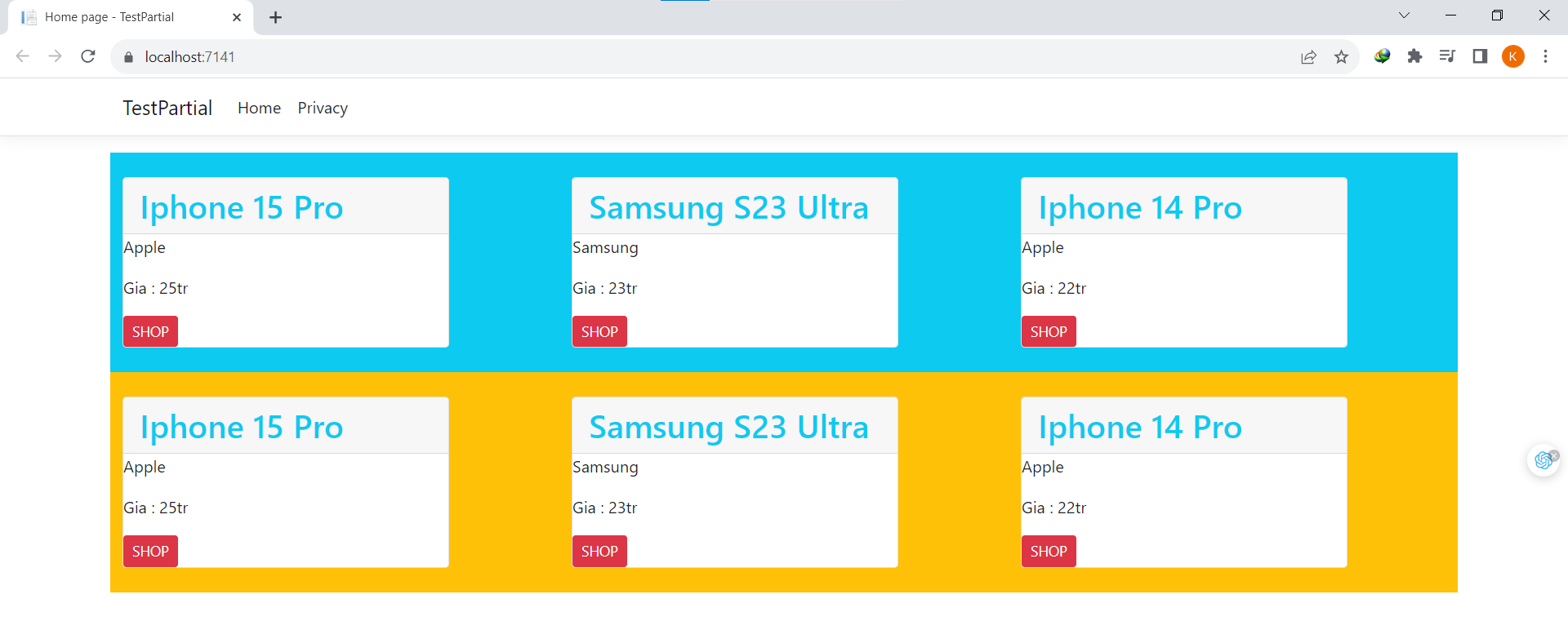
1. Code \_ProductItem.cshtml



1. Code Index.cshtml

* Đây là chỗ bạn tạo dữ liệu, sử dụng trang partial \_ProductItem và cung cấp dữ liệu cho nó.
* Biến data là một danh sách các đối tượng. Sử dụng lối gọi trang partial thông qua markup <partial name="…" />
* Để ý rằng dữ liệu cung cấp cho trang partial thông qua attribute model="…". Giá trị của model có thể là một biểu thức.

1. Chạy thử



1. **Boostrap support for editor**

* Bootstrap là một framework phổ biến trong phát triển web, được sử dụng để tạo giao diện trang web hiện đại và đáp ứng.
* Khi bạn muốn sử dụng Bootstrap để thiết kế giao diện trang web trong Razor Views hoặc Pages, bạn có thể sử dụng Bootstrap Support Editor để giúp tạo và chỉnh sửa mã HTML dễ dàng hơn.
* Bootstrap Support Editor là một công cụ hoặc extension mà bạn có thể thêm vào môi trường phát triển của mình (chẳng hạn như Visual Studio hoặc Visual Studio Code) để hỗ trợ việc tạo và chỉnh sửa mã Bootstrap trong các trang Razor