Mô hình code-behind trong Blazor

Code-behind là mô hình tổ chức code quen thuộc trong Windows Forms và Windows Presentation Foundation. Trong Razor Pages mô hình này cũng được sử dụng mặc định thông qua Model class.

Mặc dù có nhiều điểm tương đồng với các công nghệ trên, template mặc định cho component trong Blazor lại sử dụng mô hình single file – tập trung toàn bộ code vào file razor.

Bài học này sẽ hướng dẫn bạn tổ chức code của Blazor theo mô hình code behind quen thuộc.

NỘI DUNG CỦA BÀI [Ẩn]

1. Mô hình tổ chức code mặc định của Blazor
2. Mô hình code-behind và Blazor
3. Code behind trong Blazor sử dụng partial class
4. Code-behind trong Blazor sử dụng kế thừa
5. Cách xử lý component trong Blazor
6. Kết luận

Mô hình tổ chức code mặc định của Blazor

Template mặc định cho các component của Blazor sử dụng mô hình single file. Trong mô hình này tất cả code dồn vào một file razor duy nhất.

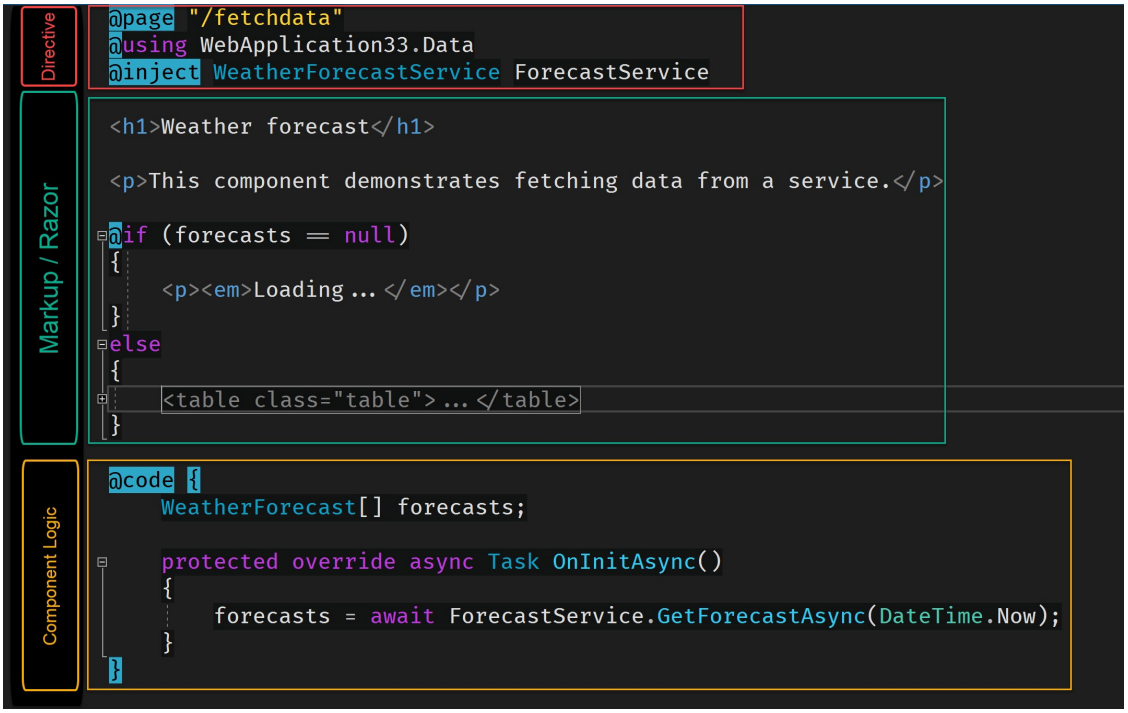
Mỗi file razor thường được chia làm 3 phần: Directive, Razor markup và Component logic.

Directive là những "lệnh" Razor đặc biệt như chỉ định route (@page), chỉ định namespace (@using), chèn object (@inject), v.v..

Razor markup thường chiếm phần lớn không gian của file razor. Đây là phần sinh ra giao diện của component. Khối này được viết bằng thứ "ngôn ngữ" trộn lẫn của C# và HTML gọi là cú pháp Razor.

Component logic nằm trong khối @code { } có nhiệm vụ xử lý logic và được viết hoàn toàn bằng C#.

Dưới đây là ví dụ về component FetchData.razor được tạo ra trong các project mẫu của Blazor Server App.



Mô hình tổ chức code này có một số vấn đề:

- Khó quản lý do trộn lẫn lộn code dành cho UI và code dành cho logic;
- Khó đọc do trong file sử dụng nhiều ngôn ngữ cùng lúc, đặc biệt khi khối lượng code lớn;
- Khó test logic do trộn lẫn với UI;
- Hỗ trợ của Visual Studio bị hạn chế: code C# trong file razor không được hỗ trợ tốt như trong file cs. Bạn có thể dễ dàng cảm nhận điều này khi viết code C# trong file razor.

Nếu code ngắn gọn, mô hình này có thể tiện lợi. Khi code dài thêm, vấn đề sẽ trở nên nghiêm trọng hơn.

Khi xây dựng các ứng dụng với khối lượng code lớn bạn không nên sử dụng mô hình tổ chức code này.

Mô hình code-behind và Blazor

Nếu bạn đã từng làm việc với Windows Forms hay Windows Presentation Foundation, bạn hẳn đã quen thuộc với mô hình code-behind. Khi học Razor Pages, bạn được khuyến khích sử dụng Model class cho content page. Model class cũng có thể xem là một dạng code-behind.

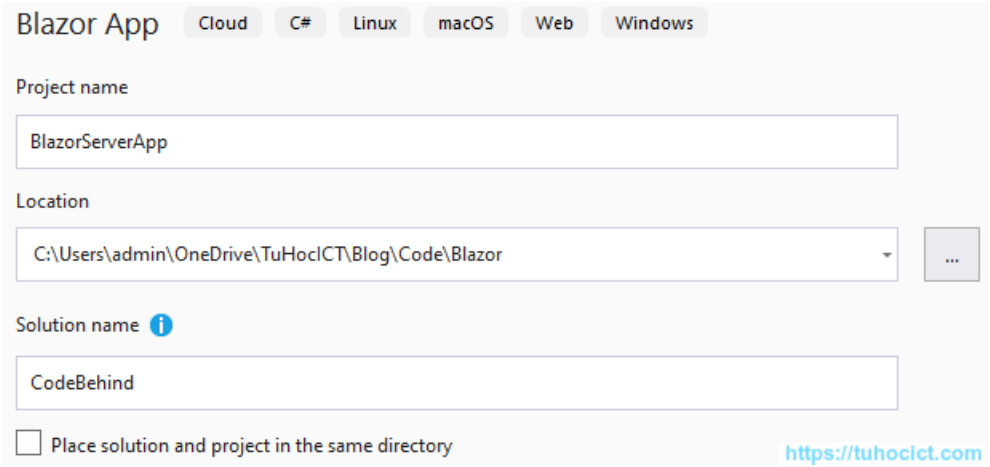
Trong mô hình này, code tạo giao diện và code xử lý logic được đặt vào các file khác nhau nhưng bạn có thể dễ dàng sử dụng chung.

Mô hình code-behind đặc biệt quan trọng trong các công nghệ trong đó logic và giao diện được tạo ra bằng các ngôn ngữ khác nhau. Ví dụ, trong Windows Presentation Foundation, giao diện được xây dựng bằng ngôn ngữ XAML, trong khi logic được xử lý bằng C#. Trong Razor Pages, logic xử lý bằng C# còn giao diện được tạo ra bằng HTML/Razor.

Trong Blazor bạn thực tế có thể rất dễ dàng vận dụng mô hình code-behind. Bạn có thể chuyển các component đã có từ trước sang mô hình code-behind. Hoặc bạn cũng có thể áp dụng ngay mô hình khi xây dựng một component mới.

Có hai cách khác nhau: (1) Sử dụng kế thừa; (2) Sử dụng partial class.

Để dễ dàng hình dung chúng ta sẽ cùng thực hiện các ví dụ. Để chuẩn bị hãy tạo một project Blazor Server App:

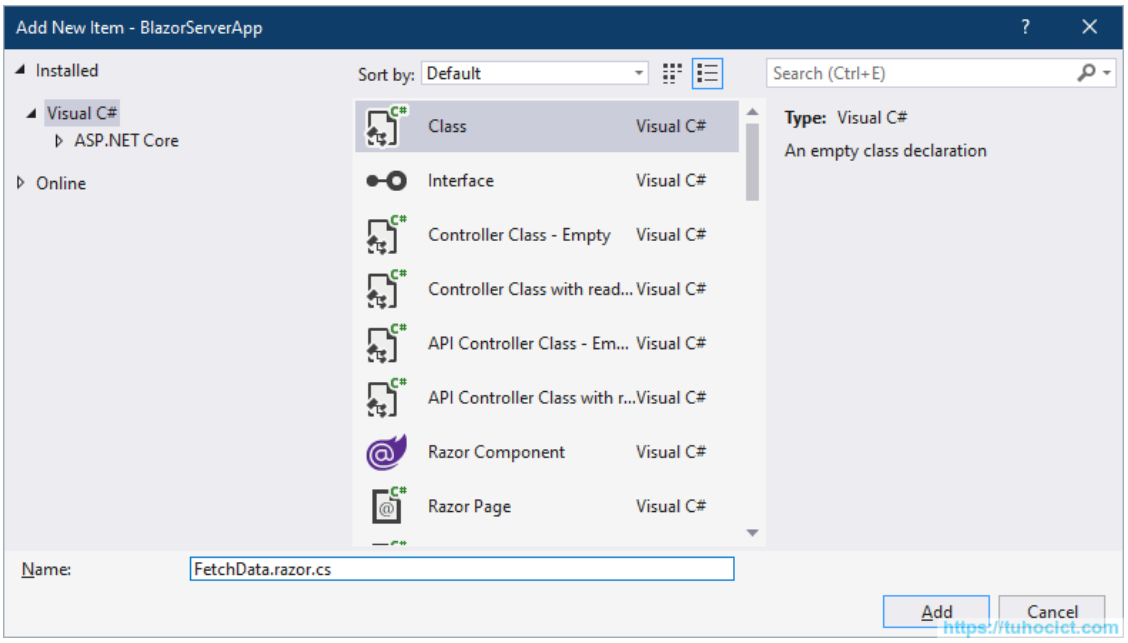


Code behind trong Blazor sử dụng partial class

Đây là mô hình tổ chức tương tự như trong Windows Forms.

Chúng ta sẽ chuyển component FetchData.razor sẵn có (trong thư mục Pages) sang mô hình code-behind sử dụng partial class.

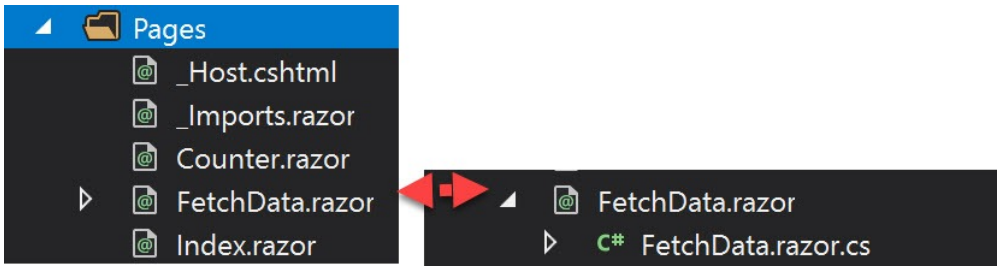
1. Tạo file code-behind FetchData.razor.cs



Lưu ý: File code behind phải nằm trong cùng thư mục với FetchData.razor.

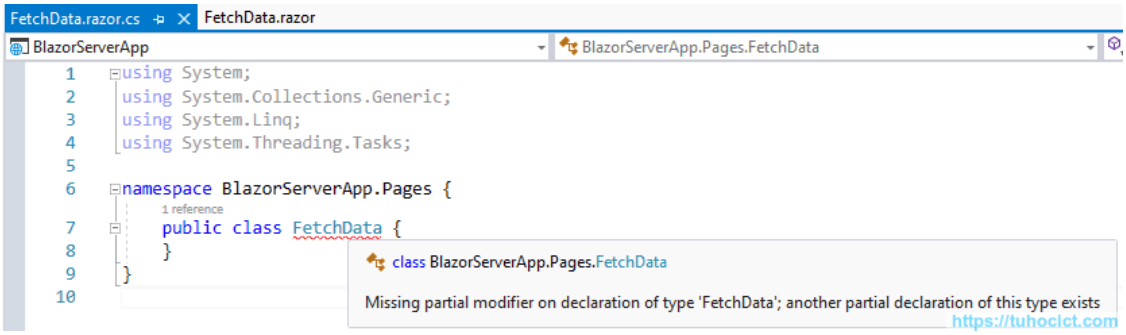
Tên file FetchData.razor.cs được đặt theo mẫu `<tên component>.razor.cs`. Đây không phải là một quy tắc. Bạn có thể đặt tên file code-behind tùy ý.

Tuy nhiên nếu đặt tên file theo mẫu trên, cơ chế file nesting của Visual Studio sẽ hiển thị file code-behind vào cùng một node với file razor. Bạn sẽ dễ quản lý file code hơn.

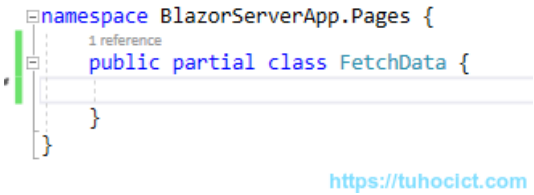


2. Chuyển FetchData thành partial class

Khi mở file FetchData.razor.cs bạn sẽ thấy C# đang báo lỗi ở class FetchData vừa tạo.



Chuyển FetchData thành partial class thì lỗi sẽ biến mất.



Chúng ta sẽ nói kỹ hơn về vấn đề này ở cuối bài.

3. Di chuyển code sang FetchData.razor.cs

Cut-paste toàn bộ code từ khối @code{ } sang class FetchData:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace BlazorServerApp.Pages {
    1 reference
    public partial class FetchData {
        private WeatherForecast[] forecasts;

        0 references
        protected override async Task OnInitializedAsync() {
            forecasts = await ForecastService.GetForecastAsync(DateTime.Now);
        }
    }
}
```


Một số chỗ vẫn báo lỗi cú pháp là do file code-behind chưa tham chiếu tới không gian tên BlazorServerApp.Data.

Cut-paste dòng `@using BlazorServerApp.Data` từ file razor và chuyển thanh `using BlazorServerApp.Data;` trong file code-behind.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using BlazorServerApp.Data;

namespace BlazorServerApp.Pages {
    1 reference
    public partial class FetchData {
        private WeatherForecast[] forecasts;

        0 references
        protected override async Task OnInitializedAsync() {
            forecasts = await ForecastService.GetForecastAsync(DateTime.Now);
        }
    }
}
```


4. Sử dụng Dependency Injection

Trong file FetchData.razor sử dụng Dependency Injection qua directive:

```
@inject WeatherForecastService ForecastService
```

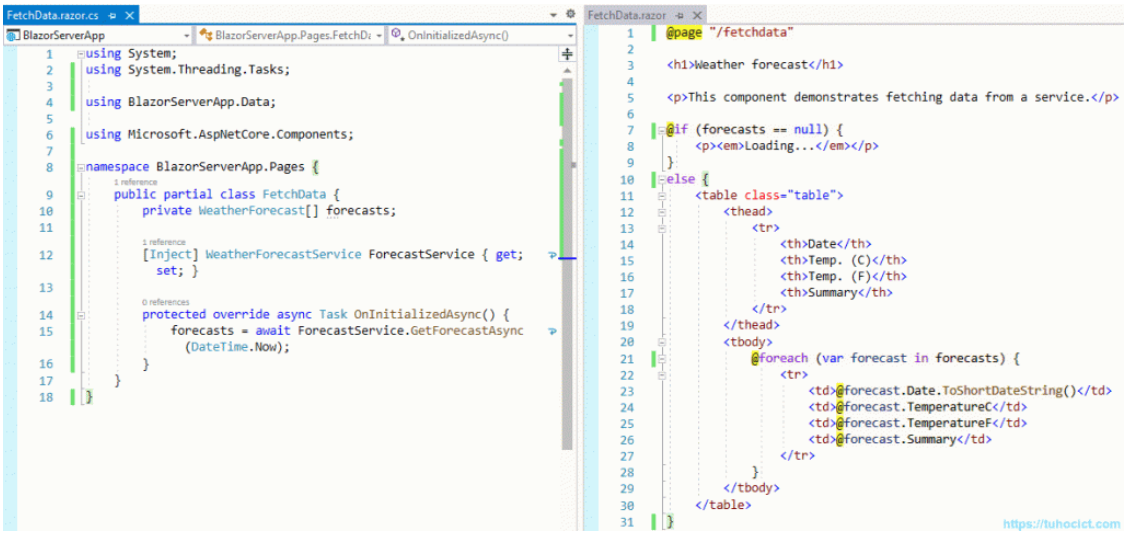
Hãy xóa bỏ dòng directive trên và thêm khai báo property sau vào code-behind:

```
[Inject] public WeatherForecastService ForecastService { get; set; }
```

Lưu ý cần thêm `using Microsoft.AspNetCore.Components;` vào đầu file code – behind.

Ở đây chúng ta sử dụng dependency injection qua property. Bạn không thể sử dụng DI qua constructor được vì Blazor bắt buộc các component phải sử dụng constructor không tham số.

Đến đây bạn đã hoàn thành chuyển đổi FetchData component sang mô hình sử dụng code-behind:



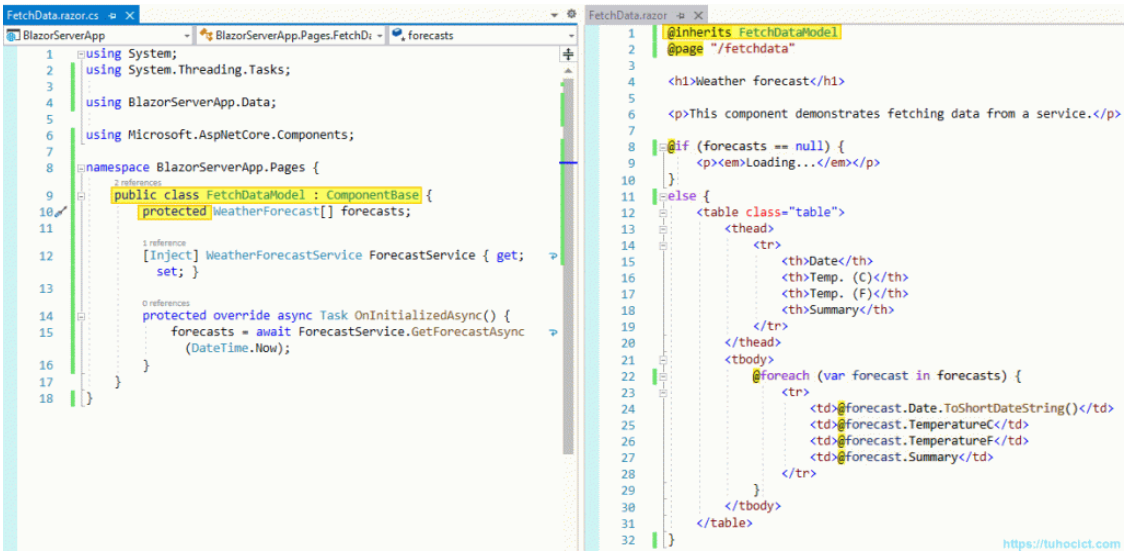
Giờ thì code C# nằm toàn bộ trong file cs, code Razor nằm trong file razor.

Thêm vào đó, trong file cs bạn có thể sử dụng tất cả những công cụ hỗ trợ C# của Visual Studio. Trong file razor, một số công cụ không sử dụng được (ví dụ Quick Action).

Code-behind trong Blazor sử dụng kế thừa

Giờ chúng ta sẽ chuyển code-behind từ partial class sang kế thừa như sau:

1. Đổi tên class FetchData thành FetchDataMode (hoặc bất kỳ tên nào)l;
2. Bỏ từ khóa partial;
3. Cho FetchDataModel kế thừa ComponentBase;
4. Thêm directive @inherits FetchDataModel vào đầu FetchData.razor.



Bạn có thể thấy, code-behind sử dụng kế thừa về cơ bản là tương tự với code-behind sử dụng partial class. Tuy nhiên có những điểm khác biệt đang lưu ý:

- Tên class không thể đặt trùng với tên component. Đối với component FetchData, bạn không thể dùng tên FetchData cho code behind mà phải chọn một tên khác.
- Code-behind classs phải kế thừa từ ComponentBase.
- Bất kỳ thành viên nào cần truy xuất từ file razor thì phải đặt độ truy cập là protected hoặc public.
- File razor sử dụng directive @inherits để chỉ định code-behind class.

Cách xử lý component trong Blazor

Bạn có thể thắc mắc tại sao chúng ta lại có thể tách rời code sử dụng một trong hai cách như trên. Điều này có liên quan đến cách thức xử lý component trong Blazor.

Khi bạn tạo một component trong file razor, Blazor sẽ tự động ra một class tương ứng trong thư mục `obj\Debug\netstandard2.1\Razor\Pages`.

Ví dụ, với FetchData component (trong file FetchData.razor), Blazor sẽ sinh ra file FetchData.razor.g.cs. Trong file này khai báo class:

```
public partial class FetchData : Microsoft.AspNetCore.Components.ComponentBase {
... }
```

Toàn bộ mã Razor sẽ chuyển thành code C# trong phương thức BuildRenderTree (ghi đè phương thức cha trong ComponentBase).

Tất cả code C# trong khối @code{ … } được chuyển nguyên vẹn sang thân class này.

Như vậy, khi xây dựng một component (razor), thực tế là bạn đang xây dựng một class cùng tên.

Do class này được khai báo là partial và kế thừa từ ComponentBase, bạn có thể lợi dụng để tổ chức code-behind như chúng ta đã thực hiện ở trên.

Khi này code xử lý, thay vì để Blazor tự chuyển sang class sinh tự động, bạn chủ động chuyển nó sang một file code khác.

> + Nếu bạn thấy site hữu ích, trước khi rời đi hãy **giúp đỡ** site bằng một hành động nhỏ để site có thể phát triển và phục vụ bạn tốt hơn.
> + Nếu bạn thấy bài viết hữu ích, hãy giúp **chia sẻ** tới mọi người.
> + Nếu có thắc mắc hoặc cần trao đổi thêm, mời bạn viết trong phần **thảo luận** cuối trang.
> Cảm ơn bạn!

Kết luận

Bài viết đưa ra hai phương pháp khác nhau để áp dụng mô hình code-behind cho Blazor component, bao gồm sử dụng partial class hoặc kế thừa ComponentBase.

Nhìn chung cả hai phương pháp này là tương tự nhau. Bạn có thể áp dụng phương pháp nào cũng được.

Trong đó phương pháp kế thừa có thêm khả năng tách rời hoàn toàn code-behind class sang một project riêng. Bạn có sự tự do tương đối hơn khi sử dụng phương pháp kế thừa.