Xây dựng ứng dụng Blazor server từ A-Z (+video)

Blazor là một thành viên trong gia đình ASP.NET Core và là công nghệ phát triển ứng dụng web client mới nhất của Microsoft sử dụng C# thay cho Javascript. Đây cũng là công nghệ được Microsoft đầu tư rất mạnh trong thời gian qua. Blazor hứa hẹn sẽ là một framework chủ đạo để xây dựng web client của ASP.NET Core trong tương lai.

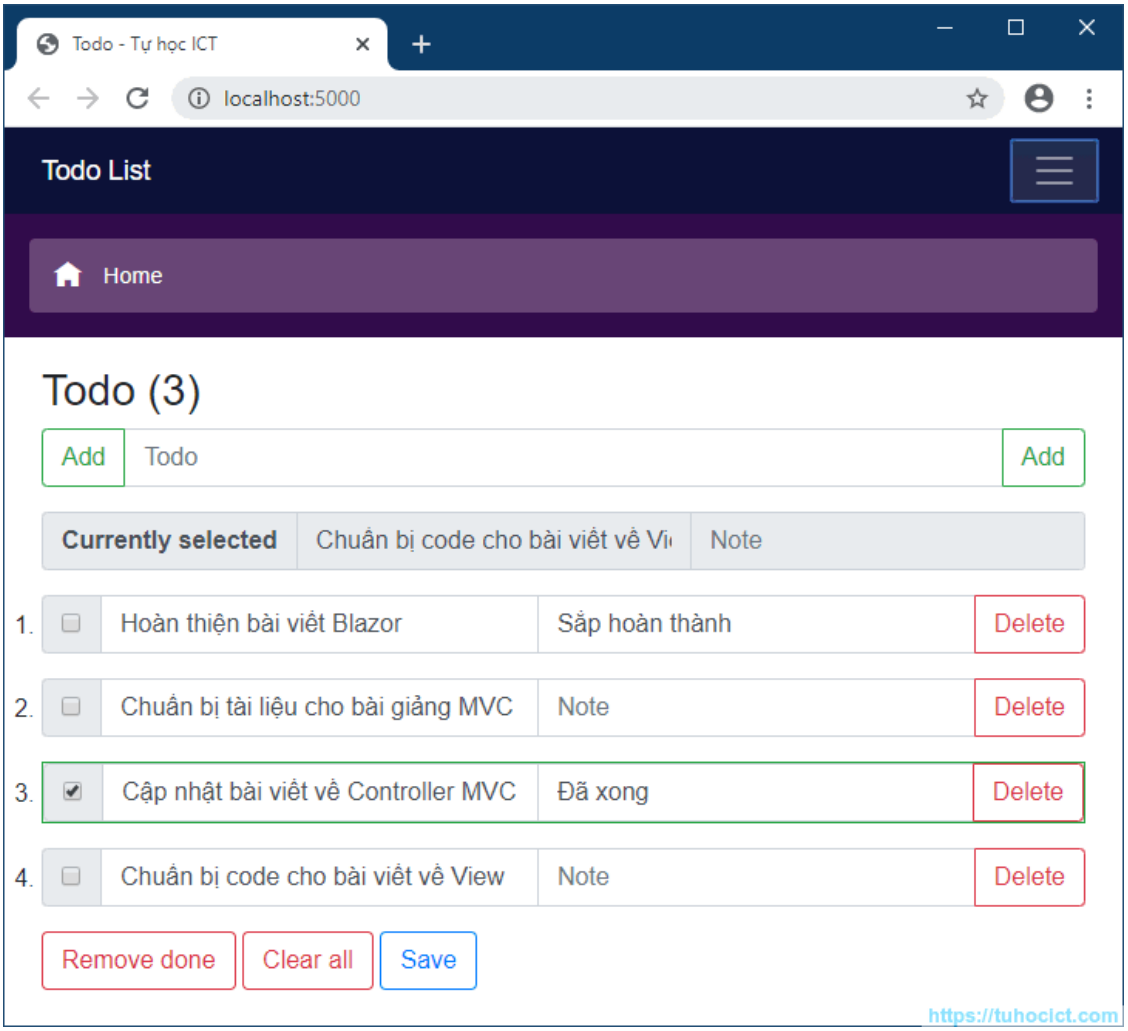
> Video hướng dẫn và mã nguồn ở phần Kết luận

NỘI DUNG CỦA BÀI [Ẩn]

Ứng dụng Blazor sẽ xây dựng

Hiện nay có hai mô hình triển khai Blazor: Blazor server và Blazor WebAssembly. Ngoài ra còn có mô hình Progressive Web App đang ở giai đoạn preview. Bài viết này sẽ hướng dẫn cách xây dựng một ứng dụng Blazor trọn vẹn sử dụng mô hình Blazor Server.

Chúng ta sẽ xây dựng là một ứng dụng "Todo list" đơn giản như sau:



Ứng dụng này cho phép bạn viết các ghi chú đơn giản và một số thao tác quản lý các ghi chú này. Ứng dụng lưu trữ dữ liệu trong một file JSON đặt trên server.

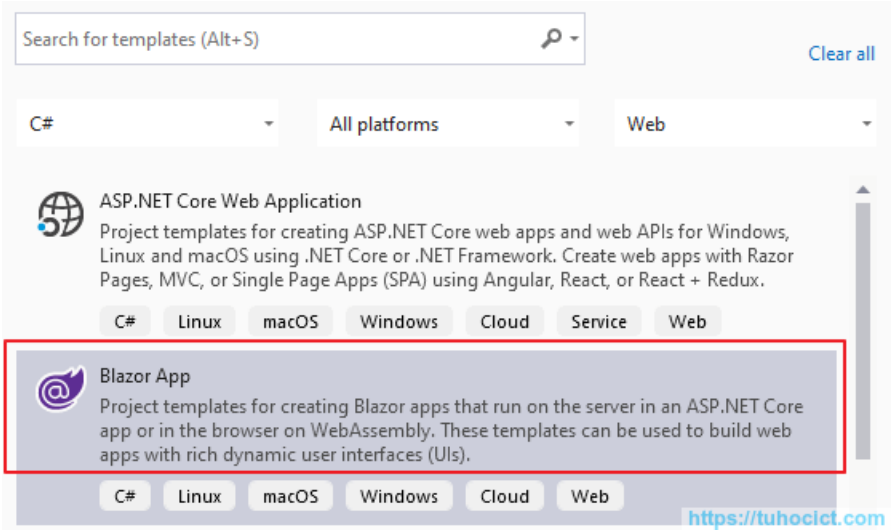
Qua bài viết này bạn sẽ làm quen với một loạt khái niệm quan trọng trong Blazor như binding, event, component.

Để thực hiện theo bài viết này, bạn cần cài đặt Visual Studio 2019 và update lên bản mới nhất (hoặc tối thiểu là bản **16.3.6**). Bạn cũng phải cài đặt workload **ASP.NET and Web development** *hoặc* **.NET Core cross-platform development**.

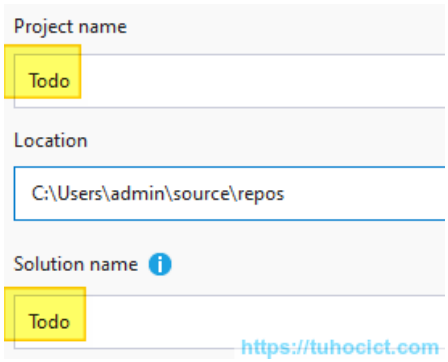
Chuẩn bị dự án

Bước 1. Tạo một project "Blazor App"

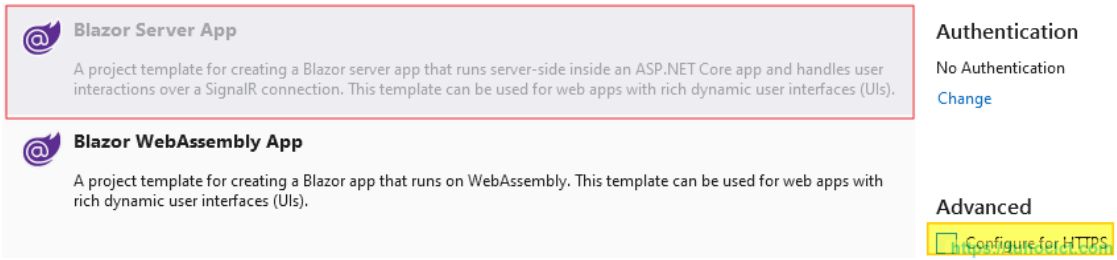
Ở bước **Create a new project** chọn **Blazor App**.



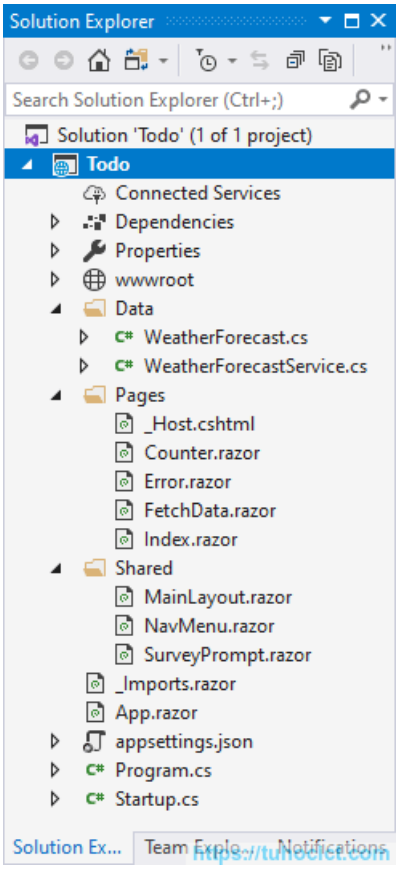
Đặt tên cho Project và Solution là **Todo**:



Ở bước Create a Blazor App chọn **Blazor Server App** và bỏ đánh dấu Configure for HTTPS.



Bạn thu được project với cấu trúc như sau:



Bước 2. Trong project này có những file dư thừa được tạo ra làm ứng dụng mẫu. Hãy xóa bớt các file sau:

- /Data/WeatherForecast.cs
- /Data/WeatherForecastService.cs
- /Pages/Counter.razor
- /Pages/FetchData.razor
- /Shared/SurveyPrompt.razor

Bước 3. Mở file Startup.cs, phương thức ConfigureServices. Tìm và xóa dòng:

```
services.AddSingleton<WeatherForecastService>();
```

Đây là dòng đăng ký khởi tạo object cho class WeatherForecastService mà bạn đã xóa từ trước.

Bước 4. Mở file /Shared/NavMenu.razor và xóa các dòng sau:

```
1.  <li class="nav-item px-3">
2.              <NavLink class="nav-link" href="counter">
3.                  <span class="oi oi-plus" aria-hidden="true"></span> Counter
4.              </NavLink>
5.          </li>
6.          <li class="nav-item px-3">
7.              <NavLink class="nav-link" href="fetchdata">
8.                  <span class="oi oi-list-rich" aria-hidden="true"></span> Fetch data
9.              </NavLink>
10.         </li>
```

Đây là những dòng thừa trong menu của ứng dụng.

Xây dựng model và dataservice

Một điểm rất hay ở Blazor là bạn có thể tiếp tục áp dụng các kỹ thuật quen thuộc của ASP.NET Core backend cho web client (frontend). Đối với Blazor Server, chuyện này còn tốt hơn nữa vì tất cả code bạn viết thực tế đều chạy trên server, mặc dù hiển thị trên client.

> Hãy đọc bài viết về mô hình triển khai của Blazor Server nếu bạn chưa biết.

Bước 1. Trong thư mục Data tạo thêm class TodoItem (file /Data/TodoItem.cs) và viết code như sau:

```
namespace Todo.Data {
    public class TodoItem {
        public string Title { get; set; }
        public bool Done { get; set; }
        public string Note { get; set; }
    }
}
```

Đây là một POCO class được sử dụng làm domain model.

Bước 2. Cài đặt thư viện Newtonsoft.Json.

Chúng ta cài đặt thư viện Newtonsoft.Json để hỗ trợ làm việc với dữ liệu Json (đọc/ghi dữ liệu với file Json).

Bạn có thể sử dụng lệnh `PM> install-package newtonsoft.json` từ Package Manager Console hoặc sử dụng NuGet Package Manager.

Bước 3. Trong thư mục Data tạo tiếp class TodoItemService (file /Data/TodoItemService.cs) và viết code như sau:

```
using System.Collections.Generic;
using System.IO;

using Newtonsoft.Json;

namespace Todo.Data {
    public class TodoItemService {
        private static List<TodoItem> _data = new List<TodoItem>() {
            new TodoItem{ Title = "Meet Donald Trump"},
            new TodoItem{ Title = "Lunch with Barack Obama"},
            new TodoItem{ Title = "Go fishing with Bill Gates"},
            new TodoItem{ Title = "Walking with Putin"},
        };

        private readonly string _file = "Data\\todo.json";

        public List<TodoItem> GetData() {
            if (File.Exists(_file)) {
                using var file = File.OpenText(_file);
                var serializer = new JsonSerializer();
                _data = serializer.Deserialize(file, typeof(List<TodoItem>)) as List<Tod
            }
            return _data;
        }

```

```
        public void SaveChanges() {
            using var file = File.CreateText(_file);
            var serializer = new JsonSerializer();
            serializer.Serialize(file, _data);
        }
    }
}
```

Bước 4. Mở file Startup.cs và điều chỉnh phương thức ConfigureServices như sau:

```
public void ConfigureServices(IServiceCollection services) {
    services.AddRazorPages();
    services.AddServerSideBlazor();
    services.AddSingleton<TodoItemService>();
}
```

Ở đây bạn đăng ký dịch vụ TodoItemService với cơ chế Dependency Injection của ASP.NET Core.

Nếu đã từng làm việc hoặc học ASP.NET Core bạn hẳn sẽ thấy các bước trên rất quen thuộc: tạo domain class => xây dựng data serivce => đăng ký dịch vụ cho Dependency Injection.

Giờ bạn có thể tiếp tục sử dụng các kỹ thuật lập trình backend quen thuộc của ASP.NET Core khi lập trình cho frontend.

Data binding trong Blazor

Mở file /Pages/Index.razor. Xóa bỏ toàn bộ code có sẵn và viết code mới như sau:

```
@page "/"
@using Data
@inject TodoItemService Service

@code{
    private List<TodoItem> _data;
    private string _current;
    private TodoItem _item;

    protected override void OnInitialized() {
        _data = Service.GetData();
        _item = _data != null && _data.Count > 0 ? _data[0] : null;
    }
}

<ol class="list-group">
    @foreach (var i in _data) {
        <li>
            <div class="input-group mb-3 @(i.Done ? "border border-success" : "")">
                <div class="input-group-prepend">
                    <span class="input-group-text">
                        <input type="checkbox" @bind="i.Done" class="custom-checkbox" />
                    </span>
                </div>
                <input type="text" @bind="i.Title" class="form-control" placeholder="Todo" />
                <input type="text" @bind="i.Note" class="form-control" placeholder="Note" />
                <div class="input-group-append">
                    <button class="btn btn-outline-danger" type="button">Delete</button>
                </div>
            </div>
        </li>
    }
</ol>
```

File /Pages/Index.razor là một **component** của Blazor. Đây cũng đồng thời là component đầu tiên được tải cùng ứng dụng với Url /. Bạn có thể hình dung component tương tự như form trong windows forms. Trên mỗi component bạn có thể đặt các điều khiển và xử lý sự kiện của các điều khiển này.

Nếu bạn đã học Razor Pages hẳn sẽ thấy code của Index component rất quen thuộc. Ngoại trừ directive @code, còn lại mọi thứ giống hệt như một trang nội dung của Razor Pages.

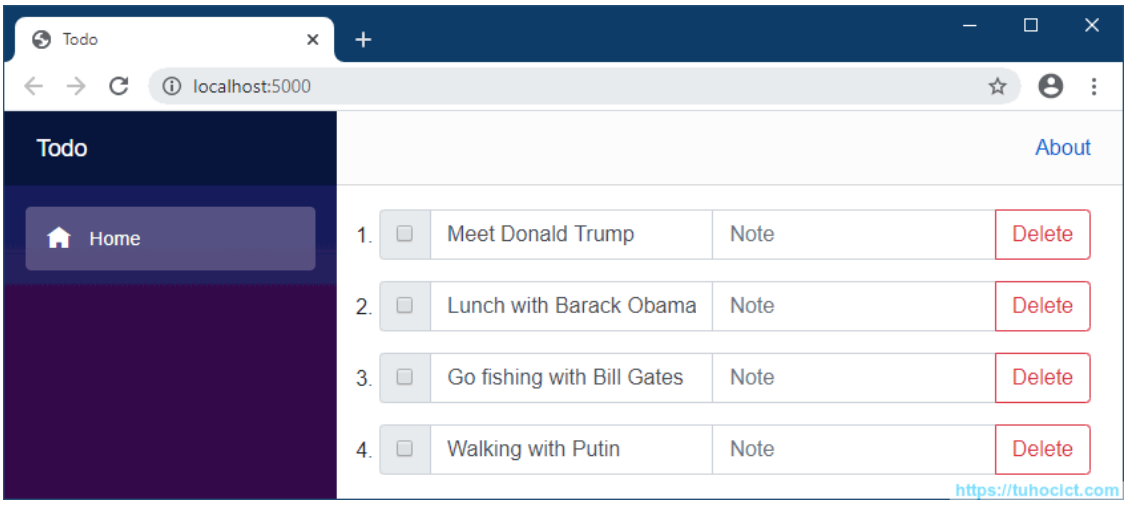
> Nếu chưa biết, bạn có thể đọc thêm về cú pháp Razor.

Directive @inject TodoItemService Service sử dụng cơ chế DI để tự động tạo object của TodoItemService. Bạn có thể thấy, chúng ta không hề khởi tạo object của class này. Tất cả được DI thực hiện tự động khi tải component Index.

Trong directive @code bạn có thể viết mọi thứ như bên trong thân một class bình thường.

Chú ý, trong @code có phương thức OnInitialized. Đây là phương thức được gọi đầu tiên khi component được tải. Code để khởi tạo các biến và thuộc tính được viết ở đây.

Nếu chạy ứng dụng bạn thu được kết quả như sau:



Trong code Razor ở trên bạn để ý thấy trong hai thẻ input có các attributet lạ @bind="i.Title" hay @bind="i.Note". Đây là cách thực hiện cơ chế data-binding của Blazor. Nó hoạt động tương tự như cơ chế data-binding trong windows forms.

Cơ chế này ghép nối dữ liệu chứa trong object với điều khiển. Nếu object thay đổi, điều khiển sẽ cập nhật. Ngược lại, nếu bạn thay đổi thông tin trên điều khiển, object sẽ thay đổi theo. Đây là cơ chế binding hai chiều.

Nhờ cơ chế databinding này, bạn không cần tự mình cập nhật giá trị điều khiển. Bạn cũng không cần quan tâm đến việc đọc dữ liệu từ điều khiển nữa. Việc lập trình cho web client, do đó, trở nên đơn giản hơn rất nhiều.

> Mặc dù có thể xuất dữ liệu ra với biểu thức Razor như bình thường, bạn nên sử dụng data binding trong Blazor để dễ dàng đồng bộ dữ liệu của object và điều khiển.

Xử lý sự kiện trong Blazor

Ở trên bạn đã xuất được dữ liệu ra. Nếu muốn thực hiện các chức năng thêm/sửa/xóa, bạn cần phải xử lý sự kiện trên các nút bấm và điều khiển tương ứng.

Bước 1. Bổ sung thêm code sau vào cuối khối @code{ … }

```
private void Add() {
    if (!string.IsNullOrWhiteSpace(_current))
        _data.Add(new TodoItem() { Title = _current });
}

private void Clear() => _data.RemoveAll(i => i.Done);

private void ClearAll() => _data.Clear();

private void DoTheThing(KeyboardEventArgs eventArgs) {
    if (eventArgs.Key == "Enter")        // fire on enter
    {
        Add();
    }
}

private void Save() => Service.SaveChanges();
```

Đây là một loạt code C# bình thường. Riêng phương thức DoTheThing có thêm tham số KeyboardEventArgs. Tuy vậy, tất cả các phương thức trên đều có thể sử dụng làm event handler trong Blazor. Đối với DoTheThing, tham số được sử dụng để lấy thông tin về phím đã bấm trên thẻ Input tương ứng.

Bước 2. Bổ sung thêm các khối code sau vào trước và sau thẻ sẵn có

```
<div class="input-group mb-3">
    <div class="input-group-prepend">
        <button @onclick="Add" class="btn btn-outline-success" type="button">Add</button
    </div>
    <input @bind="_current" type="text" class="form-control" placeholder="Todo" @onkeyup
    <div class="input-group-append">
        <button @onclick="Add" class="btn btn-outline-success" type="button">Add</button
    </div>
</div>

<ol>
<!-- code đã viết từ trước -->
</ol>

<button @onclick="Clear" class="btn btn-outline-danger">Remove done</button>
<button @onclick="ClearAll" class="btn btn-outline-danger">Clear all</button>
<button @onclick="Save" class="btn btn-outline-primary">Save</button>
```

Để đăng ký phương thức xử lý sự kiện trên điều khiển, bạn sử dụng các attribnute có dạng @onXXX như `@onkeyup`, `@onclick`. Giá trị của các attribute này là tên phương thức tương ứng đã viết trong khối @code { … }.

Bước 3. Điều chỉnh lại thẻ <button> của nút Delete như sau:

```
1.  <button class="btn btn-outline-danger" type="button" @onclick="@(e=>_data.Remove(i))">De
```

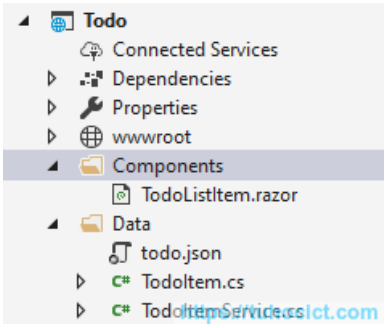
Để ý đoạn `@onclick="@(e=>_data.Remove(i))"`. Ở đây bạn gặp một cách khác để đăng ký phương thức xử lý sự kiện `@onclick` sử dụng biểu thức lambda `@(e=>_data.Remove(i))`.

Xây dựng component mới

Component là những thành phần có thể tái sử dụng trong Blazor. Bạn sử dụng component như một thẻ html tự tạo trong một component khác. Nói chung có thể hình dung component trong blazor tương tự như tag helper trong ASP.NET Core.

Chúng ta sẽ cùng xây dựng một component mới để minh họa. Component này có nhiệm vụ hiển thị thông tin về mục ToDo đang được lựa chọn.

Bước 1. Tạo thêm thư mục Components trực thuộc dự án. Trong thư mục này tạo thêm file TodoListItem.razor.



Mỗi component của Blazor chỉ là một file có đuôi razor. Không có template nào đặc biệt cho file này. Bạn chỉ cần tạo một file text và đổi đuôi thành razor là được.

Bước 2. Viết code cho TodoListItem.razor như sau:

```
@using Data

@if (Item != null) {
    <div class="input-group mb-3">
        <div class="input-group-prepend">
            <span class="input-group-text">
                <strong>Currently selected</strong>
            </span>
        </div>
        <input readonly type="text" @bind="Item.Title" class="form-control" placeholder="Todo" /
        <input readonly type="text" @bind="Item.Note" class="form-control" placeholder="Note" />
    </div>
}

@code {
    [Parameter]
    public TodoItem Item { get; set; }
}
```

Để ý trong khối @code, property public TodoItem Item { get; set; } được đánh dấu với attribute [Parameter]. Attribute này cho phép biến Item thành tham số khi sử dụng trong trang hoặc component khác.

Toàn bộ code còn lại là code Razor thông thường.

Bước 3. Quay lại file Index.razor, bổ sung @using Todo.Components vào đầu file (ngay dưới @using Data).

Đặt thẻ

```
<TodoListItem Item="_item" />
```

vào ngay trước thẻ <ol>. Bạn có thể đặt bất kỳ đâu mình muốn. Component này sẽ hiển thị mục todo nào đang được chọn.

Bước 4. Đặt đăng ký @onfocus=”@(e=>_item=i)” vào hai textbox của Title và Note:

```
1.  <input type="text" @bind="i.Title" class="form-control" placeholder="Todo" @onfocus="@(e
2.  <input type="text" @bind="i.Note" class="form-control" placeholder="Note" @onfocus="@(e=
```

Sự kiện @onfocus được kích hoạt khi người dùng click vào ô nhập Title hoặc Note. Khi sự kiện này xảy ra biến _item sẽ nhận giá trị của mục TooDoItem tương ứng trong danh sách. Khi này component <TodoListItem /> cũng sẽ được cập nhật theo.

Đến đây quá trình phát triển đã hoàn thành. Bạn có thể chạy thử nghiệm ứng dụng.

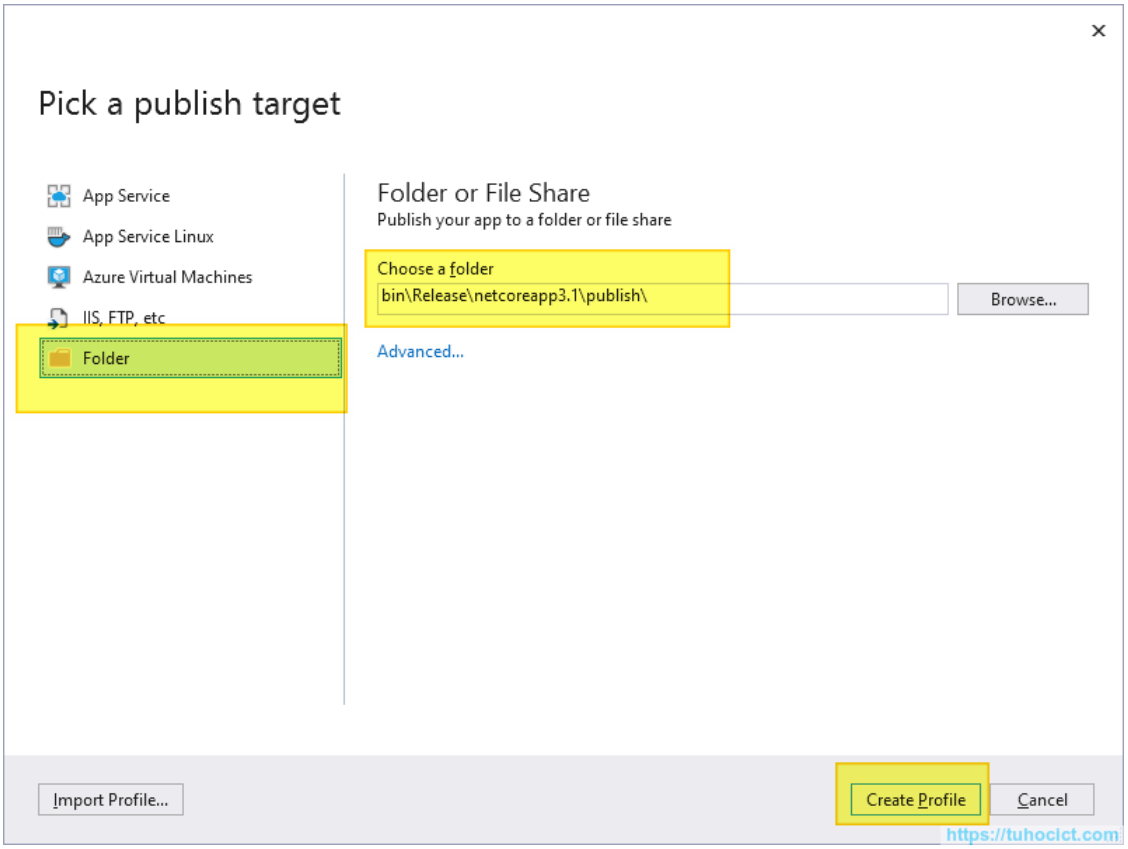
Khi bạn lưu dữ liệu lần đầu tiên, file data.json sẽ được tạo ra trong thư mục Data. Nếu không lưu dữ liệu, chương trình sẽ sử dụng dữ liệu thử nghiệm tạo sẵn trong code của TodoItemService.

Deploy ứng dụng

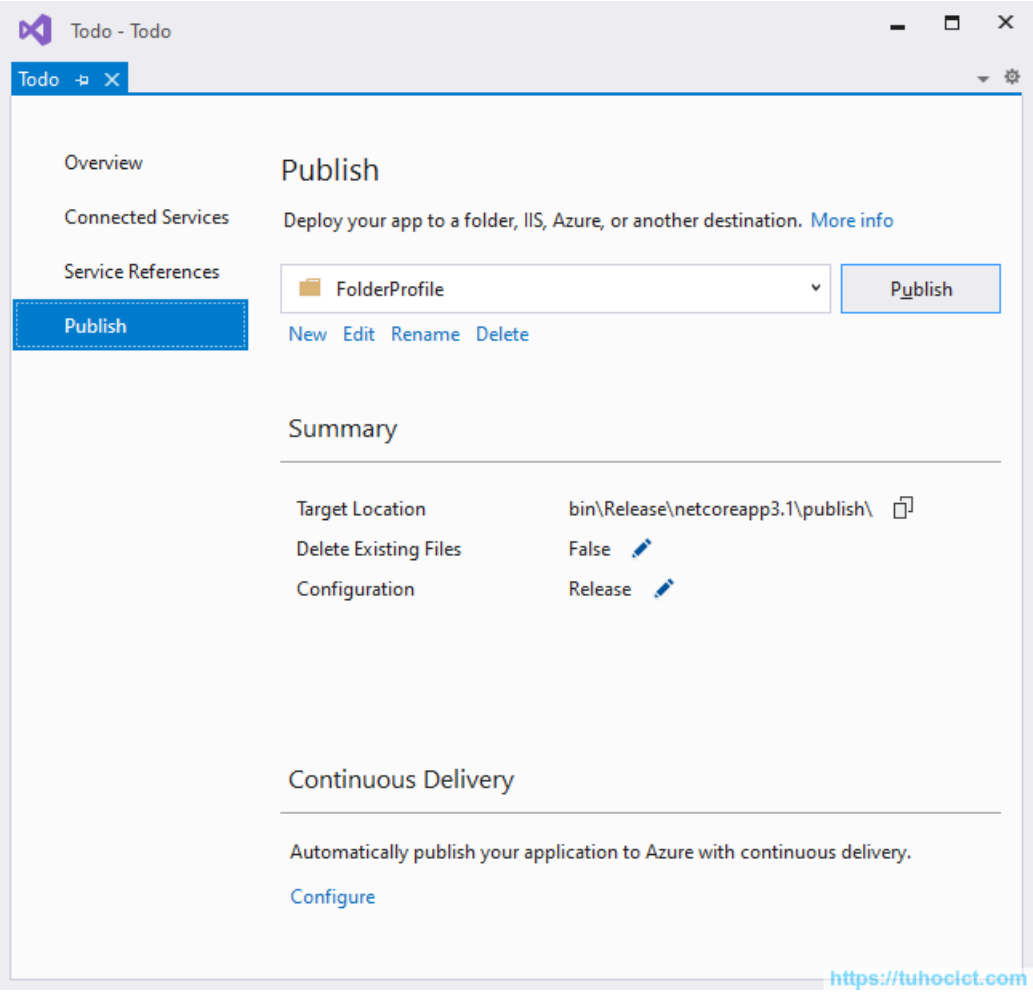
Khi quá trình phát triển hoàn thành, bạn nên deploy code để sử dụng.

Để deploy ứng dụng, bạn click phải vào dự án và chọn Publish. Bạn cũng có thể chọn từ menu Build -> Publish Todo.

Cách đơn giản nhất là công bố ứng dụng vào một thư mục trên ổ cứng. Từ đây bạn có thể chạy ứng dụng ASP.NET Core như một ứng dụng console thông thường và truy xuất giao diện đồ họa qua trình duyệt.



Sau khi chỉnh thư mục theo ý muốn, click nút Create profile. Bạn chỉ cần thực hiện các thao tác trên một lần duy nhất để tạo profile. Từ lần sau bạn sẽ vào thẳng giao diện dưới đây.

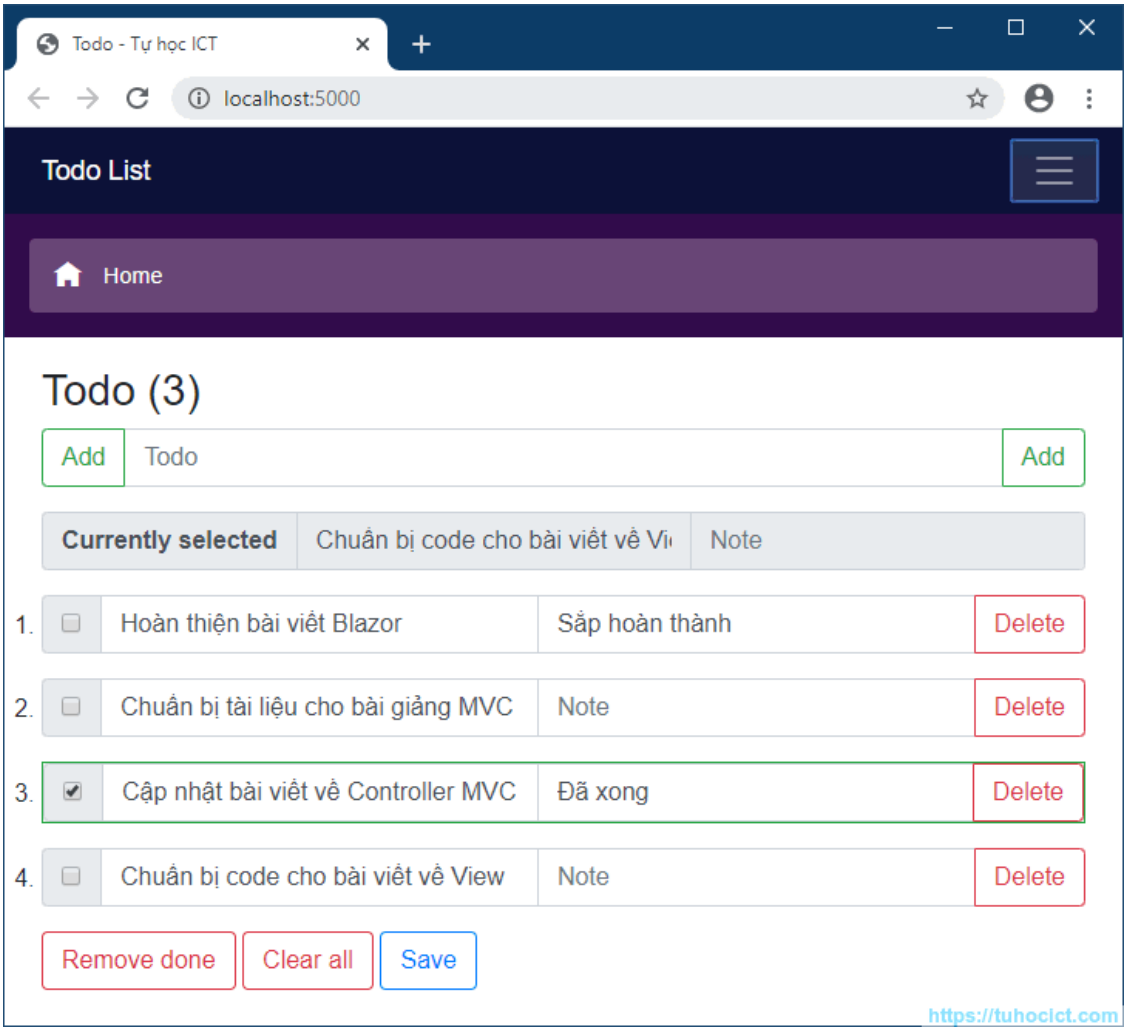


Ấn nút Publish, ứng dụng sẽ được dịch ở chế độ Release và đưa vào thư mục bin\Release\netcoreapp3.1\publish.

Khi cần mang ứng dụng đi triển khai, bạn copy thư muc trên là đủ. Bạn có thể chạy ứng dụng này trên Linux, Mac hoặc Windows.

Trên Windows, bạn tìm đến file Todo.exe trong thư mục trên và chạy nó như một chương trình console thông thường.

Khi chương trình đã chạy, bạn truy xuất ứng dụng qua trình duyệt với Url localhost:5000 hoặc localhost:5001.



Nếu không muốn sử dụng cổng 5000 hoặc 5001 như mặc định, hãy thêm mục sau vào file appsettings.json:

```
"Kestrel": {
    "EndPoints": {
      "Http": {
        "Url": "https://localhost:443"
      }
    }
  }
```

Thiết lập này yêu cầu Kestrel – web server của ASP.NET Core – tiếp nhận truy vấn từ Url theo chỉ định. Nếu sử dụng HTTPS, bạn cần nghe ở cổng 443 (mặc định). Nếu sử dụng HTTP, bạn cần nghe cổng 80.

Nếu ứng dụng nghe các cổng mặc định này, trình duyệt sẽ tự động thêm cổng phù hợp giúp bạn.

Đến đây xin chúc mừng bạn đã xây dựng được một ứng dụng Blazor hoàn chỉnh có thể sử dụng được!

Vấn đề triển khai ứng dụng Blazor

Bạn có thể hơi ngạc nhiên khi việc triển khai một ứng dụng Blazor lại đơn giản như một ứng dụng desktop console như vậy. Nó hoàn toàn không giống như triển khai một ứng dụng web trên IIS hay Apache.

Ứng dụng Blazor như bạn vừa xây dựng cũng là một ứng dụng ASP.NET Core. Do đó, việc triển khai nó cũng tương tự như triển khai bất kỳ ứng dụng ASP.NET Core nào khác.

Nếu bạn đã đọc về ASP.NET Core bạn hẳn sẽ biết rằng framework này được xây dựng bên trên .NET Core, vốn ban đầu chỉ hỗ trợ các ứng dụng console đa nền tảng. Ứng dụng ASP.NET Core cũng là một ứng dụng console đa nền tảng.

Ứng dụng ASP.NET Core sử dụng một web server tích hợp có tên gọi là Kestrel. Đây là một chương trình web server hoàn chỉnh, hiệu suất cao tích hợp với các ứng dụng ASP.NET Core.

Như vậy, khi chương trình console hoạt động, Kestrel cũng hoạt động và biến ứng dụng console trở thành một ứng dụng web nhờ khả năng tiếp nhận truy vấn HTTP.

Nếu bạn triển khai ứng dụng trong intranet, Kestrel là hoàn toàn đủ. Sử dụng Kestrel và console giúp deploy ứng dụng trong intranet trở nên vô cùng đơn giản và tiện lợi.

Tuy nhiên, Kestrel không được thiết kế để tiếp xúc với Internet. Nói cách khác, bạn không nên để Kestrel nghe truy vấn đến từ Internet. Nếu cần triển khai ứng dụng trên Internet, bạn vẫn cần đến IIS, Apache hay NGinx kết hợp với Kestrel. Khi này, IIS, Apache hay NGinx hoạt động như một reverse proxy.

Vấn đề triển khai Blazor kết hợp với reverse proxy hơi lạc đề nên chúng ta không tiếp tục nói về vấn đề này nữa.

> Đọc thêm về mô hình hoạt động của ASP.NET Core.

Kết luận

Qua bài viết này chúng ta đã cùng xây dựng một ứng dụng Blazor server hoàn chỉnh từ a -> z. Qua đây chúng ta đã làm quen với những khái niệm cơ bản nhất của Blazor.

Bạn có thể thấy, phát triển ứng dụng web client với Blazor thực sự đơn giản so với sử dụng JavaScript. Bạn cũng có thể tiếp tục sử dụng những kỹ thuật lập trình ASP.NET Core quen thuộc.

Nếu khó hình dung về quá trình làm, bạn có thể tham khảo video hướng dẫn sau.

Bạn có thể tải mã nguồn để tham khảo:
https://1drv.ms/u/s!Ar_aj4rIJ2qGkfYKT6b9heNzzgzZYQ?e=WDg4lN

+ 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!