Thực hành (1) xây dựng ứng dụng trong ASP.NET Core MVC (+video)

CRUD (Create-Retrieve-Update-Delete) là nhóm thao tác cơ bản nhất với dữ liệu mà mọi ứng dụng ASP.NET Core MVC đều phải thực hiện.

Các kiến thức bạn đã học trong các bài trước đã đủ để xây dựng một ứng dụng MVC với các chức năng CRUD cơ bản.

Trong bài học này chúng ta sẽ vận dụng tổng hợp các kiến thức đã học để xây dựng một ứng dụng nhỏ nhưng hoàn chỉnh. Qua đó bạn sẽ nhìn thấy cách các thành phần của ứng dụng MVC hoạt động và tương tác với nhau.

> Video hướng dẫn ở phần kết luận cuối bài.

Kết thúc loạt bài thực hành này bạn sẽ thu được ứng dụng như sau:

NỘI DUNG CỦA BÀI [Ẩn]

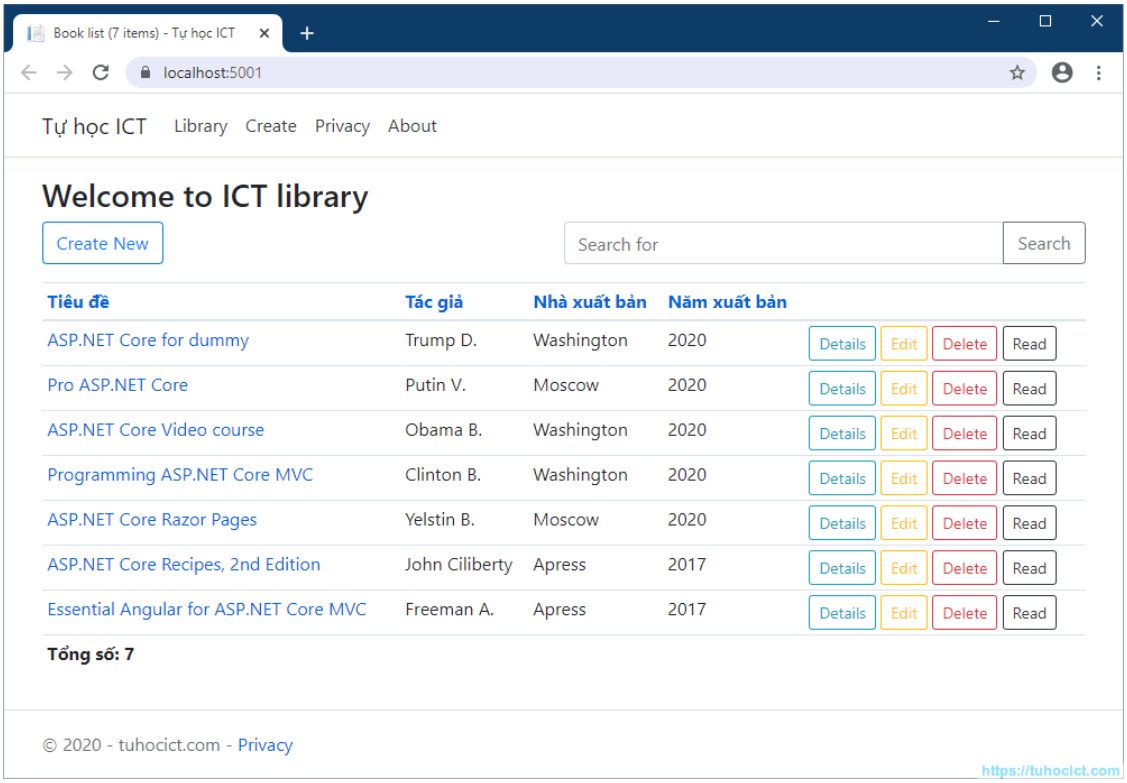
Yêu cầu

Trong bài học này bạn sẽ xây dựng một ứng dụng quản lý sách điện tử đơn giản với khả năng:

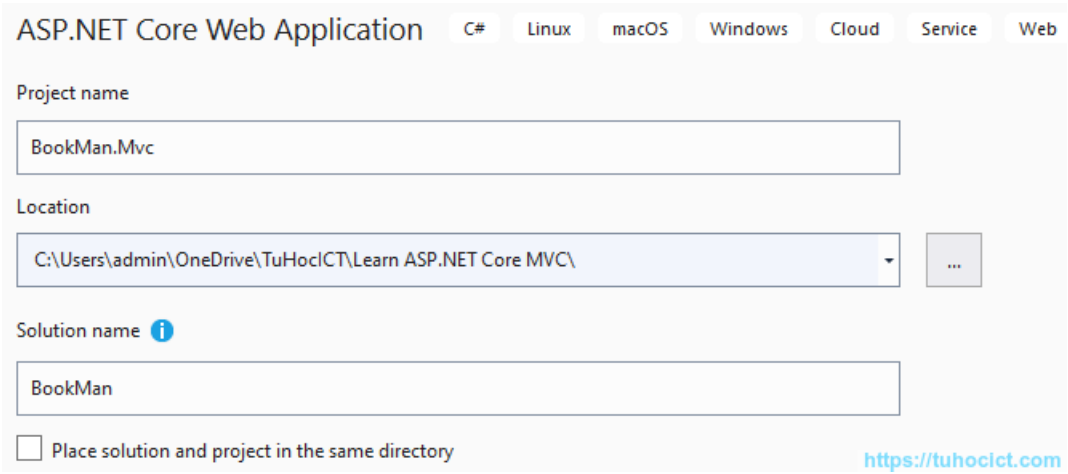
- Hiển thị danh sách các cuốn sách đang có;
- Hiện thị thông tin chi tiết một cuốn sách theo yêu cầu của người dùng;
- Cập nhật thông tin một cuốn sách;
- Xóa bỏ dữ liệu một cuốn sách;
- Thêm một cuốn sách mới;

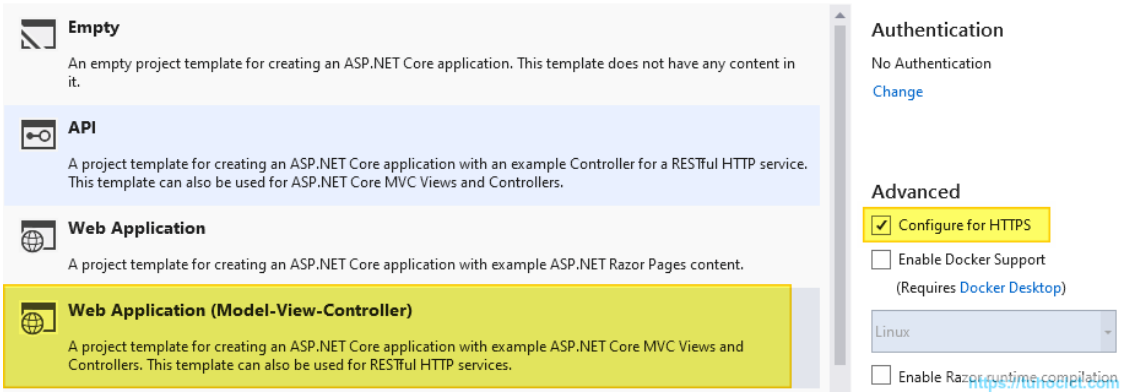
Khi cập nhật hoặc thêm mới, bạn có thể upload file pdf của cuốn sách lên server. Khi xem thông tin (danh sách hoặc chi tiết), bạn có thể tải file pdf của cuốn sách về máy để đọc.

Kết thúc bài học này bạn sẽ thu được một ứng dụng đơn giản như sau:



Để chuẩn bị, hãy tạo một project mới theo template Web Application (Model-View-Controller) như sau:





Xây dựng model

Như bạn đã học, trong một dự án MVC sử dụng nhiều loại model khác nhau. Các model chúng ta thường phải xây dựng ngay từ đầu bao gồm domain model và application model.

Domain model

Trong **thư mục Models** tạo mới class **Book** trong file **Book.cs** và viết code như sau:

```
using System;
using System.ComponentModel;
using System.ComponentModel.DataAnnotations;

namespace BookMan.Mvc.Models
{
    public class Book
    {
        public int Id { get; set; }
        [Required, DisplayName("Tiêu đề")]
        public string Name { get; set; }
        [Required, DisplayName("Tác giả")]
        public string Authors { get; set; }
        [Required, DisplayName("Nhà xuất bản")]
        public string Publisher { get; set; }
        [Required, Range(1990, int.MaxValue), DisplayName("Năm xuất bản")]
        public int Year { get; set; }
        [DisplayName("Tóm tắt")]
        public string Description { get; set; }
        [DisplayName("File")]
        public string DataFile { get; set; }
    }
}
```

Class Book mô tả các thông tin cơ bản của một cuốn sách. Đây là một *domain model* class. Nó thể hiển dữ liệu của nghiệp vụ quản lý sách.

Trong class này chúng ta sử dụng một số attribute đặc biệt như `[Required]`, `[DisplayName], [Range]`.

Các attribute này có tác dụng trong việc thẩm định dữ nhiệu người dùng ([Required], [Range]) hoặc có liên quan đến hiển thị trên view ([DisplayName]). Bạn sẽ thấy tác dụng của các attribute này khi xây dựng view.

Application model

Bước 1. Tạo Service class

Cùng trong thư mục Models tạo thêm class Service trong file Service.cs và viết code như sau:

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Xml.Serialization;

namespace BookMan.Mvc.Models
{
    public class Service
    {
        private readonly string _dataFile = @"Data\data.xml";
        private readonly XmlSerializer _serializer = new XmlSerializer(typeof(HashSet<Bo
        public HashSet<Book> Books { get; set; }

        public Service()
        {
```

```
            if (!File.Exists(_dataFile))
            {
                Books = new HashSet<Book>() {
                    new Book{Id = 1, Name = "ASP.NET Core for dummy", Authors = "Trump D
                    new Book{Id = 2, Name = "Pro ASP.NET Core", Authors = "Putin V.", Pu
                    new Book{Id = 3, Name = "ASP.NET Core Video course", Authors = "Obam
                    new Book{Id = 4, Name = "Programming ASP.NET Core MVC", Authors = "C
                    new Book{Id = 5, Name = "ASP.NET Core Razor Pages", Authors = "Yelst
                };
            }
            else
            {
                using var stream = File.OpenRead(_dataFile);
                Books = _serializer.Deserialize(stream) as HashSet<Book>;
            }
        }

        public Book[] Get() => Books.ToArray();

        public Book Get(int id) => Books.FirstOrDefault(b => b.Id == id);

        public bool Add(Book book) => Books.Add(book);

        public Book Create()
        {
            var max = Books.Max(b => b.Id);
            var b = new Book()
            {
                Id = max + 1,
                Year = DateTime.Now.Year
            };
            return b;
        }

        public bool Update(Book book)
        {
            var b = Get(book.Id);
            return b != null && Books.Remove(b) && Books.Add(book);
        }

        public bool Delete(int id)
        {
            var b = Get(id);
            return b != null && Books.Remove(b);
        }

        public void SaveChanges()
        {
            using var stream = File.Create(_dataFile);
            _serializer.Serialize(stream, Books);
        }
    }
}
```

Service là một class hỗ trợ làm việc với dữ liệu Book. Trong class này chứa đầy đủ các phương thức CRUD với Book mà chương trình cần có.

Toàn bộ dữ liệu của chương trình được lưu trong file `Data\data.xml`.

Bạn có thể xem Service tương tự như class chịu trách nhiệm làm việc với cơ sở dữ liệu. Mặc dù, ở đây, để đơn giản và tiện lợi khi test, chúng ta không sử dụng cơ sở dữ liệu.

> Lưu ý **tạo thư mục Data** trực thuộc dự án trước khi chạy chương trình lần đầu. Thiếu thư mục này chương trình sẽ báo lỗi khi tạo file dữ liệu data.xml.

Tạo controller

Bước 1. Tạo BookController

Tạo class BookController (file BookController.cs) trong thư mục Controllers và viết code như sau:

```
using BookMan.Mvc.Models;
using Microsoft.AspNetCore.Mvc;

namespace BookMan.Mvc.Controllers
{
    public class BookController : Controller
    {
        private readonly Service _service;
        public BookController(Service service)
        {
            _service = service;
        }
        public IActionResult Index()
        {
            return View(_service.Get());
        }
    }
}
```

BookController đang còn rất đơn giản, chỉ có một action Index. Trong các phần tiếp theo chúng ta sẽ lần lượt bổ sung code mới vào BookController.

Để ý trong constructor có tham số kiểu Service. Khi ứng dụng chạy, cơ chế DI sẽ tự động tạo object của Service và truyền vào cho BookController. Chúng ta sẽ đăng ký Service với DI sau.

Action Index trả về một object ViewResult thông qua phương thức hỗ trợ View. View model Index truyền sang cho view là một mảng (Book[]) do phương thức Get của Service sinh ra.

Action này cũng sử dụng view theo quy ước của ASP.NET Core MVC (/Views/Book/Index.cshtml) mà chúng ta sẽ xây dựng sau.

Bước 2. Tạo route riêng cho BookController

Điều chỉnh lời gọi `app.UserEndpoints` của phương thức `Configure` lớp Startup để thêm một route mới như sau:

```
app.UseEndpoints(endpoints =>
{
    endpoints.MapControllerRoute("book", "{action=Index}/{id?}", new { controller = "Boo
});
```

Ở đây chúng ta tạo thêm một route đặc biệt thay cho route mặc định của ứng dụng MVC.

Route này tự động sử dụng BookController và ánh xạ mỗi Url trực tiếp vào action của controller này.

Route này tương ứng với các Url như `/Index`, `/Edit`, `/Delete`, `/Details`, v.v. mà chúng ta sẽ xây dựng sau.

Bước 3. Đăng ký sử dụng Service trong Startup class

Để sử dụng Service trong BookController, bạn cần đăng ký nó trong phương thức `ConfigureServices` của `Startup` class.

```
1.  public void ConfigureServices(IServiceCollection services)
2.  {
3.      services.AddControllersWithViews();
4.      services.AddSingleton<Service>();
5.  }
```

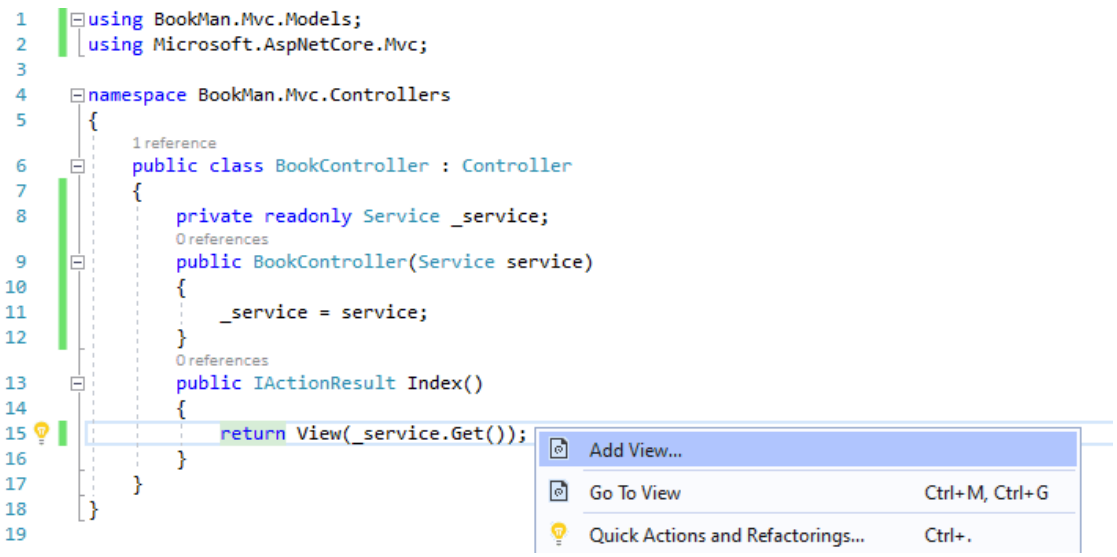
Bước này đăng ký Service với DI của ASP.NET Core.

Bạn sẽ không cần tự mình khởi tạo object của Service. Cơ chế DI của ASP.NET Core sẽ tự động khởi tạo và truyền object của Service giúp bạn.

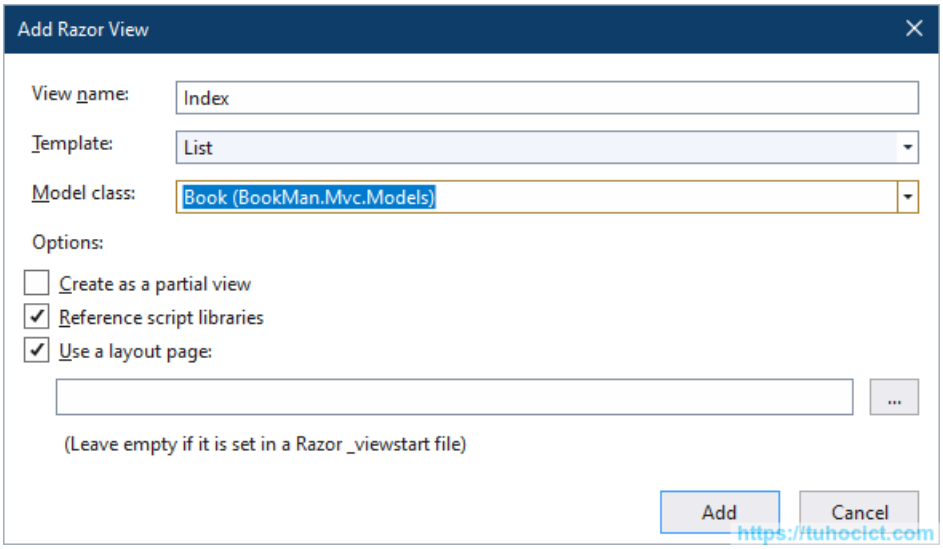
Tạo view cho Index action

Bước 1. Tự động sinh Index.cshtml view

Click phải chuột vào vị trí bất kỳ bên trong code của Index và chọn Add View… từ context menu.



Chọn View name, Template và Model class như sau:



Đây là khả năng scafolding (sinh code tự động) của ASP.NET Core. Scafolding giúp nhanh chóng sinh ra một view hoàn chỉnh sử dụng được ngay. Bạn chỉ cần tinh chỉnh view này theo nhu cầu.

Scafolding có thể tự động sinh code với các template List (danh sách), Details (chi tiết), Edit (cập nhật), Delete (xác nhận xóa), và Empty (không dùng template nào). Các template này có thể hoạt động cùng model class tương ứng.

View được tự động sinh ra trong thư mục /Views/Book và trùng tên với Action.

Qua bước này bạn đã tạo ra view /Views/Book/Index.cshtml.

Bước 2. Tinh chỉnh Index.cshtml

Bạn có thể tiếp tục sử dụng view Index.cshtml do ASP.NET Core tự động sinh ra.

Tuy nhiên để theo dõi và hoàn thành bài học này, bạn hãy điều chỉnh code của Index.cshtml như sau:

```
@model IEnumerable<BookMan.Mvc.Models.Book>

@{
    ViewData["Title"] = $"Book list ({Model.Count()} items)";
}

<p class="h3">ICT library</p>
<table class="table table-hover table-sm">
    <thead>
        <tr>
            <th>@Html.DisplayNameFor(model => model.Name)</th>
            <th>@Html.DisplayNameFor(model => model.Authors)</th>
            <th>@Html.DisplayNameFor(model => model.Publisher)</th>
            <th>@Html.DisplayNameFor(model => model.Year)</th>
            <th></th>
        </tr>
    </thead>
    <tbody>
        @foreach (var item in Model)
        {
            <tr>
                <td><a asp-route-id="@item.Id" asp-action="Details">@item.Name</a></td>
                <td>@item.Authors</td>
                <td>@item.Publisher</td>
                <td>@item.Year</td>
```

```
                <td>
                    <a asp-action="Details" asp-route-id="@item.Id" class="btn btn-sm bt
                </td>
            </tr>
        }
        <tr><td colspan="5"><strong>Tổng số: @Model.Count()</strong></td></tr>
    </tbody>
</table>
```

Trong Index.cshtml có một số điều lưu ý sau:

Model của view này có kiểu là `@model IEnumerable<BookMan.Mvc.Models.Book>.` Để ý rằng view model mà Index trả cho view có kiểu Book[] – mảng này thực thi IEnumerable<Book>.

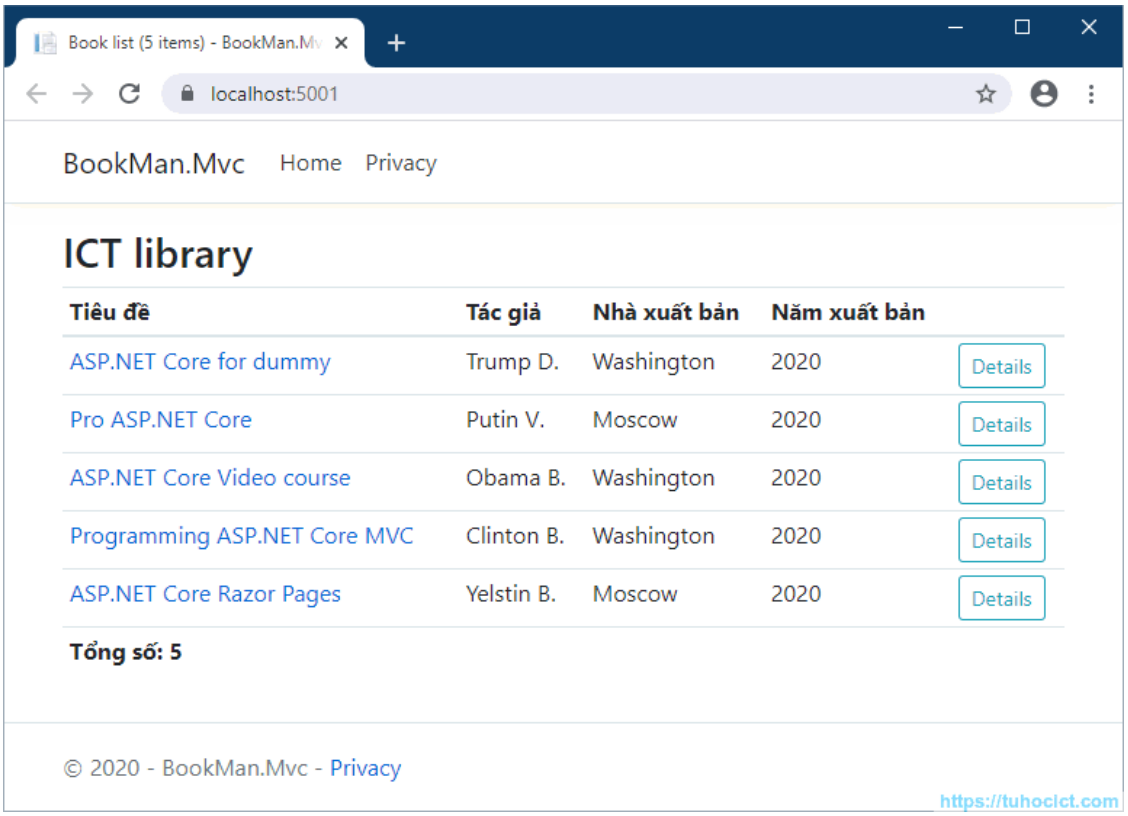
Các biểu thức có dạng @Html.DisplayNameFor(model => model.Name) được gọi là **Html helper**. Html helper là những phương thức C# hỗ trợ sinh HTML. Đây là sản phẩm từ thời ASP.NET MVC.

Một số thẻ a có các attribute lạ như <a asp-action=”Details” asp-route-id=”@item.Id> . Đây không còn là một thẻ <a> bình thường mà là một Tag helper. Tag helper này giúp sinh ra thẻ a có dạng

Tag helper là cơ chế hỗ trợ sinh Html (tương tự như Html helper) nhưng được sử dụng ở dạng thẻ (như của ngôn ngữ HTML), thay cho dạng phương thức của C#.

ASP.NET Core khuyến khích sử dụng Tag helper thay cho Html helper. Tuy vậy trong một số trường hợp bạn vẫn phải dùng đến Html helper như trong Index view ở trên.

Chạy thử chương trình bạn thu được kết quả như sau:



> Lưu ý **tạo thư mục Data** trực thuộc dự án trước khi chạy chương trình lần đầu. Thiếu thư mục này chương trình sẽ báo lỗi khi tạo file dữ liệu data.xml.

Hiển thị chi tiết

Trong phần này chúng ta bổ sung tính năng hiển thị chi tiết một cuốn sách khi người dùng click vào đường link trong danh sách.

Bước 1. Xây dựng Details action

```
public IActionResult Details(int id)
{
    var b = _service.Get(id);
    if (b == null) return NotFound();
    else return View(b);
}
```

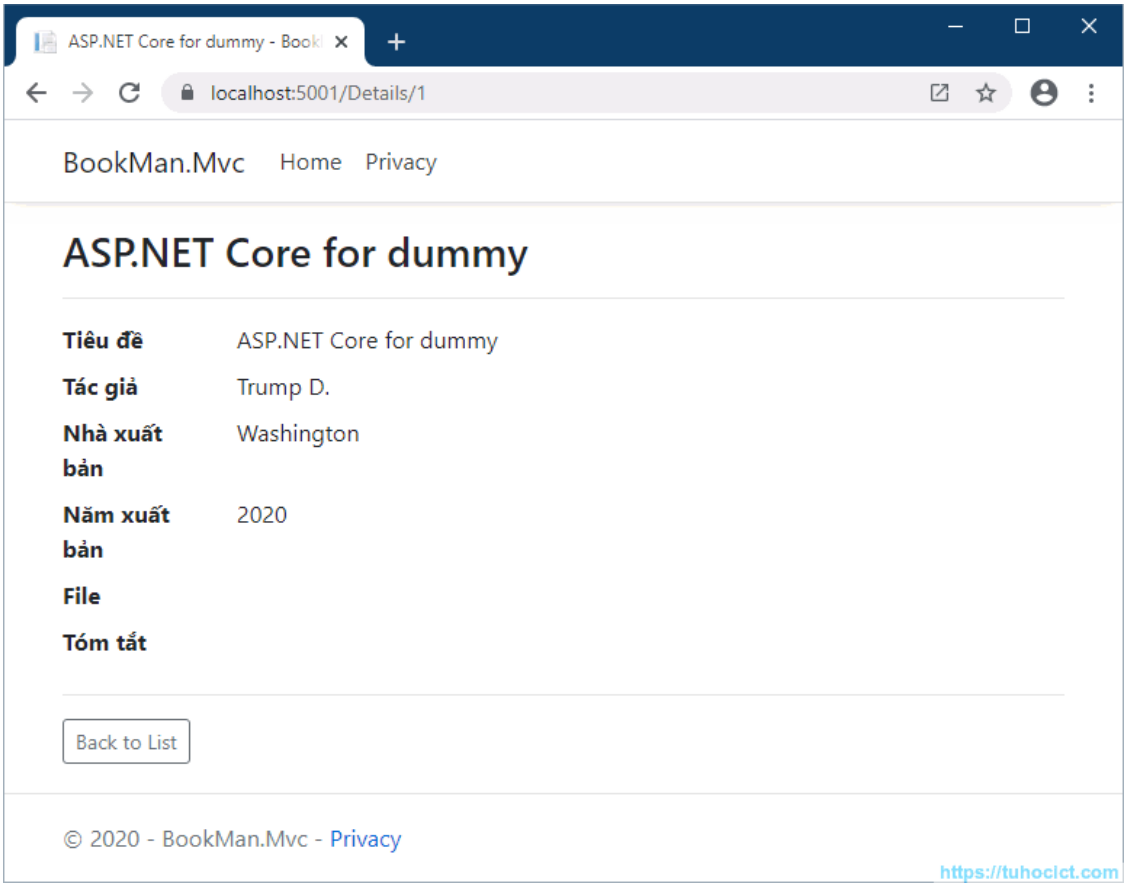
Bước 2. Xây dựng Details view

Bạn sử dụng scafolding để tạo một view mới cho Details action và tinh chỉnh lại view này như sau:

```
@model BookMan.Mvc.Models.Book

@{
    ViewData["Title"] = Model.Name;
}

<div>
    <p class="h3">@Model.Name</p>
    <hr />
    <dl class="row">
        <dt class="col-sm-2"><label asp-for="Name"></label></dt>
        <dd class="col-sm-10">@Model.Name</dd>
        <dt class="col-sm-2"><label asp-for="Authors"></label></dt>
        <dd class="col-sm-10">@Model.Authors</dd>
        <dt class="col-sm-2"><label asp-for="Publisher"></label></dt>
        <dd class="col-sm-10">@Model.Publisher</dd>
        <dt class="col-sm-2"><label asp-for="Year"></label></dt>
        <dd class="col-sm-10">@Model.Year</dd>
        <dt class="col-sm-2"><label asp-for="DataFile"></label></dt>
        <dd class="col-sm-10">@Model.DataFile</dd>
        <dt class="col-sm-2"><label asp-for="Description"></label></dt>
        <dd class="col-sm-10">@Model.Description</dd>
    </dl>
</div>

<hr />

<div>
    <a asp-action="Index" class="btn btn-sm btn-outline-secondary">Back to List</a>
</div>
```

Nếu không dùng scafolding, hãy tạo Razor view Details.cshtml trong thư mục /Views/Book và viết code như trên.

Chạy thử chương trình và click vào nút Details từ trang chủ, bạn thu được kết quả như sau:



Nếu bạn cố tình nhập một Id không có trong danh sách dữ liệu, site sẽ trả về trang 404.

Kết luận

Trong phần thứ nhất của bài thực hành tổng hợp này chúng ta đã xây dựng được chức năng đầu tiên (R – Retrieve) của một ứng dụng quản lý sách đơn giản.

Trong bài thực hành tiếp theo, chúng ta sẽ tiếp tục xây dựng các chức năng xóa, cập nhật và thêm mới.

Link tải mã nguồn: https://1drv.ms/u/s!Ar_aj4rIJ2qGkoEpRbH-TzkM6BAvog?e=9i5uqS

Video hướng dẫn trên Youtube:

> + 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!