Thực hành (1) Xây dựng ứng dụng quản lý sách (+video)

Qua loạt bài học Razor Pages từ đầu đến giờ bạn đã nắm được rất nhiều kiến thức mới. Giờ là lúc chúng ta vận dụng tổng hợp để giải quyết một bài toán (hơi hơi) thực tế. Bài học này hướng tới củng cố những gì bạn đã học trong những bài học trước đây.

Trong bài học này chúng ta sẽ cùng xây dựng một ứng dụng quản lý sách điện tử đơn giản. Do kiến thức chưa đầy đủ, chúng ta sẽ chỉ xây dựng một phần chức năng của ứng dụng, chủ yếu là chức năng xuất dữ liệu.

> Video hướng dẫn và link tải mã nguồn để ở phần kết luận.

NỘI DUNG CỦA BÀI [Ẩn]

Bài toán

Trong bài học này chúng ta sẽ cùng xây dựng một ứng dụng quản lý sách điện tử đơn giản.

Ứng dụng này có thể:

- (1) xuất ra danh sách các cuốn sách điện tử (file pdf) đang lưu trong một thư mục trên máy chủ;
- (2) thêm một cuốn sách mới;
- (3) cập nhật thông tin sách;
- (4) xóa bỏ sách.

Nói tóm lại, đây là một ứng dụng CRUD dữ liệu cơ bản và rất phổ biến.

Bạn sẽ lần lượt hoàn thiện ứng dụng này qua các bài thực hành tổng hợp theo lộ trình học và vận dụng những kiến thức học được.

Trong bài thực hành tổng hợp đầu tiên này bạn sẽ xây dựng chức năng thứ nhất: xuất dữ liệu.

Khi kết thúc bài thực hành này bạn sẽ thu được một ứng dụng web đơn giản như sau:

Tạo dự án Razor Pages

Hãy thực hiện các bước như sau để tạo dự án Razor Pages theo template Web Application:

Lưu ý ở đây chúng ta tích chọn thêm hai mục:

(1) Configure for HTTPS: cấu hình để ứng dụng hoạt động với giao thức HTTPS (có mã hóa dữ liệu). Lựa chọn này sẽ tạo ra một chứng chỉ bảo mật (certificate) giả nhằm mục đích thử nghiệm ứng dụng. Khi chạy thử nghiệm, trình duyệt sẽ đưa ra thông báo bảo mật. Bạn hãy chấp nhận đưa site vào danh sách ngoại lệ.

(2) Enable Razor runtime compilation: đây là một tính năng mới của .NET Core 3.1 cho phép dịch trang razor ngay trong khi ứng dụng đang chạy debug. Một khi bạn lưu lại những thay đổi của trang razor, bạn có thể refresh trình duyệt để tải lại nội dung mới mà không cần chạy lại ứng dụng từ đầu. Tính năng này giúp giảm bớt thời gian debug giao diện.

Tạo domain model

Tạo thư mục Model trực thuộc project. Trong thư mục này tạo file Book.cs.

Viết code cho Book.cs như sau:

```
using System;

namespace BookMan.WebApp.Model {
    public class Book {
        public int Id { get; set; }
        public string Title { get; set; } = "A new book";
        public string Authors { get; set; } = "Authors";
        public string Publisher { get; set; } = "Publisher";
        public int Year { get; set; } = DateTime.Now.Year;
        public string Description { get; set; } = "";
    }
}
```

Bạn có thể để ý thấy đây là một POCO (Plain-Old-C#-Object) – loại class C# chỉ chứa dữ liệu thuộc các kiểu cơ sở. Bạn phải sử dụng POCO để sau này có thể làm việc với Entity Framework.

Sử dụng Dependency Injection

Dependency Injection (DI) là một kỹ thuật khá "cao cấp". Đa số các bạn khi học lập trình đều xa lạ với khái niệm này. Do đây không phải là trọng tâm của bài học, chúng ta sẽ không đi vào giải thích chi tiết.

Tuy nhiên, trong ASP.NET Core, DI lại là loại kỹ thuật được sử dụng rất phổ biến và được hỗ trợ mặc định.

Cách thực hiện DI trong ASP.NET Core rất đơn giản.

Tạo interface

Trước hết tạo thư mục Interface trực thuộc project. Trong thư mục này tạo file IRepository.cs.

Viết code cho IRepository.cs như sau:

```
using System.Collections.Generic;

using BookMan.WebApp.Model;

namespace BookMan.WebApp.Interface {
    public interface IRepository {
        public HashSet<Book> Books { get; set; }
        public Book Get(int id);
    }
}
```

Ở đây bạn vừa tạo ra một interface để sử dụng với mô hình repository.

Tạo repository

Tiếp theo tạo thư mục Repository trực thuộc project. Trong thư mục này tạo thêm file BookRepository.cs.

Viết code cho BookRepository.cs như sau:

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

using BookMan.WebApp.Interface;
using BookMan.WebApp.Model;

namespace BookMan.WebApp.Repository {
    public class BookRepository : IRepository {
        public HashSet<Book> Books { get; set; } = new HashSet<Book>
        {
            new Book {Id = 1, Title = "ASP.NET Core for dummy",Publisher = "Apress",
            new Book {Id = 2,  Title = "Professional ASP.NET Core 3",Publisher = "Man
            new Book {Id = 3,  Title = "ASP.NET Core Self learning",Publisher = "Wile
            new Book {Id = 4,  Title = "ASP.NET Core quick course",Publisher = "Linda
            new Book {Id = 5,  Title = "ASP.NET Core Video Course",Publisher = "Linda
        };
        public Book Get(int id) => Books.SingleOrDefault(b => b.Id == id);
    }
}
```

Cấu hình DI

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

```
public void ConfigureServices(IServiceCollection services) {
    services
        .AddRazorPages()
        .AddRazorRuntimeCompilation();
    services.AddSingleton<IRepository, BookRepository>();
}
```

Lưu ý bổ sung hai mục using vào đầu file:

```
using BookMan.WebApp.Interface;
using BookMan.WebApp.Repository;
```

Ở đây chúng ta thực hiện hai cấu hình mới:

AddRazorRuntimeCompilation() – yêu cầu sử dụng chức năng runtime compilation – dịch page khi đang debug.

AddSingleton() – cấu hình cho Dependency Injection. Bạn sẽ nhìn thấy hoạt động của DI khi tạo các page về sau.

Index page – xuất danh sách

Page model class

Mở file Index.cshtml.cs và viết code như sau:

```
using System.Collections.Generic;
using BookMan.WebApp.Interface;
using BookMan.WebApp.Model;
using Microsoft.AspNetCore.Mvc.RazorPages;

namespace BookMan.WebApp.Pages {
    public class IndexModel : PageModel {
        private readonly IRepository _repository;
        public HashSet<Book> Books => _repository.Books;
        public int Count => _repository.Books.Count;
        public IndexModel(IRepository repository) => _repository = repository;
    }
}
```

Đây là một model class mà bạn đã được học.

Ở đây bạn có thể để ý thấy hàm tạo của IndexModel có tham số thuộc kiểu IRepository. Giá trị của tham số này sẽ được gán cho hằng _repository. Đây chính là một cách sử dụng của kỹ thuật DI – constructor DI.

Hãy nhớ lại trong file Startup.cs chúng ta đã cấu hình cho IRepository "tương ứng" với BookRepository.

Hiểu đơn giản như thế này: mỗi khi object của class IndexModel được tạo ra, Razor Pages (chính xác hơn là DI Container của Asp.net Core) sẽ tự động sinh ra một object thuộc kiểu BookRepository và truyền cho hàm tạo này. Chúng ta không cần mất công tự tạo object cho BookRepository nữa.

Kỹ thuật này vô cùng có lợi nếu bạn có một số lượng trang rất lớn, và bạn cần thay thế BookRepository bằng một class khác, ví dụ, khi bạn chuyển sang sử dụng một ORM nào đó như Entity Framework hay Entity Framework Core. Khi này bạn chỉ cần thay thế đúng một dòng code trong file Startup.cs.

Giao diện

Điều chỉnh nội dung của file Index.cshtml như sau:

```
@page
@model IndexModel
@{
    ViewData["Title"] = "Home page";
}

<div class="text-center">
    <p class="h3 p-3">Welcome to TuHocICT's library!</p>
    <div>
        <table class="table table-bordered table-sm">
            <thead class="thead-light">
            <tr>
                <th>Tiêu đề</th>
                <th>Tác giả</th>
                <th>Nhà xuất bản</th>
                <th>Năm xuất bản</th>
                <th></th>
            </tr>
            </thead>
            <tbody>
            @foreach (var b in Model.Books) {
```

```
                <tr>
                    <td><a href="/book/@b.Id">@b.Title</a></td>
                    <td>@b.Authors</td>
                    <td>@b.Publisher</td>
                    <td>@b.Year</td>
                    <td>
                        <a class="btn btn-info btn-sm" href="/book/@b.Id">Details</a>
                    </td>
                </tr>
            }
            <tr><td colspan="5"><strong>Tổng số sách trong thư viện: @Model.Count</st
            </tbody>
        </table>
    </div>
</div>
```

Đây là một file cshtml thông thường mà bạn đã tiếp xúc nhiều lần. Ở đây chúng ta sử dụng các định dạng css của Bootstrap để giao diện nhìn đàng hoàng hơn một chút.

Giờ đây nếu chạy ứng dụng bạn sẽ thu được kết quả như sau:

Thông tin chi tiết

Chúng ta thực hiện thêm chức năng hiển thị thông tin chi tiết mỗi khi người dùng click vào đường link.

Thêm page Book:

Viết code cho model class trong file Book.cshtml.cs như sau:

```
using BookMan.WebApp.Interface;
using BookMan.WebApp.Model;

using Microsoft.AspNetCore.Mvc.RazorPages;

namespace BookMan.WebApp.Pages {
    public class BookModel : PageModel {
        public enum Action { Detail, Delete, Update, Create }
        private readonly IRepository _repository;
        public BookModel(IRepository repository) => _repository = repository;
        public Action Job { get; private set; }
        public Book Book { get; private set; }

        public void OnGet(int id) {
            Job = Action.Detail;
            Book = _repository.Get(id);
            ViewData["Title"] = Book == null ? "Book not found!" : $"Detail - {Book.T
        }
    }
}
```

Trong file này bạn một lần nữa gặp cách sử dụng DI qua constructor của lớp BookModel.

Bạn cũng gặp phương thức `OnGet(int id)`. Đây là phương thức xử lý sự kiện (handler) khi trang được tải bằng truy vấn Get. Bạn sẽ học chi tiết về xử lý sự kiện trong một bài học riêng.

ViewData[“Title”] chứa chuỗi ký tự sử dụng làm tiêu đề của trang. Giá trị của ViewData[“Title”] được sử dụng trong _Layout.cshtml.

Mở file Book.cshtml và viết code như sau:

```
@page "{id:int?}"
@model BookMan.WebApp.Pages.BookModel
```

```
3.
4.  <div style="margin: auto;" class="border border-light p-3 w-50 shadow">
5.      @switch (Model.Job) {
6.          case BookModel.Action.Detail:
7.              template(readOnly: true, errorMessage: "Không tìm thấy cuốn sách bạn yêu
8.              <a class="btn btn-info btn-block mb-2" href="/">Return</a>
9.              break;
10.         case BookModel.Action.Delete:
11.             break;
12.         case BookModel.Action.Create:
13.             break;
14.         case BookModel.Action.Update:
15.             break;
16.     }
17. </div>
18.
19. @{
20.     void template(bool readOnly = true, string errorMessage = "") {
21.         if (Model.Book == null) {
22.             <p class="h5 text-center text-danger mb-4">@errorMessage</p>
23.             return;
24.         }
25.         <input name="id" type="hidden" value="@Model.Book.Id" />
26.         <p class="h4 text-center mb-4">@Model.Book.Title</p>
27.         <label for="title" class="">Tiêu đề</label>
28.         <input name="title" type="text" class="form-control mb-2" id="title" value="@
29.         <label for="authors" class="">Tác giả</label>
30.         <input name="authors" type="text" class="form-control mb-2" id="authors" valu
31.         <label for="publisher" class="">Nhà xuất bản</label>
32.         <input name="publisher" type="text" class="form-control mb-2" id="publisher"
33.         <label for="year" class="">Năm xuất bản</label>
34.         <input name="year" type="number" class="form-control mb-2" id="year" value="@
35.     }
36. }
```

Khi nhìn dòng đầu tiên bạn có thể thấy hơi lạ với directive `@page "{id:int?}"`. Đây là cách cấu hình ghi đè routing mặc định trong Razor Pages. Lối viết này báo cho Razor Pages rằng trang Book chấp nhận các Url như /book/, /book/1, /book/2, v.v.. Chúng ta cũng sẽ học chi tiết về ghi đè routing trong một bài học riêng.

Trong file này chúng ta tạo một hàm cục bộ template với nhiệm vụ in ra thông tin chi tiết của một cuốn sách (nếu có), hoặc in ra thông báo lỗi. Hàm template sử dụng khả năng chuyển đội ngôn ngữ tự động trong khối code razor. Đọc lại nội dung về code block trong razor nếu bạn không nhớ.

Trang Book có nhiệm vụ xử lý các yêu cầu về từng cuốn sách riêng rẽ, bao gồm xuất thông tin chi tiết, xóa, cập nhật, thêm mới. Tùy vào từng tình huống (căn cứ vào giá trị của property **`BookModel.Action`** trong cấu trúc switch) sẽ in ra giao diện khác nhau.

Giờ đây nếu chạy ứng dụng và click vào các đường link tương ứng trong danh sách bạn sẽ thu được thông tin chi tiết:

Nếu người dùng cố tình để trống phần id trong Url hoặc nhập một Id không tồn tại, trang Book sẽ báo lỗi:

Kết luận

Qua bài thực hành này bạn đã xây dựng được một phần chức năng của một phần mềm. Bạn đã thấy cách vận dụng những kiến thức đã học vào giải quyết một bài toán cụ thể. Qua những phần thực hành tổng hợp sau chúng ta sẽ lần lượt hoàn thiện ứng dụng theo mức độ kiến thức và kỹ năng học được.

Bạn có thể tham khảo video hướng dẫn sau:

Tải mã nguồn solution BookMan (1)

1 file(s) 0.00 KB

TẢI MÃ NGUỒN SOLUTION