

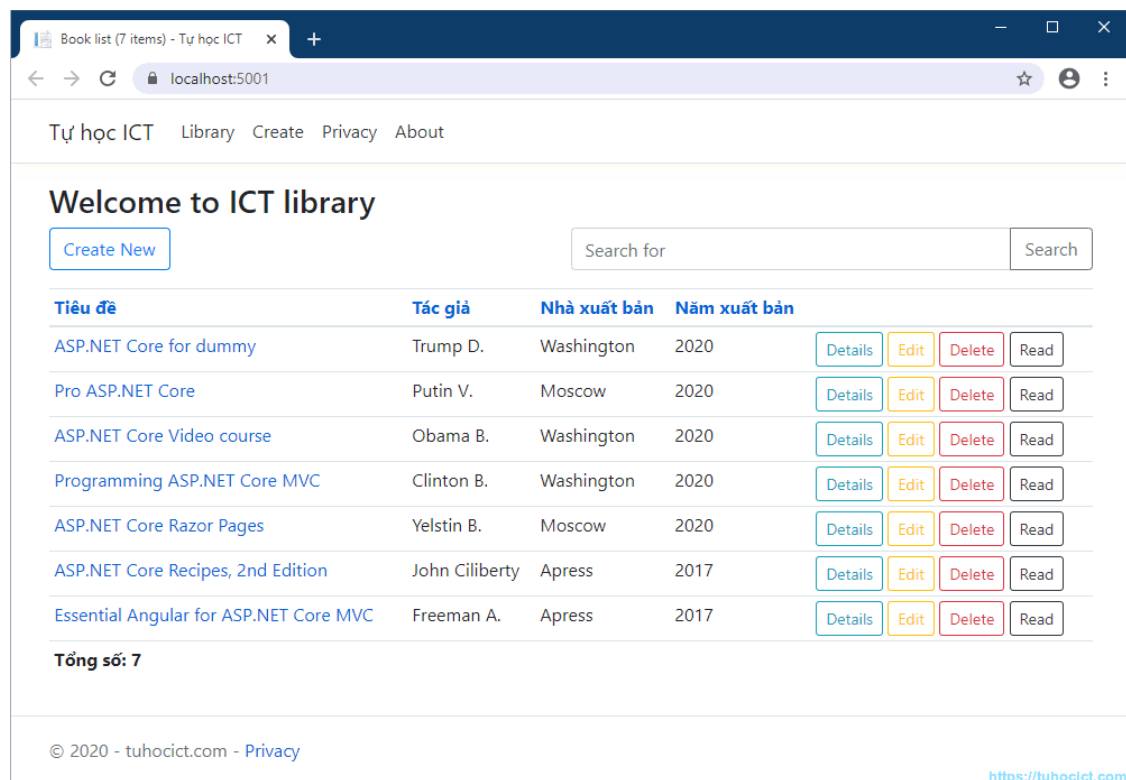
Thực hành (3) tìm kiếm, sắp xếp, phân trang, upload, download

Hướng dẫn tự học lập trình ASP.NET Core toàn tập > Thực hành (3) tìm kiếm, sắp xếp, phân trang, upl...

Trong bài học này chúng ta tiếp tục hoàn thiện ứng dụng với các chức năng thường gặp trên bảng dữ liệu, bao gồm tìm kiếm (Searching), sắp xếp (Sorting), phân trang (Pagination).

Chúng ta cũng hoàn thành khả năng upload file cho chức năng Create và Edit, khả năng download file từ server về client.

Kết thúc bài học này bạn sẽ thu được một ứng dụng MVC hoàn chỉnh.



NỘI DUNG CỦA BÀI [Ấn]

1. Upload file lên server
2. Download file từ server
3. Tìm kiếm
4. Phân trang
5. Sắp xếp
6. Kết luận

Upload file lên server

Upload file từ trình duyệt lên server là chức năng thường gặp ở các ứng dụng. Với chương trình quản lý sách chúng ta đang xây dựng, chức năng này cho phép người dùng upload các file pdf lên một thư mục trên server, đồng thời lưu đường dẫn file vào dữ liệu của cuốn sách tương ứng.

Thông tin về file dữ liệu cho phép người dùng về sau tải file pdf về máy để đọc.

Chức năng upload được sử dụng ở Create và Edit view.

Bước 1. Điều chỉnh lớp Service

Thêm hai phương thức sau vào lớp Service:

```
1. public string GetDataPath(string file) => $"Data\\{file}";
2.
3. public void Upload(Book book, IFormFile file) {
4.     if (file != null) {
5.         var path = GetDataPath(file.FileName);
6.         using var stream = new FileStream(path, FileMode.Create);
7.         file.CopyTo(stream);
8.         book.DataFile = file.FileName;
9.     }
10. }
```

IFormFile là kiểu dữ liệu của ASP.NET Core dùng để lưu thông tin về file client upload lên server. Object của IFormFile được cơ chế model binding tạo ra từ truy vấn HTTP và truyền cho action.

Để lưu lại dữ liệu của IFormFile vào một file trên ổ đĩa (của server), chúng ta tạo một luồng file tương ứng và copy dữ liệu từ IFormFile sang.

Tất cả file upload lên server đều được lưu vào thư mục Data.

Bước 2. Điều chỉnh Create và Edit action

```
1. [HttpPost]
2. public IActionResult Edit(Book book, IFormFile file) {
3.     if (ModelState.IsValid) {
4.         _service.Upload(book, file);
5.         _service.Update(book);
6.         _service.SaveChanges();
7.         return RedirectToAction("Index");
8.     }
9.     return View(book);
10. }
11.
12. [HttpPost]
13. public IActionResult Create(Book book, IFormFile file) {
14.     if (ModelState.IsValid) {
15.         _service.Upload(book, file);
16.         _service.Add(book);
17.         _service.SaveChanges();
18.         return RedirectToAction("Index");
19.     }
20.     return View(book);
21. }
```

Trong khai báo của hai phương thức này chúng ta bổ sung thêm tham số file kiểu IFormFile. Đây là kiểu tham số quy định của ASP.NET Core dành cho file upload từ client lên server. Bạn có thể sử dụng FormFile thay cho IFormFile.

Trong thân phương thức chúng ta gọi Upload của Service để lưu file vào thư mục trên Data trên server.

Bước 3. Kiểm tra các view

Trong bài thực hành (2) bạn đã tách các điều khiển trên form của Edit và Create view vào một file riêng `_Form.cshtml` và sử dụng nó như một partial view.

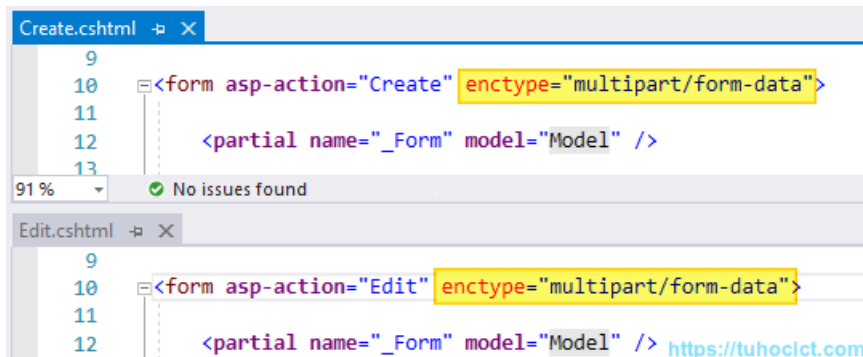
Hãy để ý trong form này đã có sẵn control dành cho upload file:

```
<div class="col-md-6">
  <div class="form-group">
    <label asp-for="Description" class="control-label"></label>
    <textarea asp-for="Description" class="form-control" rows="8"></textarea>
    <span asp-validation-for="Description" class="text-danger small"></span>
  </div>
  <div class="form-group">
    <input type="file" name="file" />
    <input type="hidden" asp-for="DataFile" />
  </div>
  <div asp-validation-summary="ModelOnly" class="text-danger"></div>
</div>
</div>
```

<https://tuhocict.com>

Để ý thêm là trong khai báo form Create và Edit chúng ta đều thiết lập giá trị `enctype="multipart/form-data"`. Đây là yêu cầu bắt buộc đối với form có hỗ trợ file upload.

Nếu không nhớ hãy đọc lại bài học về [cơ chế hoạt động của HTML form](#).



```
Create.cshtml
9
10 <form asp-action="Create" enctype="multipart/form-data">
11
12   <partial name="_Form" model="Model" />
13

Edit.cshtml
9
10 <form asp-action="Edit" enctype="multipart/form-data">
11
12   <partial name="_Form" model="Model" />
```

<https://tuhocict.com>

Chạy thử chương trình với tính năng upload file trong Edit và Create view.

Tất cả file upload lên sẽ nằm trong thư mục Data của dự án. Khi triển khai ứng dụng, file sẽ upload vào thư mục Data của chương trình. Do vậy, khi triển khai bạn sẽ phải tự mình tạo thư mục Data.

Dĩ nhiên bạn có thể thiết lập cấu hình để chọn một thư mục khác làm nơi chứa file upload. Chúng ta sẽ trình bày cách sử dụng file cấu hình sau.

Download file từ server

Tính năng này cho phép người dùng tải file pdf về máy để đọc offline. Đường link tải file sẽ xuất hiện ở trang Index và trang Details.

Cũng lưu ý rằng, chúng ta xây dựng tính năng xử lý download file riêng chứ không sử dụng khả năng cung cấp file tĩnh (StaticFiles middleware) của ASP.NET Core.

Bước 1. Thêm phương thức sau vào lớp Service

```
1. public (Stream, string) Download(Book b) {
2.     var memory = new MemoryStream();
3.     using var stream = new FileStream(GetDataPath(b.DataFile), FileMode.Open);
4.     stream.CopyTo(memory);
5.     memory.Position = 0;
6.     var type = Path.GetExtension(b.DataFile) switch
7.     {
8.         "pdf" => "application/pdf",
9.         "docx" => "application/vnd.ms-word",
10.        "doc" => "application/vnd.ms-word",
11.        "txt" => "text/plain",
12.        _ => "application/pdf"
13.    };
14.     return (memory, type);
15. }
```

Phương thức Download này có sử dụng hai tính năng mới của C# 8 là **switch expression** và **tuple**.

Phương thức này đọc một file vào một object MemoryStream, đồng thời cung cấp **mime type** của loại file tương ứng (như mime type của file pdf là application/pdf).

Do phương thức trả về cặp dữ liệu MemoryStream và string, chúng ta sử dụng type (MemoryStream, string) làm kiểu trả về. Tuple là một kiểu dữ liệu mới của C# 8. Nó rất tiện lợi khi cần nhận/trả nhiều giá trị. Khi sử dụng tuple bạn không cần xây dựng các view model class nhỏ.

Mime type được lựa chọn dựa trên phần đuôi file sử dụng một cấu trúc mới của C# 8: switch expression.

Bước 2. Xây dựng action Read trong BookController

```
1. public IActionResult Read(int id) {
2.     var b = _service.Get(id);
3.     if (b == null) return NotFound();
4.     if (!System.IO.File.Exists(_service.GetDataPath(b.DataFile))) return NotFound();
5.
6.     var (stream, type) = _service.Download(b);
7.     return File(stream, type, b.DataFile);
8. }
```

Trong action này chúng ta gọi phương thức Download của Service. Stream và mime type do Download trả về được sử dụng trong phương thức hỗ trợ File. Phương thức này trả về một object thuộc kiểu FileResult. Object này sẽ được chuyển thành luồng download ở trình duyệt.

Bước 3. Thêm link vào Index và Details view

```
1. <!-- Thêm thẻ này vào Index view -->
2. <a asp-action="Read" asp-route-id="@item.Id" class="btn btn-sm btn-outline-dark">Read</a>
3.
4. <!-- Thêm thẻ này vào Details view -->
5. <a asp-action="Read" asp-route-id="@Model.Id" class="btn btn-sm btn-outline-dark">Read</a>
```

Chạy thử chương trình với tính năng download file.

Bạn có thể dễ ý thấy rằng đây hoàn toàn không phải là cung cấp đường link tới một file tĩnh trên server. Người dùng không hề biết gì về cấu trúc lưu file vật lý trên server.

Tìm kiếm

Tìm kiếm là chức năng không thể thiếu trong các chương trình xử lý dữ liệu.

Trong ứng dụng của chúng ta, chức năng tìm kiếm hoạt động như sau:

1. Người dùng nhập vào một cụm ký tự trên Index view;
2. Chương trình tìm kiếm trong tất cả các property của các object xem có chứa cụm ký tự đó hay không;
3. Trả kết quả về Index view.

Với mỗi object, chương trình sẽ tìm kiếm trong các property: Title, Authors, Publisher, Year, Description.

Bước 1. Điều chỉnh Service

Thêm phương thức sau vào lớp Service:

```
1. public Book[] Get(string search) {
2.     var s = search.ToLower();
3.     return Books.Where(b =>
4.         b.Name.ToLower().Contains(s) ||
5.         b.Authors.ToLower().Contains(s) ||
6.         b.Publisher.ToLower().Contains(s) ||
7.         b.Description.Contains(s) ||
8.         b.Year.ToString() == s
9.     ).ToArray();
10. }
```

Trong phương thức này chúng ta sử dụng LINQ để lọc tất cả dữ liệu chứa cụm ký tự cần tìm trong tất cả các property của mỗi object.

Bước 2. Điều chỉnh BookController

Thêm action sau vào BookController

```
1. public IActionResult Search(string term) {
2.     return View("Index", _service.Get(term));
3. }
```

Bước 3. Thêm form tìm kiếm

Đặt form tìm kiếm sau vào ngay sau nút "Create New".

```
1. <form class="input-group w-50 float-right" asp-action="Search" method="get">
2.     <input type="text" class="form-control" placeholder="Search for" name="term" />
3.     <div class="input-group-append">
4.         <button class="btn btn-outline-secondary" type="submit">Search</button>
5.     </div>
6. </form>
```

```

9 <div class="mb-3">
10 <a asp-action="Create" class="btn btn-outline-primary">Create New</a>
11 <form class="input-group w-50 float-right" asp-action="Search" method="get">
12 <input type="text" class="form-control" placeholder="Search for" name="term" />
13 <div class="input-group-append">
14 <button class="btn btn-outline-secondary" type="submit">Search</button>
15 </div>
16 </form>
17 </div>

```

Đến đây chức năng tìm kiếm đã hoạt động. Bạn có thể chạy thử ứng dụng và nhập cụm "moscow" vào ô tìm kiếm rồi ấn Search (hoặc ấn enter). Kết quả thu được như sau:

The screenshot shows a web browser window with the URL `localhost:5001/Search?term=moscow`. The page displays a search results table for the ICT library. The table has columns: Tiêu đề, Tác giả, Nhà xuất bản, Năm xuất bản, and action buttons (Details, Delete, Edit, Read). Two results are shown, both with 'Moscow' as the publisher.

Tiêu đề	Tác giả	Nhà xuất bản	Năm xuất bản	
Pro ASP.NET Core	Putin V.	Moscow	2020	Details Delete Edit Read
ASP.NET Core Razor Pages	Yelstin B.	Moscow	2020	Details Delete Edit Read

Tổng số: 2

Form tìm kiếm này đang chỉ định sử dụng phương thức GET nên dữ liệu trả về server ở dạng query string `?term=...`

Phân trang

Phân trang (pagination, paging) là một yêu cầu rất quan trọng khi hiển thị số lượng dữ liệu lớn. Bạn không nên hiển thị hàng trăm hàng ngàn dòng dữ liệu trong bảng cùng lúc. Thay vào đó bạn nên hiển thị, ví dụ, 100 dòng dữ liệu đầu tiên. Một khối 100 dòng dữ liệu này được gọi là một trang (page).

Khi người dùng có nhu cầu sẽ hiển thị 100 dòng tiếp theo. Khối lượng dữ liệu này tạo thành trang thứ 2. V.v.

Người dùng có thể lựa chọn nhảy thẳng tới trang thứ n bất kỳ, nhảy về trang đầu tiên, nhảy về trang cuối cùng, nhảy tới trang kế tiếp của trang hiện tại, nhảy tới trang trước của trang hiện tại.

Để thực hiện phân trang bạn cần biết một "thuật toán" nhỏ.

Nếu bạn có tổng cộng N dòng dữ liệu, và trên mỗi trang bạn chỉ muốn hiển thị n dòng, vậy tổng cộng bạn sẽ có $\text{Math.Ceiling}(N/n)$ trang. Trong đó Ceiling là phép làm tròn lên.

Ví dụ, nếu bạn có 21 dòng dữ liệu, và trên mỗi trang bạn cần hiển thị 5 dòng, vậy bạn sẽ có tổng cộng $21/5=4,2$, làm tròn lên là 5 trang.

Giá trị n còn được gọi là *kích thước trang* (size).

Để lấy dữ liệu ở trang thứ p , bạn cần bỏ qua $(p-1) * n$ bản ghi đầu tiên.

Ví dụ, nếu người dùng muốn tải dữ liệu ở trang thứ $p=3$, mỗi trang có $n=5$ dòng, bạn sẽ phải bỏ qua $(3-1)*5 = 10$ dòng đầu tiên, nghĩa là bạn cần bắt đầu lấy từ dòng thứ 11.

Trong C#, nếu sử dụng LINQ, bạn có thể dễ dàng thực hiện hai yêu cầu trên: (1) bỏ qua m dòng đầu tiên trong danh sách sử dụng phương thức `Skip`; (2) lấy đúng n dòng dữ liệu với phương thức `Take`.

Dựa trên thuật toán cơ bản trên người ta có thể áp dụng theo nhiều kiểu khác nhau. Trong bài thực hành này chúng ta vận dụng kiểu đơn giản nhất.

Bước 1. Điều chỉnh lớp Service

Bổ sung phương thức sau vào lớp Service:

```
1. public (Book[] books, int pages, int page) Paging(int page) {
2.     int size = 5;
3.     int pages = (int)Math.Ceiling((double)Books.Count / size);
4.     var books = Books.Skip((page - 1) * size).Take(size).ToArray();
5.     return (books, pages, page);
6. }
```

Phương thức này thực hiện đúng thuật toán chúng ta đã trình bày ở trên để lấy 5 bản ghi ở một trang bất kỳ.

Chúng ta quy định cứng mỗi trang chỉ có 5 bản ghi.

Tổng số trang `pages` tính theo công thức đã trình bày ở trên.

5 bản ghi ở trang thứ `page` được lấy bằng cách kết hợp `Skip` và `Take`.

Bước 2. Điều chỉnh Index action

Vì bảng dữ liệu nằm ở Index view, chúng ta điều chỉnh Index action để sử dụng phân trang:

```
1. public IActionResult Index(int page = 1) {
2.     var model = _service.Paging(page);
3.     ViewData["Pages"] = model.pages;
4.     ViewData["Page"] = model.page;
5.     return View(model.books);
6. }
```

Index giờ đây sẽ nhận thêm giá trị `page` – số thứ tự của trang. Nếu không chỉ định `page`, biến này sẽ nhận giá trị mặc định là 1 (là trường hợp trang Index được tải lần đầu tiên).

Bước 3. Điều chỉnh Index view

Tìm và điều chỉnh dòng footer của bảng dữ liệu như sau:

```
1. <tr>
2.     <td colspan="2">
3.         @{ int pages = ViewData["Pages"]; int p = ViewData["Page"]; }
4.         <form asp-action="Index" method="get">
5.             <input type="submit" value="Go to" /> page <input type="number" max="@pages" />
```

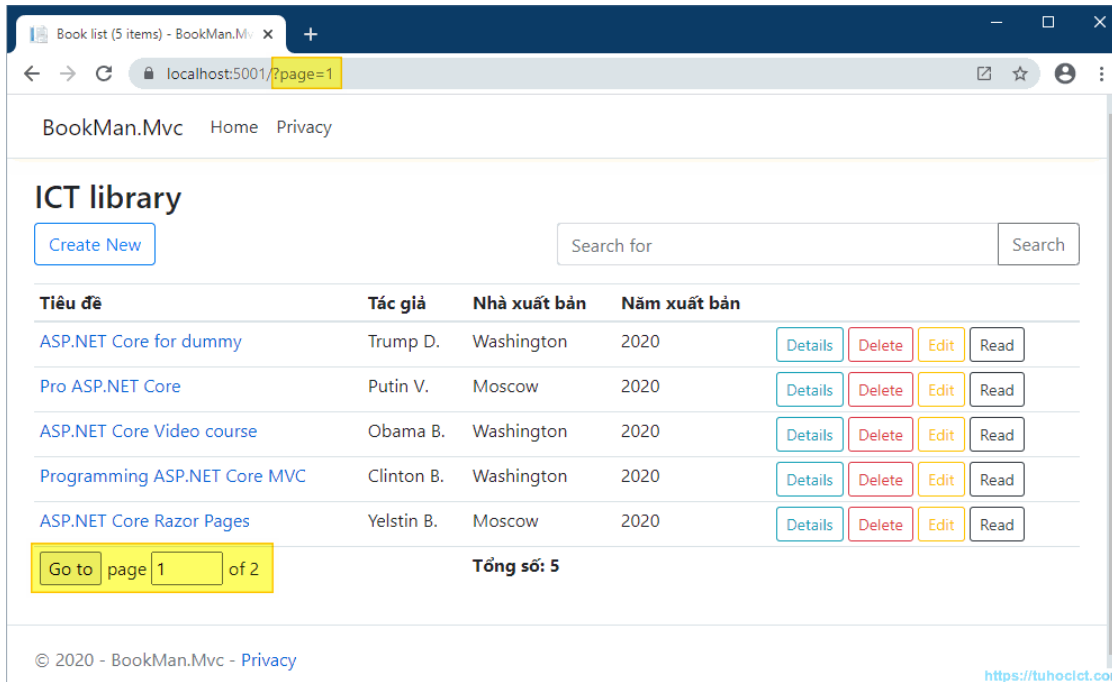
```

6.         </form>
7.     </td>
8.     <td colspan="3"><strong>Tổng số: @Model.Count()</strong></td>
9. </tr>

```

Ở đây chúng ta bổ sung thêm một cột để hiển thị form nhập số trang bên cạnh cột "Tổng số".

Đến đây chức năng phân trang đã hoàn thành. Bạn có thể chạy thử ứng dụng để kiểm tra.



Sắp xếp

Sắp xếp là một yêu cầu quan trọng không kém phân trang hay tìm kiếm. Thực tế nhóm chức năng tìm kiếm – sắp xếp – phân trang thường đi cùng với nhau khi hiển thị dữ liệu dưới dạng bảng.

Việc sắp xếp dữ liệu trong C# thực hiện rất đơn giản nhờ gọi phương thức LINQ `OrderBy()` hoặc `OrderByDescending()`:

```

// sắp xếp tăng dần
var books = Books.OrderBy(b => b.Name).ToArray();
// sắp xếp giảm dần
var books = Books.OrderByDescending(b => b.Name).ToArray();

```

Tuy nhiên, sử dụng LINQ như trên có một vấn đề.

Trong ứng dụng web, bạn thường phải sắp xếp dữ liệu theo yêu cầu của người dùng. Yêu cầu này thường đến dưới dạng query string của URL. Ở mức độ đơn giản nhất, người dùng thường muốn chỉ định sắp xếp dữ liệu theo trường nào và theo thứ tự nào (tăng dần/giảm dần).

Với dữ liệu sách, người dùng có thể muốn sắp xếp theo một trong bốn tiêu chí: tiêu đề, tác giả, năm xuất bản, nhà xuất bản.

Bạn có thể sử dụng cấu trúc if-else hoặc switch để gọi cặp `OrderBy/OrderByDescending` theo đúng tiêu chí người dùng mong muốn.

Tuy nhiên, hãy hình dung nếu dữ liệu của bạn có rất nhiều property làm tiêu chí sắp xếp. Khi này, code của bạn sẽ phình to ra.

Có một giải pháp đơn giản hơn cho vấn đề này. Bạn có thể sử dụng một thư viện class hỗ trợ sử dụng hàm LINQ nhưng với tham số là chuỗi ký tự, thay vì hàm lambda như nguyên bản.

Hãy hình dung bạn có thể gọi

```
var books = Books.OrderBy("Name").ToArray();
var books = Books.OrderBy("Name descending").ToArray();
```

Mặc dù nhìn không quá khác biệt nhưng cách gọi thứ hai có ưu điểm rất lớn trong ứng dụng web: Bạn có thể nhận trực tiếp tiêu chí sắp xếp từ truy vấn (qua string query) và truyền vào phương thức. Không cần if-else hay switch-case nữa. Nó giúp code của bạn đơn giản đi rất nhiều.

Một vấn đề khác cần lưu ý là khi kết hợp sắp xếp với phân trang, bạn có thể lựa chọn (1) sắp xếp trước – phân trang sau, hoặc (2) phân trang trước – sắp xếp sau.

Thứ tự thực hiện khác nhau dẫn đến kết quả rất khác biệt. Phương án 1 sắp xếp dữ liệu toàn cục trước rồi mới phân trang trên dữ liệu đã sắp xếp. Phương án 2 thực hiện phân trang trước rồi mới sắp xếp dữ liệu cục bộ trên từng trang.

Khi đã nắm được các vấn đề lý thuyết, giờ hãy cùng thực hiện.

Bước 1. Cài đặt thư viện `System.Linq.Dynamic.Core`

Đây là thư viện cho phép chúng ta sử dụng các hàm LINQ qua tham số dạng chuỗi, thay vì tham số dạng biểu thức lambda.

Bạn có thể cài đặt qua Nuget Package Manager hoặc Package Manager Console:

```
1. install-package System.Linq.Dynamic.Core
```

Bước 2. Điều chỉnh phương thức Paging của Service

```
1. public (Book[] books, int pages, int page) Paging(int page, string orderBy = "Name", boo
2.     int size = 5;
3.     int pages = (int)Math.Ceiling((double)Books.Count / size);
4.     var books = Books.Skip((page - 1) * size).Take(size).AsQueryable().OrderBy($"{orderBy}
5.     return (books, pages, page);
6. }
```

Ở đây chúng ta xây dựng phương thức kết hợp phân trang và sắp xếp theo trật tự phân trang trước – sắp xếp sau.

Để thực hiện sắp xếp chúng ta cần thêm hai tham số: trường sắp xếp (orderBy) và thứ tự (dsc). Trong đó giá trị true của biến dsc báo hiệu sắp xếp giảm dần. Mặc định chúng ta sẽ sắp xếp tăng dần theo tên sách.

Thư viện dynamic LINQ giúp bạn viết truy vấn theo cách khác với LINQ thông thường:

```
.AsQueryable().OrderBy($"{orderBy} {(dsc ? "descending" : "")}")
```

Bước 3. Cập nhật Index action

```
1. public IActionResult Index(int page = 1, string orderBy = "Name", bool dsc = false) {
2.     var model = _service.Paging(page, orderBy, dsc);
3.     ViewData["Pages"] = model.pages;
4.     ViewData["Page"] = model.page;
5.
6.     ViewData["Name"] = false;
7.     ViewData["Authors"] = false;
8.     ViewData["Publisher"] = false;
9.     ViewData["Year"] = false;
10.
11.     ViewData[orderBy] = !dsc;
12.
13.     return View(model.books);
14. }
```

Việc cập nhật này đảm bảo rằng Index sẽ nhận được đủ tham số từ truy vấn để gọi phương thức Paging.

Để ý khối code

```
ViewData["Name"] = false;
ViewData["Authors"] = false;
ViewData["Publisher"] = false;
ViewData["Year"] = false;

ViewData[orderBy] = !dsc;
```

Đây là một chút tiểu xảo để giúp xây dựng link mà bạn sẽ thấy sau đây.

Bước 4. Điều chỉnh Index view

Tìm đến khối <thead></thead> của bảng dữ liệu và điều chỉnh như sau:

```
<th>
<a asp-action="Index" asp-route-orderBy="Name" asp-route-dsc="@ViewData["Name"]" asp-route-page=
</th>
<th>
<a asp-action="Index" asp-route-orderBy="Authors" asp-route-dsc="@ViewData["Authors"]" asp-route-
</th>
<th>
<a asp-action="Index" asp-route-orderBy="Publisher" asp-route-dsc="@ViewData["Publisher"]" asp-
</th>
<th>
<a asp-action="Index" asp-route-orderBy="Year" asp-route-dsc="@ViewData["Year"]" asp-route-page=
```

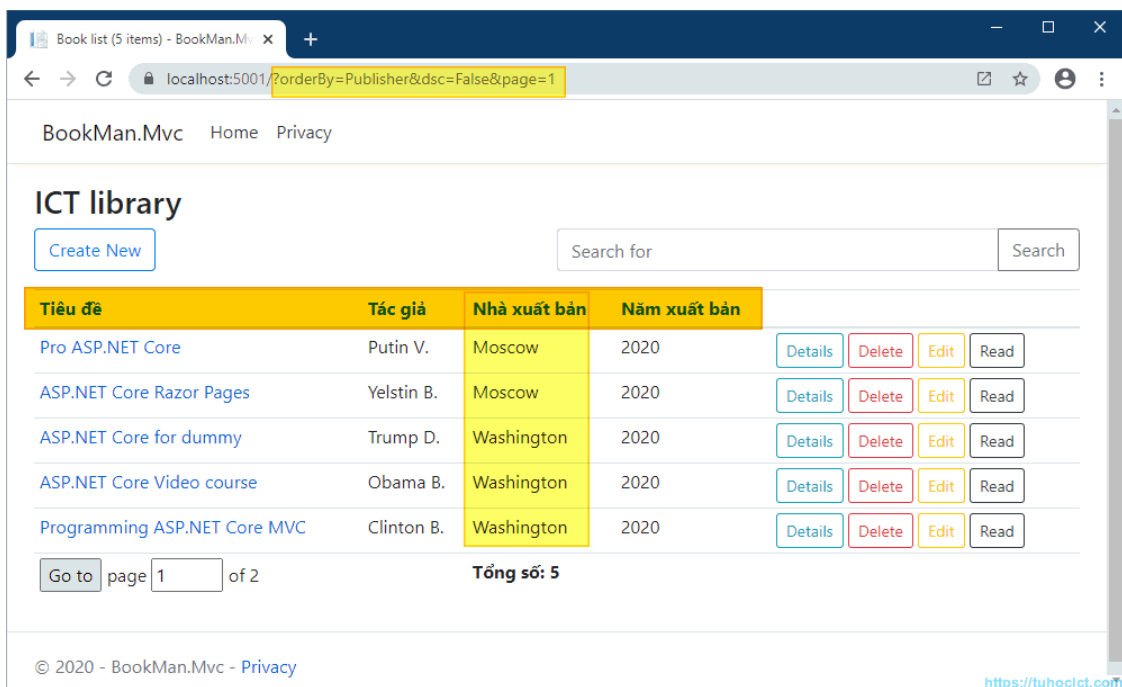
```
<th>  
<th></th>
```

Chúng ta biến mỗi header của bảng thành một đường link gọi về Index action và tạo ra string query với các tham số: orderBy, dsc, page.

Để ý cách dùng tag helper cho thẻ a để tạo ra string query: asp-route-orderBy, asp-route-dsc, asp-route-page.

Thẻ a sau khi đi qua tag helper sẽ chuyển thành HTML ở dạng `/Index?orderBy=Name&dsc=True&page=1`. String query này chứa đủ tham số mà Index cần.

Giờ bạn có thể chạy thử chương trình để kiểm tra xem sắp xếp đã hoạt động hay chưa.



Đến đây xin chúc mừng bạn đã hoàn thành một ứng dụng tương đối trọn vẹn với ASP.NET Core MVC.

Kết luận

Trong bài thực hành này chúng ta đã thực hiện những chức năng cơ bản thường gặp ở một ứng dụng quản lý dữ liệu, bao gồm chức năng tìm kiếm, sắp xếp và phân trang. Ngoài ra chúng ta cũng thực hiện thêm chức năng download/upload file riêng biệt của ứng dụng.

Bạn có thể tải mã nguồn để tham khảo:

https://1drv.ms/u/s!Ar_aj4rIJ2qGkf8rCeoYjbiNZRbg4w?e=vrEPKn

Đến đây bạn đã nắm được đầy đủ cả lý thuyết và thực hành về cách thức hoạt động của ASP.NET Core MVC để xây dựng một ứng dụng quản lý dữ liệu đơn giản.

- + 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!