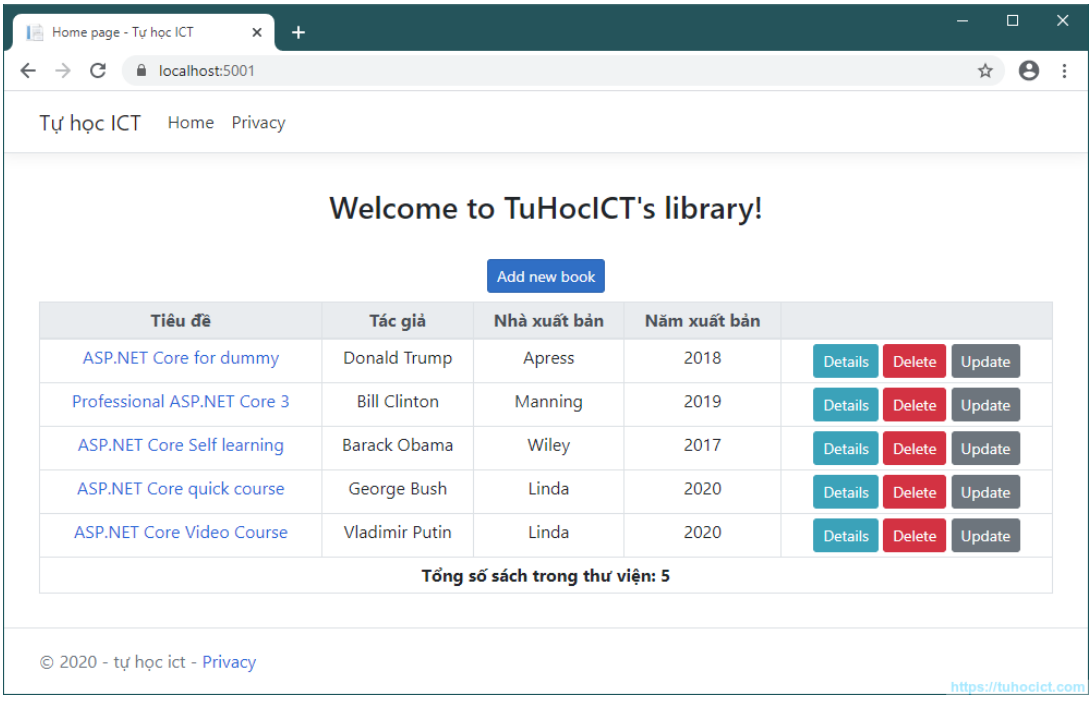
Thực hành (2) CRUD trong Razor Pages (+video)

CRUD – Create, Retrieve, Update, Delete – là nhóm chức năng cơ bản nhất của các ứng dụng quản lý (Line-of-Business, LOB). Trong Razor Pages, bạn rất dễ dàng thực hiện nhóm chức năng CRUD dữ liệu cơ bản.

Trong bài thực hành tổng hợp thứ nhất, bạn đã xây dựng được một phần của ứng dụng quản lý sách điện tử. Cụ thể, bạn đã hiển thị được danh sách tài liệu ở dạng bảng và hiển thị được thông tin chi tiết của từng tài liệu. Đây tương ứng với chức năng R – Retrieve trong CRUD.

Trong bài thực hành này chúng ta tiếp tục xây dựng nốt 3 chức năng xử lý dữ liệu còn lại: Create, Update và Delete. Qua bài thực hành này chúng ta sẽ xem xét cách vận dụng loạt kiến thức đã học trong các bài trước khi xây dựng chương trình Razor Pages.

Kết thúc bài thực hành này bạn sẽ thu được một ứng dụng đơn giản với đủ 4 chức năng CRUD dữ liệu cơ bản.



Video hướng dẫn và mã nguồn ở phần kết luận.

NỘI DUNG CỦA BÀI [Ẩn]

1. Xóa dữ liệu
   1.1. Bước 1. Bổ sung khai báo vào IRepository
   1.2. Bước 2. Bổ sung phương thức Delete vào BookRepository
   1.3. Bước 3. Bổ sung link vào trang Index
   1.4. Bước 4. Điều chỉnh model class BookModel
   1.5. Bước 5. Điều chỉnh Book.cshtml
   1.6. Chạy thử nghiệm
2. Tạo mới
3. Cập nhật

4. Kết luận
   4.1. Tải mã nguồn BookMan (2)

Xóa dữ liệu

Chúng ta bắt đầu với chức năng xóa dữ liệu.

Workflow của chức năng này xuất phát từ trang Index:
=> người dùng click vào nút hoặc đường link tương ứng
=> hiển thị thông tin chi tiết của cuốn sách và đề nghị người dùng xác nhận xóa
-> nếu người dùng xác nhận xóa => xóa dữ liệu => quay trở về trang Index
-> nếu người dùng đổi ý => quay trở về trang Index

Bước 1. Bổ sung khai báo vào IRepository

Thêm khai báo sau vào IRepository:

```
1.  public bool Delete(int id);
```

Bước 2. Bổ sung phương thức Delete vào BookRepository

Thực thi phương thức Delete trong BookRepository:

```
1.  public bool Delete(int id) {
2.      var book = Get(id);
3.      return book != null ? Books.Remove(book) : false;
4.  }
```

Sau bước này BookRepository đã có khả năng xóa bỏ object khỏi tập hợp dữ liệu Books dựa trên giá trị Id của sách.

Bước 3. Bổ sung link vào trang Index

Thêm link sau vào trang Index:

```
1.  <a class="btn btn-danger btn-sm" href="/book/@b.Id?handler=delete">Delete</a>
```

Hãy để ý đến phần `?handler=delete` trong URL /book/@b.Id?handler=delete. Ở đây chúng ta đang chuẩn bị sử dụng named handler (phương thức xử lý sự kiện có định danh).

> Ý tưởng ở đây là trang Book sẽ đảm nhiệm hết tất cả các chức năng liên quan đến xử lý dữ liệu của từng object, bao gồm hiển thị chi tiết, xóa, cập nhật, thêm mới. Mỗi loại chức năng sẽ tương ứng với một truy vấn riêng và thực hiện trong một handler riêng biệt. Do vậy chúng ta phải vận dụng named handler ở đây.

Bước 4. Điều chỉnh model class BookModel

Điều chỉnh class BookModel (Book.cshtml.cs) thêm phương thức OnGetDelete:

```
1.  public void OnGetDelete(int id) {
2.      Job = Action.Delete;
3.      Book = _repository.Get(id);
4.      ViewData["Title"] = Book == null ? "Book not found!" : $"Confim deleting: {Book.T
5.  }
6.
7.  public IActionResult OnGetConfirm(int id) {
8.      _repository.Delete(id);
9.      return new RedirectToPageResult("index");
10. }
```

Lưu ý thêm `using Microsoft.AspNetCore.Mvc;` vào đầu file code để sử dụng được `IActionResult` (sẽ học trong một bài riêng).

Ở đây chúng ta xây dựng hai named handler OnGetDelete và OnGetConfirm.

OnGetDelete được kích hoạt nếu trong URL có tham số handler=delete. OnGetConfirm sẽ chạy nếu URL chứa handler=confirm.

Nhiệm vụ của OnGetDelete là hiển thị giao diện xác nhận xóa. Khi OnGetDelete được gọi, nó thiết lập giá trị cho biến Job, đồng thời tìm kiếm cuốn sách có Id tương ứng. Trang Razor khi thấy giá trị của Job là Delete, nó sẽ hiển thị giao diện xác nhận xóa.

Nếu người dùng xác nhận xóa, OnGetConfirm sẽ được kích hoạt để thực sự xóa object khỏi danh sách, đồng thời điều hướng trở lại trang Index.

Bước 5. Điều chỉnh Book.cshtml

Điều chỉnh case ứng với trường hợp Delete trong trang Book.cshtml như sau:

```
case BookModel.Action.Delete:
    template(readOnly: true, errorMessage: "Không tìm thấy cuốn sách bạn yêu cầu.");
    if (Model.Book != null) {
        <p class="text-danger text-center">Bạn thực sự muốn xóa cuốn sách này?</p>
        <a class="btn btn-danger btn-block mb-2" href="/book/@Model.Book.Id?handler=c
    }
    <a class="btn btn-info btn-block mb-2" href="/">Return</a>
    break;
```

Hãy để ý URL của thẻ <a>: `/book/@Model.Book.Id?handler=confirm`.

URL này vừa cung cấp Id theo route data (@Model.Book.Id), vừa chỉ định named handler xử lý (OnGetConfirm).

Chạy thử nghiệm

Chạy thử nghiệm chương trình. Từ trang Index click nút Delete sẽ chuyển đến trang sau:

Đến đây chức năng xóa dữ liệu đã hoàn thành.

Tạo mới

Bước 1. Bổ sung khai báo sau vào IRepository:

```
1.  public Book Create();
2.  public bool Add(Book book);
```

Bước 2. Bổ sung cặp phương thức sau vào BookRepository để thực thi IRepository:

```
1.  public Book Create() {
2.      var max = Books.Max(b => b.Id);
3.      var book = new Book() { Id = ++max };
4.      return book;
5.  }
6.
7.  public bool Add(Book book) => Books.Add(book);
```

Hai bước này chuẩn bị các phương thức cần dùng cho trang Book.cshtml.

Bước 3. Bổ sung thẻ sau vào trang Index.cshtml:

```
1.  <a class="btn btn-primary btn-sm m-2" href="/book?handler=create">Add new book</a>
```

Bước này đặt đường link mới (Add new book) vào trang Index trước bảng dữ liệu để gọi tới trang Book => phương thức OnGetCreate.

Bước 4. Bổ sung cặp phương thức sau vào BookModel (Book.cshtml.cs):

```
public void OnGetCreate() {
    Job = Action.Create;
    Book = _repository.Create();
    ViewData["Title"] = "Create a new book";
}

public IActionResult OnPostCreate(Book book) {
    _repository.Add(book);
    return new RedirectToPageResult("index");
}
```

Ở đây bạn gặp mô hình cặp GET/POST trong xử lý dữ liệu form. Workflow của mô hình này như sau:
=> Người dùng click vào một link (truy vấn GET) để phát đi yêu cầu
=> phương thức GET hoạt động trả lại form chứa dữ liệu
=> người dùng nhập dữ liệu và submit form (truy vấn POST) để trả lại dữ liệu về server
=> phương thức POST xử lý dữ liệu từ form và trả về trang danh sách (hoàn thành).

Bạn cũng vận dụng kỹ thuật model binding (parameter binding) với object kiểu Book trong OnPostCreate để tránh phải tự trích xuất dữ liệu từ form.

Bước 5. Bổ sung hàm cục bộ sau vào Book.cshtml (trong code block ngay phía dưới hàm void template đã có sẵn):

```
async Task form(string handler = "create") {
    <form method="post" href="?handler=@handler">
        @{template(false); }
        <button type="submit" class="btn btn-primary btn-block mb-2">Save</button>
```

```
            <button type="reset" class="btn btn-secondary btn-block mb-2">Reset</button>
        </form>
        <a class="btn btn-info btn-block mb-2" href="/">Return</a>
    }
```

và thêm lệnh sau vào case tương ứng của Create:

```
    await form();
```

Riêng ở đây, vì một lý do không rõ nào đó Razor Pages lại bắt sử dụng bất đồng bộ, do đó chúng ta phải đánh dấu async và kiểu trả về là Task. Hàm đồng bộ tương tự của nó là void form(string handler = “create”){…}. Ở nơi sử dụng chúng ta phải gọi với từ khóa await.

Đến đây chức năng thêm dữ liệu đã hoàn thành:

Cập nhật

Bước 1. Thêm khai báo sau vào IRepository:

```
1.  public bool Update(Book book);
```

Bước 2. Thực thi phương thức Update trong BookRepository:

```
1.  public bool Update(Book book) {
2.      var b = Get(book.Id);
3.      return b != null ? Books.Remove(b) && Books.Add(book) : false;
4.  }
```

Bước 3. Thêm đường link vào sau nút Delete và Detail:

```
1.  <a class="btn btn-secondary btn-sm" href="/book/@b.Id?handler=update">Update</a>
```

Bước 4. Bổ sung cặp phương thức sau vào BookModel (Book.cshtml.cs):

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

public IActionResult OnPostUpdate(Book book) {
    _repository.Update(book);
    return new RedirectToPageResult("index");
}
```

Cặp phương thức này hoạt động giống hệt như cặp OnGetCreate/OnPostCreate bạn đã gặp ở bên trên.

Bước 5. Thêm case mới cho trường hợp Update:

```
case BookModel.Action.Update:
    await form(handler: "update");
    break;
```

Đến đây chức năng cập nhật đã hoàn thành:

Kết luận

Qua bài thực hành này bạn xây dựng được các chức năng CRUD dữ liệu cơ bản nhất của ứng dụng Razor Pages. Qua đây bạn đã vận dụng tương đối đầy đủ các kỹ thuật của Razor, bao gồm phương thức xử lý truy vấn, truy xuất dữ liệu từ truy vấn, model binding.

Dĩ nhiên, các chức năng này còn rất hạn chế và chưa thực tế. Ví dụ, bạn chưa hề kiểm soát dữ liệu người dùng, chưa lưu trữ được dữ liệu, chưa upload/download file sách.

Một vấn đề khá "nghiêm trọng" nữa là mặc dù ứng dụng hoạt động nhưng cách thức chúng ta xây dựng các chức năng CRUD lại không thực sự "tiêu chuẩn". Để thực hành các tính năng của ASP.NET Core Razor Pages chúng ta đã sử dụng một cách lập trình khác so với cách làm tiêu chuẩn của Razor Pages: Thay vì tạo page riêng rẽ cho từng chức năng, chúng ta dồn tất cả vào trang Book.

Nhẽ ra, đối với chức năng cập nhật, chúng ta cần tạo page Edit riêng; đối với chức năng thêm mới, chúng ta phải tạo trang Create riêng. Nghĩa là Razor Pages yêu cầu mỗi chức năng CRUD nên được xây dựng trong một page riêng với cặp handler GET-POST.

Cuối cùng, ứng dụng của chúng ta mặc dù hiển thị được bảng dữ liệu nhưng chưa thể thực hiện các chức năng đi cùng bảng dữ liệu như tìm kiếm, sắp xếp, phân trang.

Tất cả những hạn chế trên sẽ được giải quyết trong bài thực hành tổng hợp tiếp theo.

Bạn có thể xem video hướng dẫn này để tham khảo.

Tải mã nguồn BookMan (2)

1 file(s) 0.00 KB

TẢI MÃ NGUỒN

+ 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!