IT2030 Assignment 13 Matt Brown

**BookListViewModel.cs**

namespace Bookstore.Models

{

public class BookListViewModel

{

public IEnumerable<Book> Books { get; set; }

public RouteDictionary CurrentRoute { get; set; }

public int TotalPages { get; set; }

// data for filter drop-downs - one hardcoded

public IEnumerable<Author> Authors { get; set; }

public IEnumerable<Genre> Genres { get; set; }

public Dictionary<string, string> Prices =>

new Dictionary<string, string> {

{ "under7", "Under $7" },

{ "7to14", "$7 to $14" },

{ "over14", "Over $14" }

};

public int[] PageSizes => new int[] { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };

}

}

**BookController.cs**

namespace Bookstore.Controllers

{

public class BookController : Controller

{

private BookstoreUnitOfWork data { get; set; }

public BookController(BookstoreContext ctx) => data = new BookstoreUnitOfWork(ctx);

public RedirectToActionResult Index() => RedirectToAction("List");

// dto has properties for the paging, sorting, and filtering route segments defined in the Startup.cs file

public ViewResult List(BooksGridDTO values)

{

// get grid builder, which loads route segment values and stores them in session

var builder = new BooksGridBuilder(HttpContext.Session, values,

defaultSortField: nameof(Book.Title));

// create a BookQueryOptions object to build a query expression for a page of data

var options = new BookQueryOptions {

Includes = "BookAuthors.Author, Genre",

OrderByDirection = builder.CurrentRoute.SortDirection,

PageNumber = builder.CurrentRoute.PageNumber,

PageSize = builder.CurrentRoute.PageSize

};

// call the SortFilter() method of the BookQueryOptions object and pass it the builder

// object. It uses the route information and the properties of the builder object to

// add sort and filter options to the query expression.

options.SortFilter(builder);

// create view model and add page of book data, data for drop-downs,

// the current route, and the total number of pages.

var vm = new BookListViewModel {

Books = data.Books.List(options),

Authors = data.Authors.List(new QueryOptions<Author> {

OrderBy = a => a.FirstName }),

Genres = data.Genres.List(new QueryOptions<Genre> {

OrderBy = g => g.Name }),

CurrentRoute = builder.CurrentRoute,

TotalPages = builder.GetTotalPages(data.Books.Count)

};

// pass view model to view

return View(vm);

}

public ViewResult Details(int id)

{

var book = data.Books.Get(new QueryOptions<Book> {

Includes = "BookAuthors.Author, Genre",

Where = b => b.BookId == id

});

return View(book);

}

[HttpPost]

public RedirectToActionResult Filter(string[] filter, bool clear = false)

{

// get current route segments from session

var builder = new BooksGridBuilder(HttpContext.Session);

// clear or update filter route segment values. If update, get author data

// from database so can add author name slug to author filter value.

if (clear) {

builder.ClearFilterSegments();

}

else {

var author = data.Authors.Get(filter[0].ToInt());

builder.CurrentRoute.PageNumber = 1;

builder.LoadFilterSegments(filter, author);

}

// save route data back to session and redirect to Book/List action method,

// passing dictionary of route segment values to build URL

builder.SaveRouteSegments();

return RedirectToAction("List", builder.CurrentRoute);

}

[HttpPost]

public RedirectToActionResult PageSize(int pagesize)

{

var builder = new BooksGridBuilder(HttpContext.Session);

builder.CurrentRoute.PageSize = pagesize;

builder.SaveRouteSegments();

return RedirectToAction("List", builder.CurrentRoute);

}

}

}

**List.cshtml**

@model BookListViewModel

@{

ViewData["Title"] = " | Book Catalog";

// aliases to make paging, sorting, and filtering code shorter

RouteDictionary current = Model.CurrentRoute;

RouteDictionary routes = Model.CurrentRoute.Clone();

}

<h1>Book Catalog</h1>

<div class="row">

<div class="col-9">

@\* filter form \*@

<form asp-action="Filter" method="post" class="form-inline">

<label>Author: </label>

<select name="filter" class="form-control m-2"

asp-items="@(new SelectList(

Model.Authors, "AuthorId", "FullName", current.AuthorFilter))">

<option value="@BooksGridDTO.DefaultFilter">All</option>

</select>

<label>Genre: </label>

<select name="filter" class="form-control m-2"

asp-items="@(new SelectList(

Model.Genres, "GenreId", "Name", current.GenreFilter))">

<option value="@BooksGridDTO.DefaultFilter">All</option>

</select>

<label>Price: </label>

<select name="filter" class="form-control m-2"

asp-items="@(new SelectList(

Model.Prices, "Key", "Value", current.PriceFilter))">

<option value="@BooksGridDTO.DefaultFilter">All</option>

</select>

<button type="submit" class="btn btn-primary mr-2">Filter</button>

<button type="submit" class="btn btn-primary"

name="clear" value="true">

Clear

</button>

</form>

</div>

<div class="col">

@\* page-size form \*@

<form asp-action="PageSize" method="post" class="form-inline">

<label>Page Size: </label>

<select name="pagesize" class="form-control m-2"

asp-items="@(new SelectList(Model.PageSizes, current.PageSize))">

</select>

<button type="submit" class="btn btn-primary mr-2">Save</button>

</form>

</div>

</div>

@\* add to cart form \*@

<form asp-action="Add" asp-controller="Cart" method="post">

<table class="table table-bordered table-striped table-sm">

@\* for each sortable table column, update the route dictionary with the field to sort by, and then

use that route dictionary and the current route to create route segments for sorting link \*@

<thead class="thead-dark">

<tr>

<th>

@{ routes.SetSortAndDirection(nameof(Book.Title), current); }

<a asp-action="List" asp-all-route-data="@routes"

class="text-white">Title</a>

</th>

<th>Author(s)</th>

<th>

@{ routes.SetSortAndDirection(nameof(Genre), current); }

<a asp-action="List" asp-all-route-data="@routes"

class="text-white">Genre</a>

</th>

<th>

@{ routes.SetSortAndDirection(nameof(Book.Price), current); }

<a asp-action="List" asp-all-route-data="@routes"

class="text-white">Price</a>

</th>

<th></th>

</tr>

</thead>

<tbody>

@foreach (Book book in Model.Books)

{

<tr>

<td>

<a asp-action="Details" asp-route-id="@book.BookId"

asp-route-slug="@book.Title.Slug()">

@book.Title

</a>

</td>

<td>

@foreach (var ba in book.BookAuthors)

{

<p>

<a asp-action="Details" asp-controller="Author"

asp-route-id="@ba.AuthorId"

asp-route-slug="@ba.Author.FullName.Slug()">

@ba.Author.FullName

</a>

</p>

}

</td>

<td>@book.Genre?.Name</td>

<td>@book.Price.ToString("c")</td>

<td>

<button type="submit" name="id" value="@book.BookId"

class="btn btn-primary">

Add To Cart

</button>

</td>

</tr>

}

</tbody>

</table>

</form>

@\* paging links \*@

@{

@\* Reset the route dictionary to the current route values. This is necessary

to clear the changes made while creating the sorting links above. Clone so

don't overwrite current route segment values. \*@

routes = Model.CurrentRoute.Clone();

@\* update route dictionary with page number in 'i' and then use

updated route dictionary to create route segments for paging link.\*@

for (int i = 1; i <= Model.TotalPages; i++)

{

routes.PageNumber = i;

<a asp-action="List" asp-all-route-data="@routes"

class="btn btn-outline-primary @Nav.Active(i, current.PageNumber)">@i</a>

}

}