src/Library.Web/Program.cs  
  
using Library.Web.Data;  
using Library.Web.Repositories;  
using Microsoft.EntityFrameworkCore;  
  
var builder = WebApplication.CreateBuilder(args);  
  
// DbContext  
builder.Services.AddDbContext<LibraryContext>(options =>  
 options.UseSqlServer(builder.Configuration.GetConnectionString("DefaultConnection")));  
  
// MVC  
builder.Services.AddControllersWithViews();  
  
// Repositories + Unit of Work  
builder.Services.AddScoped(typeof(IGenericRepository<>), typeof(GenericRepository<>));  
builder.Services.AddScoped<IBookRepository, BookRepository>();  
builder.Services.AddScoped<IAuthorRepository, AuthorRepository>();  
builder.Services.AddScoped<IGenreRepository, GenreRepository>();  
builder.Services.AddScoped<IUnitOfWork, UnitOfWork>();  
  
var app = builder.Build();  
  
if (!app.Environment.IsDevelopment())  
{  
 app.UseExceptionHandler("/Home/Error");  
 app.UseHsts();  
}  
  
app.UseHttpsRedirection();  
app.UseStaticFiles();  
  
app.UseRouting();  
  
app.MapControllerRoute(  
 name: "default",  
 pattern: "{controller=Books}/{action=Index}/{id?}");  
  
app.Run();  
  
===== src/Library.Web/appsettings.json =====  
  
{  
 "ConnectionStrings": {  
 "DefaultConnection": "Server=YOUR\_SQL\_SERVER;Database=AdvancedLibraryDb;Trusted\_Connection=True;MultipleActiveResultSets=true;TrustServerCertificate=True"  
 },  
 "Logging": {  
 "LogLevel": {  
 "Default": "Information",  
 "Microsoft.AspNetCore": "Warning"  
 }  
 },  
 "AllowedHosts": "\*"  
}

src/Library.Web/Data/LibraryContext.cs   
  
using Library.Web.Models;  
using Microsoft.EntityFrameworkCore;  
  
namespace Library.Web.Data  
{  
 public class LibraryContext : DbContext  
 {  
 public LibraryContext(DbContextOptions<LibraryContext> options) : base(options) {}  
  
 public DbSet<Book> Books => Set<Book>();  
 public DbSet<Author> Authors => Set<Author>();  
 public DbSet<Genre> Genres => Set<Genre>();  
 public DbSet<BookAuthor> BookAuthors => Set<BookAuthor>();  
  
 protected override void OnModelCreating(ModelBuilder modelBuilder)  
 {  
 base.OnModelCreating(modelBuilder);  
  
 modelBuilder.Entity<BookAuthor>()  
 .HasKey(ba => new { ba.BookId, ba.AuthorId });  
  
 modelBuilder.Entity<BookAuthor>()  
 .HasOne(ba => ba.Book)  
 .WithMany(b => b.BookAuthors)  
 .HasForeignKey(ba => ba.BookId);  
  
 modelBuilder.Entity<BookAuthor>()  
 .HasOne(ba => ba.Author)  
 .WithMany(a => a.BookAuthors)  
 .HasForeignKey(ba => ba.AuthorId);  
  
 modelBuilder.Entity<Book>()  
 .HasOne(b => b.Genre)  
 .WithMany(g => g.Books)  
 .HasForeignKey(b => b.GenreId)  
 .OnDelete(DeleteBehavior.Restrict);  
 }  
 }  
}  
  
src/Library.Web/Models/Author.cs   
  
using System.ComponentModel.DataAnnotations;  
  
namespace Library.Web.Models  
{  
 public class Author  
 {  
 public int Id { get; set; }  
  
 [Required, StringLength(100)]  
 public string Name { get; set; } = string.Empty;  
  
 public ICollection<BookAuthor> BookAuthors { get; set; } = new List<BookAuthor>();  
 }  
}

src/Library.Web/Models/Genre.cs   
  
using System.ComponentModel.DataAnnotations;  
  
namespace Library.Web.Models  
{  
 public class Genre  
 {  
 public int Id { get; set; }  
  
 [Required, StringLength(60)]  
 public string Name { get; set; } = string.Empty;  
  
 public ICollection<Book> Books { get; set; } = new List<Book>();  
 }  
}  
  
 src/Library.Web/Models/Book.cs   
  
using System.ComponentModel.DataAnnotations;  
  
namespace Library.Web.Models  
{  
 public class Book  
 {  
 public int Id { get; set; }  
 public string Title { get; set; } = string.Empty;  
 public int? PublishedYear { get; set; }  
 public int GenreId { get; set; }  
 public Genre? Genre { get; set; }  
  
 public ICollection<BookAuthor> BookAuthors { get; set; } = new List<BookAuthor>();  
 }  
}  
  
 src/Library.Web/Models/BookAuthor.cs   
  
namespace Library.Web.Models  
{  
 public class BookAuthor  
 {  
 public int BookId { get; set; }  
 public Book Book { get; set; } = default!;  
  
 public int AuthorId { get; set; }  
 public Author Author { get; set; } = default!;  
 }  
}  
  
src/Library.Web/Repositories/IGenericRepository.cs  
  
using System.Linq.Expressions;  
  
namespace Library.Web.Repositories  
{  
 public interface IGenericRepository<T> where T : class  
 {  
 Task<T?> GetByIdAsync(object id);  
 Task<IEnumerable<T>> GetAllAsync(  
 Expression<Func<T, bool>>? filter = null,  
 Func<IQueryable<T>, IOrderedQueryable<T>>? orderBy = null,  
 string includeProperties = "",  
 int? skip = null,  
 int? take = null);  
 Task AddAsync(T entity);  
 void Update(T entity);  
 void Delete(T entity);  
 Task<int> CountAsync(Expression<Func<T, bool>>? filter = null);  
 }  
}  
  
src/Library.Web/Repositories/GenericRepository.cs

using Microsoft.EntityFrameworkCore;  
using System.Linq.Expressions;  
using Library.Web.Data;  
  
namespace Library.Web.Repositories  
{  
 public class GenericRepository<T> : IGenericRepository<T> where T : class  
 {  
 protected readonly LibraryContext \_context;  
 protected readonly DbSet<T> \_db;  
  
 public GenericRepository(LibraryContext context)  
 {  
 \_context = context;  
 \_db = \_context.Set<T>();  
 }  
  
 public async Task AddAsync(T entity) => await \_db.AddAsync(entity);  
  
 public void Delete(T entity) => \_db.Remove(entity);  
  
 public async Task<IEnumerable<T>> GetAllAsync(Expression<Func<T, bool>>? filter = null,  
 Func<IQueryable<T>, IOrderedQueryable<T>>? orderBy = null, string includeProperties = "",  
 int? skip = null, int? take = null)  
 {  
 IQueryable<T> query = \_db.AsQueryable();  
  
 if (filter != null) query = query.Where(filter);  
  
 foreach (var includeProp in includeProperties.Split(new[] { ',' }, StringSplitOptions.RemoveEmptyEntries))  
 query = query.Include(includeProp);  
  
 if (orderBy != null) query = orderBy(query);  
  
 if (skip.HasValue) query = query.Skip(skip.Value);  
 if (take.HasValue) query = query.Take(take.Value);  
  
 return await query.AsNoTracking().ToListAsync();  
 }  
  
 public async Task<T?> GetByIdAsync(object id) => await \_db.FindAsync(id);  
  
 public void Update(T entity) => \_db.Update(entity);  
  
 public async Task<int> CountAsync(Expression<Func<T, bool>>? filter = null)  
 {  
 if (filter != null) return await \_db.CountAsync(filter);  
 return await \_db.CountAsync();  
 }  
 }  
}  
src/Library.Web/Repositories/IBookRepository.cs   
  
using Library.Web.Models;  
  
namespace Library.Web.Repositories  
{  
 public interface IBookRepository : IGenericRepository<Book>  
 {  
 Task<Book?> GetDetailAsync(int id);  
 }  
}

src/Library.Web/Repositories/BookRepository.cs   
  
using Library.Web.Data;  
using Library.Web.Models;  
using Microsoft.EntityFrameworkCore;  
  
namespace Library.Web.Repositories  
{  
 public class BookRepository : GenericRepository<Book>, IBookRepository  
 {  
 public BookRepository(LibraryContext context) : base(context) { }  
  
 public async Task<Book?> GetDetailAsync(int id)  
 {  
 return await \_context.Books  
 .Include(b => b.Genre)  
 .Include(b => b.BookAuthors).ThenInclude(ba => ba.Author)  
 .AsNoTracking()  
 .FirstOrDefaultAsync(b => b.Id == id);  
 }  
 }  
}  
  
src/Library.Web/Repositories/IAuthorRepository.cs   
  
using Library.Web.Models;  
  
namespace Library.Web.Repositories  
{  
 public interface IAuthorRepository : IGenericRepository<Author> { }  
}  
  
===== src/Library.Web/Repositories/AuthorRepository.cs =====  
  
using Library.Web.Data;  
using Library.Web.Models;  
  
namespace Library.Web.Repositories  
{  
 public class AuthorRepository : GenericRepository<Author>, IAuthorRepository  
 {  
 public AuthorRepository(LibraryContext context) : base(context) { }  
 }  
}  
 src/Library.Web/Repositories/IGenreRepository.cs   
  
using Library.Web.Models;  
  
namespace Library.Web.Repositories  
{  
 public interface IGenreRepository : IGenericRepository<Genre> { }  
}  
  
 src/Library.Web/Repositories/GenreRepository.cs  
  
using Library.Web.Data;  
using Library.Web.Models;  
  
namespace Library.Web.Repositories  
{  
 public class GenreRepository : GenericRepository<Genre>, IGenreRepository  
 {  
 public GenreRepository(LibraryContext context) : base(context) { }  
 }  
}  
  
 src/Library.Web/Repositories/IUnitOfWork.cs   
  
using System.Threading.Tasks;  
  
namespace Library.Web.Repositories  
{  
 public interface IUnitOfWork  
 {  
 IBookRepository Books { get; }  
 IAuthorRepository Authors { get; }  
 IGenreRepository Genres { get; }  
 Task<int> SaveAsync();  
 }  
}  
src/Library.Web/Repositories/UnitOfWork.cs   
  
using Library.Web.Data;  
  
namespace Library.Web.Repositories  
{  
 public class UnitOfWork : IUnitOfWork  
 {  
 private readonly LibraryContext \_context;  
 public IBookRepository Books { get; }  
 public IAuthorRepository Authors { get; }  
 public IGenreRepository Genres { get; }  
  
 public UnitOfWork(LibraryContext context,  
 IBookRepository bookRepository,  
 IAuthorRepository authorRepository,  
 IGenreRepository genreRepository)  
 {  
 \_context = context;  
 Books = bookRepository;  
 Authors = authorRepository;  
 Genres = genreRepository;  
 }  
  
 public async Task<int> SaveAsync() => await \_context.SaveChangesAsync();  
 }  
}

src/Library.Web/Controllers/BooksController.cs  
  
using Library.Web.Models;  
using Library.Web.Repositories;  
using Microsoft.AspNetCore.Mvc;  
using Microsoft.EntityFrameworkCore;  
  
namespace Library.Web.Controllers  
{  
 public class BooksController : Controller  
 {  
 private readonly IUnitOfWork \_uow;  
 private readonly ILogger<BooksController> \_logger;  
  
 public BooksController(IUnitOfWork uow, ILogger<BooksController> logger)  
 {  
 \_uow = uow;  
 \_logger = logger;  
 }  
  
 // GET: /Books  
 public async Task<IActionResult> Index(string? search, int page = 1, int pageSize = 10)  
 {  
 search = search?.Trim();  
 var filter = string.IsNullOrWhiteSpace(search) ? null : (b => b.Title.Contains(search));  
 var total = await \_uow.Books.CountAsync(filter);  
 var skip = (page - 1) \* pageSize;  
  
 var list = await \_uow.Books.GetAllAsync(  
 filter: filter,  
 orderBy: q => q.OrderBy(b => b.Title),  
 includeProperties: "Genre,BookAuthors.Author",  
 skip: skip, take: pageSize  
 );  
  
 ViewBag.Total = total;  
 ViewBag.Page = page;  
 ViewBag.PageSize = pageSize;  
 ViewBag.Search = search;  
  
 if (Request.Headers["X-Requested-With"] == "XMLHttpRequest")  
 return PartialView("\_List", list);  
  
 return View(list);  
 }  
  
 // GET: /Books/Create  
 public async Task<IActionResult> Create()  
 {  
 ViewBag.Authors = await \_uow.Authors.GetAllAsync(orderBy: q => q.OrderBy(a => a.Name));  
 ViewBag.Genres = await \_uow.Genres.GetAllAsync(orderBy: q => q.OrderBy(g => g.Name));  
 return PartialView("\_CreateOrEdit", new Book());  
 }  
  
 // POST: /Books/Create (AJAX)  
 [HttpPost]  
 public async Task<IActionResult> Create([FromBody] BookDto dto)  
 {  
 try  
 {  
 if (!ModelState.IsValid) return BadRequest("Invalid payload");  
  
 var book = new Book { Title = dto.Title, GenreId = dto.GenreId, PublishedYear = dto.PublishedYear };  
 await \_uow.Books.AddAsync(book);  
 await \_uow.SaveAsync();  
  
 // authors  
 if (dto.AuthorIds != null)  
 {  
 foreach (var aid in dto.AuthorIds.Distinct())  
 {  
 book.BookAuthors.Add(new BookAuthor { BookId = book.Id, AuthorId = aid });  
 }  
 await \_uow.SaveAsync();  
 }  
  
 return Ok(new { success = true, id = book.Id });  
 }  
 catch (Exception ex)  
 {  
 \_logger.LogError(ex, "Error creating book");  
 return Problem(title: "Create failed", detail: ex.Message, statusCode: 500);  
 }  
 }  
  
 // GET: /Books/Edit/5  
 public async Task<IActionResult> Edit(int id)  
 {  
 var book = await \_uow.Books.GetDetailAsync(id);  
 if (book == null) return NotFound();  
  
 ViewBag.Authors = await \_uow.Authors.GetAllAsync(orderBy: q => q.OrderBy(a => a.Name));  
 ViewBag.Genres = await \_uow.Genres.GetAllAsync(orderBy: q => q.OrderBy(g => g.Name));  
 return PartialView("\_CreateOrEdit", book);  
 }  
  
 // PUT: /Books/Edit/5 (AJAX)  
 [HttpPut]  
 public async Task<IActionResult> Edit(int id, [FromBody] BookDto dto)  
 {  
 try  
 {  
 var existing = await \_uow.Books.GetDetailAsync(id);  
 if (existing == null) return NotFound();  
  
 existing.Title = dto.Title;  
 existing.GenreId = dto.GenreId;  
 existing.PublishedYear = dto.PublishedYear;  
  
 // reset authors  
 existing.BookAuthors.Clear();  
 if (dto.AuthorIds != null)  
 {  
 foreach (var aid in dto.AuthorIds.Distinct())  
 existing.BookAuthors.Add(new BookAuthor { BookId = existing.Id, AuthorId = aid });  
 }  
  
 \_uow.Books.Update(existing);  
 await \_uow.SaveAsync();  
 return Ok(new { success = true });  
 }  
 catch (Exception ex)  
 {  
 \_logger.LogError(ex, "Error updating book");  
 return Problem(title: "Update failed", detail: ex.Message, statusCode: 500);  
 }  
 }  
  
 // DELETE: /Books/Delete/5 (AJAX)  
 [HttpDelete]  
 public async Task<IActionResult> Delete(int id)  
 {  
 try  
 {  
 var book = await \_uow.Books.GetByIdAsync(id);  
 if (book == null) return NotFound();  
 \_uow.Books.Delete(book);  
 await \_uow.SaveAsync();  
 return Ok(new { success = true });  
 }  
 catch (DbUpdateException dbex)  
 {  
 \_logger.LogWarning(dbex, "Delete failed due to FK constraints");  
 return Problem(title: "Delete failed", detail: "Cannot delete due to related records.", statusCode: 409);  
 }  
 catch (Exception ex)  
 {  
 \_logger.LogError(ex, "Error deleting book");  
 return Problem(title: "Delete failed", detail: ex.Message, statusCode: 500);  
 }  
 }  
 }  
  
 public class BookDto  
 {  
 public string Title { get; set; } = string.Empty;  
 public int GenreId { get; set; }  
 public int? PublishedYear { get; set; }  
 public List<int>? AuthorIds { get; set; }  
 }  
}

src/Library.Web/Controllers/AuthorsController.cs   
  
using Library.Web.Models;  
using Library.Web.Repositories;  
using Microsoft.AspNetCore.Mvc;  
  
namespace Library.Web.Controllers  
{  
 public class AuthorsController : Controller  
 {  
 private readonly IUnitOfWork \_uow;  
 private readonly ILogger<AuthorsController> \_logger;  
  
 public AuthorsController(IUnitOfWork uow, ILogger<AuthorsController> logger)  
 {  
 \_uow = uow; \_logger = logger;  
 }  
  
 public async Task<IActionResult> Index()  
 {  
 var authors = await \_uow.Authors.GetAllAsync(orderBy: q => q.OrderBy(a => a.Name));  
 return View(authors);  
 }  
  
 [HttpPost]  
 public async Task<IActionResult> Create([FromForm] string name)  
 {  
 try  
 {  
 if (string.IsNullOrWhiteSpace(name)) return BadRequest("Name required");  
 await \_uow.Authors.AddAsync(new Author { Name = name.Trim() });  
 await \_uow.SaveAsync();  
 return RedirectToAction(nameof(Index));  
 }  
 catch (Exception ex) { \_logger.LogError(ex, "Create author"); return Problem("Create failed"); }  
 }  
 }  
}  
  
src/Library.Web/Controllers/GenresController.cs   
  
using Library.Web.Models;  
using Library.Web.Repositories;  
using Microsoft.AspNetCore.Mvc;  
  
namespace Library.Web.Controllers  
{  
 public class GenresController : Controller  
 {  
 private readonly IUnitOfWork \_uow;  
 private readonly ILogger<GenresController> \_logger;  
  
 public GenresController(IUnitOfWork uow, ILogger<GenresController> logger)  
 {  
 \_uow = uow; \_logger = logger;  
 }  
  
 public async Task<IActionResult> Index()  
 {  
 var genres = await \_uow.Genres.GetAllAsync(orderBy: q => q.OrderBy(g => g.Name));  
 return View(genres);  
 }  
  
 [HttpPost]  
 public async Task<IActionResult> Create([FromForm] string name)  
 {  
 try  
 {  
 if (string.IsNullOrWhiteSpace(name)) return BadRequest("Name required");  
 await \_uow.Genres.AddAsync(new Genre { Name = name.Trim() });  
 await \_uow.SaveAsync();  
 return RedirectToAction(nameof(Index));  
 }  
 catch (Exception ex) { \_logger.LogError(ex, "Create genre"); return Problem("Create failed"); }  
 }  
 }  
}  
 src/Library.Web/Views/Shared/\_Layout.cshtml  
  
<!DOCTYPE html>  
<html>  
<head>  
 <meta charset="utf-8" />  
 <meta name="viewport" content="width=device-width, initial-scale=1.0" />  
 <title>@ViewData["Title"] - Advanced Library</title>  
 <link rel="stylesheet" href="~/css/site.css" />  
 <script src="https://code.jquery.com/jquery-3.7.1.min.js"></script>  
 <script src="~/js/library.js"></script>  
</head>  
<body>  
 <div class="nav">  
 <a href="/">Books</a> | <a href="/Authors">Authors</a> | <a href="/Genres">Genres</a>  
 </div>  
 <div class="container">  
 @RenderBody()  
 </div>  
</body>  
</html>  
  
src/Library.Web/Views/Books/Index.cshtml  
  
@model IEnumerable<Library.Web.Models.Book>  
@{  
 ViewData["Title"] = "Books";  
 var total = (int)ViewBag.Total;  
 var page = (int)ViewBag.Page;  
 var pageSize = (int)ViewBag.PageSize;  
 var search = (string)ViewBag.Search ?? "";  
 var pages = (int)Math.Ceiling(total / (double)pageSize);  
}  
<div class="page-header">  
 <h2>Books</h2>  
 <div class="actions">  
 <input id="searchBox" value="@search" placeholder="Search title..." />  
 <button id="btnSearch">Search</button>  
 <button id="btnAdd">Add Book</button>  
 </div>  
</div>  
  
<div id="listContainer">  
 @await Html.PartialAsync("\_List", Model)  
</div>  
  
<div id="modalHost" class="modal" style="display:none;">  
 <div class="modal-content">  
 <span class="close" id="modalClose">&times;</span>  
 <div id="modalBody"></div>  
 </div>  
</div>

src/Library.Web/Views/Books/\_List.cshtml  
  
@model IEnumerable<Library.Web.Models.Book>  
<table class="table">  
 <thead>  
 <tr><th>Title</th><th>Genre</th><th>Authors</th><th>Published</th><th></th></tr>  
 </thead>  
 <tbody>  
 @foreach (var b in Model)  
 {  
 <tr data-id="@b.Id">  
 <td>@b.Title</td>  
 <td>@b.Genre?.Name</td>  
 <td>@string.Join(", ", b.BookAuthors.Select(ba => ba.Author.Name))</td>  
 <td>@b.PublishedYear</td>  
 <td>  
 <button class="btn-edit" data-id="@b.Id">Edit</button>  
 <button class="btn-delete" data-id="@b.Id">Delete</button>  
 </td>  
 </tr>  
 }  
 </tbody>  
</table>  
  
src/Library.Web/Views/Books/\_CreateOrEdit.cshtml  
  
@model Library.Web.Models.Book  
@{  
 var isEdit = Model?.Id > 0;  
 var authors = (IEnumerable<Library.Web.Models.Author>)ViewBag.Authors;  
 var genres = (IEnumerable<Library.Web.Models.Genre>)ViewBag.Genres;  
 var selectedAuthorIds = Model?.BookAuthors?.Select(ba => ba.AuthorId).ToHashSet() ?? new HashSet<int>();  
}  
<form id="bookForm" data-mode="@(isEdit ? "edit" : "create")" data-id="@Model?.Id">  
 <div>  
 <label>Title</label>  
 <input id="title" value="@Model?.Title" required />  
 </div>  
 <div>  
 <label>Genre</label>  
 <select id="genreId" required>  
 <option value="">--select--</option>  
 @foreach (var g in genres)  
 {  
 <option value="@g.Id" @(Model?.GenreId == g.Id ? "selected" : "")>@g.Name</option>  
 }  
 </select>  
 </div>  
 <div>  
 <label>Published Year</label>  
 <input id="publishedYear" type="number" value="@Model?.PublishedYear" min="1000" max="9999" />  
 </div>  
 <div>  
 <label>Authors</label>  
 <div class="checkboxes">  
 @foreach (var a in authors)  
 {  
 <label>  
 <input type="checkbox" class="author" value="@a.Id" @(selectedAuthorIds.Contains(a.Id) ? "checked" : "") />  
 @a.Name  
 </label>  
 }  
 </div>  
 </div>  
 <div class="form-actions">  
 <button type="submit">@(isEdit ? "Update" : "Create")</button>  
 </div>  
</form>  
  
src/Library.Web/Views/Authors/Index.cshtml  
  
@model IEnumerable<Library.Web.Models.Author>  
@{  
 ViewData["Title"] = "Authors";  
}  
<h2>Authors</h2>  
<form method="post" asp-action="Create">  
 <input name="name" placeholder="Author name" required />  
 <button type="submit">Add</button>  
</form>  
  
<table class="table">  
<thead><tr><th>Name</th></tr></thead>  
<tbody>  
@foreach (var a in Model) { <tr><td>@a.Name</td></tr> }  
</tbody>  
</table>  
  
src/Library.Web/Views/Genres/Index.cshtml  
  
@model IEnumerable<Library.Web.Models.Genre>  
@{  
 ViewData["Title"] = "Genres";  
}  
<h2>Genres</h2>  
<form method="post" asp-action="Create">  
 <input name="name" placeholder="Genre name" required />  
 <button type="submit">Add</button>  
</form>  
  
<table class="table">  
<thead><tr><th>Name</th></tr></thead>  
<tbody>  
@foreach (var g in Model) { <tr><td>@g.Name</td></tr> }  
</tbody>  
</table>  
  
 src/Library.Web/wwwroot/css/site.css  
  
body { font-family: system-ui, Arial, sans-serif; margin:0; }  
.nav { background:#111; color:#eee; padding:.75rem 1rem; }  
.nav a { color:#eee; text-decoration:none; margin-right:1rem; }  
.container { padding: 1rem; }  
.table { width:100%; border-collapse: collapse; }  
.table th, .table td { border:1px solid #ddd; padding:.5rem; }  
.actions { margin-bottom:1rem; display:flex; gap:.5rem; }  
.modal { position:fixed; inset:0; background-color:rgba(0,0,0,.5); }  
.modal-content { background:#fff; margin:10vh auto; padding:1rem; width:min(800px, 95%); border-radius:.5rem; }  
.close { float:right; cursor:pointer; font-size:1.5rem; }  
.checkboxes { display:grid; grid-template-columns: repeat(auto-fill,minmax(180px,1fr)); gap:.25rem .75rem; max-height:40vh; overflow:auto; padding:.5rem; border:1px solid #ddd; }  
.form-actions { margin-top:.75rem; }  
#searchBox { width:240px; }  
  
 src/Library.Web/wwwroot/js/library.js  
  
// AJAX helpers for Books page (uses fetch + partial rendering)  
document.addEventListener('DOMContentLoaded', () => {  
 const listContainer = document.querySelector('#listContainer');  
 const btnSearch = document.querySelector('#btnSearch');  
 const searchBox = document.querySelector('#searchBox');  
 const btnAdd = document.querySelector('#btnAdd');  
 const modalHost = document.querySelector('#modalHost');  
 const modalBody = document.querySelector('#modalBody');  
 const modalClose = document.querySelector('#modalClose');  
  
 const openModal = (html) => { modalBody.innerHTML = html; modalHost.style.display = 'block'; };  
 const closeModal = () => { modalHost.style.display = 'none'; modalBody.innerHTML = ''; };  
 if (modalClose) modalClose.addEventListener('click', closeModal);  
 window.addEventListener('click', e => { if (e.target === modalHost) closeModal(); });  
  
 const loadList = async (query = '') => {  
 const resp = await fetch('/Books' + query, { headers: { 'X-Requested-With': 'XMLHttpRequest' } });  
 if (!resp.ok) { alert('Failed to load list'); return; }  
 listContainer.innerHTML = await resp.text();  
 wireListButtons();  
 };  
  
 if (btnSearch) btnSearch.addEventListener('click', () => {  
 const q = searchBox.value.trim();  
 const query = q ? `?search=${encodeURIComponent(q)}` : '';  
 loadList(query);  
 });  
  
 if (btnAdd) btnAdd.addEventListener('click', async () => {  
 const resp = await fetch('/Books/Create', { headers: { 'X-Requested-With': 'XMLHttpRequest' } });  
 openModal(await resp.text());  
 wireForm();  
 });  
  
 function wireListButtons() {  
 document.querySelectorAll('.btn-edit').forEach(btn => {  
 btn.addEventListener('click', async () => {  
 const id = btn.dataset.id;  
 const resp = await fetch(`/Books/Edit/${id}`, { headers: { 'X-Requested-With': 'XMLHttpRequest' } });  
 openModal(await resp.text());  
 wireForm();  
 });  
 });  
  
 document.querySelectorAll('.btn-delete').forEach(btn => {  
 btn.addEventListener('click', async () => {  
 const id = btn.dataset.id;  
 if (!confirm('Delete this book?')) return;  
 const resp = await fetch(`/Books/Delete/${id}`, { method: 'DELETE' });  
 if (resp.ok) loadList(location.search);  
 else {  
 const data = await resp.json().catch(() => ({}));  
 alert(data?.detail || 'Delete failed');  
 }  
 });  
 });  
 }  
  
 function wireForm() {  
 const form = document.querySelector('#bookForm');  
 if (!form) return;  
 form.addEventListener('submit', async (e) => {  
 e.preventDefault();  
 const mode = form.dataset.mode;  
 const id = form.dataset.id;  
 const authorIds = Array.from(document.querySelectorAll('input.author:checked')).map(el => parseInt(el.value));  
 const body = JSON.stringify({  
 title: document.querySelector('#title').value.trim(),  
 genreId: parseInt(document.querySelector('#genreId').value),  
 publishedYear: parseInt(document.querySelector('#publishedYear').value) || null,  
 authorIds  
 });  
 const headers = { 'Content-Type': 'application/json' };  
 let url = '/Books/Create', method = 'POST';  
 if (mode === 'edit') { url = `/Books/Edit/${id}`; method = 'PUT'; }  
 const resp = await fetch(url, { method, headers, body });  
 if (resp.ok) { closeModal(); loadList(location.search); }  
 else {  
 const data = await resp.json().catch(() => ({}));  
 alert(data?.detail || 'Operation failed');  
 }  
 });  
 }  
 wireListButtons();  
});