

.NET TUTORIAL

Thomasina Cummings
GCS TRAINING Jubilee Court, Swansea

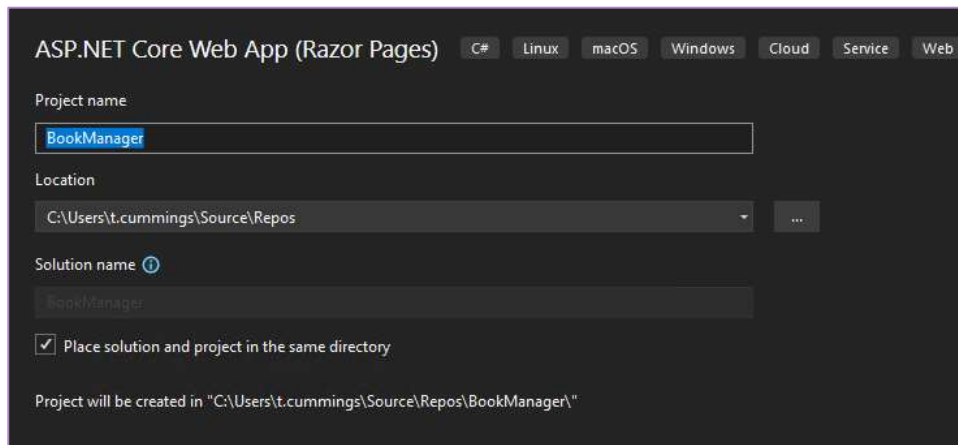
Setup

Install VS 2022 with the ASP.NET and web development workload.

Start Visual Studio and select Create a New project.

In the Create a new project dialog, select ASP.NET Core Web App (Razor Pages) > Next.

In the Configure your new project dialog, enter BookManager for Project name. It's important to name the project BookManager, including matching the capitalization, so the namespaces will match when you copy and paste example code.



Select Next.

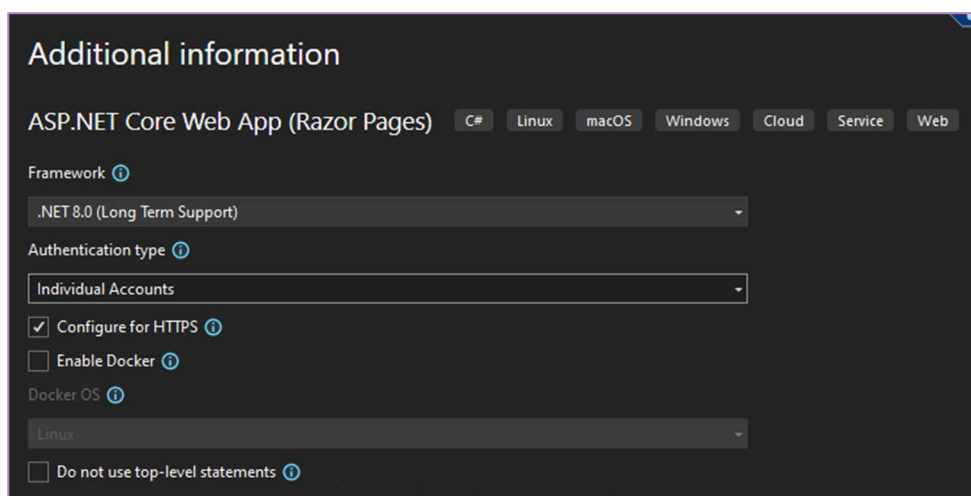
In the Additional information dialog:

Select .NET 8.0 (Long Term Support).

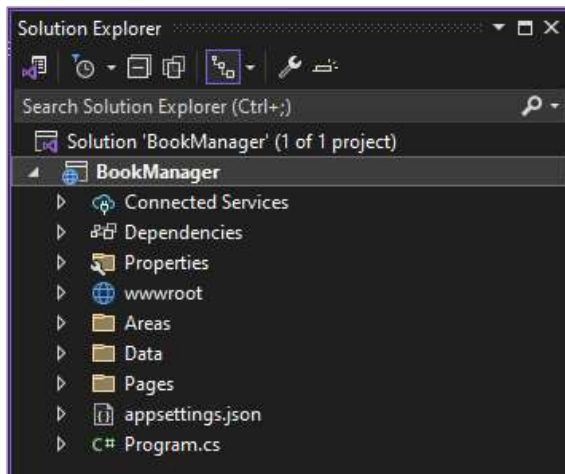
Individual Accounts

Verify: Do not use top-level statements is unchecked.

Select Create.



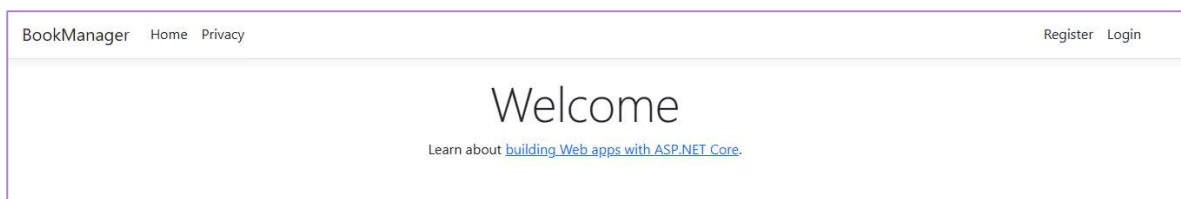
A starter project will be created and the BookManager folder should contain the following by default.



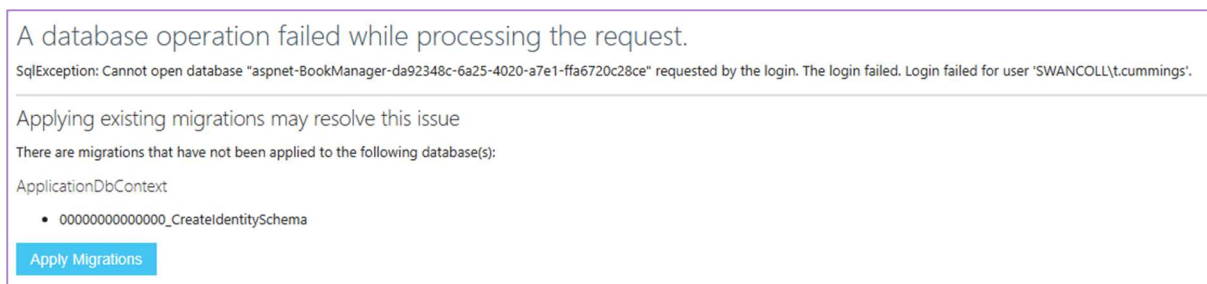
Run the app without the debugger, using CTRL + F5 (you may need to add Fn if on laptop).

Security

If the project is not yet configured to use SSL, so click Yes to install the development certificate. Visual studio will launch the Kestrel server which, in turn, will launch the default browser to ***http:localhost.<port>***.

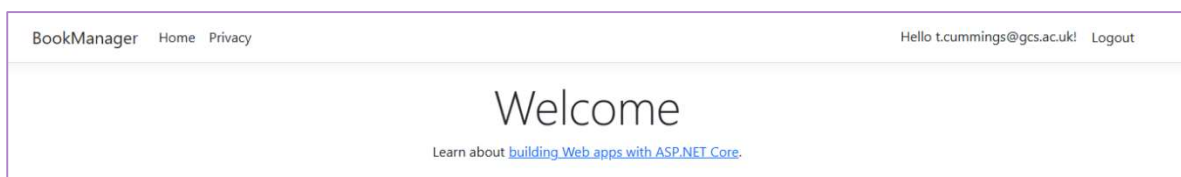


Click to Register an account – when it fails, click Apply Migrations, and refresh when prompted.



Reload page and click to confirm your account – you can set up for a real email sender later ([optional](#)).

Log in to test your account. Add this test to your test log with OK or NOK result.



Close the browser.

Pages Folder

The Pages folder contains Razor pages and supporting files.

Each Razor page contains a pair of files.

- .cshtml – a HTML markup with C# code using Razor syntax
- .cshtml.cs – a file containing C# code to handle page events

Files beginning with _ are supporting files (such as _Layout.cshtml that configure user interface elements that are common to all pages and _Layout.cshtml which sets up navigation).

wwwroot Folder

This folder contains static assets like HTML files, JavaScript files, and CSS files.

appsettings.json

This contains configuration data, like connection strings.

Program.cs

Contains the C# code for the application. It also contains the code that will be used to build the web application.

Important Coding Note

Code is provided, but it is also important you understand what the code is doing. If in doubt, check the Microsoft help pages for more details.

Throughout, you should also note that Book, book, books, and Books are **not** the same. If you get errors, check the correct instance is used. Don't put your trust too much in the version in the code, as autocorrect can sometimes update without the writer noticing.

Add a data model

In the Solution Explorer, right click the BookManager project > Add > New Folder. Name the folder Models.

Right click the Models folder. Select Add > Class. Name the class Book.

Add the following code in place of the default.

```
using System.ComponentModel.DataAnnotations;
namespace BookManager.Models;
public class Book
{
    public int Id { get; set; }
    public string? Title { get; set; }
    [DataType(DataType.Date)]
    public DateTime ReleaseDate { get; set; }
    public string? Genre { get; set; }
    public decimal Price { get; set; }
}
```

- ID is a unique primary key.
- The ? after string makes it *Nullable*, or context aware, so that it is less likely to throw exception errors.

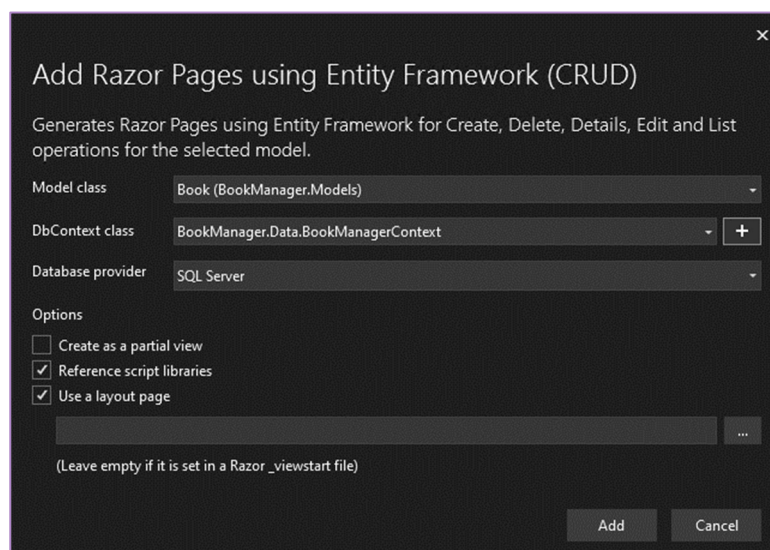
With a model like books, you need to be able to Create, Read, Update and Delete entries. This can be done quickly by **Scaffolding**.

Right click on the pages folder > Add > New Folder.

Name the folder Books

Right click on the Pages/Books Folder > Add > New Scaffolded Item

In the menu, Add New Scaffold by selecting Razor Pages using Entity Framework (CRUD). Complete as per the next image. Click add and wait for the scaffolding to generate.



The migrations feature in Entity Framework Core provides a way to create the database schema and keep it up to date, while preserving existing data.

From the Tools menu, select NuGet Package Manager > Package Manager Console (PMC)

In the PMC type:

```
Add-Migration BookCreate -context BookManagerContext
Update-Database -context BookManagerContext
```

Ignore any errors about price. These will be fixed later.

Use the debug and run option to open the browser and add /Books/Create to the URL

Create some new books

You can also test the Edit, Delete and Details links are working.

Add tests to your test log with OK or NOK results.

Delete all books and close the browser before continuing.

Update the Layout

You can now start to customise the layout.

Change the <title> element in the Pages/Shared/_Layout.cshtml file to display Book Manager instead of BookManager

Find the anchor element and change the line Book Manager to Book Manager

Test the Book Manager: Create, Edit and Delete links.

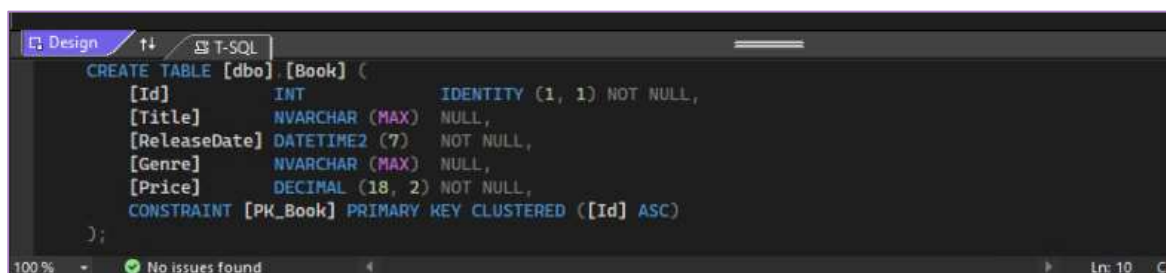
Again, you should add the tests to your test log with OK or NOK status.

Delete all books and close browser.

SQL Server Express Local DB

From the view menu, open SQL Server Object Explorer (SSOX)

Right click on dbo.Book and select View Designer



Seeding Data

In the Models folder, create a new class named SeedData with the following code – add your own book choices. Repeat new Book code for as many books as you want to add.

```
using Microsoft.EntityFrameworkCore;
using BookManager.Data;
namespace BookManager.Models;
public static class SeedData
{
    public static void Initialize(IServiceProvider serviceProvider)
    {
        using (var context = new BookManagerContext(
            serviceProvider.GetRequiredService<
                DbContextOptions<BookManagerContext>>()))
        {
            if (context == null || context.Book == null)
            {
                throw new ArgumentNullException("Null
BookManagerContext");
            }
            // Look for any book.
            if (context.Book.Any())
            {
                return; // DB has been seeded
            }
            context.Book.AddRange(
                new Book
                {
                    Title = "Add book here",
                    ReleaseDate = DateTime.Parse("yyyy-mm-dd"),
                    Genre = "Add genre here",
                    Price = 7.99M
                },
                new Book
                {
                    Title = "Add book here ",
                    ReleaseDate = DateTime.Parse("yyyy-mm-dd"),
                    Genre = "Add genre here",
                    Price = 8.99M
                }
            );
            context.SaveChanges();
        }
    }
}
```

Add the seed initialiser

- Open Program.cs
- Add using BookManager.Models; to the using section
- Add the following lines below the var app = builder.Build(); section.

```
using (var scope = app.Services.CreateScope())
{
    var services = scope.ServiceProvider;

    SeedData.Initialize(services);
}
```

Run the app to check if the seed method runs, then update Models/Books.cs with the following. Reminder: log all tests in your test log.

```
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace BookManager.Models;

public class Book
{
    public int Id { get; set; }
    public string Title { get; set; } = string.Empty;

    [Display(Name = "Release Date")]
    [DataType(DataType.Date)]
    public DateTime ReleaseDate { get; set; }
    public string Genre { get; set; } = string.Empty;

    [Column(TypeName = "decimal(18, 2)")]
    public decimal Price { get; set; }
}
```

Add a Search

Add the following code to Pages/Books/Index.cshtml.cs

```
public class IndexModel : PageModel
{
    private readonly BookManager.Data.RazorPagesBookContext _context;

    public IndexModel(BookManager.Data.RazorPagesBookContext context)
    {
        _context = context;
    }

    public IList<Book> Book { get; set; } = default!;

    [BindProperty(SupportsGet = true)]
    public string? SearchString { get; set; }

    public SelectList? Genres { get; set; }

    [BindProperty(SupportsGet = true)]
    public string? BookGenre { get; set; }
```

Update the Index page's OnGetAsync method with the following code:

```
public async Task OnGetAsync()
{
    var books = from b in _context.Book
                select b;
    if (!string.IsNullOrEmpty(SearchString))
    {
        books = books.Where(s => s.Title.Contains(SearchString));
    }

    Book = await books.ToListAsync();
}
```

Add the following to Pages/Books/Index.cshtml above <table class="table"> to create a search input box.

```
<form>
    <p>
        Title: <input type="text" asp-for="SearchString" />
        <input type="submit" value="Filter" />
    </p>
</form>
```

Add a New Field

Open the Models/Books.cs file and add a Comment property using public string
Comment {get; set; } = string.Empty

Add a Comment field to the Pages/Books/Index.cshtml file

```
<th>
    @Html.DisplayNameFor(model => model.Book[0].Comment)
</th>

Further down:

<td>
    @Html.DisplayFor(modelItem => item.Comment)
</td>
```

You should also update the following pages with a Comment field

[Pages/Books/Create.cshtml.](#)

[Pages/Books/Delete.cshtml.](#)

[Pages/Books/Details.cshtml.](#)

[Pages/Books/Edit.cshtml.](#)

You should also update the seed class to add Comment = "R"

Build the App

CTRL + SHIFT + B, logging and fixing any errors and tests.

Add Comment to Migration

In the PMC, enter the following:

```
Add-Migration Comment -context BookManagerContext
Update-Database -context BookManagerContext
```

Validation

Using DRY (**Don't Repeat Yourself**) software development principles is key.

Update Books.cs file to take advantage of build in validation.

```
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace BookManager.Models;

public class Book
{
    public int Id { get; set; }

    [StringLength(60, MinimumLength = 3)]
    [Required]
    public string Title { get; set; } = string.Empty;

    [DataType(DataType.Date)]
    public DateTime ReleaseDate { get; set; }

    [Range(1, 100)]
    [DataType(DataType.Currency)]
    [Column(TypeName = "decimal(18, 2)")]
    public decimal Price { get; set; }

    [RegularExpression(@"^[A-Z]+[a-zA-Z\s]*$")]
    [Required]
    [StringLength(30)]
    public string Genre { get; set; } = string.Empty;

    [RegularExpression(@"^[A-Z]+[a-zA-Z0-9""'\s-]*$")]
    [StringLength(250)]
    [Required]
    public string Comment { get; set; } = string.Empty;
}
```

Run the app and navigate to Pages/Books.

Select the Create New link and check that validation is working.

Remember to log all tests.

At this point you should create a migration so that the schema is consistent with the model.

```
Add-Migration New_DataAnnotations -context BookManagerContext  
Update-Database -context BookManagerContext
```

For your submission you will need add content, test fully and develop appropriately.

See planning notes.

You should also provide test logs.