[**HOW TO RETRIEVE DATA FROM DIFFERENT Tables IN DATATABLES WITH ASP.NET MVC CORE 2.2**](https://github.com/LayersOfAbstraction/Pitcher)

You could use a Entity Framework to create the UI for CRUD operations but that would mean more code, a performance reduction and not nearly as elegant if you design it yourself.

DataTables takes the heartache out of that. For example, I will show you the code for one controller and view built entirely in Entity Framework Core. Don't worry you don't have to memorise all the code in the samples right now especially in EF Core.

**THE VIEW**

@model PaginatedList<Pitcher.Models.Registration>

@{

ViewData["Title"] = "Registrations";

}

<**h1**>Registrations</**h1**>

<**p**>

<**a** asp-action="Create">Create New</**a**>

</**p**>

@\*COPY AND PASTE THIS TAG HELPER METHOD TEXTBOX CUSTOMIZATION INTO OTHER VIEWS TO ENABLE SEARCHING.\*@

<**form** asp-action="Index" method="get">

<**div** class="form-actions no-color">

<**p**>

Find by name: <**input** type="text" name="SearchString" value="@ViewData["currentFilter"]" />

<input type="submit" value="Search" class="btn btn-primary" /> |

<**a** asp-action="Index">Back to Full List</**a**>

</**p**>

</**div**>

</**form**>

<**table** class="table table-hover">

<**thead**>

<**tr**>

<**th**>

<**a** asp-action="Index" asp-route-sortOrder="@ViewData["RegDateSortParm"]" asp-route-currentFilter="@ViewData["CurrentFilter"]">RegistrationDate</a>

</th>

<th>

@\* COPY AND PASTE THIS METHOD CUSTOMIZATION INTO OTHER VIEWS BOUND TO COMPOSITE TABLES. Enables sorting. \*@

<a asp-action="Index" asp-route-sortOrder="@ViewData["FullNameSortParam"]" asp-route-currentFilter="@ViewData["CurrentFilter"]">Full Name</a>

</th>

<th>

<a asp-action="Index" asp-route-sortOrder="@ViewData["JobTitleSortParam"]" asp-route-currentFilter="@ViewData["CurrentFilter"]">Job Title</a>

</th>

<th></th>

</tr>

</thead>

<tbody>

@**foreach** (var item in Model) {

<tr>

<td>

@Html.DisplayFor(modelItem => item.RegistrationDate)

</td>

<td>

@Html.DisplayFor(modelItem => item.User.UserFullname)

</td>

<td>

@Html.DisplayFor(modelItem => item.Job.JobTitle)

</td>

<td>

<a asp-action="Edit" asp-route-id="@item.ID" button type="button" class="btn btn-primary btn-block">Edit</**a**>

<**a** asp-action="Details" asp-route-id="@item.ID" button type="button" class="btn btn-info btn-block">Details</a>

<a asp-action="Delete" asp-route-id="@item.ID" button type="button" class="btn btn-primary btn-block">Delete</**a**>

</**td**>

</**tr**>

}

</**tbody**>

</**table**>

@{

var prevDisabled = !Model.HasPreviousPage ? "disabled" : "";

var nextDisabled = !Model.HasNextPage ? "disabled" : "";

}

<**a** asp-action="Index"

asp-route-sortOrder="@ViewData["CurrentSort"]"

asp-route-pageNumber="@(Model.PageIndex - 1)"

asp-route-currentFilter="@ViewData["CurrentFilter"]"

class="btn btn-secondary @prevDisabled"

button type="button">

Previous

</**a**>

<**a** asp-action="Index"

asp-route-sortOrder="@ViewData["CurrentSort"]"

asp-route-pageNumber="@(Model.PageIndex + 1)"

asp-route-currentFilter="@ViewData["CurrentFilter"]"

class="btn btn-secondary @nextDisabled"

button type="button">

Next

</**a**>

**CONTROLLER WITH INDEX METHOD**

**public** **async** Task **Index**(**string** sortOrder, **string** currentFilter, **string** searchString, **int**? pageNumber)

{

ViewData["CurrentSort"] = sortOrder;

ViewData["FullNameSortParam"] = String.IsNullOrEmpty(sortOrder) ? "fullName\_desc" : "";

ViewData["JobTitleSortParam"] = sortOrder == "jobTitle" ? "jobTitle\_desc" : "jobTitle";

ViewData["RegDateSortParm"] = sortOrder == "Date" ? "date\_desc" : "Date";

ViewData["CurrentFilter"] = searchString;

IQueryable registrations = \_context.Registrations.Include(r => r.Job).Include(r => r.User);

**if** (searchString != null)

{

pageNumber = 1;

}

**else**

{

searchString = currentFilter;

}

**if** (!String.IsNullOrEmpty(searchString))

{

registrations = registrations.Where(r => r.User.ToString().Contains(searchString)

|| r.Job.ToString().Contains(searchString));

}

**switch** (sortOrder)

{

**case** "fullName\_desc":

registrations = registrations.OrderByDescending(r => r.User);

**break**;

**case** "jobTitle\_desc":

registrations = registrations.OrderByDescending(r => r.Job);

**break**;

**case** "Date":

registrations = registrations.OrderBy(r => r.RegistrationDate);

**break**;

**case** "date\_desc":

registrations = registrations.OrderByDescending(r => r.RegistrationDate);

**break**;

**case** "jobTitle":

registrations = registrations.OrderBy(r => r.User);

**break**;

//By default FullName is in ascending order when entity is loaded.

**default**:

registrations = registrations.OrderBy(r => r.Job);

**break**;

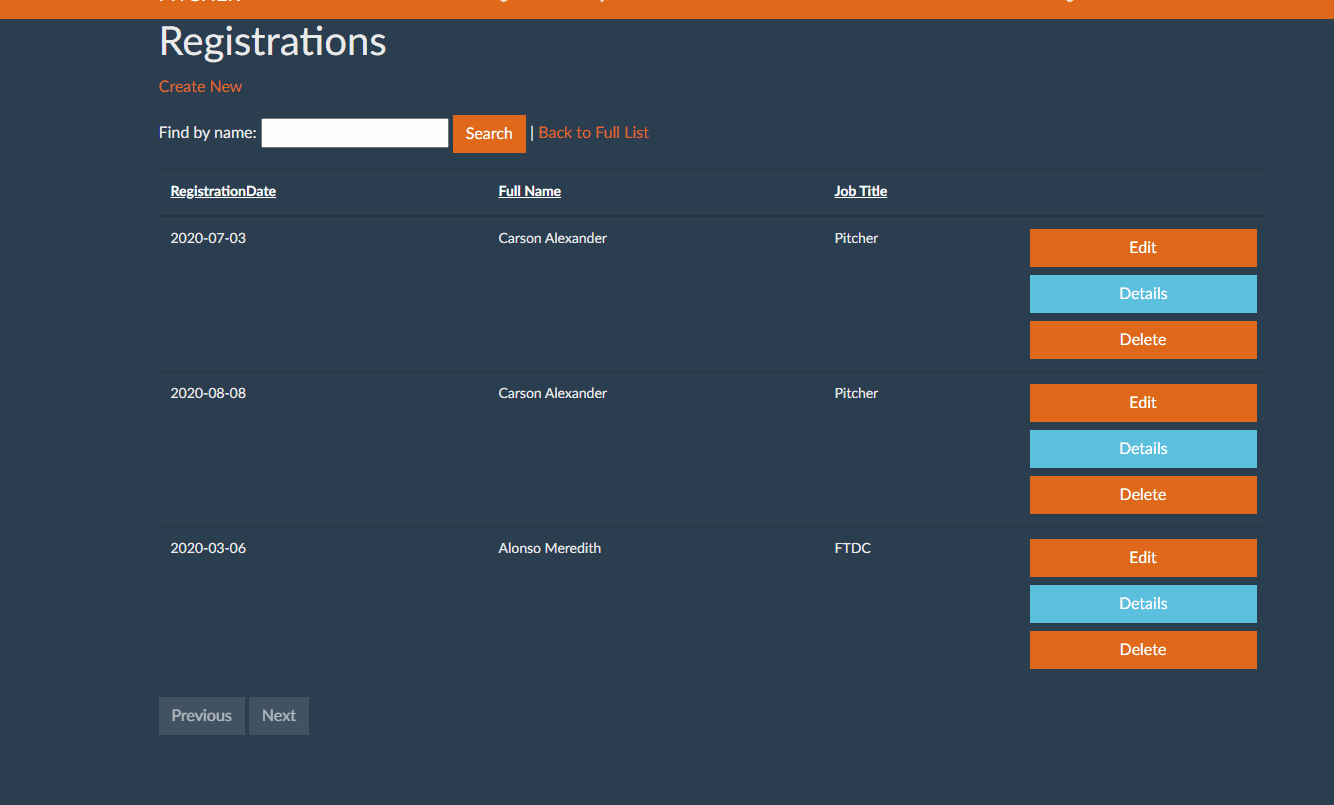
}

**int** pageSize = 20;

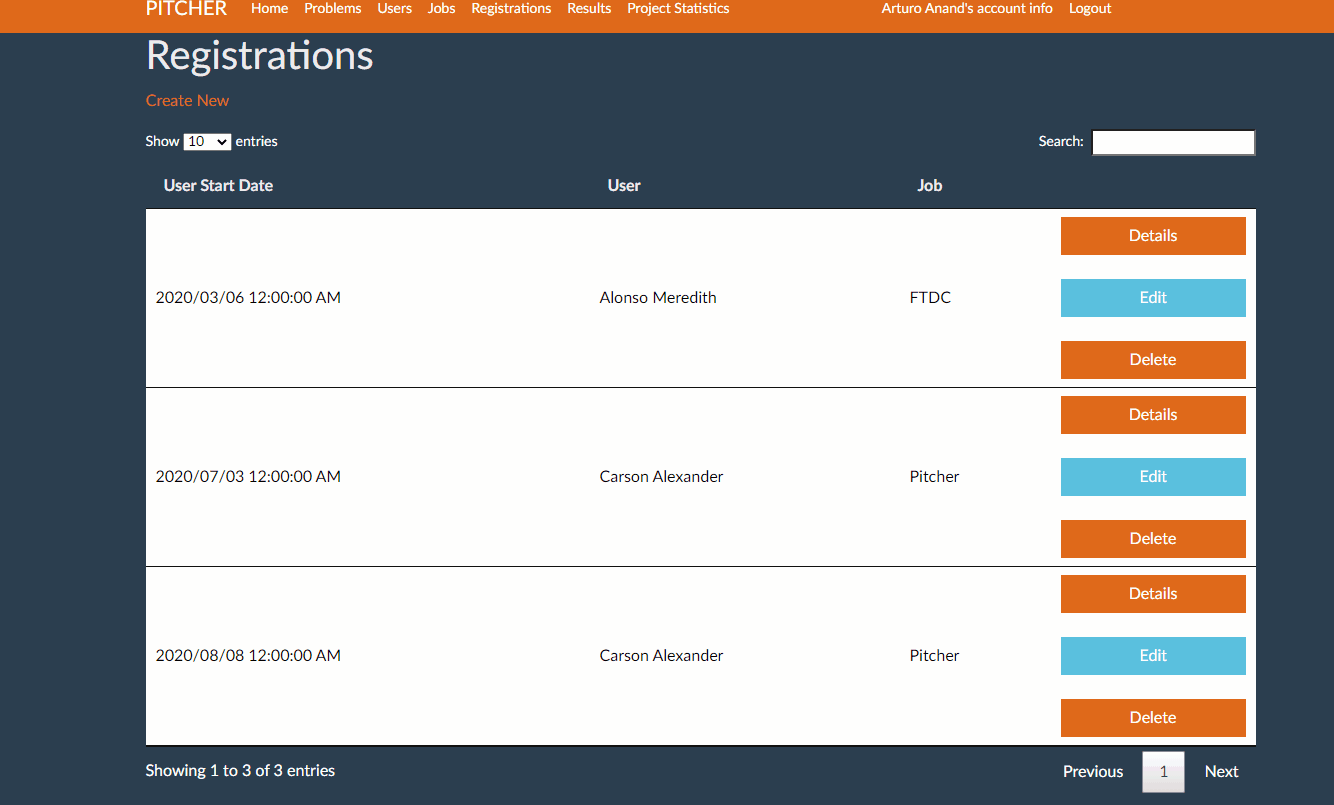
**return** View(**await** PaginatedList.CreateAsync(registrations.AsNoTracking(), pageNumber ?? 1, pageSize));

}

Now I will show you what the UI looks like when I compile the code.



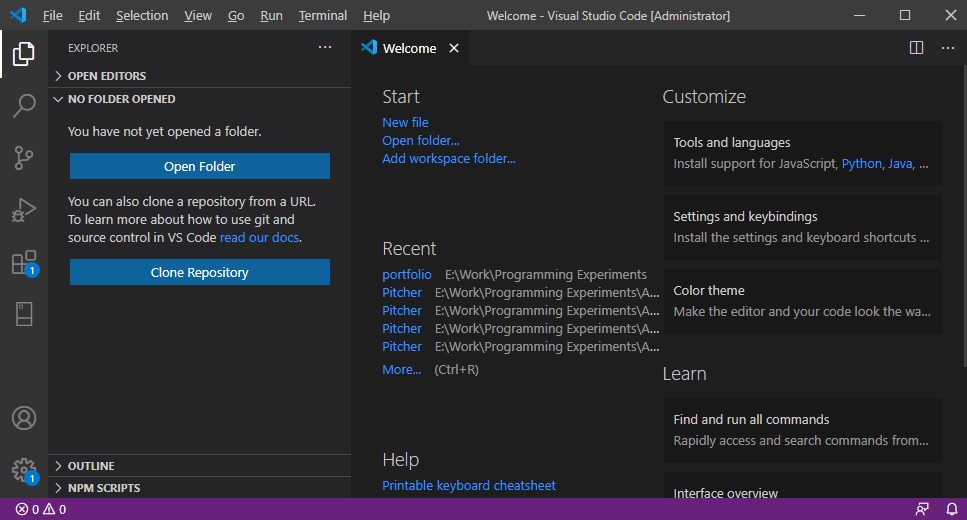
Now if we write the code in DataTables Editor in the Index method for the same controller and the view, it will be much shorter. I will show you what it would look like if we run that.



Obviously there is heaps more functionality in the UI, the code (which I will show later) is cleaner and has less performance penalty. Let's replicate this in a new project. I will do this on Windows 10. You should be able to run and Entity Framework Core on Linux too as .NET Core is open source though if you do Mac then you will have to use Docker to install SQL Server which this tutorial sadly does not cover.

First open Visual Studio Code and use the Explorer window to navigate to whatever folder you want to create the app in. Make sure you open the Terminal if it's not already. Make sure you have downloaded .NET Core SDK 2.2 or later.

<https://dotnet.microsoft.com/download/dotnet-core>



Now we will use the .NET Core CLI. If the terminal is not present, left click on Terminal in the and select "New Terminal" then copy these commands into it and press enter.

dotnet new mvc -o DTEditorLeftJoinSample  
code -r DTEditorLeftJoinSample

A dialog box appears with**Required assets to build and debug are missing from 'DTEditorLeftJoinSample'. Add them?** Select **Yes**. We have now created and loaded the project. Now Press F5 to Debug and select .Net Core from the command pallet to run the project. It should use the default browser you have. I advise you use Chrome if you are not sure otherwise. If the sample project displays the home page we are good.

We will now create a Recipe database 3 different models, Recipe, RecipeIngredient and Ingredient.

**RECIPE**

**using** System.Collections.Generic;

**using** System.ComponentModel.DataAnnotations;

**using** System.ComponentModel.DataAnnotations.Schema;

**namespace** **DTEditorLeftJoinSample**.**Models**

{

**public** **class** **Recipe**

{

**public** **int** ID { **get**; **set**; }

**public** **string** Title {**get**;**set**;}

**public** **string** Descriptions {**get**;**set**;}

**public** **string** Directions {**get**;**set**;}

**public** ICollection RecipeIngredients {**get**;**set**;}

}

}

**RECIPEINGREDIENT**

**using** System.Collections.Generic;

**using** System.ComponentModel.DataAnnotations;

**using** System.ComponentModel.DataAnnotations.Schema;

**namespace** **DTEditorLeftJoinSample**.**Models**

{

**public** **class** **RecipeIngredient**

{

**public** **int** ID {**get**;**set**;}

[Display(Name = "Recipe ID")]

**public** **int** RecipeID { **get**; **set**; }

[Display(Name = "Ingredient ID")]

**public** **int** IngredientID { **get**; **set**; }

**public** **int** quantity {**get**;**set**;}

**public** Recipe Recipe {**get**;**set**;}

**public** Ingredient Ingredient {**get**;**set**;}

}

}

Now enter these packages into the terminal.

dotnet tool install --global dotnet-ef

dotnet tool install --global dotnet-aspnet-codegenerator

dotnet add package Microsoft.VisualStudio.Web.CodeGeneration.Design

dotnet add package Microsoft.EntityFrameworkCore.Design

dotnet add package Microsoft.EntityFrameworkCore.SqlServer

Even though we cannot integrate Entity Framework Core directly with DataTables Editor, we can still create the database to use with the library. We will do this by creating the database context class.

using DTEditorLeftJoinSample.Models;

using Microsoft.EntityFrameworkCore;

namespace DTEditorLeftJoinSample.Data

{

    public class CookingContext : DbContext

    {

        public CookingContext (DbContextOptions< CookingContext> options) : base(options)

        {

        }

        public DbSet<Recipe> Recipe { get; set; }

        public DbSet<Ingredient> Ingredient {get;set;}

        public DbSet<RecipeIngredient> RecipeIngredient {get;set;}

        protected override void OnModelCreating(ModelBuilder modelBuilder)

        {

            modelBuilder.Entity<Recipe>().ToTable("tblRecipe");

            modelBuilder.Entity<Ingredient>().ToTable("tblIngredient");

            modelBuilder.Entity<RecipeIngredient>().ToTable("tblRecipeIngredient ");

        }

    }

}

Register the CookingContext as a service in Startup.cs using dependency injection where the ConfigureServices method is. You can do that by adding this code to the method.

services.AddDbContext<CookingContext>(options =>

options.UseSqlServer(Configuration.GetConnectionString("DefaultConnection")));

Now add these statements to the startup file.

using DTEditorLeftJoinSample.Data;

using Microsoft.EntityFrameworkCore;

using Microsoft.AspNetCore.Http;

Now we want to seed the database with test data. This is an optional step but highly beneficial. If it does not work for you, the data can be entered manually. In the Data folder create this this file DbInitializer.cs and insert this code.

using DTEditorLeftJoinSample.Models;

using System;

using System.Linq;

using Microsoft.EntityFrameworkCore;

using Microsoft.Extensions.DependencyInjection;

namespace DTEditorLeftJoinSample.Data

{

    public static class DbInitializer

    {

        public static void Initialize(IServiceProvider serviceProvider)

        {

            using (var context = new CookingContext(

                serviceProvider.GetRequiredService<

                    DbContextOptions<CookingContext>>()))

            {

                context.Database.EnsureCreated();

                // Look for any tables.

                if (context.Recipe.Any() && context.Ingredient.Any() && context.RecipeIngredient.Any())

                {

                    return;   // DB has been seeded

                }

                var recipes = new Recipe[]

                {

                    new Recipe { Title =" Korean-Style Steak and Noodles with Kimchi",

                    Description="Lorem ipsum dolor sit amet, consectetur adipiscing elit. Praesent sed pharetra neque. Curabitur laoreet eu lectus eu tempus. Fusce elementum arcu ut justo tincidunt mattis.",

                    Direction="1.Cras dignissim in neque a placerat." + "\r\n" + "2.Vestibulum vel vestibulum nunc." + "\r\n" +  "3. Vestibulum interdum est tellus, nec porta metus dignissim ut."

                    },

                    new Recipe { Title =" Mashed Potatoes with Savory Thyme Granola",

                    Description=" Etiam aliquam, magna quis lobortis facilisis, lorem eros dignissim nulla, ultrices pulvinar orci lectus a ligula.",

                    Direction="1. Morbi fringilla, justo eu venenatis tempus, mauris leo ultricies magna, et aliquet mi lectus at nisi. Pellentesque vel gravida nunc. Donec in tortor lectus." + "\r\n" + "2.Vestibulum vel vestibulum nunc." + "\r\n" +  "3. Vestibulum interdum est tellus, nec porta metus dignissim ut."},

                    new Recipe { Title ="Lemon Garlic Mashed Potatoes",

                    Description="Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas.",

                    Direction="1. Maecenas ultricies pretium quam id placerat. Mauris in ligula gravida, vehicula justo faucibus, semper neque." + "\r\n" + "2. Proin sodales aliquam erat quis venenatis." + "\r\n" +  "3. Morbi consectetur libero id sagittis vestibulum."},

                    new Recipe { Title =" Sour Cream and Corn Mashers",

                    Description=" Donec posuere pellentesque mi, ac suscipit tellus finibus id.",

                    Direction="1. Nulla placerat erat lorem, eget pellentesque dolor egestas vitae." + "\r\n" + "2. Proin sodales aliquam erat quis venenatis." + "\r\n" +  "3. Suspendisse ac purus lacinia, mollis velit aliquet, finibus arcu. Pellentesque molestie est in diam pulvinar, quis mattis justo volutpat."}

                                };

                foreach (Recipe r in recipes)

                {

                    context.Recipe.AddRange(r);

                }

                context.SaveChanges();

                var ingredients = new Ingredient[]

                {

                    new Ingredient{IngredientName="Duis eu ligula felis"},

                    new Ingredient{IngredientName="Donec id mollis arcu"},

                    new Ingredient{IngredientName="Cras nec enim luctus"}

                };

                foreach (Ingredient i in ingredients)

                {

                    context.Ingredient.AddRange(i);

                }

                context.SaveChanges();

                var recipeIngredients = new RecipeIngredient[]

                {

                    new RecipeIngredient{RecipeID=1, IngredientID=1, Quantity =4},

                    new RecipeIngredient{RecipeID=2, IngredientID=2, Quantity =3},

                    new RecipeIngredient{RecipeID=3, IngredientID=3, Quantity =15}

                };

                foreach (RecipeIngredient ri in recipeIngredients)

                {

                    context.RecipeIngredient.AddRange(ri);

                }

                context.SaveChanges();

            }

        }

    }

}

We want to get the database context instance from dependency injection container. Now we have to call the context instance, the seed method and pass it to the context. Then dispose the context when the seeding is complete. In Program.cs add this code in the Main method to do that.

var host = CreateHostBuilder(args).Build();

            using (var scope = host.Services.CreateScope())

            {

                var services = scope.ServiceProvider;

                try

                {

                    DbInitializer.Initialize(services);

                }

                catch (Exception ex)

                {

                    var logger = services.GetRequiredService<ILogger<Program>>();

                    logger.LogError(ex, "An error occurred seeding the DB.");

                }

            }

            host.Run();

You could write it all the views in DataTables but it is easier to auto generate all the CRUD view pages and controllers using Entity Framework Core from the models we made and edit the pages later. We will generate the views for our RecipeIngredient model with this code in the terminal.

dotnet aspnet-codegenerator controller -name RecipeIngredientController -m RecipeIngredient -dc CookingContext --relativeFolderPath Controllers --useDefaultLayout –referenceScriptLibraries

If all is ok then the RecipeIngredient folder generates with all the views, Index, Edit,Details,Delete. Now in the second div tag of the header add this list item to the navbar.

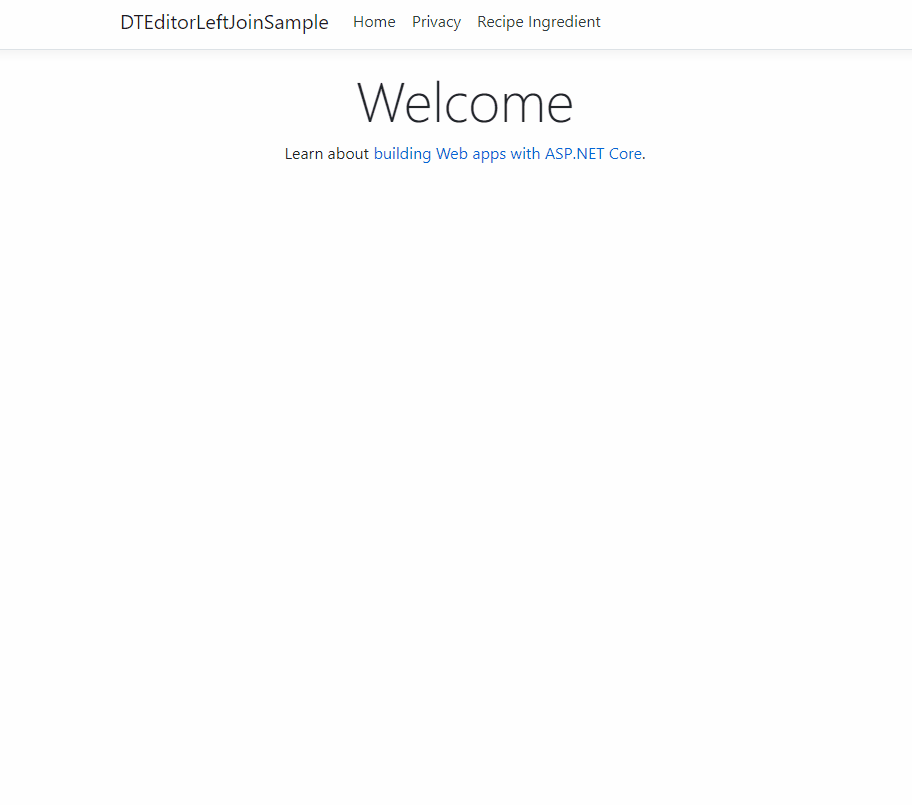
                        <li class="nav-item">

                            <a class="nav-link text-dark" asp-area="" asp-controller="RecipeIngredient" asp-action="Index">Recipe Ingredient</a>

                        </li>

Press F5. Running the program will generate the database and fill out the tables. If it doesn’t please install the SQL Server (mssql) extension to fill the tables manually.

If all goes well you should be able to go directly to the table in the Index view and render the project.



We will have to edit the index to enable sorting, paging and searching. But first let us install DataTables Editor serverside libraries to do this.

To start with Data Tables Editor let us install DataTables. We are going to do this over a Content Delivery Network CDN.