Walkthrough 9 - EF Core Database Scaffolding, Sorting and Searching

Setup

This walkthrough will create an MVC application to help manage a hospital and demonstrate the capability of the database scaffolder and introduce sorting and searching.

1. Start SQL Server.
2. Start Visual Studio.
3. Click Create a new project.
4. Set language to **C#** and project type to **Web**.
5. Select the ASP.NET Core Web App (Model-View-Controller) template, click Next.
6. Set Project name to **Hospital**.
7. Set Location to a **folder of your choosing**.
8. Ensure Place solution and project in the same directory is not selected, click Next.
9. Set version to **.NET 5.0**, unselect Configure for HTTPS, click Create.
10. Download SQL script: CHDB.sql and drag it into the project in the Solution Explorer.

CHDB.sql

1. Open CHDB.sql.
2. In the upper-left, click the Execute button.
3. You will be prompted to connect to a database, set Server Name to **localhost\sqlexpress**, click Connect.
4. The script should complete and create the database.

appsettings.json

1. Add a connection string for the CHDB database.
2.
```
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft": "Warning",
      "Microsoft.Hosting.Lifetime": "Information"
    }
  },
  "AllowedHosts": "*",
  "ConnectionStrings": {
    "CHDB": "Server=localhost\\sqlexpress;Database=CHDB;Trusted_Connection=True"
  }
}
```
3. Save the file.

Scaffold-DbContext

1. Open the Package Manager Console (PMC) and issue the following commands.
2. **Install-Package Microsoft.EntityFrameworkCore.Tools**
3. **Install-Package Microsoft.EntityFrameworkCore.SqlServer**
4. **Scaffold-DbContext -Connection name=CHDB -Provider Microsoft.EntityFrameworkCore.SqlServer -OutputDir Models -ContextDir Data -DataAnnotations**
5. A Data and a Models folder will be created.
6. The CHDBContext.cs file will be open; briefly examine it.
7. Open the Models folder and note that all CHDB tables have been modeled.
8. Open the Patients.cs file and examine the class.

Startup.cs

1. The scaffolder doesn't inject the database into the startup class. Do this manually. Add the **using Hospital.Data;** and **using Microsoft.EntityFrameworkCore;** directives.
2.
```
public void ConfigureServices(IServiceCollection services)
{
    services.AddControllersWithViews();
    services.AddDbContext<CHDBContext>(options =>
        options.UseSqlServer(Configuration.GetConnectionString("CHDB")));
}
```
3. Save the file.

MedicationsController.cs

1. Right-click the Controller folder and select Add / Controller... .
2. Choose MVC Controller with views, using Entity Framework, click Add.
3. Set Model class to **Medications (Hospital.Models)**.
4. Set Data context class to **CHDBContext (Hospital.Data)**.
5. Accept default name of MedicationsController, click Add.
6. Examine the code.

_Layout.cshtml

1. Open Views / Shared / _Layout.cshtml and add a link for the Medications controller.
2.
```
...
<li class="nav-item">
    <a class="nav-link text-dark" asp-area="" asp-controller="Home" asp-action="Index">Home</a>
</li>
<li class="nav-item">
    <a class="nav-link text-dark" asp-area="" asp-controller="Medications" asp-action="Index">Medications</a>
</li>
<li class="nav-item">
    <a class="nav-link text-dark" asp-area="" asp-controller="Home" asp-action="Privacy">Privacy</a>
</li>
...
```
3. Save the file.
4. From the PMC, issue the command **dotnet watch run --project hospital** to run the site.
5. Click the Medications link.

Medication.cs

1. Add some annotations to enhance output.
2.
```
[Table("medications")]
public partial class Medication
{
    public Medication()
    {
        UnitDoseOrders = new HashSet<UnitDoseOrder>();
    }

    [Key]
    [Column("medication_id")]
    public int MedicationId { get; set; }
    [Required]
    [Column("medication_description")]
    [StringLength(25)]
    [Display(Name = "Description")]
    public string MedicationDescription { get; set; }
    [Column("medication_cost", TypeName = "decimal(9, 2)")]
    [Display(Name = "Cost")]
    public decimal? MedicationCost { get; set; }
    [Column("package_size")]
    [StringLength(20)]
    [Display(Name = "Package Size")]
    public string PackageSize { get; set; }
    [Column("strength")]
    [StringLength(20)]
    public string Strength { get; set; }
    [Column("sig")]
    [StringLength(20)]
    public string Sig { get; set; }
    [Column("units_used_ytd")]
    [Display(Name = "Units Used YTD")]
    public int? UnitsUsedYtd { get; set; }
    [Column("last_prescribed_date", TypeName = "date")]
    [Display(Name = "Last Prescribed")]
    [DataType(DataType.Date)]
    public DateTime? LastPrescribedDate { get; set; }

    [InverseProperty(nameof(UnitDoseOrder.Medication))]
    public virtual ICollection<UnitDoseOrder> UnitDoseOrders { get; set; }
}
```
3. Save the file, the browser should refresh automatically.

Index.cshtml

1. Trim a couple of columns from the view to make it narrower.
2.
```
...
<th>
    @Html.DisplayNameFor(model => model.PackageSize)
</th>
~~<th>~~
    ~~@Html.DisplayNameFor(model => model.Strength)~~
~~</th>~~
~~<th>~~
    ~~@Html.DisplayNameFor(model => model.Sig)~~
~~</th>~~
<th>
    @Html.DisplayNameFor(model => model.UnitsUsedYtd)
</th>
...
<td>
    @Html.DisplayFor(modelItem => item.PackageSize)
</td>
~~<td>~~
```

```
    ~~@Html.DisplayFor(modelItem => item.Strength)~~
~~</td>~~
~~<td>~~
    ~~@Html.DisplayFor(modelItem => item.Sig)~~
~~</td>~~
<td>
    @Html.DisplayFor(modelItem => item.UnitsUsedYtd)
</td>
...
```

3. Save the file.
4. Cost and units used YTD are numeric and should be right-justified. Add a CSS class for this.
5.
```
...
<th class="float-right">
    @Html.DisplayNameFor(model => model.MedicationCost)
</th>
...
<th class="float-right">
    @Html.DisplayNameFor(model => model.UnitsUsedYtd)
</th>
...
<td **class="float-right"**>
    @Html.DisplayFor(modelItem => item.MedicationCost)
</td>
...
<td **class="float-right"**>
    @Html.DisplayFor(modelItem => item.UnitsUsedYtd)
</td>
...
```

6. Save the file.

Create.cshtml

1. Click the Create New link.
2. Notice that the primary key is available to be specified by the user. This isn't ideal. It happened because the scaffolder recognized that the medications table doesn't have an auto-increment for medication id.
3. Delete MedicationId and give autofocus to MedicationDescription.
4.
```
...
<form asp-action="Create">
    <div asp-validation-summary="ModelOnly" class="text-danger"></div>
    ~~<div class="form-group">~~
        ~~<label asp-for="MedicationId" class="control-label"></label>~~
        ~~<input asp-for="MedicationId" class="form-control" />~~
        ~~<span asp-validation-for="MedicationId" class="text-danger"></span>~~
    ~~</div>~~
    <div class="form-group">
        <label asp-for="MedicationDescription" class="control-label"></label>
        <input asp-for="MedicationDescription" class="form-control" **autofocus** />
        <span asp-validation-for="MedicationDescription" class="text-danger"></span>
    </div>
...
```

5. Save the file.

MedicationsController.cs

1. Update the Post Create method to compute the new medication id.
2.
```
[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> Create([Bind("MedicationId,MedicationDescription,MedicationCost,PackageSize,Strength,Sig,UnitsUsedYtd,Las
tPrescribedDate")] Medication medication)
{
    if (ModelState.IsValid)
    {
        **var maxId = _context.Medications.Max(m => m.MedicationId);**
        **medication.MedicationId = maxId + 1;**
        _context.Add(medication);
        await _context.SaveChangesAsync();
        return RedirectToAction(nameof(Index));
    }
    return View(medication);
}
```

3. Save the file.
4. Add a new medication.

ErrorViewModel.cs

1. Attempt to delete an existing medication, it should fail with a referential integrity exception.
2. Add a description property to the error model.
3.
```
public class ErrorViewModel
{
    public string RequestId { get; set; }

    public bool ShowRequestId =< !string.IsNullOrEmpty(RequestId);

    **public string Description { get; set; }**
}
```

4. Save the file.

Error.cshtml

1. Add some code for the new Description property and remove the Development Mode text.
2.
```
@model ErrorViewModel
@{
    ViewData["Title"] = "Error";
}

<h1 class="text-danger">Error.</h1>
<h2 class="text-danger">An error occurred while processing your request.</h2>

@if (Model.ShowRequestId)
{
    <p><strong>Request ID:</strong> <code>@Model.RequestId</code></p>
}

<p><strong>Description:</strong> @Model.Description</p>

~~<h3>Development Mode</h3>~~
~~<p>~~
    ~~Swapping to <strong>Development</strong> environment will display more detailed information about the error that occurred.~~
~~</p>~~
~~<p>~~
    ~~<strong>The Development environment shouldn't be enabled for deployed applications.</strong>~~
    ~~It can result in displaying sensitive information from exceptions to end users.~~
    ~~For local debugging, enable the <strong>Development</strong> environment by setting the <strong>ASPNETCORE_ENVIRONMENT</strong> enviro~~
~~nment variable to <strong>Development</strong>~~
    ~~and restarting the app.~~
~~</p>~~
```
3. Save the file.

MedicationsController.cs

1. Add a try/catch to DeleteConfirmed.
2.
```
[HttpPost, ActionName("Delete")]
[ValidateAntiForgeryToken]
public async Task<IActionResult> DeleteConfirmed(int id)
{
    try
    {
        var medication = await _context.Medications.FindAsync(id);
        _context.Medications.Remove(medication);
        await _context.SaveChangesAsync();
        return RedirectToAction(nameof(Index));
    }
    catch
    {
        return View("Error",
            new ErrorViewModel
            {
                RequestId = id.ToString(),
                Description = $"Unable to delete medication id {id}. It is referenced by other data in the system."
            });
    }
}
```
3. Save the file.
4. Attempt another delete, the improved error message will be displayed.
5. In the PMC, click the stop button to end the web server.

Sorting

1. We are going to add functionality to allow the user to sort the data by either the MedicationDescription or MedicationCost column.
2. Update the Index method to accept a parameter for sort order.
3. Add two ViewBag entries that will track the state of the sort parameters.
4.
```
public async Task<IActionResult> Index(string sortOrder)
{
    ViewBag.DescriptionSortParm = string.IsNullOrEmpty(sortOrder) ? "description_desc" : "";
    ViewBag.CostSortParm = sortOrder == "cost" ? "cost_desc" : "cost";

    return View(await _context.Medications.ToListAsync());
}
```
5. Use LINQ to query the database.
6.
```
public async Task<IActionResult> Index(string sortOrder)
{
    ViewBag.DescriptionSortParm = string.IsNullOrEmpty(sortOrder) ? "description_desc" : "";
    ViewBag.CostSortParm = sortOrder == "cost" ? "cost_desc" : "cost";

    var medications = from m in _context.Medications
                      select m;

    return View(await _context.Medications.ToListAsync());
}
```
7. Sort the result set according to the current sort parameter.
8.
```
public async Task<IActionResult> Index(string sortOrder)
{
    ViewBag.DescriptionSortParm = string.IsNullOrEmpty(sortOrder) ? "description_desc" : "";
    ViewBag.CostSortParm = sortOrder == "cost" ? "cost_desc" : "cost";
    var medications = from m in _context.Medications
                      select m;

    switch (sortOrder)
    {
        case "description_desc":
            medications = medications.OrderByDescending(m => m.MedicationDescription);
            break;
        case "cost":
            medications = medications.OrderBy(m => m.MedicationCost);
            break;
        case "cost_desc":
            medications = medications.OrderByDescending(m => m.MedicationCost);
            break;
        default:
            medications = medications.OrderBy(m => m.MedicationDescription);
            break;
    }

    return View(await _context.Medications.ToListAsync());
}
```

9. Return the medications result set, which is sorted, with change tracking disabled.
10.
```
public async Task<IActionResult> Index(string sortOrder)
{
    ViewBag.DescriptionSortParm = string.IsNullOrEmpty(sortOrder) ? "description_desc" : "";
    ViewBag.CostSortParm = sortOrder == "cost" ? "cost_desc" : "cost";

    var medications = from m in _context.Medications
                        select m;

    switch (sortOrder)
    {
        case "description_desc":
            medications = medications.OrderByDescending(m => m.MedicationDescription);
            break;
        case "cost":
            medications = medications.OrderBy(m => m.MedicationCost);
            break;
        case "cost_desc":
            medications = medications.OrderByDescending(m => m.MedicationCost);
            break;
        default:
            medications = medications.OrderBy(m => m.MedicationDescription);
            break;
    }

    return View(await ~~_context.Medications~~medications.AsNoTracking().ToListAsync());
}
```

11. Save the file.

Index.cshtml

1. Open Views / Medications / Index.cshtml and add column heading hyperlinks to MedicationDescription and MedicationCost.
2.
```
...
<th>
    <a asp-action="Index"
       asp-route-sortOrder="@ViewBag.DescriptionSortParm">
        @Html.DisplayNameFor(model => model.MedicationDescription)
    </a>
</th>
<th>
    <a asp-action="Index"
       asp-route-sortOrder="@ViewBag.CostSortParm">
        @Html.DisplayNameFor(model => model.MedicationCost)
    </a>
</th>
...
```

3. Update the title and heading.
4.
```
@model IEnumerable<Hospital.Models.Medications>

@{
    ViewData["Title"] = "~~Index~~Medications";
}

<h1>~~Index~~@ViewData["Title"]</h1>

<p>
    <a asp-action="Create">Create New</a>
</p>
```

5. Save the file.

MedicationsController.cs

1. Add a breakpoint (F9) to the beginning of the Index method. Run in debug mode (F5).
2. Notice the new hyperlinks, hover over them and notice their URLs, try each of them several times and note what is happening with sortOrder.
3. Once done, stop debugging (Shift+F5) and remove the breakpoint.

Searching

1. We are going to add functionality to allow the user to search by the MedicationDescription column.

2. Update the Index method to accept an additional parameter for search.
3. Add a ViewBag entry that will track the search parameter.
4.
```
public async Task<IActionResult> Index(string sortOrder, string searchString)
{
    ViewBag.DescriptionSortParm = string.IsNullOrEmpty(sortOrder) ? "description_desc" : "";
    ViewBag.CostSortParm = sortOrder == "cost" ? "cost_desc" : "cost";
    ViewBag.SearchString = searchString;

    var medications = from m in _context.Medications
    ...
```
5. Use LINQ to search the result set.
6.
```
public async Task<IActionResult> Index(string sortOrder, string searchString)
{
    ViewBag.DescriptionSortParm = string.IsNullOrEmpty(sortOrder) ? "description_desc" : "";
    ViewBag.CostSortParm = sortOrder == "cost" ? "cost_desc" : "cost";
    ViewBag.SearchString = searchString;

    var medications = from m in _context.Medications
                      select m;

    if (!string.IsNullOrEmpty(searchString))
    {
        medications = medications.Where(m => m.MedicationDescription.Contains(searchString));
    }

    switch (sortOrder)
    ...
```
7. Save the file.

Index.cshtml

1. Add a text box for the search string, a button to perform the search and a link to clear the search.
2.
```
@model IEnumerable<Hospital.Models.Medications>

@{
    ViewData["Title"] = "Medications";
}

<h1>@ViewData["Title"]</h1>

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

<form asp-action="Index" method="get">
    <p>
        Search by Description: <input type="text" name="searchString" value="@ViewBag.SearchString" />
        <input type="submit" value="Search" class="btn btn-primary" /> |
        <a asp-action="Index">All Medications</a>
    </p>
</form>

<table class="table">
```
3. Save the file.
4. Run the site and test the search.
5. Do a search, then do a sort. Notice that the search is lost.
6. Update the routing for the search links with the search string.
7.
```
...
<th>
    <a asp-action="Index"
        asp-route-sortOrder="@ViewBag.DescriptionSortParm"
        asp-route-searchString="@ViewBag.SearchString">
        @Html.DisplayNameFor(model => model.MedicationDescription)
    </a>
</th>
<th>
    <a asp-action="Index"
        asp-route-sortOrder="@ViewBag.CostSortParm"
        asp-route-searchString="@ViewBag.SearchString">
        @Html.DisplayNameFor(model => model.MedicationCost)
    </a>
</th>
...
```
8. Save the file.
9. Run the site and test the search.
10. Do a search, then do a sort. Notice that the search isn't lost.