Walkthrough 10 - MVC Paging

Setup

This walkthrough will add paging to the Hospital site.

1. Start SQL Server.
2. Open Hospital from the end of the previous walkthrough.

PaginatedList.cs

1. In the Controllers folder, add a new class named **PaginatedList**.
2. Make the class handle any type and inherit from the List class.

3.
   ```
   public class PaginatedList<T> : List<T>
   {

   }
   ```
4. Add two properties to keep track of the current page and the total number of pages.

5.
   ```
   public class PaginatedList<T> : List<T>
   {
       public int PageIndex { get; private set; }
       public int TotalPages { get; private set; }
   }
   ```
6. Add a constructor that accepts a list of any type, the count of all items, the current page and the page size. The constructor updates the current page, the total pages and adds the items to the list.

7.
   ```
   public PaginatedList(List<T> items, int count, int pageIndex, int pageSize)
   {
       PageIndex = pageIndex;
       TotalPages = (int)Math.Ceiling(count / (double)pageSize);
       AddRange(items);
   }
   ```
8. Add two computed properties that determine if there is a previous page and a next page.

9.
   ```
   public bool HasPreviousPage
   {
       get
       {
           return PageIndex > 1;
       }
   }

   public bool HasNextPage
   {
       get
       {
           return PageIndex < TotalPages;
       }
   }
   ```
10. Add a static method that returns an instance of the class asynchronously. The method accepts a queryable list of any type, the current page and the page size.
11. The method counts the number of records in the data and then returns a page of them, add the **using Microsoft.EntityFrameworkCore;** directive.

12.
    ```
    public static async Task<PaginatedList<T>> CreateAsync(IQueryable<T> source, int pageIndex, int pageSize)
    {
        var count = await source.CountAsync();
        var items = await source.Skip((pageIndex - 1) * pageSize).Take(pageSize).ToListAsync();
        return new PaginatedList<T>(items, count, pageIndex, pageSize);
    }
    ```
13. Save the file.

MedicationsController.cs

1. Update the Index method to accept a parameter for current search and an optional page number.

2.
   ```
   public async Task<IActionResult> Index(
       string sortOrder,
       string searchString,
       string currentSearch,
       int? pageNumber)
   {
       ViewBag.DescriptionSortParm = String.IsNullOrEmpty(sortOrder) ? "description_desc" : "";
       ...
   ```
3. Add a ViewBag entry that will track the current sort order.

4.
   ```
   public async Task<IActionResult> Index(
       string sortOrder,
       string searchString,
       string currentSearch,
   ```

```
    int? pageNumber)
{
    ViewBag.CurrentSort = sortOrder;
    ViewBag.DescriptionSortParm = String.IsNullOrEmpty(sortOrder) ? "description_desc" : "";
    ViewBag.CostSortParm = sortOrder == "cost" ? "cost_desc" : "cost";
    ...
```

5. If the search string is changed during paging, the page has to be reset to 1, because the new search will likely result in different data to display. The search string is changed when a value is entered in the text box and the Submit button is pressed. In that case, the searchString parameter isn't null.
6.
```
public async Task<IActionResult> Index(
    string sortOrder,
    string searchString,
    string currentSearch,
    int? pageNumber)
{
    ViewBag.CurrentSort = sortOrder;
    ViewBag.DescriptionSortParm = String.IsNullOrEmpty(sortOrder) ? "description_desc" : "";
    ViewBag.CostSortParm = sortOrder == "cost" ? "cost_desc" : "cost";

    if (searchString != null)
    {
        pageNumber = 1;
    }
    else
    {
        searchString = currentSearch;
    }

    ViewBag.CurrentSearch = searchString;

    var medications = from m in _context.Medications
    ...
```

7. Set the page size to 3 and return a PaginatedList of Medications to a single page in a collection type that supports paging. That single page of medications is then passed to the view.
8.
```
public async Task<IActionResult> Index(
    string sortOrder,
    string searchString,
    string currentSearch,
    int? pageNumber)
{
    ViewBag.CurrentSort = sortOrder;
    ViewBag.DescriptionSortParm = String.IsNullOrEmpty(sortOrder) ? "description_desc" : "";
    ViewBag.CostSortParm = sortOrder == "cost" ? "cost_desc" : "cost";

    if (searchString != null)
    {
        pageNumber = 1;
    }
    else
    {
        searchString = currentSearch;
    }

    ViewBag.CurrentSearch = searchString;

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

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

    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;
    }

    var pageSize = 3;
    ~~return View(await medications.AsNoTracking().ToListAsync());~~
    return View(await PaginatedList<Medication>.CreateAsync(medications.AsNoTracking(), pageNumber ?? 1, pageSize));
}
```

9. Save the file.

Index.cshtml

1. Open Views / Medications / Index.cshtml and change the model to the paginated list of medications.
2.
```
~~@model IEnumerable<Hospital.Models.Medication>~~
@model Hospital.Controllers.PaginatedList<Medication>

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

3. Update the column headings to use the FirstOrDefault method on the model to retrieve their values.
4.
```
<table class="table">
    <thead>
        <tr>
            <th>
                <a asp-action="Index"
                   asp-route-sortOrder="@ViewBag.DescriptionSortParm"
                   asp-route-searchString="@ViewBag.CurrentSearch">
                    @Html.DisplayNameFor(model => model.FirstOrDefault().MedicationDescription)
                </a>
            </th>
            <th>
                <a asp-action="Index"
                   asp-route-sortOrder="@ViewBag.CostSortParm"
                   asp-route-searchString="@ViewBag.CurrentSearch">
                    @Html.DisplayNameFor(model => model.FirstOrDefault().MedicationCost)
                </a>
            </th>
            <th>
                @Html.DisplayNameFor(model => model.FirstOrDefault().PackageSize)
            </th>
            <th>
                @Html.DisplayNameFor(model => model.FirstOrDefault().UnitsUsedYtd)
            </th>
            <th>
                @Html.DisplayNameFor(model => model.FirstOrDefault().LastPrescribedDate)
            </th>
            <th></th>
        </tr>
    </thead>
    ...
```
5. Add two variables to control when paging links are disabled.
6.
```
...
</table>

@{
    var prevDisabled = Model.HasPreviousPage ? "" : "disabled";
    var nextDisabled = Model.HasNextPage ? "" : "disabled";
}
```
7. Add First, Previous, Next and Last links.
8.
```
...
@{
    var prevDisabled = Model.HasPreviousPage ? "" : "disabled";
    var nextDisabled = Model.HasNextPage ? "" : "disabled";
}

<ul class="pagination">
    <li class="page-item @prevDisabled">
        <a asp-action="Index"
           asp-route-sortOrder="@ViewBag.CurrentSort"
           asp-route-pageNumber="1"
           asp-route-currentSearch="@ViewBag.CurrentSearch"
           class="page-link">First</a>
    </li>
    <li class="page-item @prevDisabled">
        <a asp-action="Index"
           asp-route-sortOrder="@ViewBag.CurrentSort"
           asp-route-pageNumber="@(Model.PageIndex - 1)"
           asp-route-currentSearch="@ViewBag.CurrentSearch"
           class="page-link">Previous</a>
    </li>
    <li class="page-item @nextDisabled">
        <a asp-action="Index"
           asp-route-sortOrder="@ViewBag.CurrentSort"
           asp-route-pageNumber="@(Model.PageIndex + 1)"
           asp-route-currentSearch="@ViewBag.CurrentSearch"
           class="page-link">Next</a>
    </li>
    <li class="page-item @nextDisabled">
        <a asp-action="Index"
           asp-route-sortOrder="@ViewBag.CurrentSort"
           asp-route-pageNumber="@Model.TotalPages"
           asp-route-currentSearch="@ViewBag.CurrentSearch"
           class="page-link">Last</a>
    </li>
</ul>
```
9. Save the file.
10. Run the site and try the paging.
11. Search for **tyl**. Note that both the search and paging function, but the search term is lost from the text box.
12. Update the textbox to use either SearchString or CurrentSearch.
13.
```
...
<form asp-action="Index" method="get">
    <p>
        Search by Description: <input type="text" name="searchString" value="@(ViewBag.SearchString ?? ViewBag.CurrentSearch" />
        <input type="submit" value="Search" class="btn btn-primary" /> |
        <a asp-action="Index">All Medications</a>
    </p>
</form>
...
```
14. Save the file.
15. Run the site, search and page. The search term persists.
16. Search for **tyl**, then sort by Cost. The search term is lost.

17. Update the routing for the sorts to include the current search.
18.
```
...
<th>
    <a> asp-action="Index"
        asp-route-sortOrder="@ViewBag.DescriptionSortParm"
        asp-route-searchString="@ViewBag.SearchString"
        asp-route-currentSearch="@ViewBag.CurrentSearch">
        @Html.DisplayNameFor(model => model.FirstOrDefault().MedicationDescription)
    </a>
</th>
<th class="float-right">
    <a> asp-action="Index"
        asp-route-sortOrder="@ViewBag.CostSortParm"
        asp-route-searchString="@ViewBag.SearchString"
        asp-route-currentSearch="@ViewBag.CurrentSearch"&gt;
        @Html.DisplayNameFor(model => model.FirstOrDefault().MedicationCost)
    </a>
</th>
...
```
19. Save the file.
20. Run the site and search for **tyl**, then sort by cost. Functionality is now correct.

MedicationsController.cs

1. In the Index method, change the page size to 6.
2.
```
    ...
    }

    int pageSize = ~~3~~6;
    return View(await PaginatedList<Medications>.CreateAsync(medications.AsNoTracking(), pageNumber ?? 1, pageSize));
}
```
3. Save the file.
4. Refresh the page.

Index.cshtml

1. Add an alternate style of pagination.
2.
```
...
</ul>

<ul class="pagination">
    <li class="page-item @prevDisabled">
        <a asp-action="Index"
           asp-route-sortOrder="@ViewBag.CurrentSort"
           asp-route-pageNumber="@(Model.PageIndex - 1)"
           asp-route-currentSearch="@ViewBag.CurrentSearch"
           class="page-link">Previous</a>
    </li>
    @for (var i = 1; i <= Model.TotalPages; i++)
    {
        <li class="page-item @(i == Model.PageIndex ? "active" : "")">
            <a asp-action="Index"
               asp-route-sortOrder="@ViewBag.CurrentSort"
               asp-route-pageNumber="@i"
               asp-route-currentSearch="@ViewBag.CurrentSearch"
               class="page-link">@i</a>
        </li>
    }
    <li class="page-item @nextDisabled">
        <a asp-action="Index"
           asp-route-sortOrder="@ViewBag.CurrentSort"
           asp-route-pageNumber="@(Model.PageIndex + 1)"
           asp-route-currentSearch="@ViewBag.CurrentSearch"
           class="page-link">Next</a>
    </li>
</ul>
```
3. Save the file and run the site.