**ContosoUniversity**

public class IndexModel : PageModel

{

private readonly ContosoUniversity.Data.SchoolContext \_context;

private readonly IConfiguration Configuration;

public IndexModel(SchoolContext context, IConfiguration configuration)

{

\_context = context;

Configuration = configuration;

}

public string NameSort { get; set; }

public string DateSort { get; set; }

public string CurrentFilter { get; set; }

public string CurrentSort { get; set; }

public string CurrentPage { get; set; }

public string MaxPage { get; set; }

public PaginatedList<Student> Students { get; set; }

// public IList<Student> Students { get;set; } = default!;

public async Task OnGetAsync(string sortOrder, string currentFilter, string searchString, int? pageIndex)

{

CurrentSort = sortOrder;

CurrentFilter = searchString;

NameSort = String.IsNullOrEmpty(sortOrder) ? "name\_desc" : "";

DateSort = sortOrder == "Date" ? "date\_desc" : "Date";

if (searchString != null)

{

pageIndex = 1;

}

else

{

searchString = currentFilter;

}

IQueryable<Student> studentsIQ = from s in \_context.Students select s;

if (!String.IsNullOrEmpty(searchString))

{

studentsIQ = studentsIQ.Where(s => s.LastName.Contains(searchString) || s.FirstMidName.Contains(searchString));

}

switch (sortOrder)

{

case "name\_desc":

studentsIQ = studentsIQ.OrderByDescending(s => s.LastName);

break;

case "Date":

studentsIQ = studentsIQ.OrderBy(s => s.EnrollmentDate);

break;

case "date\_desc":

studentsIQ = studentsIQ.OrderByDescending(s => s.EnrollmentDate);

break;

default:

studentsIQ = studentsIQ.OrderBy(s => s.LastName);

break;

}

var pageSize = Configuration.GetValue("PageSize", 4);

Students = await PaginatedList<Student>.CreateAsync(

studentsIQ.AsNoTracking(), pageIndex ?? 1, pageSize);

CurrentPage = pageIndex==null ? "1":pageIndex.ToString();

MaxPage = pageSize.ToString();

}

}

@page

@model ContosoUniversity.Pages.Students.IndexModel

@{

ViewData["Title"] = "Index";

}

<h1>Index</h1>

<p>

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

</p>

<**form** **asp-page**="./Index" method="get">

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

<p>

Find by name:

<input type="text" name="SearchString" value="@Model.CurrentFilter" />

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

<**a** **asp-page**="./Index">Back to full List</**a**>

</p>

</div>

</**form**>

<table class="table">

<thead>

<tr>

<th>

<**a** **asp-page**="./Index" **asp-route-sortOrder**="@Model.NameSort">

@Html.DisplayNameFor(model => model.Students[0].LastName)

</**a**>

</th>

<th>

@Html.DisplayNameFor(model => model.Students[0].FirstMidName)

</th>

<th>

<**a** **asp-page**="./Index" **asp-route-sortOrder**="@Model.DateSort">

@Html.DisplayNameFor(model => model.Students[0].EnrollmentDate)

</**a**>

<th></th>

</tr>

</thead>

<tbody>

@foreach (var item in Model.Students) {

<tr>

<td>

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

</td>

<td>

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

</td>

<td>

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

</td>

<td>

<**a** **asp-page**="./Edit" **asp-route-id**="@item.ID">Edit</**a**> |

<**a** **asp-page**="./Details" **asp-route-id**="@item.ID">Details</**a**> |

<**a** **asp-page**="./Delete" **asp-route-id**="@item.ID">Delete</**a**>

</td>

</tr>

}

</tbody>

</table>

@{

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

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

}

<**a** **asp-page**="./Index"

**asp-route-sortOrder**="@Model.CurrentSort"

**asp-route-pageIndex**="@(Model.Students.PageIndex - 1)"

**asp-route-currentFilter**="@Model.CurrentFilter"

class="btn btn-primary @prevDisabled">

Previous

</**a**>

@Model.CurrentPage / @Model.MaxPage

<**a** **asp-page**="./Index"

**asp-route-sortOrder**="@Model.CurrentSort"

**asp-route-pageIndex**="@(Model.Students.PageIndex + 1)"

**asp-route-currentFilter**="@Model.CurrentFilter"

class="btn btn-primary @nextDisabled">

Next

</**a**>

public class CreateModel : PageModel

{

private readonly ContosoUniversity.Data.SchoolContext \_context;

public CreateModel(ContosoUniversity.Data.SchoolContext context)

{

\_context = context;

}

public IActionResult OnGet()

{

return Page();

}

//[BindProperty]

//public Student Student { get; set; } = default!;

[BindProperty]

public StudentVM StudentVM { get; set; }

// To protect from overposting attacks, see https://aka.ms/RazorPagesCRUD

public async Task<IActionResult> OnPostAsync()

{

if (!ModelState.IsValid)

{

return Page();

}

var entry = \_context.Add(new Student());

entry.CurrentValues.SetValues(StudentVM);

await \_context.SaveChangesAsync();

return RedirectToPage("./Index");

}

//public async Task<IActionResult> OnPostAsync()

//{

// var emptyStudent = new Student();

// if (await TryUpdateModelAsync<Student>(

// emptyStudent,

// "student", // Prefix for form value.

// s => s.FirstMidName, s => s.LastName, s => s.EnrollmentDate))

// {

// \_context.Students.Add(emptyStudent);

// await \_context.SaveChangesAsync();

// return RedirectToPage("./Index");

// }

// return Page();

//}

//public async Task<IActionResult> OnPostAsync()

//{

// if (!ModelState.IsValid)

// {

// return Page();

// }

// \_context.Students.Add(Student);

// await \_context.SaveChangesAsync();

// return RedirectToPage("./Index");

//}

}

@page

@model ContosoUniversity.Pages.Students.CreateModel

@{

ViewData["Title"] = "Create";

}

<h1>Create</h1>

<h4>StudentVM</h4>

<hr />

<div class="row">

<div class="col-md-4">

<**form** method="post">

<**div** **asp-validation-summary**="ModelOnly" class="text-danger"></**div**>

<div class="form-group">

<**label** **asp-for**="StudentVM.LastName" class="control-label"></**label**>

<**input** **asp-for**="StudentVM.LastName" class="form-control" />

<**span** **asp-validation-for**="StudentVM.LastName" class="text-danger"></**span**>

</div>

<div class="form-group">

<**label** **asp-for**="StudentVM.FirstMidName" class="control-label"></**label**>

<**input** **asp-for**="StudentVM.FirstMidName" class="form-control" />

<**span** **asp-validation-for**="StudentVM.FirstMidName" class="text-danger"></**span**>

</div>

<div class="form-group">

<**label** **asp-for**="StudentVM.EnrollmentDate" class="control-label"></**label**>

<**input** **asp-for**="StudentVM.EnrollmentDate" class="form-control" />

<**span** **asp-validation-for**="StudentVM.EnrollmentDate" class="text-danger"></**span**>

</div>

<div class="form-group">

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

</div>

</**form**>

</div>

</div>

<div>

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

</div>

@section Scripts {

@{await Html.RenderPartialAsync("\_ValidationScriptsPartial");}

}

public class PaginatedList<T> : List<T>

{

public int PageIndex { get; private set; }

public int TotalPages { get; private set; }

public PaginatedList(List<T> items, int count, int pageIndex, int pageSize)

{

PageIndex = pageIndex;

TotalPages = (int)Math.Ceiling(count / (double)pageSize);

this.AddRange(items);

}

public bool HasPreviousPage => PageIndex > 1;

public bool HasNextPage => PageIndex < TotalPages;

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);

}

}