2019-2022

.NET Core

REUSABLE SERVICES & HELPERS

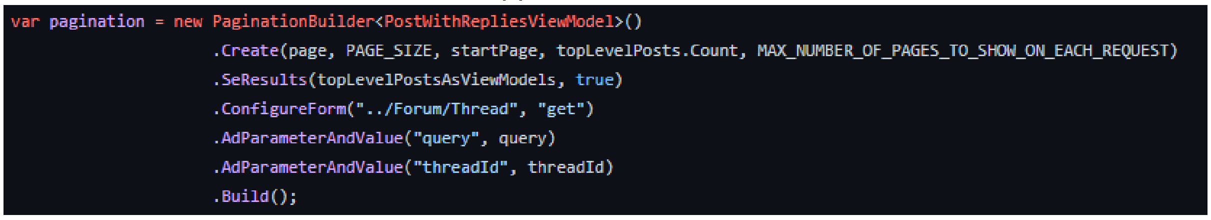
SONYA NADESAN

SONYA | https://github.com/SonyaNadesan | www.linkedin.com/in/sonya-nadesan |

Table of Contents

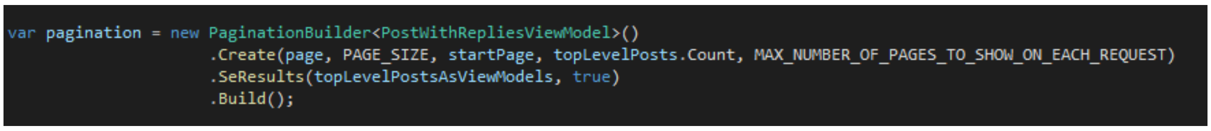
Pagination

Here is an example of creating a simple pagination for a list of results of type PostsWithRepliesViewModel, in an application where an MVC architecture is used.



```
var pagination = new PaginationBuilder<PostWithRepliesViewModel>()
                    .Create(page, PAGE_SIZE, startPage, topLevelPosts.Count, MAX_NUMBER_OF_PAGES_TO_SHOW_ON_EACH_REQUEST)
                    .SeResults(topLevelPostsAsViewModels, true)
                    .ConfigureForm("../Forum/Thread", "get")
                    .AdParameterAndValue("query", query)
                    .AdParameterAndValue("threadId", threadId)
                    .Build();
```

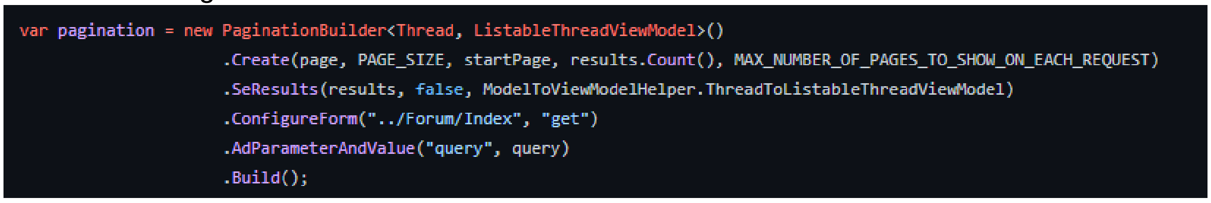
For single page applications or async features where page reload is not required, developers can ignore form configuration and data peristence:



```
var pagination = new PaginationBuilder<PostWithRepliesViewModel>()
                    .Create(page, PAGE_SIZE, startPage, topLevelPosts.Count, MAX_NUMBER_OF_PAGES_TO_SHOW_ON_EACH_REQUEST)
                    .SeResults(topLevelPostsAsViewModels, true)
                    .Build();
```

The variable pagination would hold information on the items to be displayed, the current page number, the start page, the last page, the page size, the total number of results, the total number of pages, the maximum number of pages to show on each request as well as information on form configuration and data persistence. All this information can be used to build a reusable pagination UI component – whether that be in .NET Razor or a JavaScript framework such as Anglar or React.

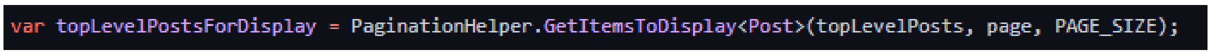
In the below example, we are converting Thread to a ListableThreadViewModel. Here, we pass in two arguments – the data class and the view-model class.



```
var pagination = new PaginationBuilder<Thread, ListableThreadViewModel>()
                    .Create(page, PAGE_SIZE, startPage, results.Count(), MAX_NUMBER_OF_PAGES_TO_SHOW_ON_EACH_REQUEST)
                    .SeResults(results, false, ModelToViewModelHelper.ThreadToListableThreadViewModel)
                    .ConfigureForm("../Forum/Index", "get")
                    .AdParameterAndValue("query", query)
                    .Build();
```

Other Classes/Interfaces Available for Pagination

PaginationHelper



```
var topLevelPostsForDisplay = PaginationHelper.GetItemsToDisplay<Post>(topLevelPosts, page, PAGE_SIZE);
```

Filtering

Like pagination, filtering is a common functionality in many web applications and there are many solutions to it - whether that be via SQL queries and stored procedures or ORMs such as Entity Framework (EF). Whatever you use, each has its own advantages and drawbacks. For example, SQL may be faster whilst an EF approach would allow developers to write cleaner code. In my solution, I have assumed that an ORM such as EF is used. I have developed an interface called IFilter<T>, where T is the type that the filter is for. This interface enforces developers to define a string property, Description – this is for debugging purposes, as well as a method that returns whether the item is valid against the given filter. To create a filter, simply create a class and implement the interface – remember to specify what the filter is for. For example, this a filter built for an item of type Thread for a forum. Once a collection of filters is created, the filter service can be used to run the filtering. To do this, create an instance of IFilterService for Threads, using Dependancy Injection, before calling the GetFilteredList method :

```
private readonly IFilterService<Thread> _threadFilterService;
```

```
var results = _threadFilterService.GetFilteredList(allThreads, filters).OrderByDescending(t => t.DateTime);
```

Other Classes/Interfaces Available for Filtering:

FilterBuilder<T>

IFilterBuilder<T>

Random List Generator

The RandomListGenerator is a generic helper class, allowing developers to get random items from a list of items, by specifying the list and the number of items to retrieve. If return count is null, the list is returned in a random order.

```
static List<T> GetRandomItems<T>(List<T> listOfItems, int? returnCount)
```

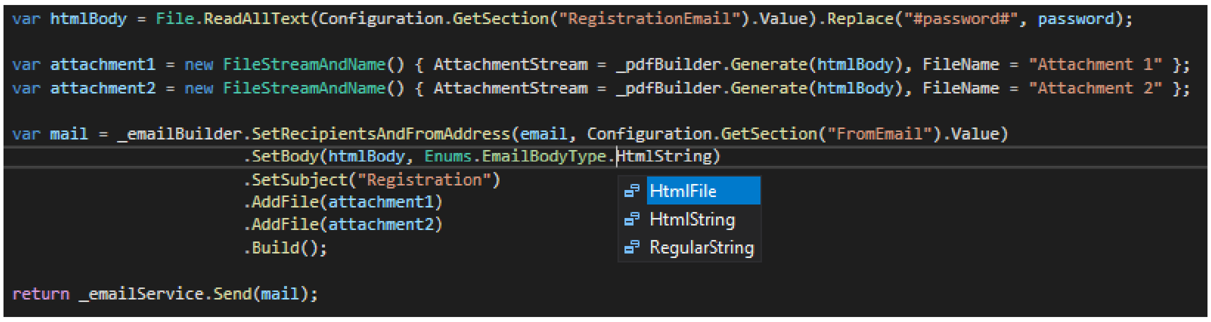
HTML to PDF Generation

Simply call the *Generate* method and pass in the html. You can use IPdfBuilder and PdfBuilder to enforce dependency injection:

```
_pdfBuilder.Generate(htmlBody)
```

Email Generation & Sending

The Email Builder allows developers to easily generate emails. The first parameter of the *SetBody* method is of type string. This string could be a path to a HTML file or a string that is itself HTML or simply just regular text; the second parameter is how we define the type. To send an email, use the EmailSenderService which implements IEmalSenderSrvice.



```
var htmlBody = File.ReadAllText(Configuration.GetSection("RegistrationEmail").Value).Replace("#password#", password);

var attachment1 = new FileStreamAndName() { AttachmentStream = _pdfBuilder.Generate(htmlBody), FileName = "Attachment 1" };
var attachment2 = new FileStreamAndName() { AttachmentStream = _pdfBuilder.Generate(htmlBody), FileName = "Attachment 2" };

var mail = _emailBuilder.SetRecipientsAndFromAddress(email, Configuration.GetSection("FromEmail").Value)
                        .SetBody(htmlBody, Enums.EmailBodyType.HtmlString)
                        .SetSubject("Registration")
                        .AddFile(attachment1)
                        .AddFile(attachment2)
                        .Build();

return _emailService.Send(mail);
```

File Upload

To upload a file, user the FileUploadService (and IFileUploadService to enforce Dependency Injection).

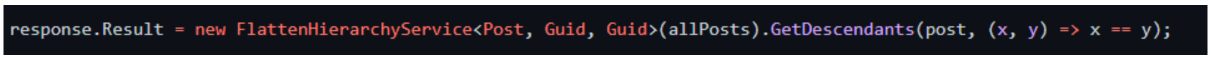
Flatten Hierarchy

This service is useful for requirements that are concerned with displaying hierarchical data. For example, navigation or comments on a forum. In terms of back-end, this is fairly straight forward to model. However, when it comes to rendering the data on the front-end, it is much easier to have a flat list of items in the order that they are to be rendered, as opposed to a hierarchical model. This service helps flatten a hierarchical structure into a flat list. There are two classes that represent FlattenHierarchyService:

- FlattenHierarchyService<T>
- FlattenHierarchyService<T, TId, TParentId>.

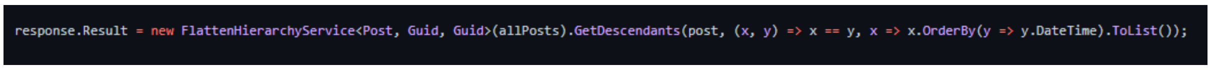
FlattenHierarchyService<T, TId, TParentId>

This service provides methods that allow developers to either get descendants or ancestors. The constructor expects all items available. Once this has been provided, developers can call the getDescendants or getAncestors method to get a flat list of items, by passing in an item followed by a delegate that would compare an Id with a ParentId. Developers may be wondering why this is necessary as most of the time, the Id and ParentId would be of the same type – but this may not always be the case. For example, an Id may be of type int, whilst a ParentId may be of type int? (nullable int).



```
response.Result = new FlattenHierarchyService<Post, Guid, Guid>(allPosts).GetDescendants(post, (x, y) => x == y);
```

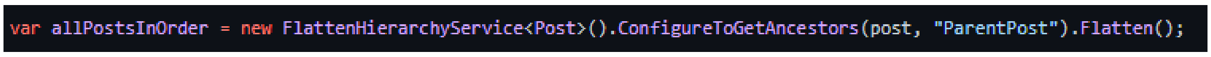
There is also an optional parameter, where developers can sort items at each level in the hierarchy in a specific order; in the following example, all items that share the same parent are sorted by date/time.



```
response.Result = new FlattenHierarchyService<Post, Guid, Guid>(allPosts).GetDescendants(post, (x, y) => x == y, x => x.OrderBy(y => y.DateTime).ToList());
```

FlattenHierarchyService<T>

This service provides ConfigureToGetAncestors or ConfigureToGetDescendants method, followed by a call to the Flatten method as shown below. Here, only the root object is passed along with the property name of the parent/children. Please note that where possible, use the FlattenHierarchyService<T, TId, TParentId> class as methods tend be more performant.



```
var allPostsInOrder = new FlattenHierarchyService<Post>().ConfigureToGetAncestors(post, "ParentPost").Flatten();
```

When retrieving descendants using this service, there is an optional parameter in the Flatten method, which allows developers to sort items that share the same parent in a specific way.