

Mariusz Wieczorek

mariusz.wieczorek@kabat.pl

Repozytorium, UnitOfWork, Serwisy

Jak w praktyce użyć tych wzorców

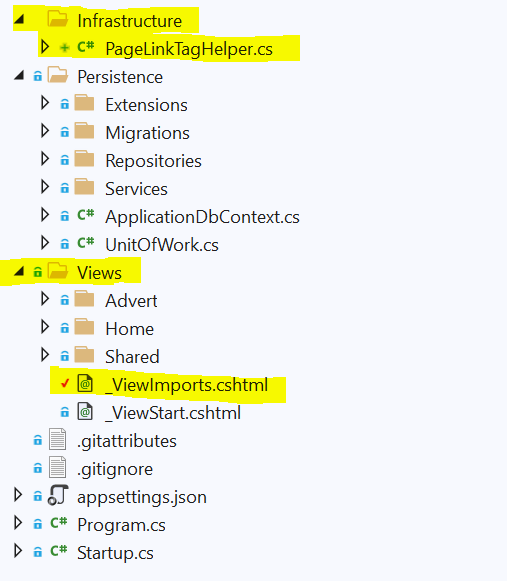
asp.net Core

Klasa Atrybutu Pomocniczego

**Dodanie klasy atrybutu pomocniczego**

Tworzymy katalog **Infrastructure** - w nim przechowujemy klasy komponentów aplikacji nie powiązanych z domeną

W tym katalogu tworzymy plik: **PageLinkTagHelper.cs**



Atrybuty pomocnicze znaczników w odróżnieniu od widoków i kontrolerów muszą być rejestrowanie robimy to w pliku: **Views\\_ViewImports.cshtml**

|  |
| --- |
| @using MyAdvert  @using MyAdvert.Models  @using MyAdvert.Core.ViewModels  @addTagHelper \*, Microsoft.AspNetCore.Mvc.TagHelpers  @addTagHelper MyAdvert.Infrastructure.\*, MyAdvert |

**Dodawanie Stronicowania ( paginacji )**

Aby zapewnić możliwość prawidłowego działania atrybutów pomocniczych znaczników musimy przekazać informacje o liczbie dostępnych stron, bieżącej stronie oraz całkowitej liczbie pozycji w repozytorium.

Najprostszym sposobem zrealizowania tego zadania jest utworzenie modelu widoku, który będzie używany do przekazywania danych między kontrolerem i widokiem.  
Tworzymy katalog Models/ViewModels i tworzymy w nim plik klasy PagingInfo.cs

|  |
| --- |
| namespace MyAdvert.Core.ViewModels  {  public class PagingInfo  {  public int TotalItems { get; set; }  public int ItemsPerPage { get; set; }  public int CurrentPage { get; set; }  public int TotalPages =>  (int)Math.Ceiling((decimal)TotalItems / ItemsPerPage);  }  } |

Musimy przekazać obiekt klasy **PagingInfo** do widoku, dołączamy go więc do modelu widoku listy ogłoszeń.

|  |
| --- |
| namespace MyAdvert.Core.ViewModels  {  public class AdvertsViewModel  {  public IEnumerable<Advert> Adverts { get; set; }  public IEnumerable<Category> Categories { get; set; }  public FilterAdverts FilterAdverts { get; set; }  public PagingInfo PagingInfo { get; set; }  }  } |

Metoda w Repozytorium

|  |
| --- |
| public IEnumerable<Advert> GetAdverts(FilterAdverts filterTasks, PagingInfo pagingInfo)  {  var adverts = \_context.Adverts  .Include(x => x.Category).AsQueryable();  if (filterTasks.IsActive == true)  adverts = adverts.Where(x => x.IsActive == filterTasks.IsActive);  if (filterTasks.CategoryId != 0)  adverts = adverts.Where(x => x.CategoryId == filterTasks.CategoryId);  if (!string.IsNullOrWhiteSpace(filterTasks.Title))  adverts = adverts.Where(x => x.Title.Contains(filterTasks.Title));  if (pagingInfo != null)  {  adverts = adverts  .Skip((pagingInfo.CurrentPage - 1) \* pagingInfo.ItemsPerPage)  .Take(pagingInfo.ItemsPerPage);  }  return adverts.OrderBy(x => x.StartDate).ToList();  } |

Wyświetlanie łączy stron  
Mamy model widoku, który zawiera dane stronicowania  
został zaktualizowany kontroler aby te dane zostały przekazane do widoku  
czyli metoda List wywołuje metodę View przekazując do niej repozytorium oraz model widoku

Pozostało dodanie elementu HTML, który będzie przetwarzany przez atrybut pomocniczy znacznika  
w celu utworzenia łączy stron

|  |
| --- |
| using Microsoft.AspNetCore.Mvc;  using Microsoft.AspNetCore.Mvc.Rendering;  using Microsoft.AspNetCore.Mvc.Routing;  using Microsoft.AspNetCore.Mvc.ViewFeatures;  using Microsoft.AspNetCore.Razor.TagHelpers;  using MyAdvert.Core.ViewModels;  using System.Collections.Generic;  // klasa atrybutu pomocniczego znacznika  // umieszcza on w elemencie <div> znaczniki <a> odpowiadające stronom produktów  // interfejs IUrlHelper i metody Action() jest najwygodniejszym sposobem generowania łączy wychodzących  // w atrybucie pomocniczym znacznika  namespace MyAdvert.Infrastructure  {  [HtmlTargetElement("div", Attributes = "page-model")]  public class PageLinkTagHelper : TagHelper  {  private IUrlHelperFactory urlHelperFactory;  public PageLinkTagHelper(IUrlHelperFactory helperFactory)  {  urlHelperFactory = helperFactory;  }  [ViewContext]  [HtmlAttributeNotBound]  public ViewContext ViewContext { get; set; }  public PagingInfo PageModel { get; set; }  public string PageAction { get; set; }  // sposobem na otrzymywanie informacji dodatkowych z widoku bez konieczności dodawania kolejnych właściwości  // do klasy atrybutu pomocniczego jest wykorzystanie przyjaznej funkcji pozwalającej  // wszystkie właściwości o takim samym prefiksie utrzymywać w jednej kolekcji  // oznaczenie właściwości atrybutu pomocniczego znacznika  // atrybutem HtmlAttributeName pozwala na określenie prefixu dla nazw atrybutu elementu w omawianym przykładzie to page-url-  // Wartość dowolnego atrybutu, którego nazwa rozpoczyna się od podanego prefiksu do słownika przypisanego właściwości PageUrlValues  // Ten słownik zostanie przekazany metodzie IUrlHelper.Action()  /\* poniższy kod w pliku List.cshtml odpowiedzialny za wstawienie numeracji stron,  \* dzięki powyższemu możemy wpisać: page-url-category="@Model.CurrentCategory"  \* gdy użytkownik kliknie na łącze kategoria będzie przekazana do metody akcji list()  \* i filter zostanie zachowany  \*  <div page-model="@Model.PagingInfo" page-action="List" page-classes-enabled="true"  page-class="btn"  page-class-normal="btn-secondary"  page-class-selected="btn-primary"  page-url-category="@Model.CurrentCategory"  class="btn-group btn-group-sm m-1">  </div>  \*/  [HtmlAttributeName(DictionaryAttributePrefix = "page-url-")]  public Dictionary<string, object> PageUrlValues { get; set; }  = new Dictionary<string, object>();  public bool PageClassesEnabled { get; set; } = false;  public string PageClass { get; set; }  public string PageClassNormal { get; set; }  public string PageClassSelected { get; set; }  // tworzymy w pętli znaczniki <a> - tyle znaczników ile  public override void Process(TagHelperContext context, TagHelperOutput output)  {  IUrlHelper urlHelper = urlHelperFactory.GetUrlHelper(ViewContext);  TagBuilder result = new TagBuilder("div");  for (int i = 1; i <= PageModel.TotalPages; i++)  {  TagBuilder tag = new TagBuilder("a");  PageUrlValues["currentPage"] = i;  // Metoda Action() dostarczana przez interfejs IUrlHelper jest najwygodniejszym sposobem generowania łączy wychodzących  tag.Attributes["href"] = urlHelper.Action(PageAction, PageUrlValues);  if (PageClassesEnabled)  {  tag.AddCssClass(PageClass);  tag.AddCssClass(i == PageModel.CurrentPage  ? PageClassSelected : PageClassNormal);  }  tag.InnerHtml.Append(i.ToString());  result.InnerHtml.AppendHtml(tag);  }  output.Content.AppendHtml(result.InnerHtml);  }  }  } |

do pliku Views\Product\List.cshtml dodajemy:

<div page-model="@Model.PagingInfo" page-action="List"></div>

jest to bardzo prosty zestaw linków bez żadnych ozdobników