Лабораторная работа №6

Вспомогательные классы тэгов (tag-helpers). Ajax. Маршрутизация

1. Цель работы.

Подробное знакомство с тэг-хелперами. Написание своих тэг-хелперов.

Знакомство с технологией Ajax.

Знакомство с системой маршрутизации ASP.NET Core

2. Общие сведения.

3. Выполнение работы

3.1. Исходные данные

Используйте проект из лабораторной работы №5.

3.2. Задание №1

Опишите тэг-хелпер, реализующий вывод на страницу пейджера (навигация по страницам списка товаров). Пейджер должен представлять собой кнопки доступных номеров страниц. При клике на цифру должен осуществиться переход на выбранную страницу.

Разметку пейджера оформите в виде частичного представления.

Для оформления пейджера используйте компонент bootstrap pagination (https://getbootstrap.com/docs/4.3/components/pagination/).

Пример применения такого тэг-хепера:

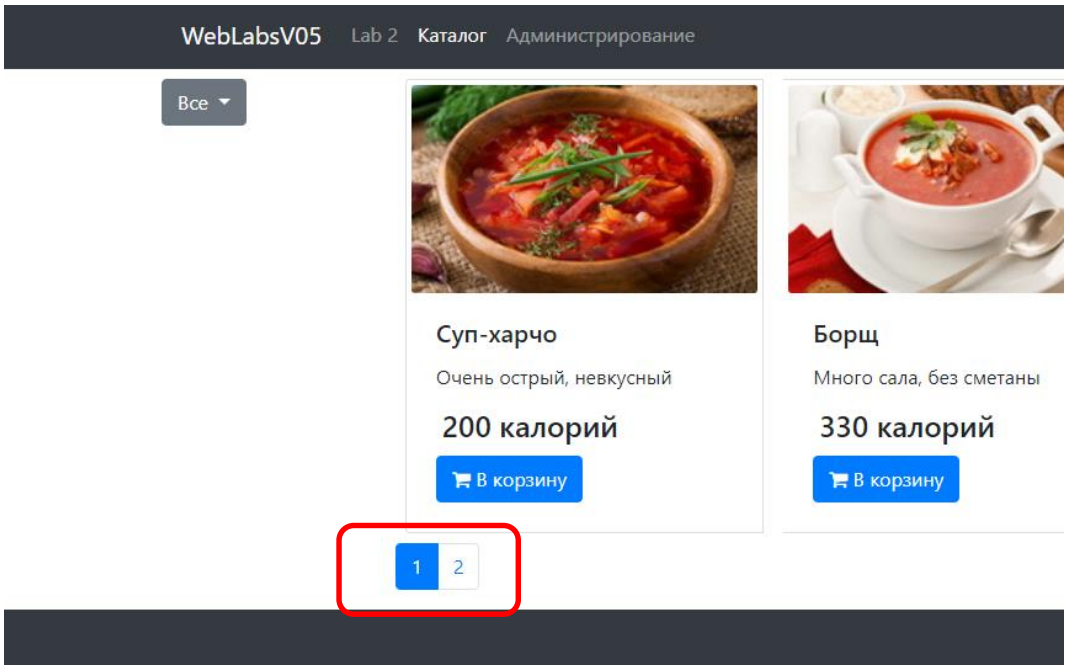
```
<pager page-current="@Model.CurrentPage"
      page-total="@Model.TotalPages"
      action="Index"
      controller="Product"
      group-id="currentGroup"></pager>
```

Назначение атрибутов:

- page-current : номер текущей страницы;
- page-total : количество страниц;
- action : имя действия;

- controller : имя контроллера;
- group-id : id группы объектов.

Пример оформления пейджера:



3.2.1. Рекомендации к заданию №1

Для создания пейджера вам понадобится следующая информация: номер текущей страницы и общее количество страниц. Данная информация содержится в модели представления - классе ListViewModel (см. лабораторную работу №6).

Id группы объектов (currentGroup) уже есть на представлении Index (использовался при формировании списка групп).

3.3. Задание №2

В частичном представлении _UserPartial.cshtml изображение аватара получается с помощью метода контроллера.

Напишите тэг-хелпер для тэга <img> , преобразующий атрибуты «img-action» и «img-controller» в атрибут «src» с правильным адресом.

Пример использования:

```
<img img-action="GetAvatar"
     img-controller="Image" />
```

Результирующая разметка:

```
<img src = "/Image/GetAvatar">
```

3.4. Задание №3

При переключении между страницами обновление списка должно происходить асинхронно по технологии Ajax (без обновления всей страницы)

3.4.1. Рекомендации к заданию №3

В методе Index контроллера Product необходимо выполнить проверку, получен обычный или асинхронный запрос. В случае асинхронного запроса метод должен вернуть частичное представление. Проверяется что заголовок «**x-requested-with**» запроса имеет значение «**XMLHttpRequest**».

Для того, чтобы при асинхронном запросе изменилась адресная строка браузера, можно использовать функцию javasript:

```
history.pushState(null, null, url)
```

где url – адрес, по которому отправлялся асинхронный запрос.

При переключении страницы не забудьте изменить выделение текущей страницы в пейджере.

3.5. Задание №4

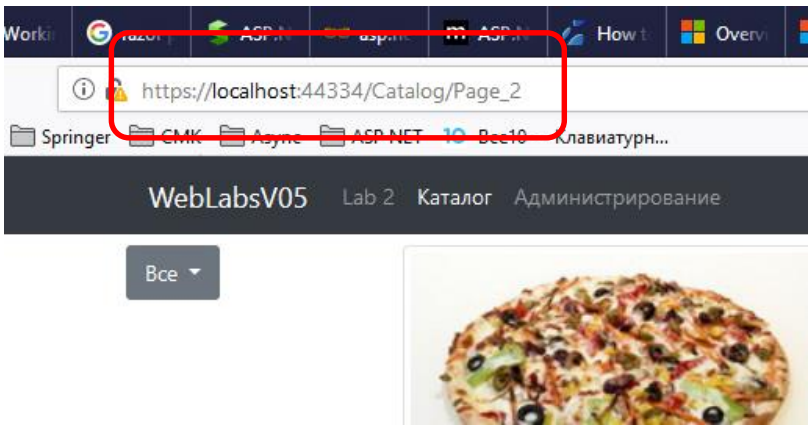
Оформите проверку запроса Ajax в виде расширяющего метода класса HttpRequest. Измените код метода Index с использованием созданного расширяющего метода.

3.6. Задание №5

При обращении к методу Index контроллера Product адресная строка браузера должна выглядеть так:

localhost:xxxx/Catalog или **localhost:xxxx/Catalog/Page_n**,

где n – номер выбранной страницы:



3.7. Задание №6

Разместить информацию об объектах и группах объектов в базе данных. Заполнить базу начальными данными.

3.7.1. Рекомендации к заданию №6

Добавьте классы сущностей в контекст базы данных. Выполните миграцию базы данных.

Заполнение начальными данными выполните в классе DbInitializer. Для заполнения базы используйте код инициализации данных в классе ProductController.

Поскольку в нашей базе данных ключевые поля назначаются автоматически, уберите назначение DishId и DishGroupId в коде инициализации.

3.8. Задание №7

Измените код контроллера Product для использования базы данных

3.8.1. Рекомендации к заданию №7

Внедрите в конструктор контроллера контекст базы данных

4. Пример выполнения работы

4.1. Тэг-хелпер для пейджера

4.1.1. Создание тэг-хелпера

Добавьте в проект папку TagHelpers.

Добавьте в папку TagHelpers класс PagerTagHelper:

```
using Microsoft.AspNetCore.Mvc.Rendering;
using Microsoft.AspNetCore.Razor.TagHelpers;
using Microsoft.AspNetCore.Routing;

namespace WebLabs_BSUIR_V01.TagHelpers
{
    public class PagerTagHelper : TagHelper
    {
        LinkGenerator _linkGenerator;
        // номер текущей страницы
        public int PageCurrent { get; set; }
        // общее количество страниц
        public int PageTotal { get; set; }
        // дополнительный css класс пейджера
        public string PagerClass { get; set; }
        // имя action
        public string Action { get; set; }
        // имя контроллера
        public string Controller { get; set; }
        public int? GroupId { get; set; }

        public PagerTagHelper(LinkGenerator linkGenerator)
        {
            _linkGenerator = linkGenerator;
        }

        public override void Process(TagHelperContext context,
TagHelperOutput output)
        {
            // контейнер разметки пейджера
            output.TagName = "nav";

            // пейджер
            var ulTag = new TagBuilder("ul");
            ulTag.AddCssClass("pagination");
            ulTag.AddCssClass(PagerClass);

            for (int i = 1; i <= PageTotal; i++)
            {
                var url = _linkGenerator.GetPathByAction(Action,
Controller,
```

```
                    new
                        {
                            pageNo = i,
                            group = GroupId == 0
                                        ? null
                                        : GroupId
                        });

                // получение разметки одной кнопки пейджера
                var item = GetPagerItem(
                        url: url, text: i.ToString(),
                        active: i == PageCurrent,
                        disabled: i == PageCurrent
                    );
                // добавить кнопку в разметку пейджера
                ulTag.InnerHtml.AppendHtml(item);
            }
            // добавить пейджер в контейнер
            output.Content.AppendHtml(ulTag);
        }

        /// <summary>
        /// Генерирует разметку одной кнопки пейджера
        /// </summary>
        /// <param name="url">адрес</param>
        /// <param name="text">текст кнопки пейджера</param>
        /// <param name="active">признак текущей страницы</param>
        /// <param name="disabled">запретить доступ к кнопке</param>
        /// <returns>объект класса TagBuilder</returns>
        private TagBuilder GetPagerItem(string url, string text,
                                        bool active = false,
                                        bool disabled = false)
        {
            // создать тэг <li>
            var liTag = new TagBuilder("li");
            liTag.AddCssClass("page-item");
            liTag.AddCssClass(active ? "active" : "");
            //liTag.AddCssClass(disabled ? "disabled" : "");

            // создать тэг <a>
            var aTag = new TagBuilder("a");
            aTag.AddCssClass("page-link");
            aTag.Attributes.Add("href", url);
            aTag.InnerHtml.Append(text);

            // добавить тэг <a> внутрь <li>
            liTag.InnerHtml.AppendHtml(aTag);

            return liTag;
        }
    }
```

```
}
```

4.1.2. Регистрация тэг-хелпера

Откройте файл _ViewImports.cshtml. Добавьте строку:

```
@addTagHelper " XXXX.TagHelpers.*, XXXX"
```

Здесь XXXX – имя вашего проекта.

4.1.3. Использование тэг-хелпера

На представлении Index добавьте разметку:

```
<div class="col-10">
    <div class="card-group">
        @foreach (var item in Model)
        {
            <partial name="_ListItemPartial" model="@item" />
        }
    </div>
    <pager page-current="@Model.CurrentPage"
           page-total="@Model.TotalPages"
           action="Index"
           controller="Product"
           group-id="@currentGroup"></pager>
</div>
```

Запустите проект. Проверьте, что пейджер отображается на странице и что страницы переключаются.

4.2. Тэг-хелпер аватара

4.2.1. Создание тэг-хелпера

Добавьте в папку TagHelpers класс ImageTagHelper:

```
using Microsoft.AspNetCore.Razor.TagHelpers;
using Microsoft.AspNetCore.Routing;

namespace WebLabs_BSUIR_V01.TagHelpers
{
    [HtmlTargetElement(tag:"img", Attributes ="img-action, img-
controller")]
    public class ImageTagHelper : TagHelper
    {
        public string ImgAction { get; set; }
        public string ImgController { get; set; }
        LinkGenerator _linkGenerator;
```

```
        public ImageTagHelper(LinkGenerator linkGenerator)
        {
            _linkGenerator = linkGenerator;
        }
        public override void Process(TagHelperContext context,
TagHelperOutput output)
        {
            var uri = _linkGenerator.GetPathByAction(ImgAction,
ImgController);
            output.Attributes.Add("src", uri);
        }
    }
}
```

4.2.2. Использование тэг-хелпера

Измените разметку аватара на частичном представлении _UserPartial.cshtml:

```
@if (signInManager.IsSignedIn(User))
{
    @await Component.InvokeAsync("Cart")
    <div class="dropdown ml-4 nav-color">
        . . .
            <img img-action="GetAvatar"
                 img-controller="Image"
                 width="30" alt="User"
                 class="rounded bg-light" />
        </div>
        <div class="dropdown-menu" aria-
labelledby="dropdownMenuButton">
            <div class="dropdown-item-text">
                <img img-action="GetAvatar"
                     img-controller="Image"
                     width="30" alt="User"
                     class="rounded bg-light" />
                @User.Identity.Name
            </div>
            . . . .
    </div>
}
```

Запустите проект, проверьте результат.

4.3. Асинхронный вызов страницы

4.3.1. Создание частичного представления

В нашем проекте асинхронно должна обновляться разметка списка и пейджера (разметка внутри тэга `<div class="card-group">`)

Создайте частичное представление _ListPartial в папке Views/Product.

```
@model ListViewModel<Dish>
      @foreach (var item in Model)
      {
            <partial name="_ListItemPartial" model="@item" />
      }
```

Измените разметку представления Index:

```
<div class="card-group" id="list">
            <partial name="_ListPartial" model="@Model" />
</div>
```

Обратите внимание, что добавлен атрибут «id» для контейнера div. Это нужно для поиска контейнера при выполнении запроса Ajax.

4.3.2. Скрипт для асинхронного запроса

Откройте файл site.js и поместите в него следующий код:

```
$(document).ready(function () {
    // подписать кнопки пейджера на событие click
    $(".page-link").click(function (e) {
        e.preventDefault();
        // получить адрес
        var uri = this.attributes["href"].value;
        // отправить асинхронный запрос и поместить ответ
        // в контейнер с id="list"
        $("#list").load(uri);
        // снять выделение с кнопки
        $(".active").removeClass("active disabled");
        // выделить текущую кнопку
        $(this).parent().addClass("active");
        // изменить адрес в адресной строке браузера
        history.pushState(null, null, uri);
    });
});
```

4.3.3. Изменение кода контроллера Product

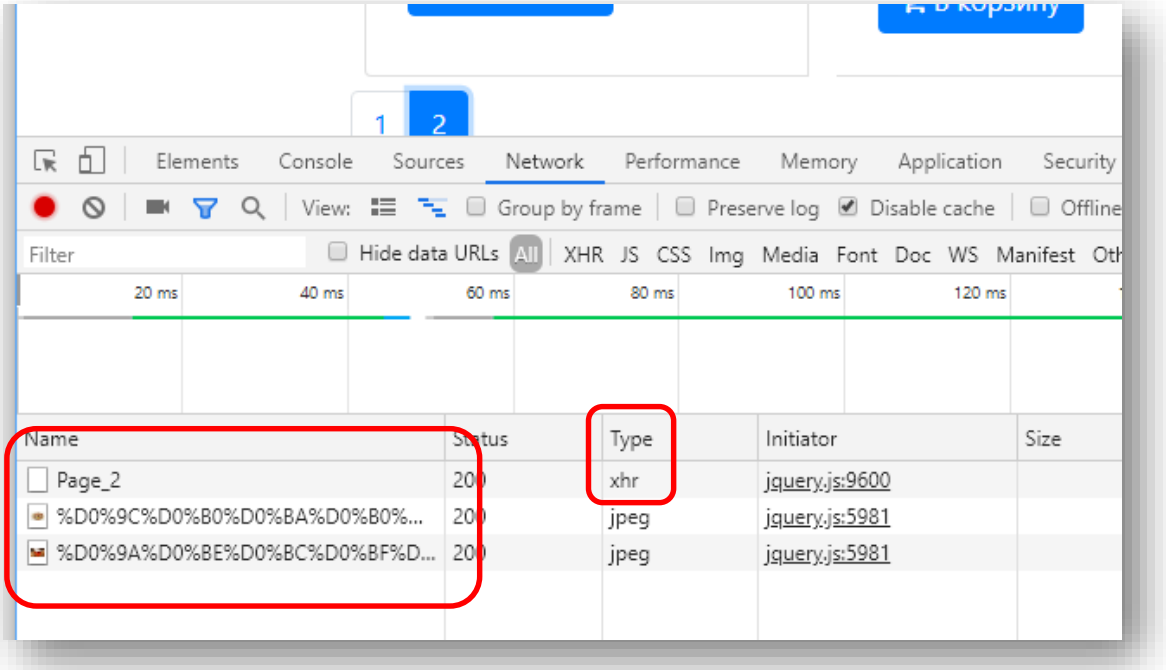
В методе Index контроллера Product добавьте проверку типа запроса:

```
public IActionResult Index(int? group, int pageNo=1)
{
            . . .

var model = ListViewModel<Dish>.GetModel(dishesFiltered, pageNo,
_pageSize);
if (Request.Headers["x-requested-with"]
      .ToString().ToLower().Equals("xmlhttprequest"))
      return PartialView("_listpartial", model);
else
      return View(model);
}
```

4.3.4. Проверка работы асинхронных вызовов

Запустите проект. Откройте режим разработчика в браузере (клавиша F12). Перейдите ко вкладке Сеть (Network). Выполняйте переключение между страницами. Убедитесь, что присылается только часть страницы, и что запрос посылается асинхронно:



4.4. Расширяющий метод класса HttpRequest

В проекте создайте папку Extensions.

В папку Extensions добавьте файл RequestExtensions.

```
using Microsoft.AspNetCore.Http;

namespace WebLabs_BSUIR_V01.Extensions
{
    public static class RequestExtensions
    {
        public static bool IsAjaxRequest(this HttpRequest request)
        {
            return request
                      .Headers["x-requested-with"]
                      .Equals("XMLHttpRequest");
        }
    }
}
```

Измените код контроллера Product для использования созданного расширяющего метода:

```
if (Request.IsAxaxRequest())
      return PartialView("_listpartial", model);
else
      return View(model);
```

4.5. Регистрация маршрута

В контроллере Product добавьте атрибуты маршрутизации перед методом Index:

```
[Route("Catalog")]
[Route("Catalog/Page_{pageNo}")]
        public IActionResult Index(int? group, int pageNo=1)
      {...}
```

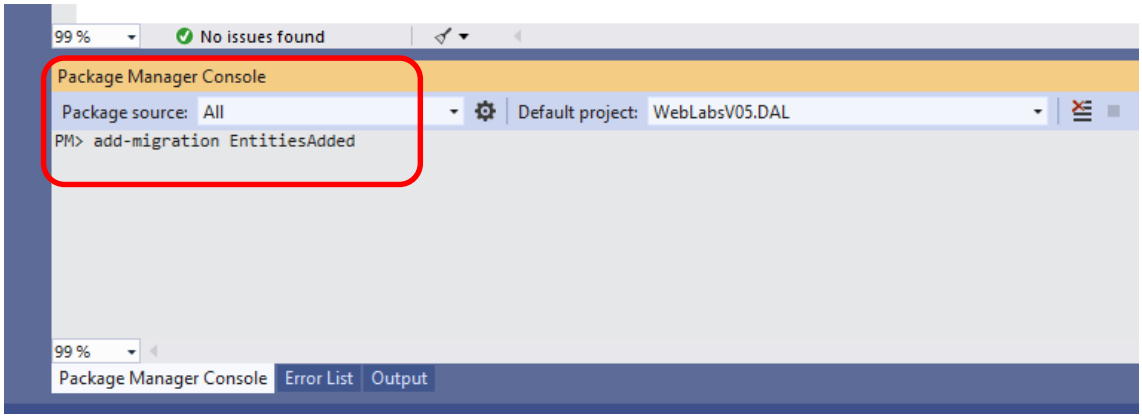
Запустите проект. Проверьте правильность формирования адреса при обращении к пункту меню Каталог и при переключении страниц списка.

4.6. Изменения в базе данных

В класс ApplicationDbContext добавьте сущности вашей предметной области, например:

```
public DbSet<Dish> Dishes { get; set; }
public DbSet<DishGroup> DishGroups { get; set; }
```

Выполните миграцию базы данных



В классе DbInitializer добавьте инициализацию данных сущностных классов:

```
//проверка наличия групп объектов
if(!context.DishGroups.Any())
{
      context.DishGroups.AddRange(
            new List<DishGroup>
            {
                  new DishGroup {GroupName="Стартеры"},
                  new DishGroup {GroupName="Салаты"},
                  new DishGroup {GroupName="Супы"},
                  new DishGroup {GroupName="Основные блюда"},
                  new DishGroup {GroupName="Напитки"},
                  new DishGroup {GroupName="Десерты"}
            });
            await context.SaveChangesAsync();
}

// проверка наличия объектов
if(!context.Dishes.Any())
{
      context.Dishes.AddRange(
            new List<Dish>
            {
                  new Dish {DishName="Суп-харчо",
                  Description="Очень острый, невкусный",
                  Calories =200, DishGroupId=3, Image="Суп.jpg" },
                  new Dish {DishName="Борщ",
                  Description="Много сала, без сметаны",
                  Calories =330, DishGroupId=3, Image="Борщ.jpg" },
                  new Dish {DishName="Котлета пожарская",
                  Description="Хлеб - 80%, Морковь - 20%",
                  Calories =635, DishGroupId=4, Image="Котлета.jpg" },
                  new Dish {DishName="Макароны по-флотски",
                  Description="С охотничьей колбаской",
                  Calories =524, DishGroupId=4, Image="Макароны.jpg" },
                  new Dish {DishName="Компот",
                  Description="Быстро растворимый, 2 литра",
```

```
                Calories =180, DishGroupId=5, Image="Компот.jpg" }
            });
        await context.SaveChangesAsync();
}
```

4.7. Использование базы данных в контроллере Product

Удалите локальные поля _dishes и _dishGroups, уберите вызов метода GetData из конструктора контроллера (данные будут получаться из контекста). Внедрите контекст базы данных в контроллер

```
using System.Collections.Generic;
using System.Linq;
using Microsoft.AspNetCore.Mvc;
using WebLabs_BSUIR_V01.Entities;
using WebLabsV05.Models;
using WebLabsV05.Extensions;
using WebLabs_BSUIR_V01.Data;

namespace WebLabsV05.Controllers
{
    public class ProductController : Controller
    {
        ApplicationDbContext _context;

        int _pageSize;

        public ProductController(ApplicationDbContext context)
        {
            _pageSize = 3;
            _context = context;
        }
        [Route("Catalog")]
        [Route("Catalog/Page_{pageNo}")]
        public IActionResult Index(int? group, int pageNo=1)
        {
            var dishesFiltered = _context.Dishes
                .Where(d => !group.HasValue || d.DishGroupId ==
group.Value);

            // Поместить список групп во ViewData
            ViewData["Groups"] = _context.DishGroups;

            // Получить id текущей группы и поместить в ViewData
            . . .;
        }
```

Запустите проект, проверьте результат.