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

Время выполнения работы: 6 часов

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.

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

Пример применения такого тэг-хепера:

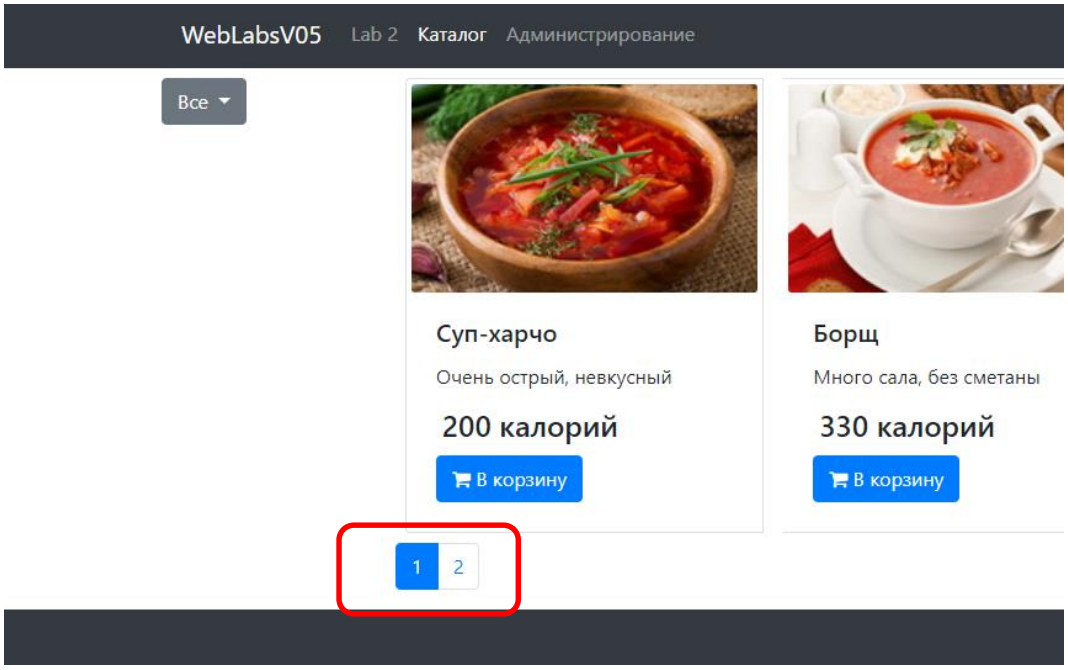
```
<pager page-current="@Model.CurrentPage"
       page-total="@Model.TotalPages"
       action="Index"
       controller="Product"
       group-id="currentGroup"></pager>
```

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 (для выделения текущей кнопки пейджера) и общее количество страниц (общее количество кнопок пейджера). 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).

Не забудьте зарегистрировать тэг-хэлпер в файле _ViewImports.cshtml:

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

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">
```

3.3.1. Рекомендации к заданию 2

Для генерирования правильного адреса внедрите в конструктор класса тэг-хелпера объект класса `LinkGenerator.`

В классе тэг-хелпера укажите, для какого элемента будет применяться данный тэг-хелпер, например:

```
[HtmlTargetElement(tag:"img",Attributes="img-action, img-controller")]
public class ImageTagHelper : TagHelper
{ . . . }
```

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

Создайте частичное представление _ListPartial в папке Views/Product. Данное частичное представление должно содержать разметку списка объектов и пейджер. Используйте его в представлении Index:

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

В методе Index контроллера Product необходимо выполнить проверку, получен обычный или асинхронный запрос. В случае асинхронного запроса метод должен вернуть частичное представление _ListPartial. Проверяется что

заголовок «**x-requested-with**» запроса имеет значение «**XMLHttpRequest**». Пример:

```
if (Request.Headers["x-requested-with"]
      .ToString().ToLower().Equals("xmlhttprequest"))
      return PartialView("_listpartial", model);
else
      return View(model);
```

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.

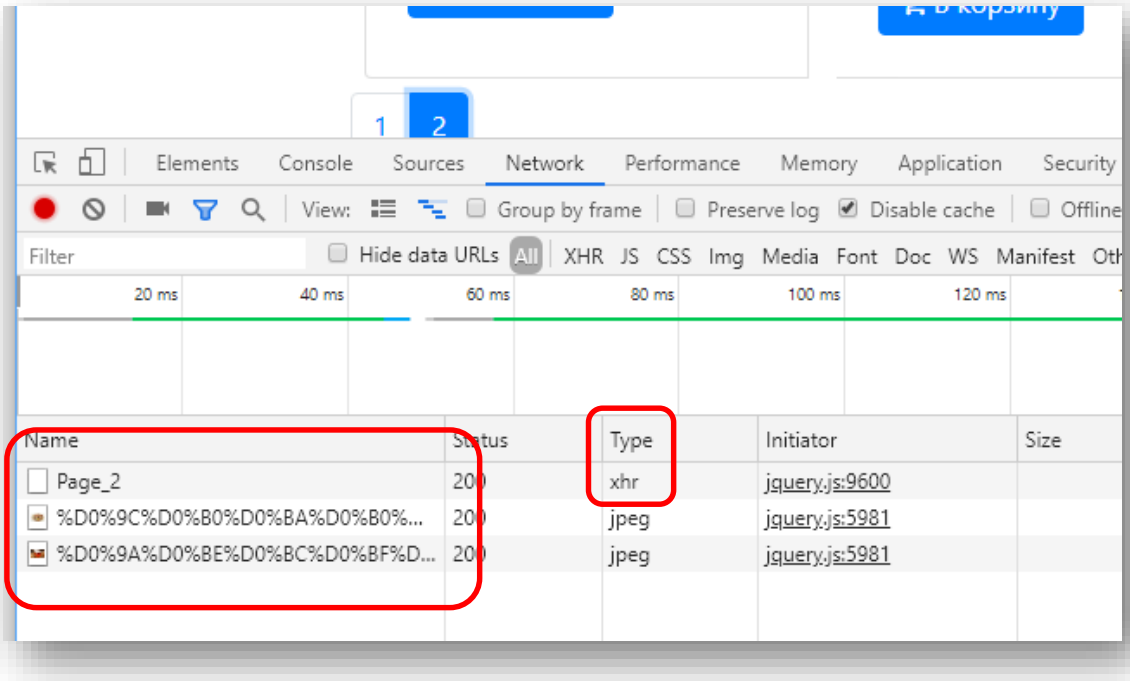
При переключении страницы не забудьте изменить выделение текущей страницы в пейджере.

Пример скрипта для асинхронной загрузки списка (в случае использования пейджера Bootstrap - bootstrap pagination версии 4.3):

```
$(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);
    });
});
```

3.5. Проверьте работу 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:



3.6. Задание №4

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

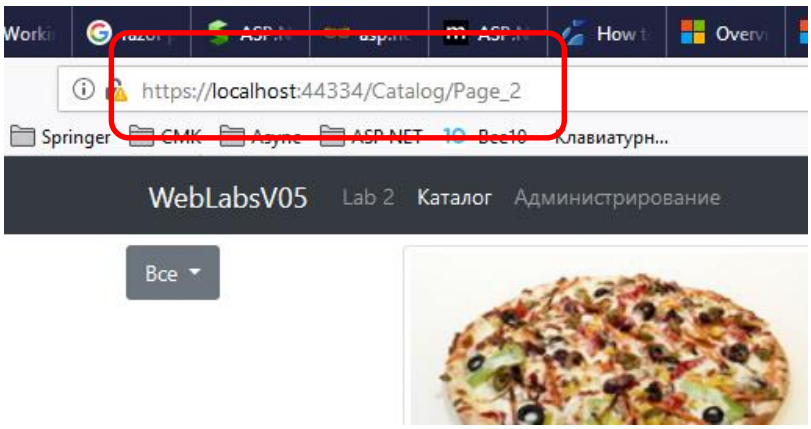
```
if (Request.IsAxaxRequest())
                return PartialView("_listpartial", model);
else
                return View(model);
```

3.7. Задание №5

Зарегистрируйте маршрут, чтобы при обращении к методу Index контроллера Product адресная строка браузера выглядела так:

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

где n – номер выбранной страницы:



3.7.1. Рекомендации к заданию 5

Используйте маршрутизацию с помощью атрибутов.

3.8. Задание №6

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

3.8.1. 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.

3.9. Задание №7

Измените код контроллера Product для использования базы данных

3.9.1. Рекомендации к заданию №7

Внедрите в конструктор контроллера контекст базы данных

4. Контрольные вопросы

Что такое tag-helpers?

Как в тэг-хелпере указать, к какому элементу применяется тэг-хелпер?

Как в тэг-хелпере получить объект текущего элемента разметки?

Как в тэг-хелпере сгенерировать url-адрес?

Как зарегистрировать в проекте использование тэг-хелперов?

Какие тэг-хелперы, предлагаемые Microsoft, вы знаете?

Какое преимущество использования тег-хелперов ссылок и форм?

Приложение.

Пример кода тэг-хелпера пейджера

(для bootstrap – pagination Ver. 4.3)

```
using Microsoft.AspNetCore.Mvc.Rendering;
using Microsoft.AspNetCore.Razor.TagHelpers;
using Microsoft.AspNetCore.Routing;
namespace XXX.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++)
            {
                // получение Url для одной кнопки пейджера
               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;
        }
    }
}
```