Лабораторная работа №6

Модульное тестирование.

1. Цель работы.

Знакомство с xUnit. Использование xUnit для тестирования классов контроллера.

2. Общие сведения.

Для проверки результата выполнения теста xUnit предоставляет класс Assert, который содержит статические методы – утверждения:

- **Contains** – проверяет, что строка содержит указанную подстроку или коллекция содержит указанный элемент;
- **DoesNotContain** – проверяет, что строка НЕ содержит указанную подстроку или коллекция НЕ содержит указанный элемент;
- **DoesNotThrow** – проверяет, что код НЕ генерирует исключение;
- **Equal** – проверяет, что два объекта эквивалентны;
- **False** – проверяет условие на false;
- **InRange** – проверяет, что величина находится в указанных границах;
- **IsAssignableFrom** – проверяет, что объект принадлежит указанному типу или наследуется от него;
- **IsNotType** – проверяет, что объект НЕ принадлежит указанному типу;
- **IsType** – проверяет, что объект принадлежит указанному типу;
- **NotEmpty** – Проверяет, что коллекция содержит элементы;
- **NotEqual** – проверяет, что два объекта НЕ эквивалентны;
- **NotInRange** – проверяет, что величина находится ВНЕ указанных границ;
- **NotNull** – проверяет, что объект не Null;

- **NotSame** – проверяет, что два объекта НЕ представляют собой одну и ту же сущность;
- **Null** – проверяет, что объект равен Null;
- **Same** – проверяет, что два объекта представляют собой одну и ту же сущность;
- **Throws** – проверяет, что код генерирует исключение;
- **True** – проверяет условие на true.

Пример проверки на генерирование исключения:

```
[Fact]
public void StackLimitIsControlled()
{
      //arrange
      var stack = new StackDemo();
      stack.StackLimit = 0;
      //act
      Action testCode = () => stack.Push("test");
      //assert
      Assert.Throws<StackDemo.StackFullException>(testCode);
         }
```

xUnit поддерживает два разных типа модульных тестов - Fact и Theory. Тип теста указывается в виде атрибута.

Fact используется, когда у нас есть некоторые критерии, которые всегда должны выполняться независимо от данных.

Theory зависит от множества параметров и ее данных. Тест пройдет для определенного набора данных, а не других. Для передачи набора исходных данных используются атрибуты [InlineData], [ClassData], [MemberData]

InlineData - позволяет передавать простые объекты:

```
[Theory]
 [InlineData(2, 4, .5)]
 [InlineData(2, 4, 1)]
 [InlineData(3, 4, (double)3/4)]
public void CanDivide(int a, int b, double c)
```

```
{

}
```

ClassData – атрибут, позволяющий в качестве источника набора данных использовать класс. Класс, используемый в качестве источника данных, должен наследоваться от `IEnumerable<object[]>`.

Пример использования:

```
[Theory]
[ClassData(typeof(DataSource))]
public void CanDivide(int a, int b, double c)
{

}
```

MemberData - атрибут позволяющий в качестве источника тестовых данных использовать метод. Метод должен возвращать `IEnumerable<object[]>`.

Пример использования:

```
[Theory]
[MemberData(nameof(DataSource.GetTestData),
                MemberType =typeof(DataSource))]
public void CanDivide(int a, int b, double c)
{

}
```

3. Выполнение работы

3.1. Исходные данные

Используйте проект из лабораторной работы №5.

3.2. Задание №1

Реализуйте разбиение списка товаров на страницы (по 3 товара на странице). При обращении к нужной странице (например ко 2-й) адрес должен иметь вид /Product/Index?pageNo=2.

3.2.1. Рекомендации к заданию №1

В классе контроллера Product создайте свойство _pageSize – количество объектов на странице.

В метод Index контроллера Product передайте номер текущей страницы.

3.3. Задание №2

Добавьте в решение новый проект – «xUnit Test project (.Net Core)».

Напишите тест для метода Index контроллера Product. Тест должен проверить, что в представление передается список выбранной страницы.

3.3.1. Рекомендации к заданию №2

Для возможности передачи в контроллер тестового списка объектов сделайте свойство _dishes публичным (это противоречит парадигме инкапсуляции, но на данном этапе сделаем так)

3.4. Задание №3

В папке Models основного проекта создайте модель представления - класс ListViewModel, - имеющий свойства:

- номер текущей страницы;
- общее количество страниц;
- список объектов для отображения на текущей странице.

Класс должен сам делать выборку объектов из общего списка для текущей страницы.

Для создания класса удобно воспользоваться шаблоном проектирования «декоратор»: унаследуйте ваш класс от класса List<T>, добавив функционал для выборки данных и хранения информации о разбиении на страницы.

Для создания объекта класса ListViewModel опишите ***статический*** фабричный метод GetModel, который принимает исходный список объектов, номер текущей страницы и количество объектов на странице. Метод должен вернуть объект класса ListViewModel с уже сформированным списком объектов и значениями номера текущей страницы и общего количества страниц.

3.5. Задание №4

Напишите модульный тест для проверки класса метода GetModel класса ListViewModel.

3.6. Задание №5

Используйте класс ListViewModel в действии Index контроллера Product.

Измените соответственно модель в представлении Index.

3.7. Задание №6

Оформите разметку одного элемента списка в виде частичного представления

3.8. Задание №7

Добавьте на страницу index возможность выбора категории объектов. При выборе категории в метод Index должен передаваться id выбранной категории.

Выбор категории разместить на странице Index слева от списка объектов.

3.8.1. Рекомендации к заданию №7

Поскольку в метод Index должны передаваться два параметра (id группы и номер страницы), измените тест контроллера Product для передачи этих параметров.

Список категорий представляет собой набор ссылок (тэг <a>) на метод Index контроллера Product с передачей параметра «group» - id выбранной группы.

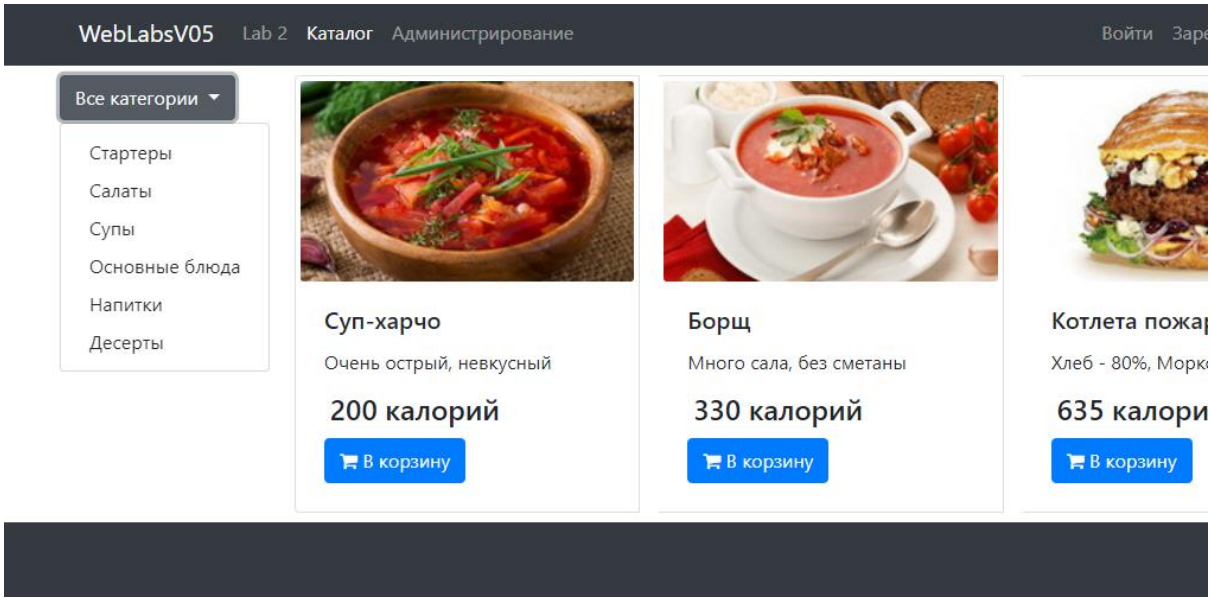
Список категорий можно оформить классом bootstrap «dropdown» (см. https://getbootstrap.com/docs/4.3/components/dropdowns/)

Список объектов категорий можно передать через ViewData. Также для выделения текущего выбора в представление нужно передать id текущей выбранной группы или «0», если выбраны все группы

id выбранной группы можно получить из запроса:

```
HttpContext.Request.Query["group"]
```

Для размещения выбора группы разбейте страницу на две колонки, например, «col-2» и «col-10».



3.9. Задание №8 (необязательное)

Напишите модульный тест, проверяющий, что метод Index контроллера Product делает правильную выборку объектов по группе.

4. Пример выполнения работы

4.1. Вывод по три объекта на страницу

4.1.1. Изменения в классе контроллера

```
public class ProductController : Controller
    {
        List<Dish> _dishes;
        List<DishGroup> _dishGroups;

        int _pageSize;
```

```
        public ProductController()
        {
            _pageSize = 3;
            SetupData();
        }

        public IActionResult Index(int pageNo=1)
        {
            var items = _dishes
                          .Skip((pageNo - 1)*_pageSize)
                          .Take(_pageSize)
                          .ToList();
            return View(items);
        }
    . . .
}
```

Запустите проект. На страницу должны выводиться 3 первых объекта.

Добавьте в строку запроса «?pageNo=2». Запустите проект. Должен выводиться список объектов 2-й страницы.

4.2. Тестирование контроллера

4.2.1. Изменение в классе контроллера

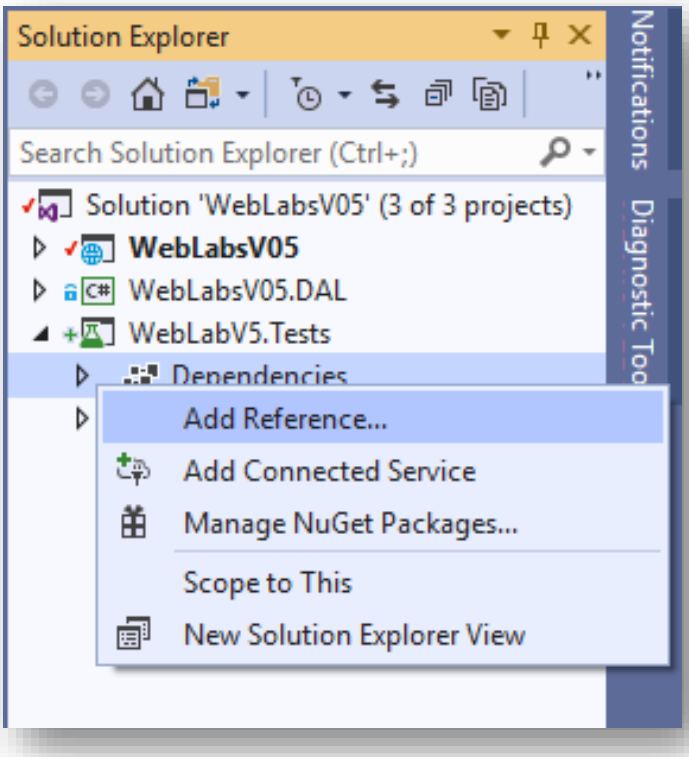
```
public List<Dish> _dishes;
```

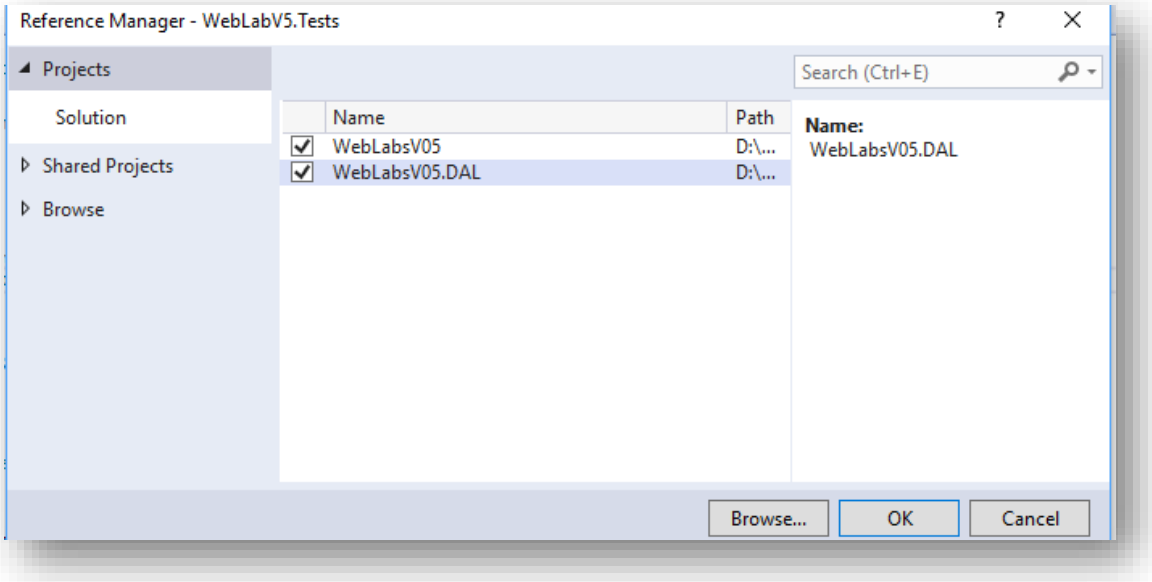
4.2.2. Добавление проекта модульных тестов

Добавьте проект тестов xUnit для .Net Core.

Назовите проект «XXX.Tests», где XXX-название вашего основного проекта.

Добавьте в проект ссылки на основной проект и проект XXX.DAL:





Добавьте в проект NuGet пакет Microsoft.AspNetCore.Mvc.

4.2.3. Модульный тест метода Index контроллера Product

Добавьте в проект класс ProductControllerTests. Добавьте пространства имен:

```
using Microsoft.AspNetCore.Mvc;
using System.Collections.Generic;
using WebLabsV05.Controllers;
using WebLabsV05.DAL.Entities;
using Xunit;
```

Напишите тест для метода Index:

```
[Theory]
[InlineData(1,3,1)] // 1-я страница, кол. объектов 3, id первого
объекта 1
[InlineData(2, 2, 4)] // 2-я страница, кол. объектов 2, id первого
объекта 4
public void ControllerGetsProperPage(int page, int qty, int id)
{
      // Arrange
      var controller = new ProductController();

      controller._dishes = new List<Dish>
      {
            new Dish{ DishId=1},
            new Dish{ DishId=2},
            new Dish{ DishId=3},
            new Dish{ DishId=4},
            new Dish{ DishId=5}
      };

      // Act
      var result = controller.Index(page) as ViewResult;
      var model = result?.Model as List<Dish>;

      // Assert
      Assert.NotNull(model);
      Assert.Equal(qty, model.Count);
      Assert.Equal(id, model[0].DishId);
}
```

Запустите тесты. Убедитесь, что тесты пройдены.

4.3. Создание модели представления

В паке Models основного проекта добавьте класс ListViewModel.

Унаследуйте класс от класса List<T>

Добавьте свойства CurrentPage (номер текущей страницы) и TotalPages (общее количество страниц).

Добавьте статический метод GetModel. Метод должен принимать список объектов, номер текущей страницы и количество объектов на странице, а

возвращать объект ListViewModel<T>, содержащий список из объектов для текущей страницы и значения CurrentPage и TotalPages:

```
public class ListViewModel<T> : List<T> where T:class
{
      public int TotalPages { get; set; }
      public int CurrentPage { get; set; }
      private ListViewModel(IEnumerable<T> items,
                            int total,
                            int current):base(items)
      {
            TotalPages = total;
            CurrentPage = current;
      }
      /// <summary>
      /// Получить модель представления списка объектов
      /// </summary>
      /// <param name="list">исходный список объектов</param>
      /// <param name="current">номер текущей страницы</param>
      /// <param name="itemsPerPage">кол. объектов на странице</param>
      /// <returns>объект класса ListViewModel</returns>
      public static ListViewModel<T> GetModel(IEnumerable<T> list,
                                        int current, int itemsPerPage)
      {
            var items = list
                            .Skip((current - 1)*itemsPerPage)
                            .Take(itemsPerPage)
                            .ToList();
            var total = (int)Math.Ceiling((double)list.Count() /
      itemsPerPage);
            return new ListViewModel<T>(items, total, current);
      }
}
```

4.4. Тестирование класса ListViewModel

4.4.1. Изменения в классе ProductControllerTests

Для тестирования класса ListViewModel понадобятся те же исходные данные, что и в тестах метода ControllerGetsProperPage. Чтобы избежать дублирования кода создайте отдельный класс (например TestData), со статическими методами, возвращающими список объектов и список исходных данных для тестирования (номер страницы, кол. объектов на выбранной странице и id первого объекта на странице).

```
public class TestData
    {
```

```
        public static List<Dish> GetDishesList()
        {
            return new List<Dish>
                {
                    new Dish{ DishId=1},
                    new Dish{ DishId=2},
                    new Dish{ DishId=3},
                    new Dish{ DishId=4},
                    new Dish{ DishId=5}
                };
        }
        public static IEnumerable<object[]> Params()
        {
            // 1-я страница, кол. объектов 3, id первого объекта 1
            yield return new object[] { 1, 3, 1 };
            // 2-я страница, кол. объектов 2, id первого объекта 4
            yield return new object[] { 2, 2, 4 };
        }
}
```

Замените [InlineData] на [MemberData]:

```
public class ProductControllerTests
{
      [Theory]
      [MemberData(nameof(TestData.Params), MemberType =
typeof(TestData))]
      public void ControllerGetsProperPage(int page, int qty, int id)
        {
            // Arrange
            var controller = new ProductController();

            controller._dishes = TestData.GetDishesList();

            // Act
            var result = controller.Index(page) as ViewResult;
            var model = result?.Model as List<Dish>;

            // Assert
            Assert.NotNull(model);
            Assert.Equal(qty, model.Count);
            Assert.Equal(id, model[0].DishId);
        }
}
```

Проверьте результат (выполните тесты контроллера)

4.4.2. Тесты класса ListViewModel

Предлагается написать три теста (проверка, исходя из того, что тестовый список содержит 5 объектов с id = 1, 2, 3, 4 и 5 соответственно):

проверка правильности подсчета общего количества страниц (ожидаемое значение 2)

проверка правильности количества объектов на выбранной странице (ожидаемые значения: 3 на первой странице и 2 на второй)

проверка правильности данных – id первого объекта на странице (ожидаемые значения: 1 на первой странице и 4 на второй странице)

Создайте класс ListViewModelTests и напишите предлагаемые тесты:

```
using WebLabsV06.DAL.Entities;
using WebLabsV06.Models;
using Xunit;

namespace WebLabsV06.Tests
{
    public class ListViewModelTests
    {
        [Fact]
        public void ListViewModelCountsPages()
        {
            // Act
            var model = ListViewModel<Dish>
                      .GetModel(TestData.GetDishesList(), 1, 3);

            // Assert
            Assert.Equal(2, model.TotalPages);
        }
        [Theory]
        [MemberData(memberName: nameof(TestData.Params),
                    MemberType = typeof(TestData))]
        public void ListViewModelSelectsCorrectQty(int page, int qty,
int id)
        {
            // Act
            var model = ListViewModel<Dish>
                      .GetModel(TestData.GetDishesList(), page, 3);

            // Assert
            Assert.Equal(qty, model.Count);
        }
        [Theory]
        [MemberData(memberName: nameof(TestData.Params),
                    MemberType = typeof(TestData))]
```

```
        public void ListViewModelHasCorrectData(int page, int qty, int
id)
        {
            // Act
            var model = ListViewModel<Dish>
                      .GetModel(TestData.GetDishesList(), page, 3);

            // Assert
            Assert.Equal(id, model[0].DishId);
        }

    }
}
```

4.4.3. Изменения в классе ProductController

В методе Index используйте класс ListViewModel<Dish>:

```
public IActionResult Index(int pageNo=1)
{
      return View(ListViewModel<Dish>.GetModel(_dishes, pageNo,
_pageSize));
}
```

В представлении Index измените модель:

```
@model ListViewModel<Dish>
```

Запустите проект. Проверьте, что список выводится правильно.

Запустите тест контроллера Product. Убедитесь, что тест пройден.

4.5. Оформление разметки списка

4.5.1. Создание частичного представления

В папке Views/Product создайте частичное представление _ListItemPartial.cshtml. В качестве модели укажите класс Dish. Скопируйте разметку карточки (и т.д.). Замените @item на @Model:

```
@model Dish

<div class='card m-2 p-1 text-center col-4'>
    <img src="~/Images/@Model.Image"
         alt='NoImage'
```

```
         class="card-img-top" />
    <div class="card-body">
        <h5 class="card-title">
            @Model.DishName
        </h5>
        <p class="card-text">
            @Model.Description
        </p>
        <div class="card-text badge badge-secondary">
            <h6>@Model.Calories калорий</h6>
        </div>
        @{ // Получение текущего адреса
            var request = ViewContext.HttpContext.Request;
            var returnUrl = request.Path +
request.QueryString.ToUriComponent();
        }
        <!--Разметка кнопки добавления в корзину-->
        <p class="mt-2">
            <a asp-action="Add"
               asp-controller="Cart"
               asp-route-id="@Model.DishId"
               asp-route-returnUrl="@returnUrl"
               class="btn btn-primary">
                <i class="fa fa-shopping-cart"></i> В корзину
            </a>
        </p>
    </div>
</div>
```

4.5.2. Изменение представления Index

Вместо разметки элемента списка поместите вызов частичного представления _ListItemPartial:

```
<div class="row">
    <div class="card-deck col">
        @foreach (var item in Model)
        {
            <partial name="_ListItemPartial" model="@item" />
        }
    </div>
</div>
```

Запустите проект, проверьте результат.

4.6. Выбор группы объектов

4.6.1. Разметка меню выбора группы

На страницу Index добавьте разметку:

```
@model ListViewModel<Dish>

@{
    ViewData["Title"] = "Меню";

    var categories = ViewData["Groups"] as IEnumerable<DishGroup>;
    int currentGroup = (int)ViewData["CurrentGroup"];

    var text = currentGroup !=0
        ? categories
            .FirstOrDefault(g => g.DishGroupId == currentGroup)?
            .GroupName
        : "Все";
}

<div class="row">
    <div class="col-2">
        <div class="dropdown mt-2">
            <a class="btn btn-secondary dropdown-toggle"
               asp-action="Index" asp-controller="Product"
               role="button"
               id="dropdownMenuLink"
               data-toggle="dropdown"
              aria-haspopup="true" aria-expanded="false">
                @text
            </a>

            <div class="dropdown-menu"
                 aria-labelledby="dropdownMenuLink">
                <a class="dropdown-item"
                   asp-action="Index"
                   asp-controller="Product">Все</a>

                @foreach (var item in categories)
                {
                    <a class="dropdown-item"
                       asp-action="Index"
                       asp-controller="Product"
                       asp-route-group="@item.DishGroupId"
                      asp-route-pageNo="1"
                       >@item.GroupName</a>
                }
            </div>
        </div>
    </div>
```

```
    <div class="col-10">
        <div class="card-deck">
            @foreach (var item in Model)
            {
                <partial name="_ListItemPartial" model="@item" />
            }
        </div>
    </div>
</div>
```

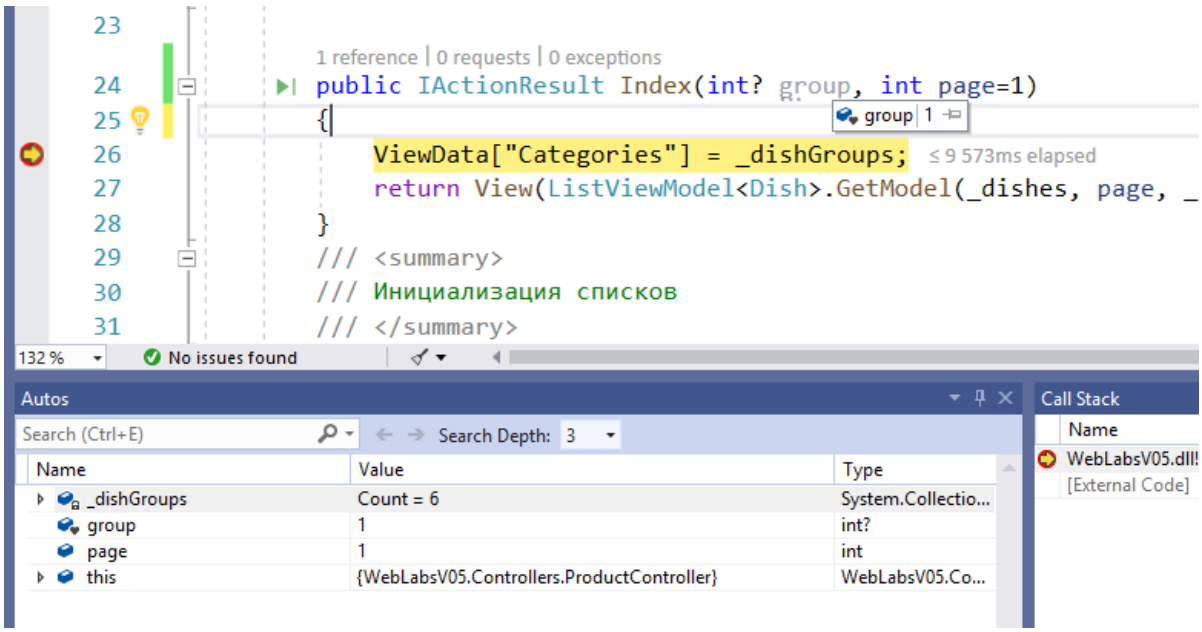
4.6.2. Изменения в методе Index контроллера Product

Добавьте еще один параметр в метод. Передайте в представление список групп и номер текущей группы.

```
public IActionResult Index(int? group, int pageNo=1)
{
    // Поместить список групп во ViewData
        ViewData["Groups"] = _dishGroups;

    // Получить id текущей группы и поместить в TempData
        ViewData["CurrentGroup"] = group ?? 0;. . .
}
```

Запустите проект в режиме отладки. Сделайте точку останова внутри метода Index. Убедитесь, что параметр «group» передается в контроллер при выборе группы в списке:



Сформируйте список объектов, отфильтрованный по группе и передайте его методу GetModel:

```
public IActionResult Index(int? group, int pageNo=1)
{
    var dishesFiltered = _dishes
    .Where(d => !group.HasValue || d.DishGroupId == group.Value);
    ViewData["Groups"] = _dishGroups;
     . . .
    return View(ListViewModel<Dish>.GetModel(dishesFiltered,
                                    pageNo, _pageSize));
}
```

Запустите проект и проверьте, что отображаются объекты из выбранной группы.

4.6.3. Изменения в тесте контроллера

```
public void ControllerGetsProperPage(int page, int qty, int id)
        {
            // Arrange
            var controller = new ProductController();
            controller._dishes = GetDishesList();

            // Act
            var result = controller
                          .Index(pageNo:page, group:null) as
                    ViewResult;
            var model = result?.Model as List<Dish>;

            // Assert
            Assert.NotNull(model);
            Assert.Equal(qty, model.Count);
            Assert.Equal(id, model[0].DishId);
}
```

4.7. Модульный тест для проверки выбора группы объектов

4.7.1. Изменения в методе GetDishesList

```
private List<Dish> GetDishesList()
        {
            return new List<Dish>
            {
                new Dish{ DishId=1, DishGroupId=1},
                new Dish{ DishId=2, DishGroupId=1},
                new Dish{ DishId=3, DishGroupId=2},
                new Dish{ DishId=4, DishGroupId=2},
                new Dish{ DishId=5, DishGroupId=3}
            };
```

```
}
```

4.7.2. Вспомогательный класс для сравнения двух объектов

В проект «XXX.Tests» добавьте класс Comparer:

```
class Comparer<T> : IEqualityComparer<T>
{
	public static Comparer<T> GetComparer(Func<T, T, bool> func)
	{
		return new Comparer<T>(func);
	}

	Func<T, T, bool> comparerFunction;

	public Comparer(Func<T, T, bool> func)
	{
		comparerFunction = func;
	}
	public bool Equals(T x, T y)
	{
		return comparerFunction(x,y);
	}

	public int GetHashCode(T obj)
	{
		throw new NotImplementedException();
	}
}
```

4.7.3. Тест контроллера

При наборе данных метода GetDishesList (см п.4.7.1) для группы №2 на страницу должны выводиться 2 объекта. Первый объект списка – это объект №3 (индекс 2, если считать от 0) из метода GetDishesList. Тест может иметь следующий код:

```
[Fact]
public void ControllerSelectsGroup()
{
	// arrange
	var controller = new ProductController();
	var data = TestData.GetDishesList();
	controller._dishes = data;

	var comparer = Comparer<Dish>
			.GetComparer((d1, d2) => d1.DishId.Equals(d2.DishId));

	// act
	var result = controller.Index(2) as ViewResult;
```

```
        var model = result.Model as List<Dish>;

        // assert
        Assert.Equal(2, model.Count);
        Assert.Equal(data[2], model[0], comparer);
}
```

Запустите тест. Убедитесь, что тест пройден.