

# Лабораторная работа №5

## Передача данных представлению. Передача файлов.

### 1. Цель работы.

Дальнейшее изучение механизма передачи данных представлению.  
Изучение работы с файлами в ASP.NET Core.

### 2. Общие сведения.

#### 2.1. Работа со статическими файлами

Статические файлы, такие как HTML, CSS, изображения и JavaScript, приложение ASP.NET Core может предоставлять непосредственно клиенту. Статические файлы как правило располагаются по пути Web Root:

`<content-root>/wwwroot`

В качестве **Content-root** обычно выбирается папка, в которой размещены все файлы проекта

Возможность работы со статическими файлами задается в методе Configure класса Startup:

```
public void Configure(IApplicationBuilder app, IHostingEnvironment
env, ILoggerFactory loggerFactory)
{
    app.UseStaticFiles();
    app.UseIdentity();
    app.UseMvcWithDefaultRoute();
}
```

Пример доступа к файлу `<content-root>/wwwroot/css/site.css`:

```
<link asp-href-include="/css/site.css" rel="stylesheet" />
```

Для работы с файлами в ASP.NET Core используются поставщики файлов (file providers).

Поставщики файлов - это абстракция над файловыми системами. Основным интерфейсом является **IFileProvider**.

**IFileProvider** предоставляет методы для получения информации о файлах (**IFileInfo**), информации о каталоге (**IDirectoryContents**) и настройке уведомлений об изменениях (с использованием **IChangeToken**).

**IFileInfo** предоставляет методы и свойства отдельных файлов или каталогов. Он имеет два булевых свойства: **Exists** и **IsDirectory**, а также свойства, описывающие имя файла, длину (в байтах) и дату **LastModified**.

Читать из файла можно с помощью метода **CreateReadStream**.

Конкретной реализацией интерфейса **IFileProvider** является **PhysicalFileProvider**, который обеспечивает доступ к физической файловой системе. Он оборачивает тип **System.IO.File** (для физического поставщика), просматривая все пути к каталогу и его дочерним элементам. Это ограничивает доступ только к определенному каталогу и его дочерним элементам, предотвращая доступ к файловой системе за пределами этой границы.

Готовые провайдеры для **Content\_Root** и **Web\_Root** можно получить из объекта **IHostingEnvironment**:

```
public SomeController(IHostingEnvironment env)
{
    var webFileProvider = env.WebRootFileProvider;
    var contentFileProvider = env.ContentRootFileProvider;
}
```

## 2.2. Передача файлов на сервер

Чтобы поддерживать загрузку файлов, HTML-форма должна иметь атрибут

**enctype="multipart/form-data"**

Доступ к отдельным файлам, загруженным на сервер, можно получить через привязку модели с использованием интерфейса **IFormFile**.

Интерфейс **IFormFile** описывает следующие методы и свойства:

```
public interface IFormFile
{
    string ContentType { get; }
    string ContentDisposition { get; }
```

```

IHeaderDictionary Headers { get; }
long Length { get; }
string Name { get; }
string FileName { get; }
Stream OpenReadStream();
void CopyTo(Stream target);
Task CopyToAsync(Stream target,
                  CancellationToken cancellationToken = null);
}

```

Пример сохранения файла в папке «wwwroot/Files»:

```

[HttpPost]
public async Task<IActionResult> Upload(
    [FromServices]IHostingEnvironment env,
    [FromForm]IFormFile uploadedFile)
{
    var path = env.WebRootPath + "/Files/" + uploadedFile.FileName;
    using (var stream = new FileStream(path, FileMode.Create))
    {
        await uploadedFile.CopyToAsync(stream);
    };
    return RedirectToAction("Index");
}

```

Пример сохранения файла в байтовый массив «byte[] AvatarImage»:

```

await uploadedFile
    .OpenReadStream()
    .ReadAsync(AvatarImage, 0, (int)uploadedFile.Length);

```

## 2.3. Передача файлов клиенту методом контроллера

Для отправки клиенту файлов предназначен абстрактный класс `FileResult`, который реализуется в классах:

- **FileContentResult**: отправляет клиенту массив байтов, считанный из файла;
- **VirtualFileResult**: представляет простую отправку файла напрямую с сервера по виртуальному пути;
- **FileStreamResult**: создает поток - объект `System.IO.Stream`, с помощью которого считывает и отправляет файл клиенту;

- **PhysicalFileResult:** для отправки используется реальный физический путь;

Пример отправки файла «Picture.jpg» из папки «wwwroot/images»:

```
public IActionResult GetImage([FromServices] IHostingEnvironment env)
{
    var provider = env.WebRootFileProvider;
    var path = Path.Combine("images", "Picture1.jpg");
    var fInfo = provider.GetFileInfo(path);
    var ext = Path.GetExtension(fInfo.Name);
    var extProvider = new FileExtensionContentTypeProvider();
    return File(fInfo.CreateReadStream(),
                extProvider.Mappings[ext]);
}
```

### 3. Выполнение работы

#### 3.1. Исходные данные

Используйте проект из лабораторной работы №4.

#### 3.2. Задание №1

Добавьте в проект возможность загрузки аватара пользователя.

Изображение аватара должно храниться в базе данных.

Выполните миграцию базы данных.

##### 3.2.1. Рекомендации к заданию №1

Для хранения изображения в классе ApplicationUser добавьте свойство типа byte[]. Также можно добавить свойство, описывающее MIME-тип изображения.

Для получения MIME-типа изображения можно воспользоваться классом FileExtensionContentTypeProvider:

```
var extProvider = new FileExtensionContentTypeProvider();
var mimeType = extProvider.Mappings[".png"];
```

Для передачи изображения создайте контроллер, метод GetAvatar() которого будет передавать аватар клиенту, а при его отсутствии – файл из

папки «images». Для получения данных пользователя понадобится внедрить в контроллер класс UserManager, а для доступа к папке wwwroot – объект IHostingEnvironment.

Для получения изображения в разметке в качестве значения атрибута *src* тэга *img* нужно указать адрес «Имя контроллера/GetImage». Для получения адреса воспользуйтесь вспомогательным методом @Url.Action.

Изображение общего аватара поместите в папку wwwroot/Images.

### 3.3. Задание №2

На панели навигации, в меню пользователя должен отображаться аватар, сохраненный при регистрации пользователя. Если аватар отсутствует, то должен выводиться общий аватар из папки «wwwroot/images»

### 3.4. Задание №3

Выберите любую предметную область. Для одной сущности из выбранной предметной области создайте В папке Entities проекта XXX.DAL создайте класс, содержащий следующие свойства:

- ID – уникальный номер;
- Название – короткое название конкретного объекта;
- Описание – дополнительное описание конкретного объекта;
- Категория – свойство для объединения объектов в группы;
- Цена/Вес/Расстояние – выберите любой параметр, который можно в дальнейшем обработать математически, например, просуммировать;
- Изображение – имя файла изображения объекта
- Mime тип изображения

В той же папке создайте класс, описывающий категорию объекта. Отношение должно быть один-ко-многим: одна категория описывает много объектов.

### 3.5. Задание №4

Создайте контроллер с именем Product. Данный контроллер будет отвечать за отображение информации об объектах из задания №3.

### Примечание:

Вы можете выбрать другое имя для контроллера, чтобы название соответствовало имени класса объекта из задания 3. В этом случае также измените название контроллера в компоненте меню (см. лабораторную работу №3)

Представление Index контроллера должно выводить список объектов.

#### 3.5.1. Рекомендации к заданию №4

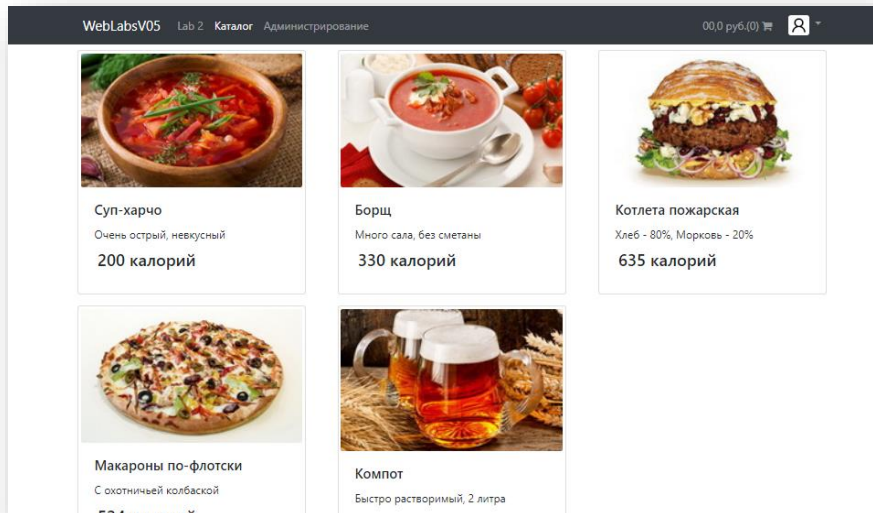
В классе контроллера создайте список типа `List<>` и проинициализируйте его 4-5 объектами. Этот список используйте в качестве модели представления Index.

#### 3.6. Задание №5

Оформите список с помощью класса bootstrap «card» (см. <https://getbootstrap.com/docs/4.3/components/card/#using-grid-markup>):

- информация об одном объекте должна занимать 1/3 контейнера экрана
- в верхней части расположить изображение объекта
- название объекта оформить стилем «card-title»
- текст описания объекта оформить стилем «card-text»
- числовую характеристику оформите стилем «card-subtitle badge»

Пример оформления списка:



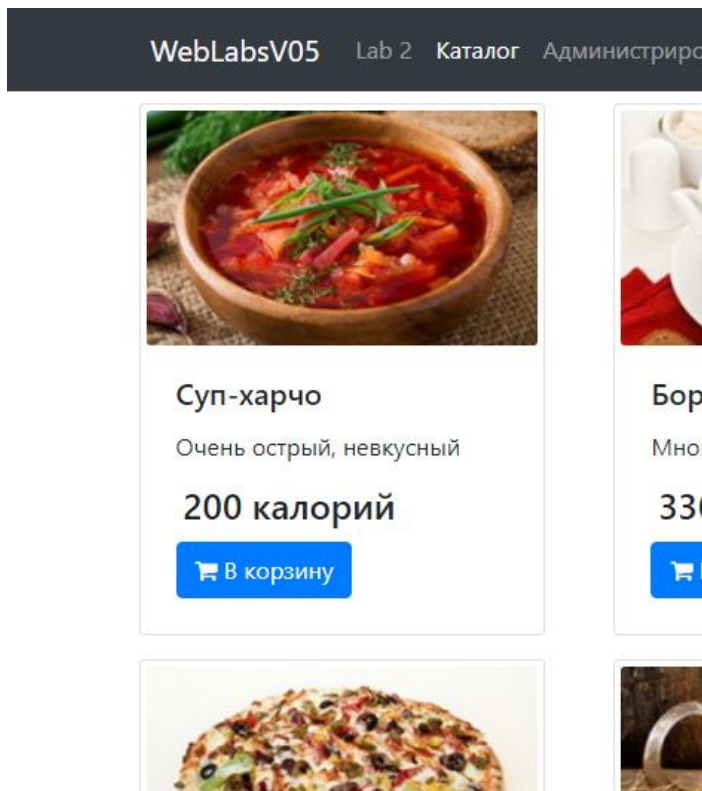
### 3.7. Задание №6

Для каждого объекта списка добавьте ссылку (тэг <a>) «В корзину».

- ссылка должна адресовать к методу Add контроллера Cart (будут созданы в дальнейшем).
- ссылка должна передавать id добавляемого элемента и адрес текущей страницы для возврата (можно использовать имя returnUrl)
- ссылку оформить стилем «btn btn-primary»
- рядом с текстом «В корзину» должна быть иконка shopping-cart из библиотеки font-awesome (см.

<https://fontawesome.com/icons/shopping-cart?style=solid>)

Пример оформления:



### 3.7.1. Рекомендации к заданию №6

Адрес текущей страницы можно получить так:

```
@{  
    // Получение текущего адреса  
    var request = ViewContext.HttpContext.Request;  
    var returnUrl = request.Path + request.QueryString.ToUriComponent();  
}
```

## 4. Контрольные вопросы

## 5. Пример выполнения работы

### 5.1. Предварительная информация

#### ВНИМАНИЕ:

Проект, используемый в качестве примера, имеет название **WebLabsV05**. Следовательно, все пространства имен в примерах начинаются с WebLabsV05, например, WebLabsV05.DAL.



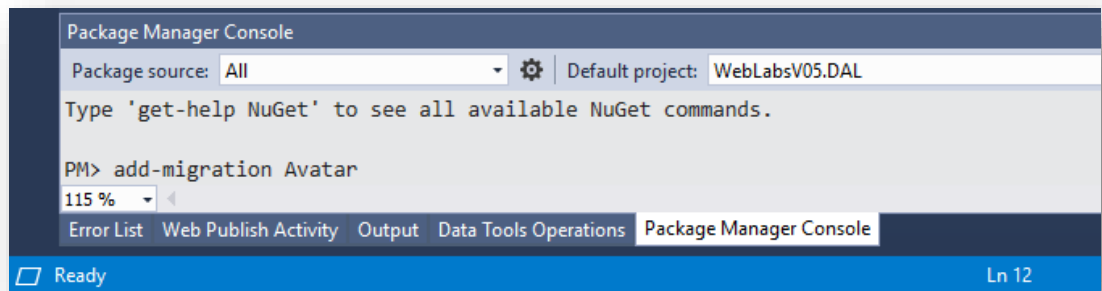
## 5.2. Добавление аватара пользователя

### 5.2.1. Изменение класса ApplicationUser

Добавьте класс ApplicationUser свойства, описывающие данные аватара и его Mime тип:

```
public class ApplicationUser:IdentityUser
{
    public byte[] AvatarImage { get; set; }
}
```

Выполните миграцию базы данных



Выполните команду «update-database»

Убедитесь, что в таблицеAspNetUsers появились новые поля.

### 5.2.2. Изменение страницы Register

В коде страницы внесите следующие изменения.

Добавьте в класс InputModel свойство:

```
public IFormFile Avatar { get; set; }
```

Измените метод OnPostAsync для сохранения изображения в базе данных:

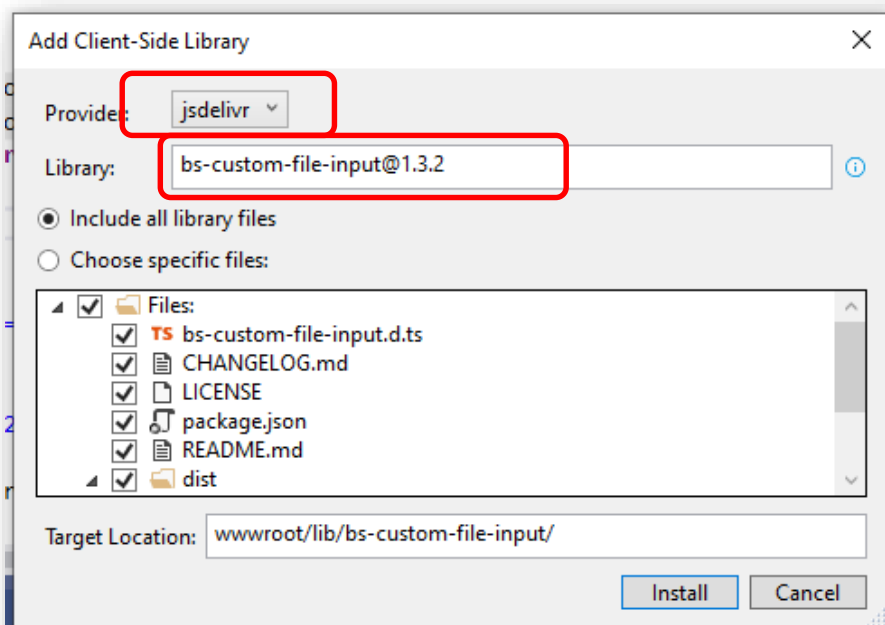
```
var user = new ApplicationUser { UserName = Input.Email, Email =
Input.Email };
if(Input.Avatar!=null)
{
    user.AvatarImage = new byte[(int)Input.Avatar.Length];
    await Input.Avatar
        .OpenReadStream()
        .ReadAsync(
            user.AvatarImage,
            0,
            (int)Input.Avatar.Length);
}
```

Измените страницу Register, чтобы пользователь при регистрации мог указать имя файла аватарки и отослать его на сервер:

```
<form asp-route-returnUrl="@Model.ReturnUrl"
      method="post"
      enctype="multipart/form-data">
  <h4>Create a new account.</h4>
  .
  .
  .
  <div class="form-group">
    <label asp-for="Input.Avatar"></label>
    <input asp-for="Input.Avatar" type="file"
          class="form-control" />
  </div>
  <button type="submit"
        class="btn btn-primary">Register</button>
</form>
```

Вариант оформления с помощью библиотеки bs-custom-file-input (см. <https://getbootstrap.com/docs/4.4/components/forms/#file-browser>, <https://www.npmjs.com/package/bs-custom-file-input>)

Добавьте clien-side library:



В конце страницы, в секцию «Scripts», подключите загруженную библиотеку и активируйте ее:

```
@section Scripts {  
    <partial name=" ValidationScriptsPartial" />  
    <script src="~/lib/bs-custom-file-input/dist/bs-custom-file-  
input.js"></script>  
    <script>  
        $(document).ready(function() {  
            bsCustomFileInput.init()  
        })  
    </script>  
}
```

Оформите разметку для выбора аватарки:

```
<div class="form-group">  
    <label asp-for="Input.Avatar"></label>  
    <div class="custom-file">  
        <input asp-for="Input.Avatar" type="file" class="custom-  
file-input" id="customFile">  
        <label class="custom-file-label" for="customFile">Выберите  
файл</label>  
    </div>  
</div>
```

Зарегистрируйте пользователя с аватаром. Убедитесь, что данные сохраняются в базе данных.

### 5.3. Отображение аватара пользователя

#### 5.3.1. Создание контроллера

В папке Controllers проекта создайте контроллер Image. Опишите метод GetAvatar(), который вернет изображение клиенту:

```
namespace WebLabsV06.Controllers  
{  
    public class ImageController : Controller  
    {  
        UserManager<ApplicationUser> _userManager;  
        IWebHostEnvironment _env;  
  
        public ImageController(UserManager<ApplicationUser>  
userManager, IWebHostEnvironment env)  
        {  
            _userManager = userManager;  
            _env = env;  
        }  
    }  
}
```

```

public async Task<FileResult> GetAvatar()
{
    var user = await _userManager.GetUserAsync(User);
    if (user.AvatarImage != null)
        return File(user.AvatarImage, "image/...");
    else
    {
        var avatarPath = "/Images/anonymous.png";

        return File(_env.WebRootFileProvider
            .GetFileInfo(avatarPath)
            .CreateReadStream(), "image/...");
    }
}
}
}
}

```

### 5.3.2. Изменение представления \_UserPartial

```

<div class="dropdown ml-4 nav-color">
    <div class="dropdown-toggle"
        id="dropdownMenuButton"
        data-toggle="dropdown" aria-haspopup="true"
        aria-expanded="false">
        
    </div>
    <div class="dropdown-menu"
        aria-labelledby="dropdownMenuButton">
        <div class="dropdown-item-text">
            
            @User.Identity.Name
        </div>
    </div>

```

Запустите проект. Войдите в систему под разными учетными данными. Убедитесь, что аватар отображается правильно.

### 5.4. Создание классов предметной области

#### **ВНИМАНИЕ:**

В предлагаемом примере используется предметная область — предприятие питания. Сущность предметной области — Dish (блюдо).

Для выполнения задания добавьте в папку «wwwroot/images» несколько изображений объектов вашей предметной области. Для удобства отображения желательно, чтобы изображения были одного размера или, хотя бы, имели одинаковое соотношение сторон.

Классы предметной области поместите в папку Entities проекта XXX.DAL

Пример класса Dish:

```
public class Dish
{
    public int DishId { get; set; } // id блюда
    public string DishName { get; set; } // название блюда
    public string Description { get; set; } // описание блюда
    public int Calories { get; set; } // кол. калорий на порцию
    public string Image { get; set; } // имя файла изображения

    // Навигационные свойства
    /// <summary>
    /// группа блюд (например, супы, напитки и т.д.)
    /// </summary>
    public int DishGroupId { get; set; }
    public DishGroup Group { get; set; }
}
```

Пример класса DishGroup:

```
public class DishGroup
{
    public int DishGroupId { get; set; }
    public string GroupName { get; set; }
    /// <summary>
    /// Навигационное свойство 1-ко-многим
    /// </summary>
    public List<Dish> Dishes { get; set; }
}
```

## 5.5. Вывод списка объектов на страницу

### 5.5.1. Создание контроллера

В папке Controllers основного проекта создайте пустой контроллер Product (имя класса контроллера – ProductController).

Опишите приватные поля, содержащие список объектов и список групп объектов, например:

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

```
List<DishGroup> _dishGroups;
```

В отдельном методе выполните заполнение списков. Количество элементов в списке объектов должно быть 4-5. Количество элементов в списке групп значения не имеет. Вызовите метод в конструкторе контроллера.

Передайте список объектов в качестве модели представлению:

```
namespace WebLabsV05.Controllers
{
    public class ProductController : Controller
    {
        List<Dish> _dishes;
        List<DishGroup> _dishGroups;

        public ProductController()
        {
            SetupData();
        }

        public IActionResult Index()
        {
            return View(_dishes);
        }

        /// <summary>
        /// Инициализация списков
        /// </summary>
        private void SetupData()
        {
            _dishGroups = new List<DishGroup>
            {
                new DishGroup {DishGroupId=1, GroupName="Стартеры"},
                new DishGroup {DishGroupId=2, GroupName="Салаты"},
                new DishGroup {DishGroupId=3, GroupName="Супы"},
                new DishGroup {DishGroupId=4, GroupName="Основные
блюда"},
                new DishGroup {DishGroupId=5, GroupName="Напитки"},
                new DishGroup {DishGroupId=6, GroupName="Десерты"}
            };

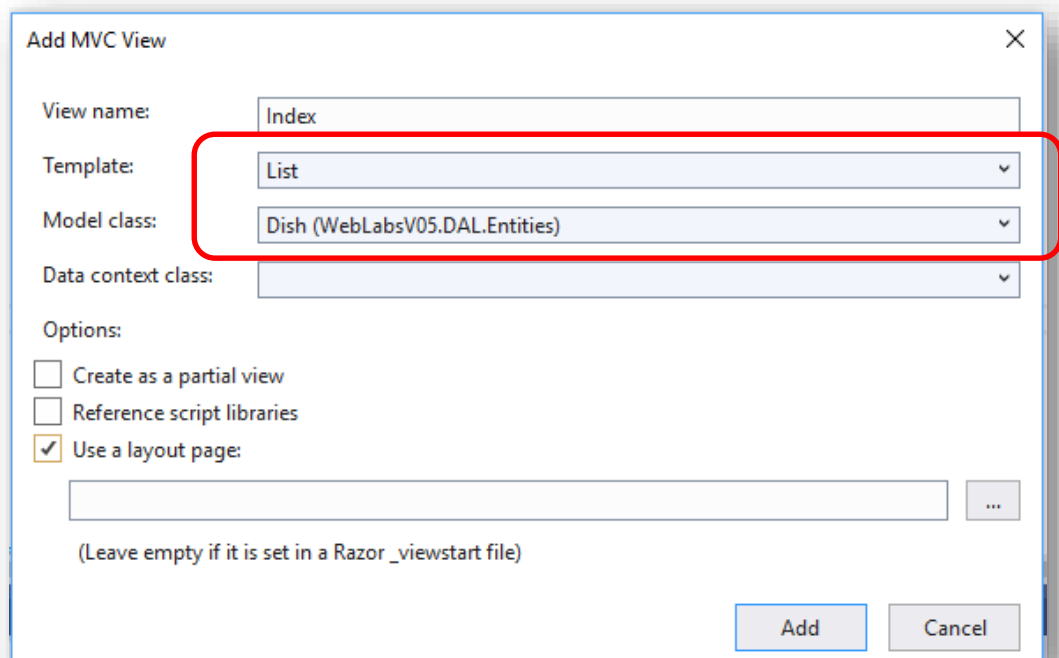
            _dishes = new List<Dish>
            {
                new Dish {DishId = 1, DishName="Суп-харчо",
                    Description="Очень острый, невкусный",
                    Calories = 200, DishGroupId=3, Image="Суп.jpg" },
                new Dish { DishId = 2, DishName="Борщ",
                    Description="Много сала, без сметаны",
                    Calories = 330, DishGroupId=3, Image="Борщ.jpg" },
                new Dish { DishId = 3, DishName="Котлета пожарская",
                    Description="Хлеб - 80%, Морковь - 20%",
```

```

        Calories = 635, DishGroupId=4, Image="Котлета.jpg" },
        new Dish { DishId = 4, DishName="Макароны по-флотски",
        Description="С охотничьей колбаской",
        Calories = 524, DishGroupId=4, Image="Макароны.jpg" },
        new Dish { DishId = 5, DishName="Компот",
        Description="Быстро растворимый, 2 литра",
        Calories = 180, DishGroupId=5, Image="Компот.jpg" }
    };
}
}
}

```

### 5.5.2. Создание представления с помощью scaffolding



В представлении описана модель:

```
@model IEnumerable<XXX.DAL.Entities.Dish>
```

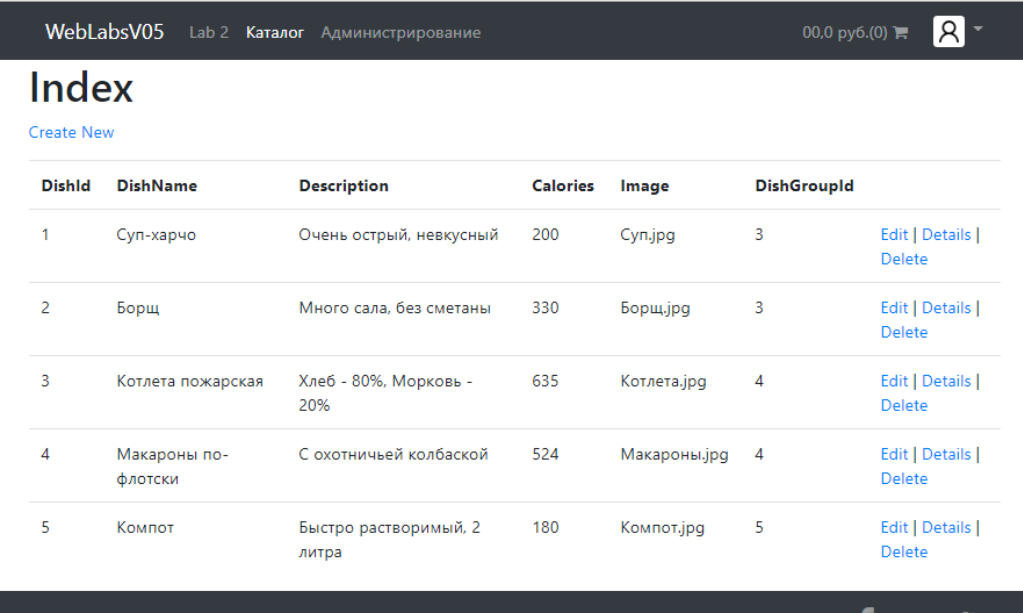
В файл \_ViewImports добавьте:

```
@using XXX.DAL.Entities
```

Измените модель в представлении Index:

```
@model IEnumerable<Dish>
```

Запустите проект. Выберите пункт меню «Каталог». Убедитесь, что выводится список объектов:



DishId	DishName	Description	Calories	Image	DishGroupId	
1	Суп-харчо	Очень острый, невкусный	200	Суп.jpg	3	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>
2	Борщ	Много сала, без сметаны	330	Борщ.jpg	3	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>
3	Котлета пожарская	Хлеб - 80%, Морковь - 20%	635	Котлета.jpg	4	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>
4	Макароны по-флотски	С охотничьей колбаской	524	Макароны.jpg	4	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>
5	Компот	Быстро растворимый, 2 литра	180	Компот.jpg	5	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>

### 5.5.3. Оформление представления Index

Уберите всю разметку из представления Index, кроме:

```
@model IEnumerable<Dish>
```

```
@{  
    ViewData["Title"] = "Index";  
}
```

Измените значение Title на более понятное название, например:

```
@{  
    ViewData["Title"] = "Меню";  
}
```

Используйте компонент `card` библиотеки `bootstrap` (<https://getbootstrap.com/docs/4.3/components/card/>):

```
@model IEnumerable<Dish>  
  
@{  
    ViewData["Title"] = "Меню";  
}  
  
<div class="row">
```



```

<div class="card-deck">
  @foreach (var item in Model)
  {
    <div class='card m-2 p-1 text-center col-4'>
  
  <div class="card-body">
    <h5 class="card-title">
      @item.DishName
    </h5>
    <p class="card-text">
      @item.Description
    </p>
    <div class="card-text badge badge-secondary">
      <h6>@item.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="@item.DishId"
        asp-route-returnUrl="@returnUrl"
        class="btn btn-primary">
        <i class="fa fa-shopping-cart"></i> В корзину
      </a>
    </p>
  </div>
</div>
  }
</div>
</div>

```

Запустите проект и проверьте результат.