Лабораторная работа №2

Тема: разработка блога на ASP.NET Core 2 и Microsoft SQL Server 2014

Цели и задачи:

1. Вспомнить основы веб-программирования: язык HTML, CSS, JavaScript
2. Познакомиться с Active Server Pages (ASP) – технологией от Microsoft для разработки сайтов
3. Узнать, что такое Model-View-Controller структура. Научиться строить логику, основываясь на ней
4. Разработать веб-систему для ведения личного блога.

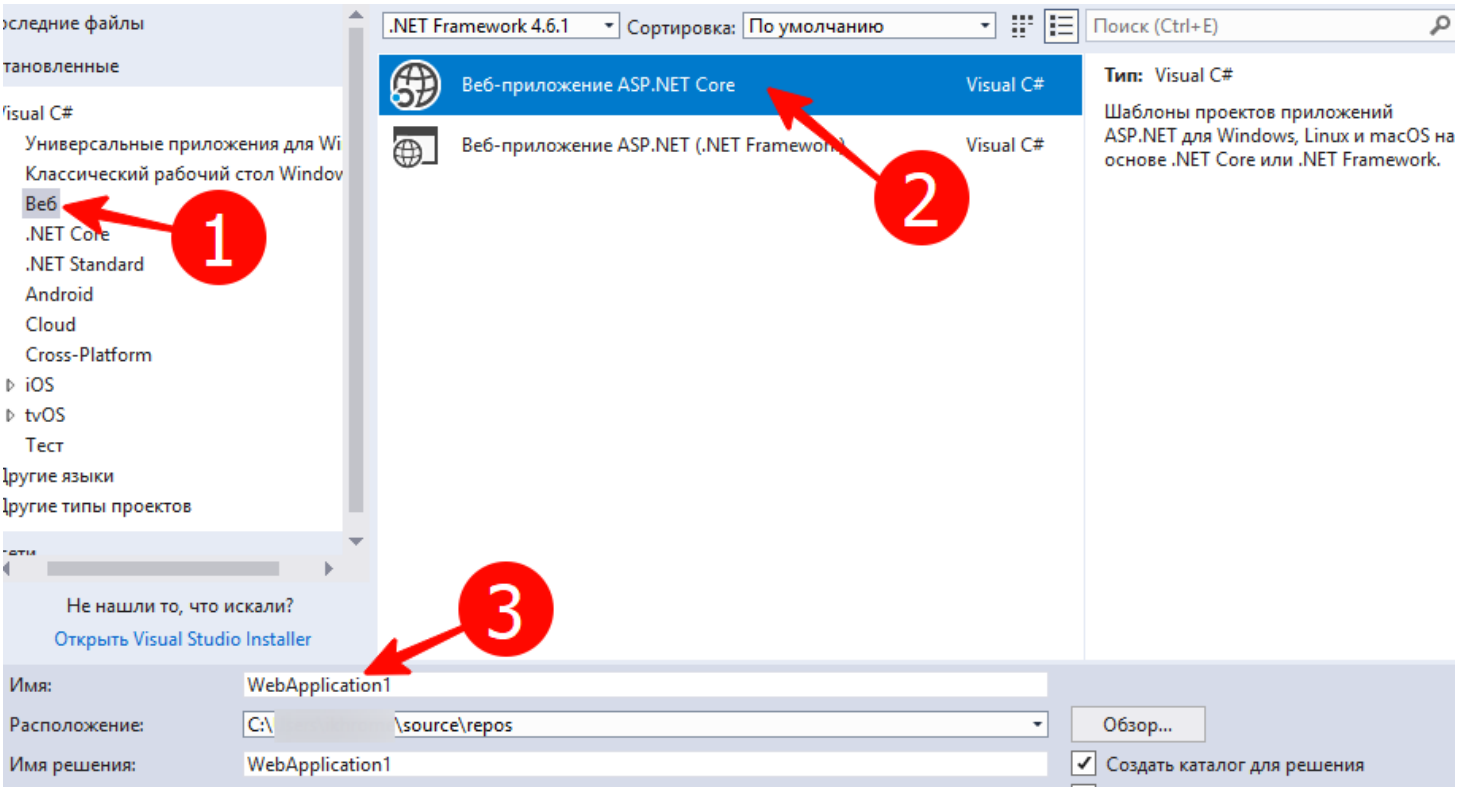
Подготовка

Убедитесь, что у вас на ПК установлены:

1. Microsoft SQL Server 2014
2. Microsoft Visual Studio 2017 с инструментами веб-разработки.

Начало работы

Откройте Visual Studio (в дальнейшем просто VS). Выберите команду Файл – Создать – Проект, либо используйте сочетание клавиш Ctrl+Shift+N.



Назовите ваш проект любым именем на английском языке. Например, “weblog”.

Теперь необходимо выбрать параметры для проекта (см. скриншот ниже)

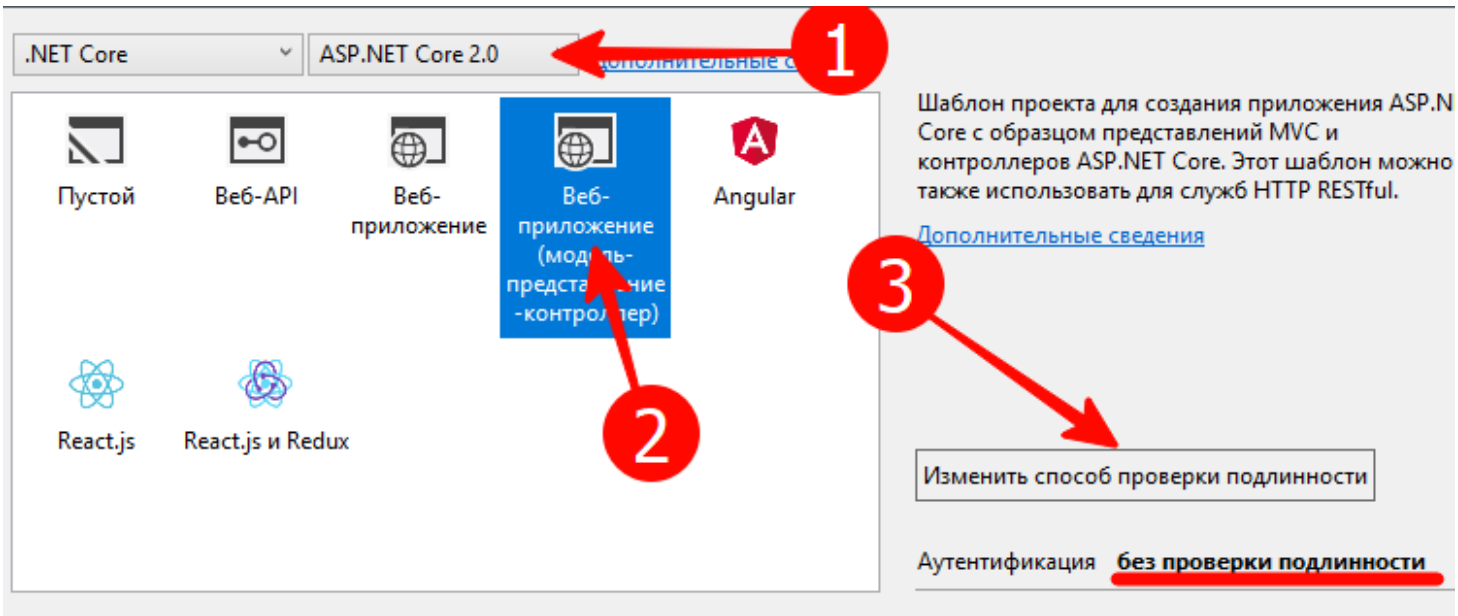
В комбобоксах вверху формы будет версия .NET Core и тип ASP.NET Core 2.0.

Необходимо выбрать тип приложения «Веб-приложение (модель-представление-контроллер)

Далее жмем на «Изменить способ проверки подлинности» и меняем его на «без проверки подлинности».

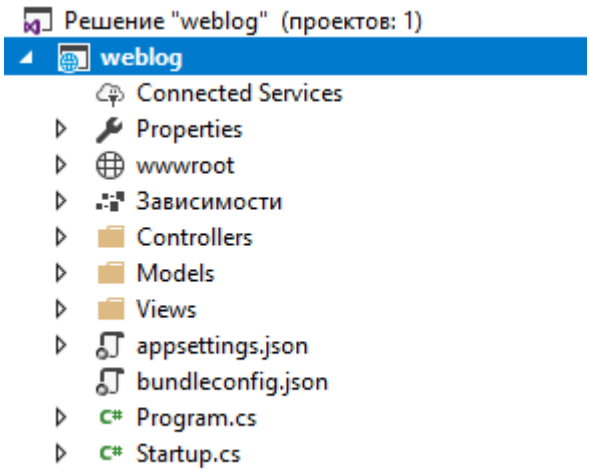
Убираем, если есть «Поддержка Docker».

Таким образом, мы выбрали платформу, тип и параметры будущего приложения.



Будет создано веб-приложение.

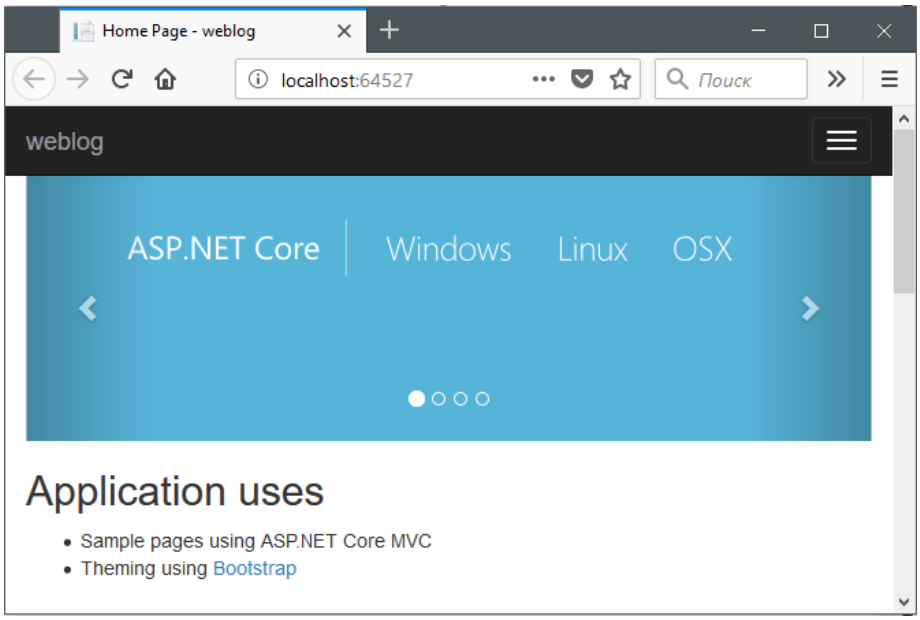
Рассмотрим его структуру:



1. **Controllers** – контроллеры приложения. Здесь мы опишем всю логику работы с веб-сервером, а также манипулирование данными.
2. **Models** – данные приложения, а так же представления данных
3. **Views** – «лицо» нашего приложения. Именно здесь мы определяем, какой HTML код будет отображать приложение.
4. **Program.cs** – файл-входная точка программы. Здесь происходит связывание всех компонентов воедино.
5. **Startup.cs** – файл, отвечающий за включение и отключение сервисов приложения.
6. **wwwroot** – все статические файлы (CSS, JS, images etc.)

Изучите файл `Controllers -> HomeController.cs`. Как видите, все в приложении завязано на контроллерах и действиях. Действие – обычный метод, например, метод типа `IActionResult`

Нажмем Ctrl+F5 для запуска сервера. Через пару десятков секунд откроется веб-браузер:



Поздравляю. Вы только что создали веб-проект в VS!

Изменение шаблона начальной страницы

Изменим представление, которое отвечает за главную страницу. Откроем файл `~/Views/Home/Index.cshtml` и изменим шаблон, добавив туда, например, элемент H1, пару параграфов, и пустую якорную ссылку:

```
Index.cshtml   HomeController.cs
1  @{
2      ViewData["Title"] = "Home Page";
3  }
4
5  <h1 class="text-muted">Блог на ASP.NET Core 2</h1>
6  <p>
7      Сюда мы сможем поместить наш текст о блоге
8  </p>
9  <a class="btn btn-primary btn-lg">Все записи &rsaquo;</a>
```

Здесь `@{ ... }` – указатели на т.н. «встроенный»(embedded) код на языке C#. Система визуализации представлений в ASP.NET Core называется Razor.

Небольшое пояснение: здесь, в приложении, используется Bootstrap – готовый набор CSS классов, помогающий сразу же начать визуализацию любого проекта. В проекте используется версия 3.3.7. На данный момент последняя версия 4.0.0-beta. Прочитать подробнее о нем можно на сайте http://getbootstrap.com

Задание: попробуйте изменить `<title>` страницы, чтобы он стал, например, «Блог – weblog».

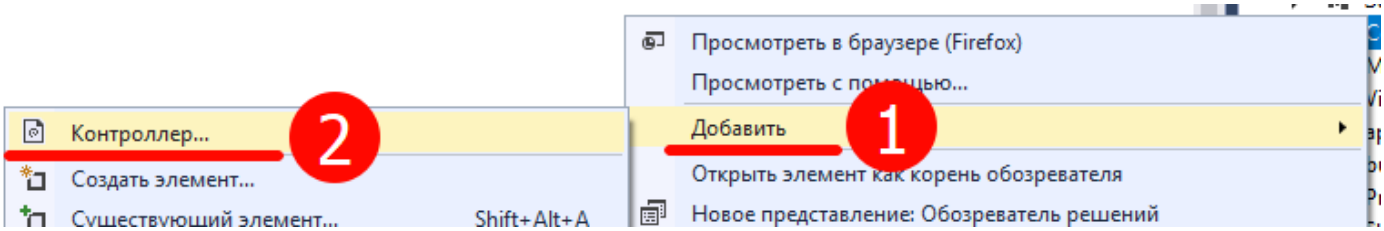
Давайте сделаем так, чтобы ссылка «Все записи» указывала на адрес `/Blog/`. Тут и наступает пора узнать про встроенные помощники (хелперы). В нашем случае – хелпер для работы с тегом <a>. Он добавляет атрибуты:

`asp-controller`	Принимает как аргумент имя контроллера. Например, • Файл контроллера – `HomeController.cs` • Имя контроллера – Home • Значит, `<a asp-controller="Home" … >Home</a>`
`asp-action`	Принимает как аргумент имя контроллера. Например, • Файл контроллера – `HomeController.cs` • Имя контроллера – Home • Имя действия – `public IActionResult ShowAll {}` • Значит, `<a asp-controller="Home" asp-action="ShowAll >Home</a>` Если имя действия Index – asp-action вернет пустой результат.

Задание: сделать ссылку на `/Blog/` с использованием хелперов.

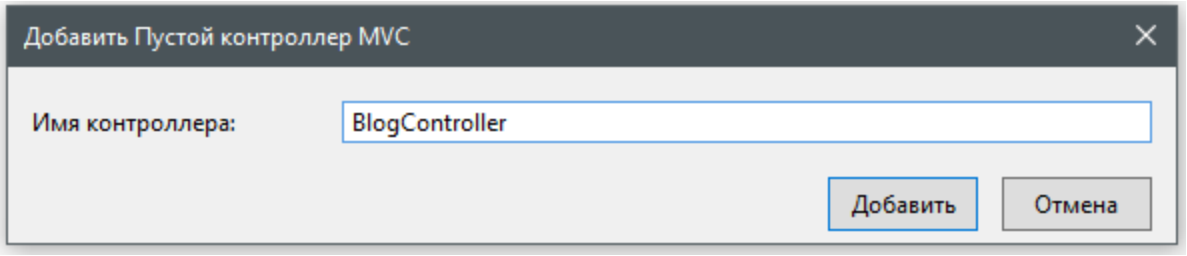
Создание контроллера BlogController

Пришло время создать контроллер. Для этого щелкаем ПКМ по папке «Controllers» в «Обозреватель решений». В меню «Создать» выберем «Контроллер».



В появившемся окне выбираем «Контроллер MVC – Пустой» и нажимаем OK.

Вводим имя контроллера, чтобы получилось «BlogController».

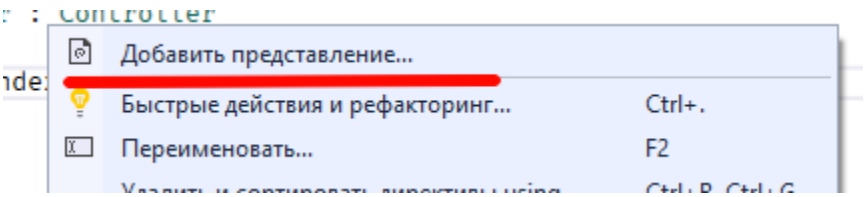


Окно выполнения операции начнет сбор информации о проекте и восстановление пакетов NuGet. Это может занять ~ 5’ времени.

По окончании будет сформирован контроллер и одно действие – Index:

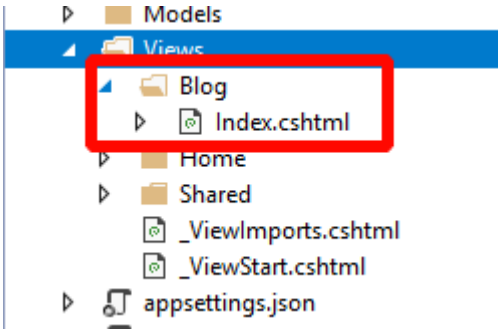
```
namespace weblog.Controllers
{
    public class BlogController : Controller
    {
        public IActionResult Index()
        {
            return View();
        }
    }
}
```

Кликаем ПКМ по Index() и получим вот такое меню:



Нажимаем «Добавить представление…». И просто нажимаем ОК в появившемся окне.

Как мы можем заметить, VS создала для нас в папке `Views` папку `Blog`, и добавила представление `Index.cshtml`. Мы создали начальную структуру.



<u>Задание:</u> Добавьте таким же образом представление `Show`. Добавьте метод, и создайте файл представления.

Теперь перейдем к разработке моделей.

Для начала, добавим строку подключения. Сделаем это немного необычным для вас способом.

Добавление строки подключения

Откроем файл appsettings.json. JSON (читается как «джейсóн») – формат для структурирования данных. Отлично подходит для хранения конфигурационных настроек приложений. Добавим после { следующее:

```
"ConnectionStrings": {
    "BlogDb": "<спросите преподавателя>"
},
```

Вспомним работу с WinForms – мы добавляли узел `connectionStrings` в App.config. Теперь все гораздо проще.

Перейдем к разработке контекста приложения. Создадим две модели: `Post` и `Category`, а также контекст приложения в папке Models.

Контекст и модели данных

Создадим контекст данных, используя подход Code-First. Делаем так, как делали в WindowsForms. Добавим файл `BlogContext.cs` в папку `Models`.

И наполним его кодом:

```
using Microsoft.EntityFrameworkCore;
using System;

namespace weblog.Models
{
    public class BlogContext : DbContext
    {
      public BlogContext(DbContextOptions<BlogContext> options) :
      base(options) { }
    }
}
```

Теперь перейдем непосредственно к моделям. Добавляем их в папку Models:

Post.cs:

```
public class Post
{
   public Guid Id { get; set; }
   public string Title { get; set; }
   public DateTime PublishDate { get; set; }
   public bool IsDraft { get; set; }
   public string Content { get; set; }

   public int CategoryId { get; set; }

   public virtual Category Category { get; set; }
}
```

В данном случае мы выбрали для Id тип GUID. Entity Framework Core создаст для нас поле в БД с типом UNIQUEIDENTIFIER и автоматически запускать функцию NEWID() при добавлении записи.

Category.cs:

```
public class Category
{
   public int Id { get; set; }
   public string Title { get; set; }
   public string Description { get; set; }
}
```

Далее добавим их в контекст:

```
public DbSet<Post> Posts { get; set; }
public DbSet<Category> Categories { get; set; }
```

И зарегистрируем на уровне приложения работу с SQL Server:

Файл Startup.cs отвечает за конфигурирование всех сервисов приложения. Для того, чтобы добавить поддержку SQL Server в наш проект находим метод `ConfigureServices` (строка 22) и перед `services.UseMvc()` пишем:

```
string connection = Configuration.GetConnectionString("BlogDb");
services.AddDbContext<BlogContext>(options =>
     options.UseSqlServer(connection)
);
```

Далее поступаем очень стандартно. Мы уже это проходили:

Создаем миграцию:

```
Add-Migration Initial
```

И обновляем базу данных:

```
Update-Database
```

Если все успешно – переходим к следующему этапу.

Наполнение БД

Самый простой шаг. Наполните БД примерно 10 записями и 3 категориями. Используйте SQL Server 2014 Management Studio.

Вывод записей блога

Изменим файл `Controllers/BlogController.cs`, добавив в него свойство `db` типа `BlogContext` и в конструкторе присвоим ему значение:

```
private readonly BlogContext db;

public BlogController(BlogContext ctx)
{
    db = ctx;
}
```

В данном коде мы воспользовались т.н. «внедрением зависимостей» (dependency injection). Подробнее об этом, если интересно, можно узнать у преподавателя.

Теперь нам необходимо написать логику получения записей из БД, проводя определенную сортировку. Изменим в файле `Controllers/BlogController.cs` метод Index следующим образом:

```
public async Task<IActionResult> Index()
{
     var posts = await db.Posts.Include(c => c.Category)
                .Where(d => !d.IsDraft).ToListAsync();
     return View(posts);
}
```

Это пример асинхронного метода. Он будет вызван в отдельном потоке, и все вычисления, которые он производит, так же будут вызваны отдельно.

Эта концепция важна для современной веб-разработки. Именно асинхронность порой спасает от «зависания» процесса.

Мы производим сортировку по полю IsDraft по условию «не».

Итак, пришло время создать шаблон для этого действия.

Вывод записей в блоге

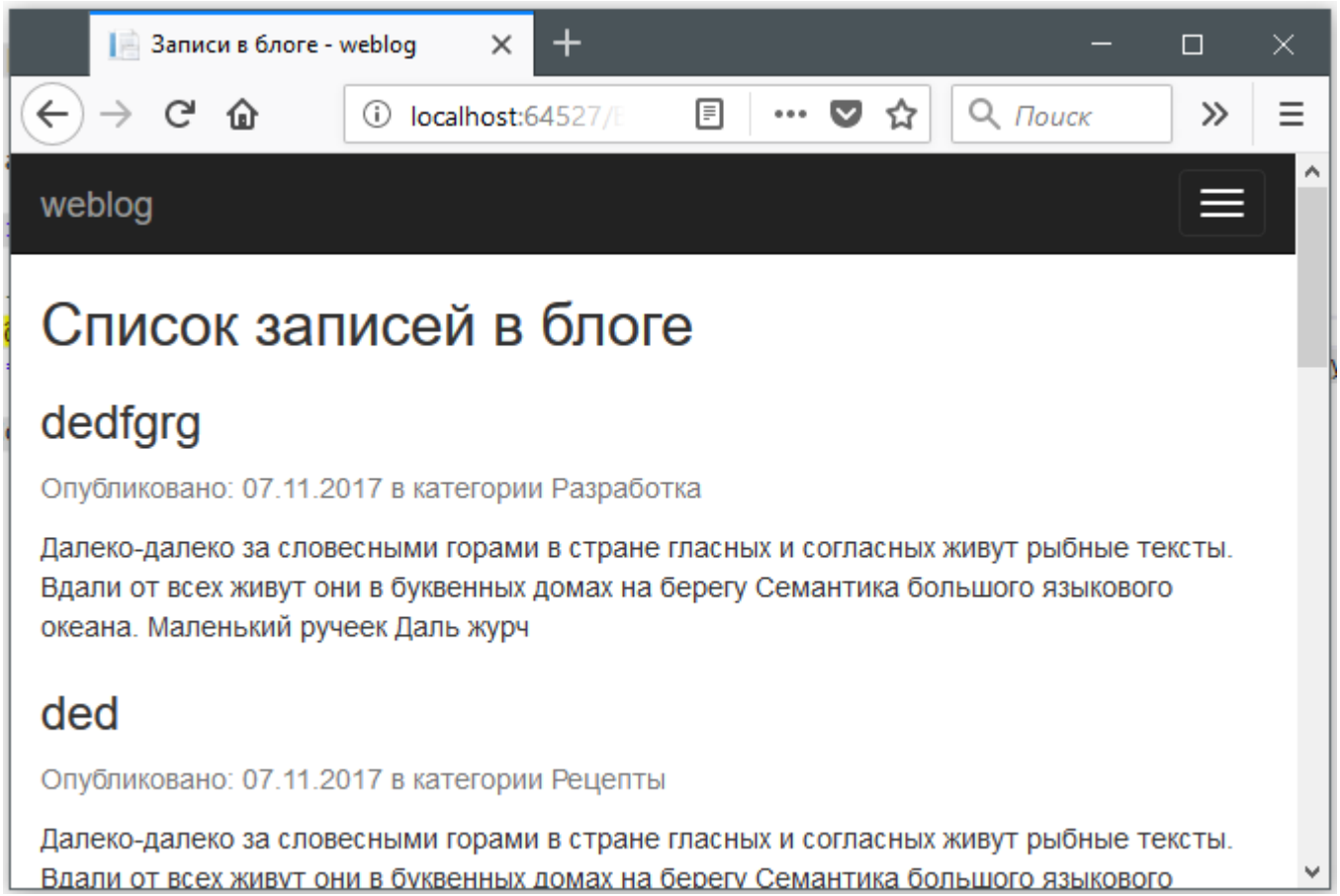
Помните, как мы в `return View()` передали как аргумент список всех записей? Так вот, мы сделали так незря.

Для доступа к коллекции записей воспользуемся свойством каждого представления, которое называется Model.

Изменим файл ~/Views/Blog/Index.cshtml, добавив туда следующий код (HTML разметка – **на усмотрение студента**):

```
...
@foreach(var p in Model)
{
    @p.Title <!-- заголовок записи -->
    @p.Category.Title <!-- доступ к Foreign Key -->
}
...
```

Таким образом, у меня получилось вывести все записи, не являющиеся черновиком:



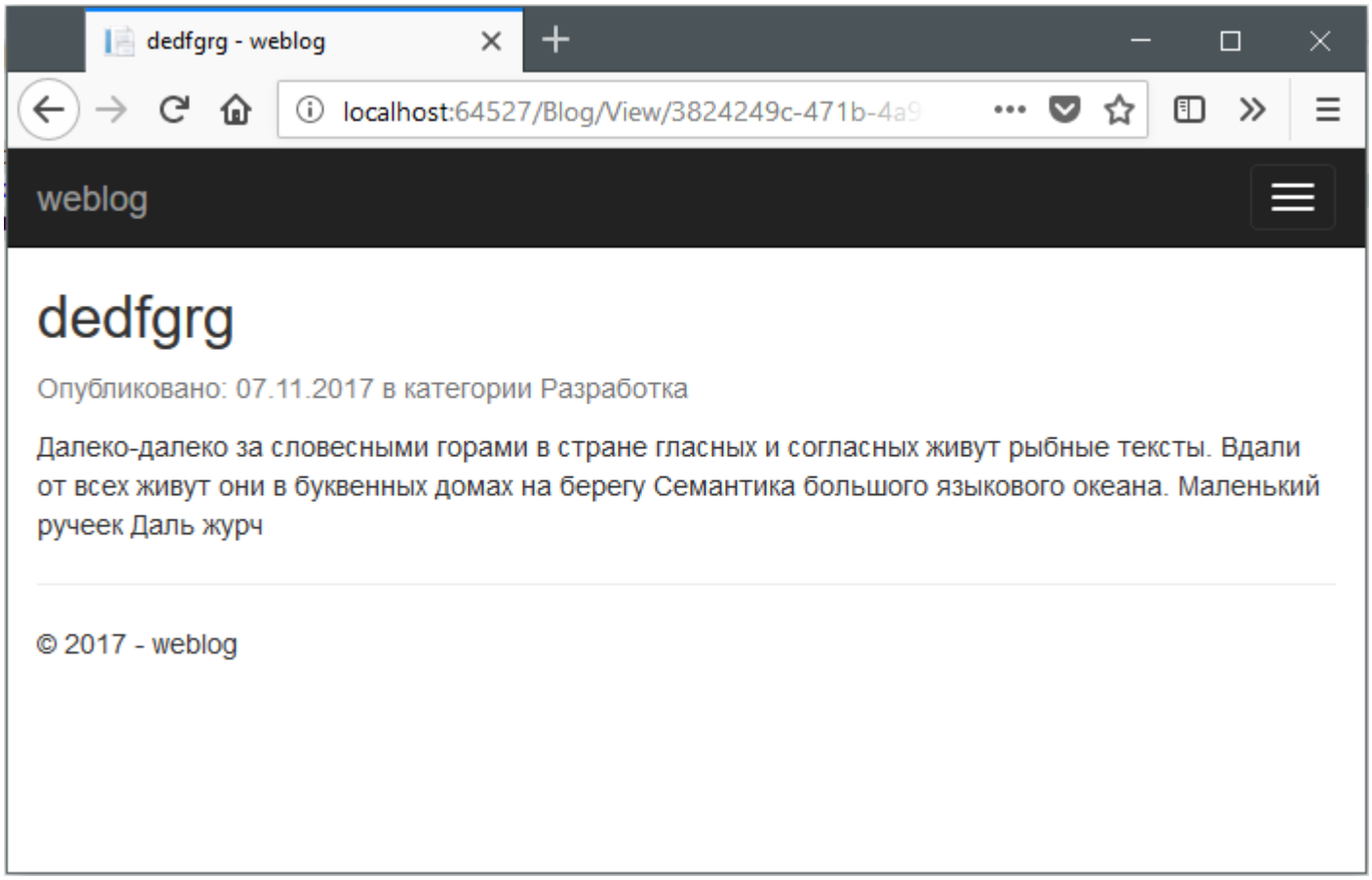
Просмотр отдельной записи

Напишем действие, которое будет отвечать за отображение записи. Мы уже создали его, просто так же сделаем асинхронным, и добавим к нему аргумент `id` типа `Guid`. Изменим в этом контроллере метод `View`:

```
public async Task<IActionResult> View(Guid id)
{
    Post post;
    try
    {
        post = await db.Posts.Include(c => c.Category).SingleAsync(i =>
i.Id == id);
```

```
    }
    catch(Exception e)
    {
        return NotFound();
    }
    return View(post);
}
```

И таким же образом, отобразите поля в представлении `~/Views/Blog/View.cshtml`. Только уже нет необходимости использовать `foreach` – это не коллекция. Вместо этого используем просто `@Model.<имя свойства>`



Задание: добавить в `title` заголовок записи.

Таким образом мы написали небольшое по своим масштабам приложение на ASP.NET Core 2. Поздравляем.

Покажите результат преподавателю.

Автор: Хромов Иван Андреевич, ГАПОУ СО «Асбестовский политехникум», преподаватель

Специальность: Программирование в компьютерных системах

Предмет: Технологии разработки баз данных

Раздел Язык C# в веб-программировании

Курс: 4

Асбест, 2017