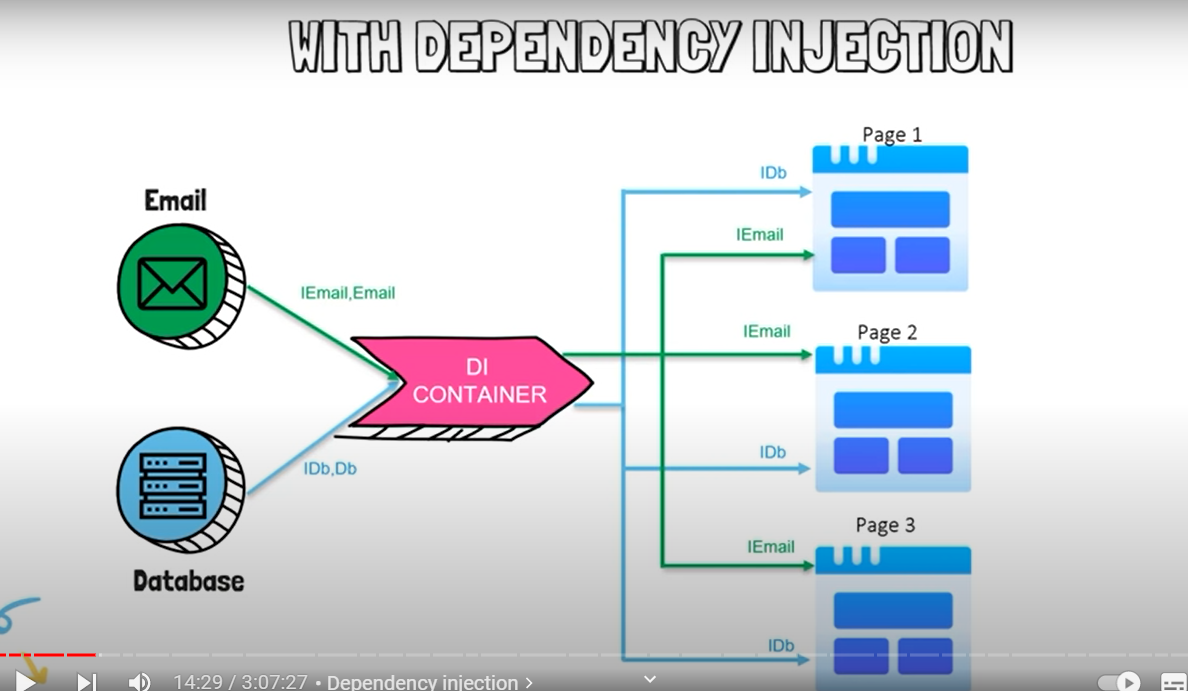
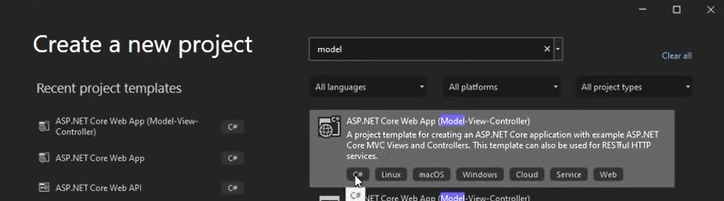
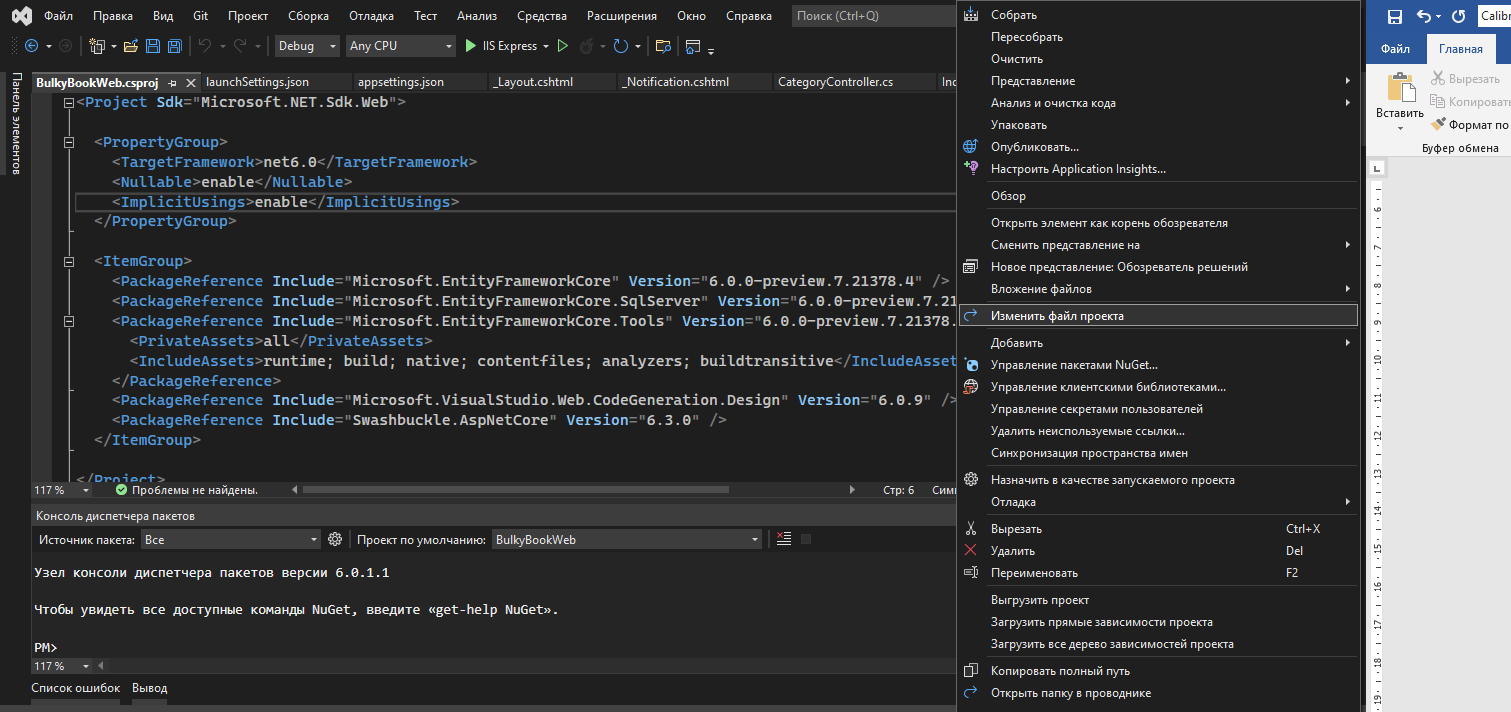
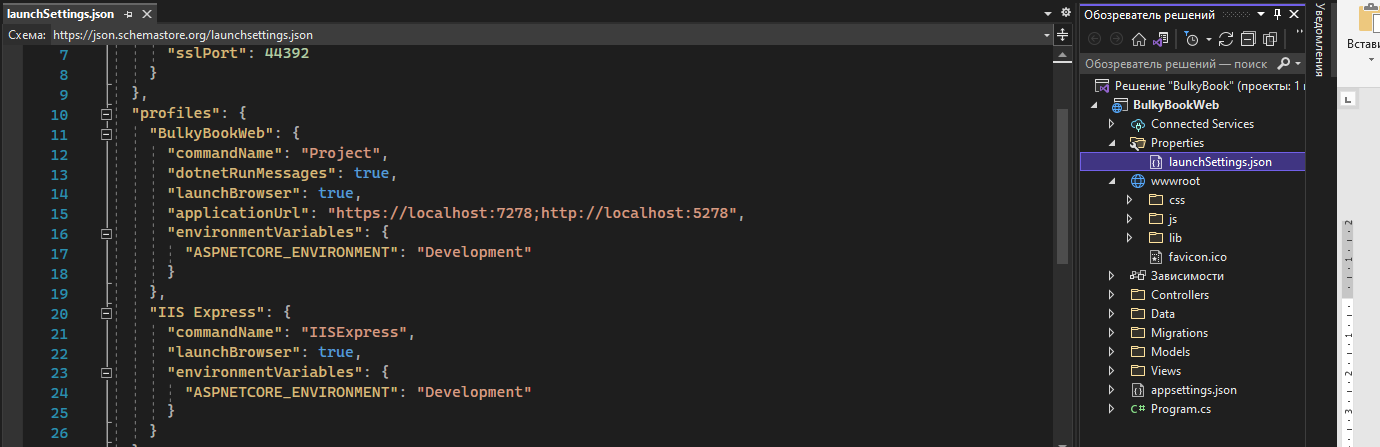
.NET 6.0 ASP MVC Entity





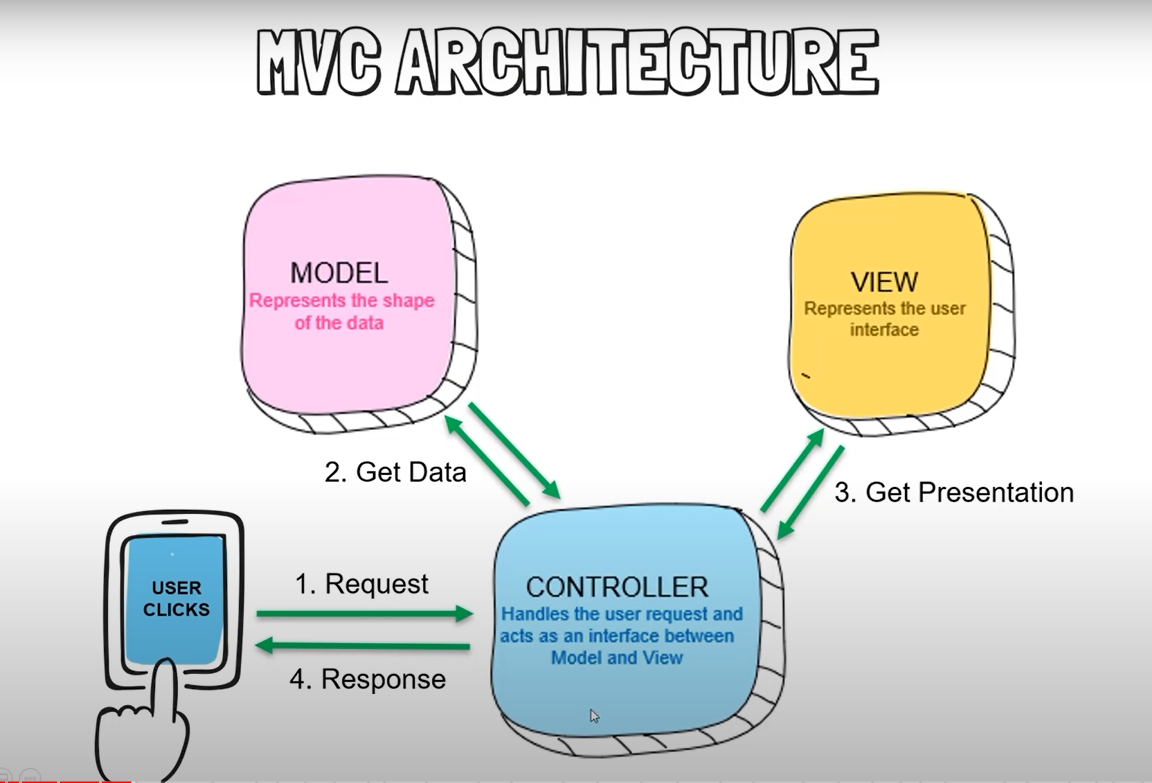


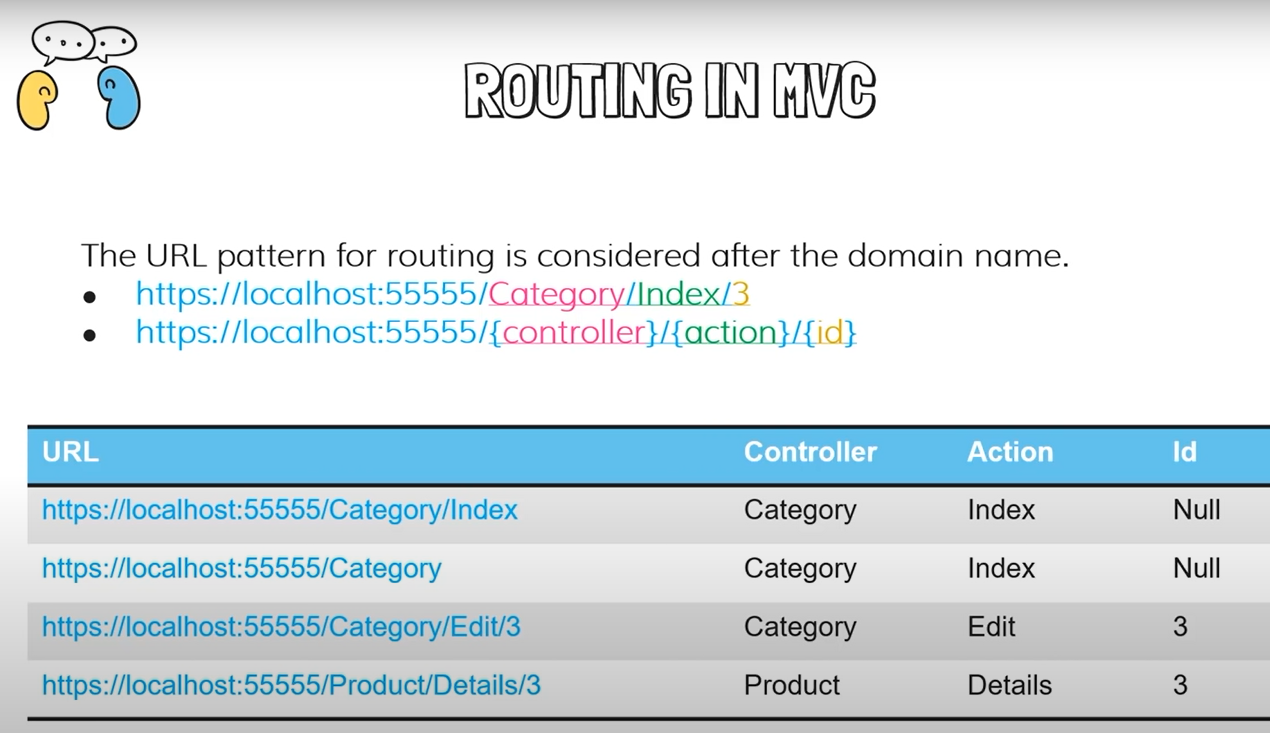
Клик на BulkyBookWeb 🡪 Изменить файл проекта. Тут описана версия .Net и все установленные пакеты NuGet



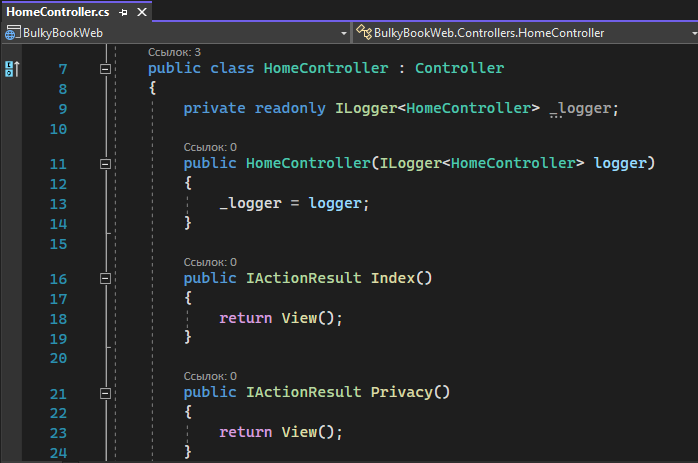
Какой способ для загрузки выбрать , номер Url , порт IIS или наш проект

Wwwroot – папка для статических данных: css js jquery img png bootstrap icons…





Есть контроллер HomeController (по умолчанию)

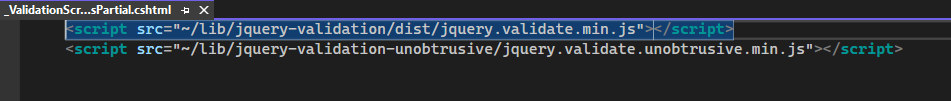


Этот Home контроллер имеет два дейсвтия Index и privacy. Это значит что в папке Views должна быть папка Home. А в папке Home должны быть файлы Index.cshtml и Privacy.cshtml. Они будут вызываться в Render.body() в середине \_Layout (Шаблона главной страницы). Суть заключается в том, что мы создаём кнопку button или поле перехода, связываем эту кнопку с нашим действием из контроллера с помощью asp-controller и asp-action . Пользователь нажимает на кнопку, запускается логика действия контроллера и пользователю возвращается вид из папки View/Home/action.

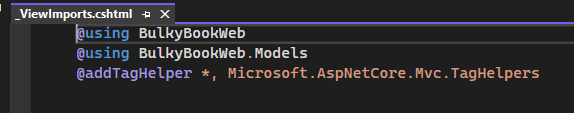
Папка Shared состоит из частичных видов

\_Layout – частичный вид (partial view), шаблон для построения большинства страниц, вверху прописываем css, потом header с менюшками, потом RenderBody() – вызов вида действия контроллера, потом футтер, потом Jquery, потом js .

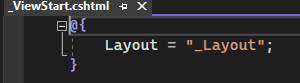
\_ValidationScriptsPartial – если в каком-то действии хотим включить проверку, то вызовем его



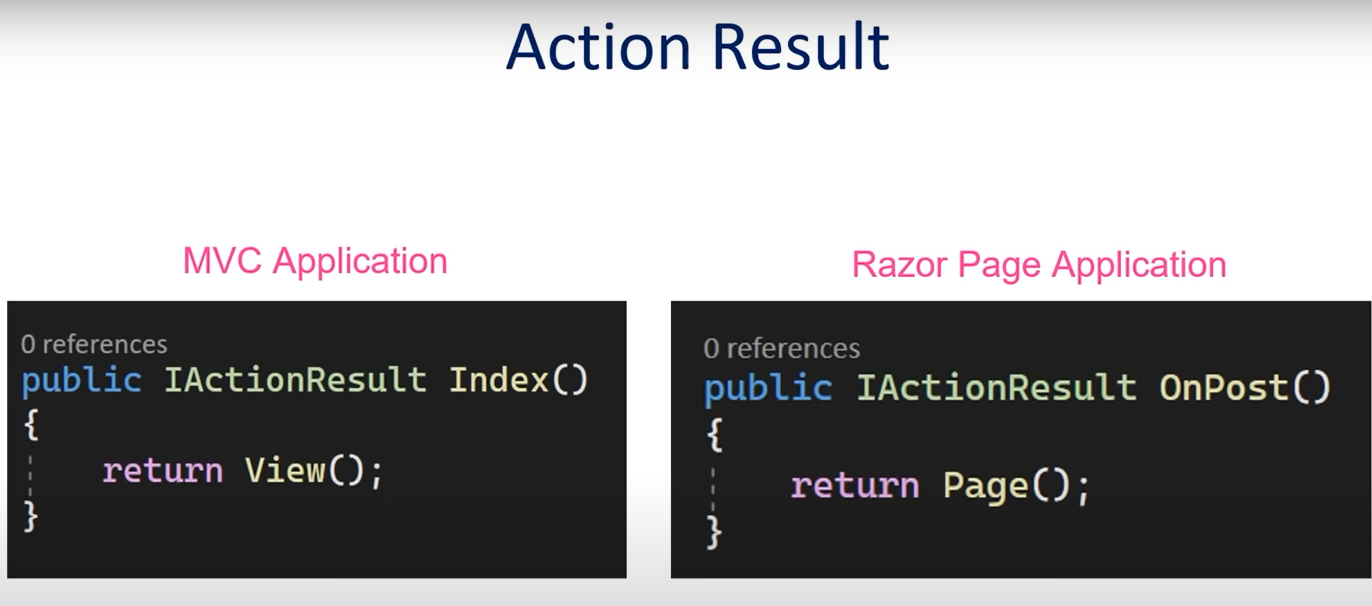
\_ViewImports – частичное представление в папке Views. Является глобальным пространством имён для всего проекта. Прописываем подключения всякие

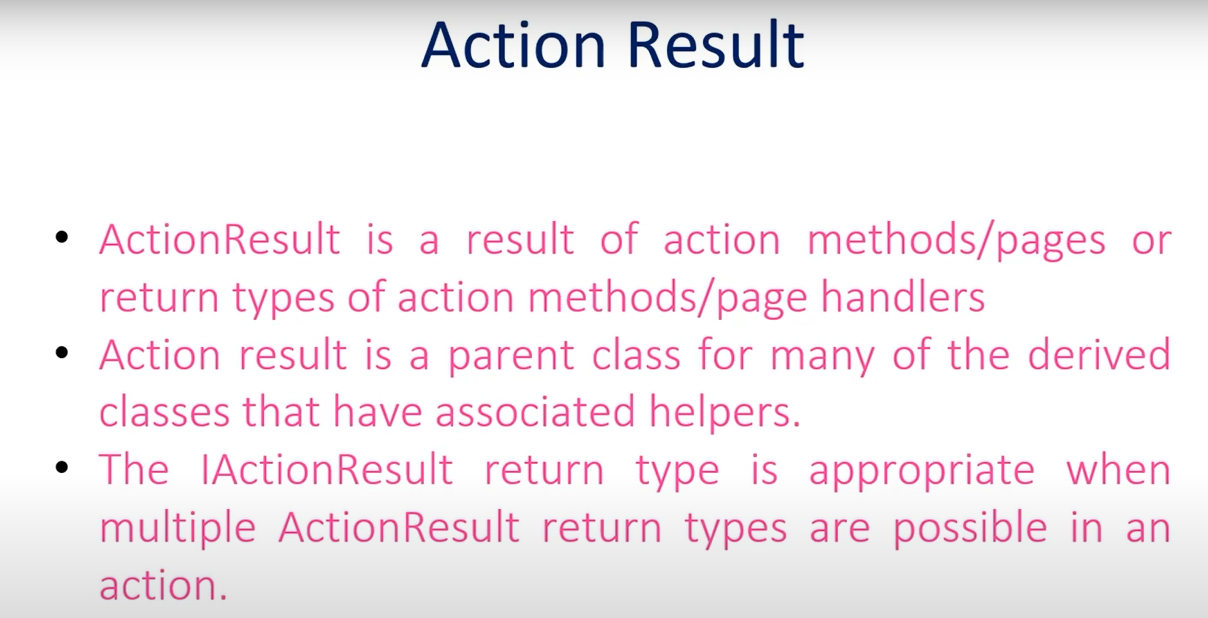


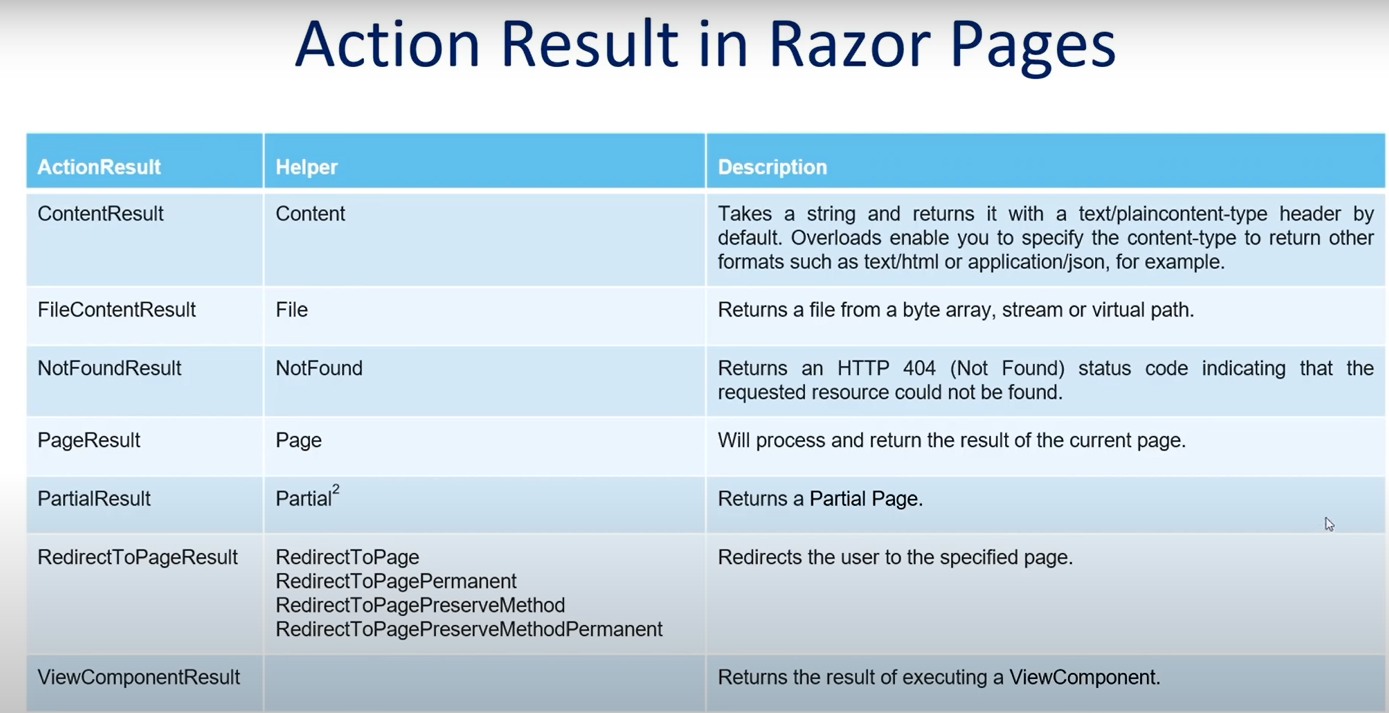
Также тут подключаются tagHelpers для всего проекта – это asp-controller asp-action asp-area и тд.

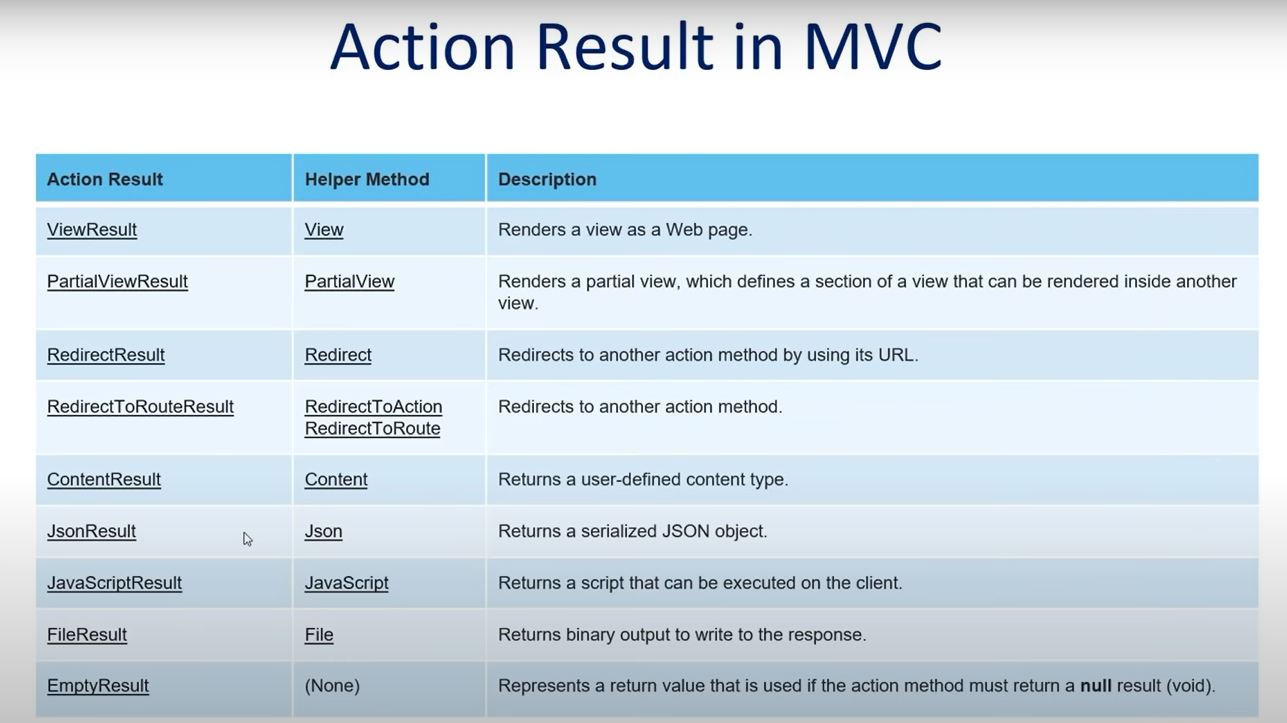


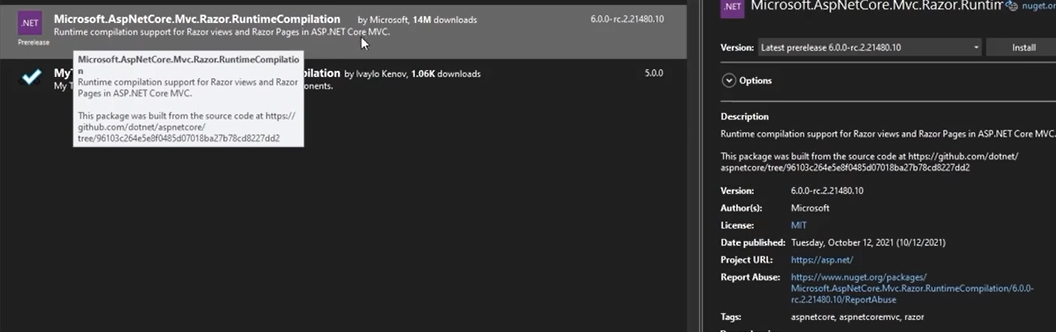
\_ViewStart – шаблон страинцы по умолчанию







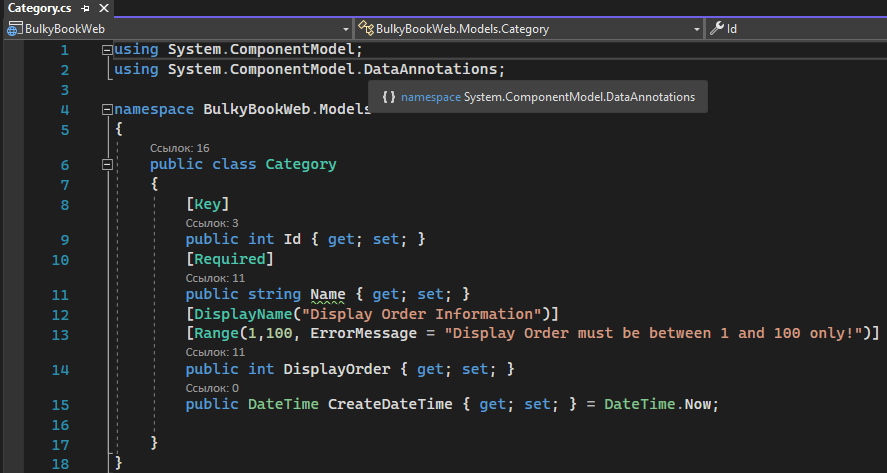




NuGet пакет для работы с razer страницами

Шаг 1

Создаём класс cs в папке Models и прописываем все свойства класса, присущие объекту. На основе этих свойств будет построена таблица в БД.

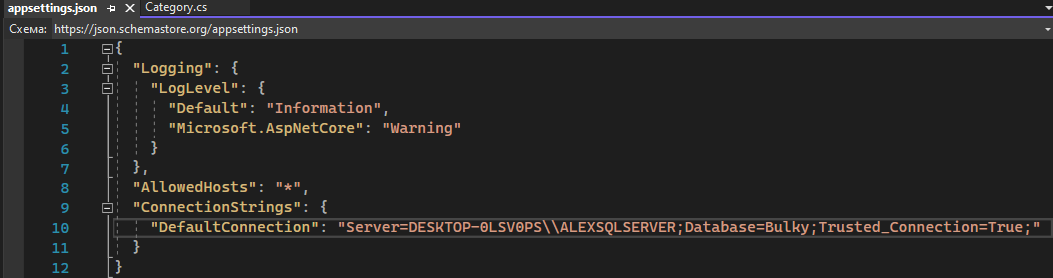


Обратите внимнаие на аннотацию данных, она подключается через using

Аннотация [key] – задаёт первичный ключ, [required] – обязательное поле …

Шаг 2

Заходим в sql server management studio. Надо убедиться , что есть доступ к любому среверу, в моём случае – это DESKTOP-0LSV0PS\\ALEXSQLSERVER. Копируем имя и преходим обратно в проект в файл appsettings.json

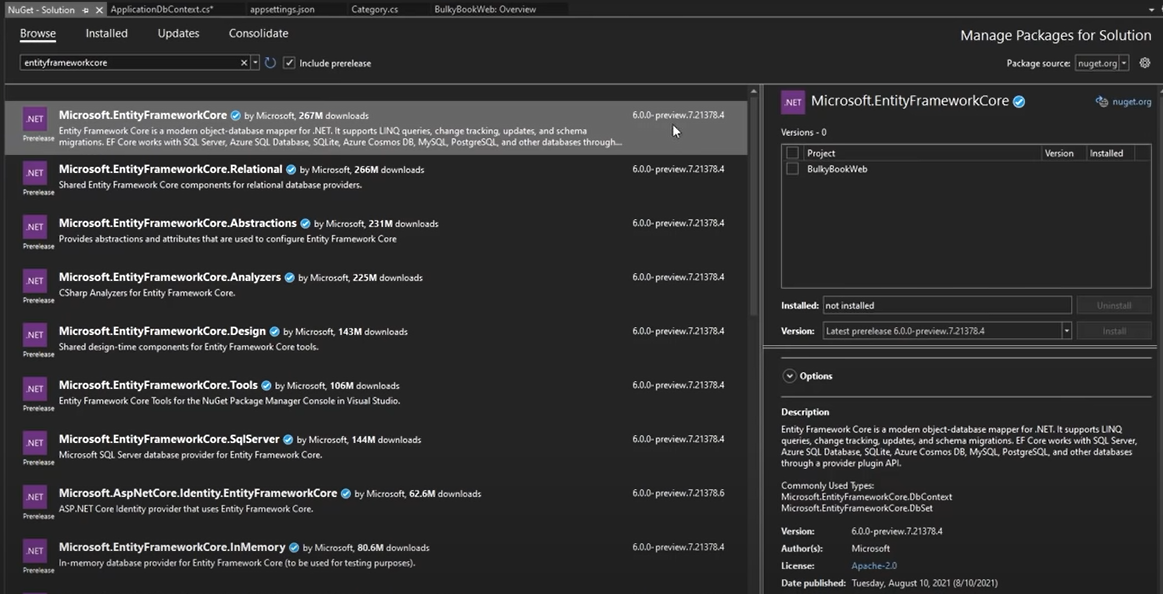


Мы прописали с 8 строки (,) по 12. В сервер вставили скопированное значение, название базы данных рэндом, соглашение подключения истина.

Шаг 3

Создаём новую папку Data для всех изменений с нашей базой данных. Создаём в этой папке класс ApplicationDbContext.cs .

Установка NuGet пакета Microsoft.EntityFrameworkCore

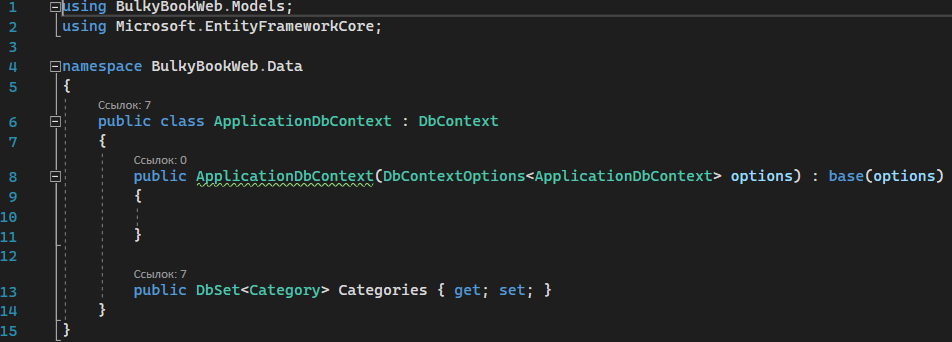


В файле ApplicationDbContext.cs используем using Microsoft.EntityFrameworkCore. А также наследуем класс от DbContext. Далее необходимо наладить соединеине с entity для этого в конструкторе класса ApplicationDbContext.cs передаём настройки родительскому классу



Также в этом классе задаётся таблица через DbSet

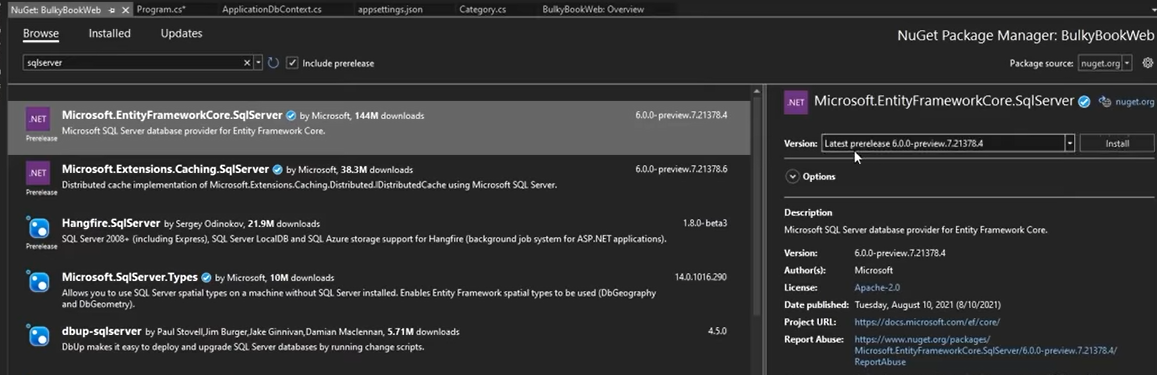
Сначала подключаем директорию нашей модели, потом через DbSet задаём таблицу и её имя. Весь класс ApplicationDbContext.cs



Далее надо подключить ConnectionString , котоый укажет путь и sql server. Для этого преходим в Program.cs . Тут подключаем дирректории Data .



Далее устанавливаем NuGet пакет Microsoft.EntityFrameworkCore.sqlServer



Добавляем новую директорию using Microsoft.EntityFrameworkCore;

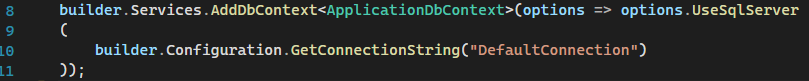


Далее в эдСервисис добавляем builder.Services.AddDbContext<ApplicationDbContext>(options => options.UseSqlServer

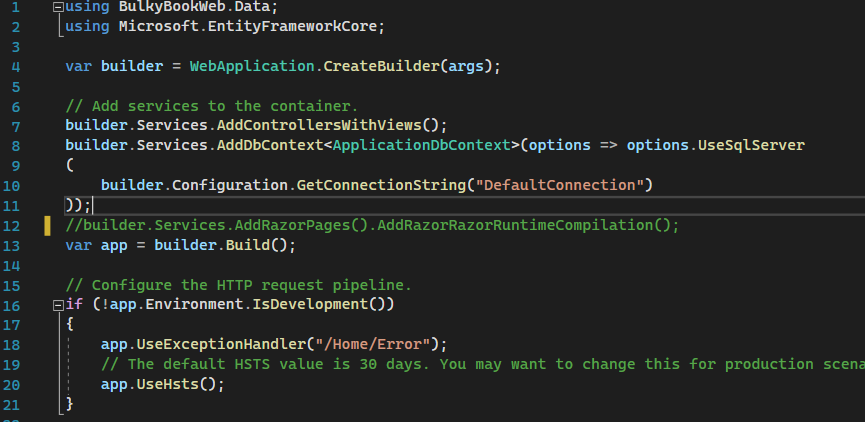
(

builder.Configuration.GetConnectionString("DefaultConnection")

));

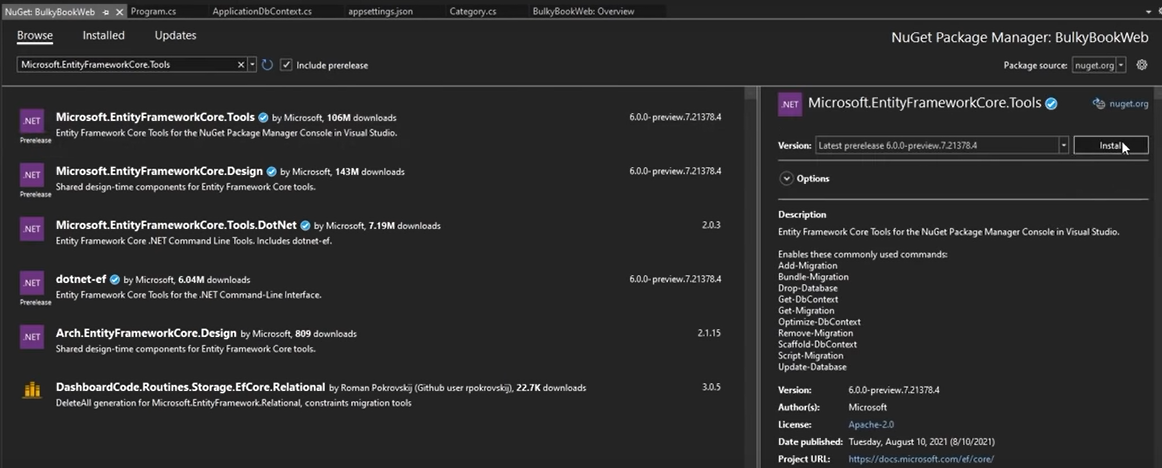


И получилось вот так



Шаг 4

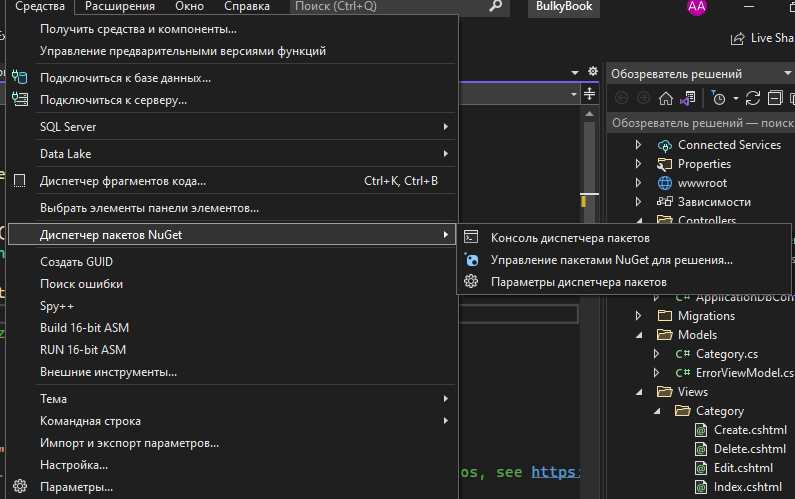
Устанвливаем NuGet пакет Microsoft.EntityFrameworkCore.Tools



Шаг 5

Устанвливаем миграцию для отпрвки изменений в нашу базу данных

Заходим в средства(Tools) в верхней панели 🡪 Диспетчер пакетов NuGet 🡪 Консоль диспетчера пакетов



Снизу откроется консоль и пропишем

add-migration NameOfMigration

Example: ( add-migration AddCategoryToDatabase )

После наступает success и в нашем проекте появляется новая папка Migrations

Сразу же открыт один из файдов с методами Up и Down.

Up – то , что должно произойти внутри миграции.

Down – то, что должно произойти , если что-то пойдёт не так. Будет запрос на основе скюэля и создастся база данных по нашему макету.

Как только мы убедились, что запрос праильный и соответсвует нашему ожиданию , то внутри прошлой консоли прописываем

update-database

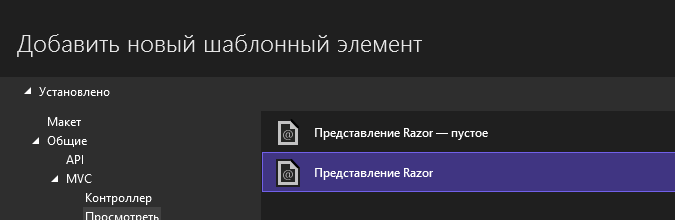
После success миграция произошла и можно посмотреть в sql server готовую бд и таблицу на основе нашего запроса

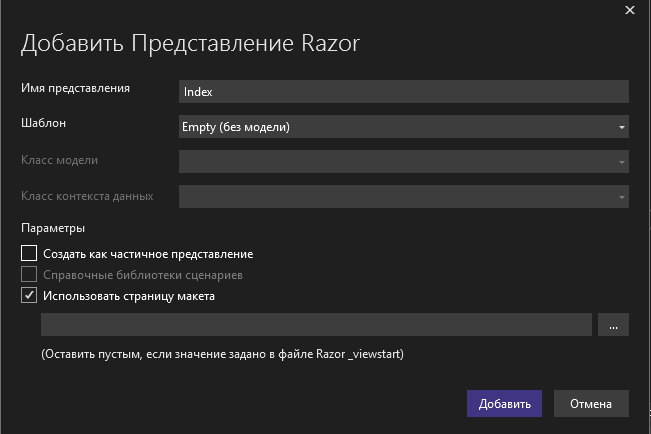
Шаг 6

Создаём в паке контроллерс новый контроллер CategoryControllers.cs

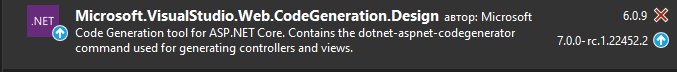
по шаблону MVC Controller – Empty

После этого в данном контроллере правым кликом на дейсвтвие Index 🡪 Добавить представление 🡪 Razor View





Нажмём добавить, если будет ошибка, то устаноим NuGet пакет



И повторим создание вида заново

После успеха создастся папка с именем контроллера в Views и в ней будет наше представление с имнем действия

Шаг 7

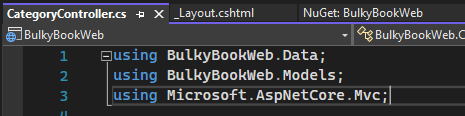
Наша цель отобразить в контроллере Category действии Index все данные таблицы. Для этого надо добавить соответсвующее поле в \_Layout, которое будет являться переходом на Category/Index. Для этого в \_Layout добавляем



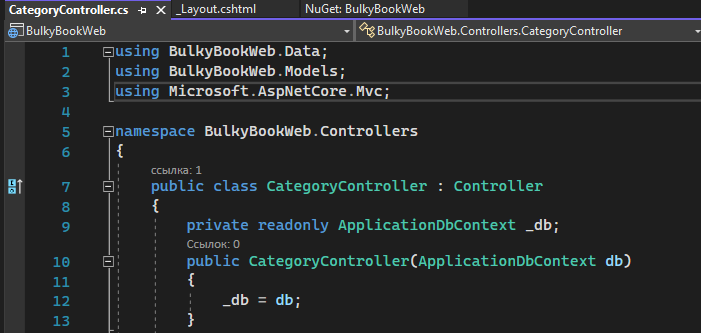
То есть при нажатии на Category будет переход с Home/Index на Category/Index.

Далее в Sql manager заполняем таблицу данными для демонстрации(не обязательно)

Далее в CategoryController подключаем новые директории Models и Data



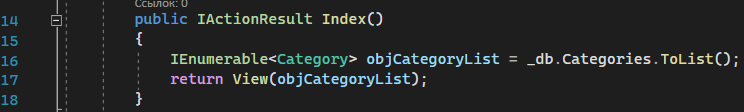
Далее создаём private readonly экземпля класса ApplicationDbContext с названием \_db и в конструкторе CategoryController пишем так



Так мы получили доступ к контексту, где уже создана база данных

Шаг 8

Прописываем действие Index в CategoryController



Мы говорим , что переменной objCategoryList типа списка из элементов типа Category присвоить даныые из экземпляра контекста.Таблицы.Преобразовать в список(). Вернуть objCategoryList в действие Index в качестве модели (model)

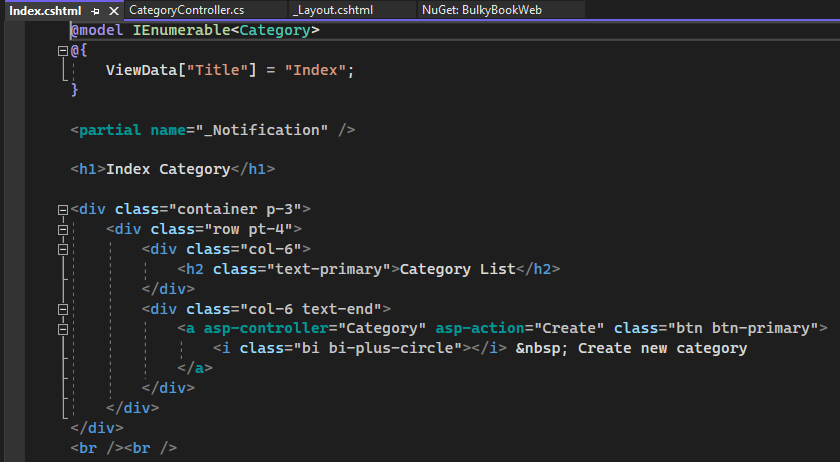
Чтобы получить данные objCategoryList в качестве модели, надо в Index прописать @model и тип переменной Ienumirable<Category>

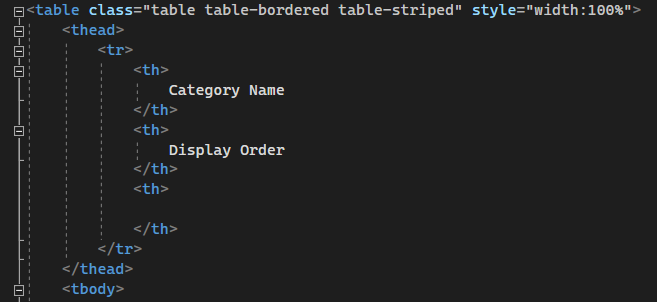


Шаг 9

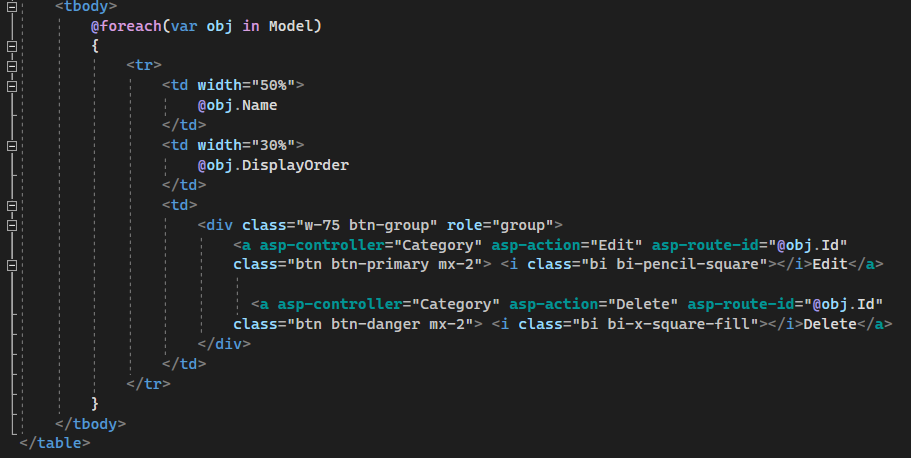
Razor страницы – такие страницы , где можно писать код с помощью html css , а также C#.

Index Razor page of CategoryController





Обрати внимание на следующую часть



Конец Index page

Для доступа к С# исспользуется @. Model – это наш полученный объект objCategoryList.

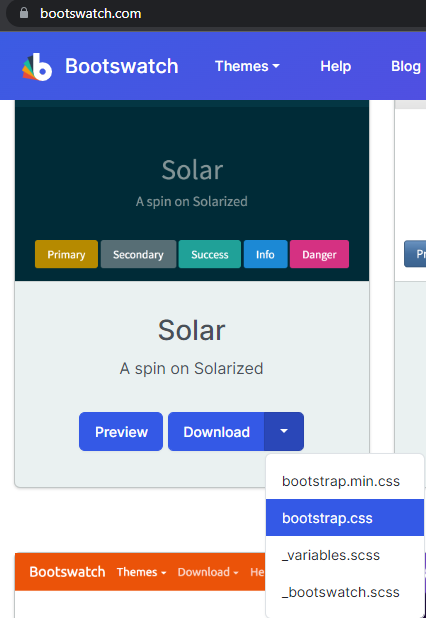
Для прехода на контроллер/действие/айди есть TagHelper: asp-controller, asp-action, asp-route-id

После написния Index можно запустить проект и перейти на Category, у которой по умолчанию вызовется действие Index и мы увидим всю нашу таблицу.

Шаг 10

Подключаем тему bootstrap…

Заходим на bootswatch.com, скролим вниз и выбираем нужную тему, в данном случае Solar, в download выбираем bootstrap.css



Файл скачается в блокнот, копируем всё и создаём новый Style Sheet(Таблица стилей) в wwwroot/css , называем bootswatchTheme.css и вставляем содержимое скачаного файла.

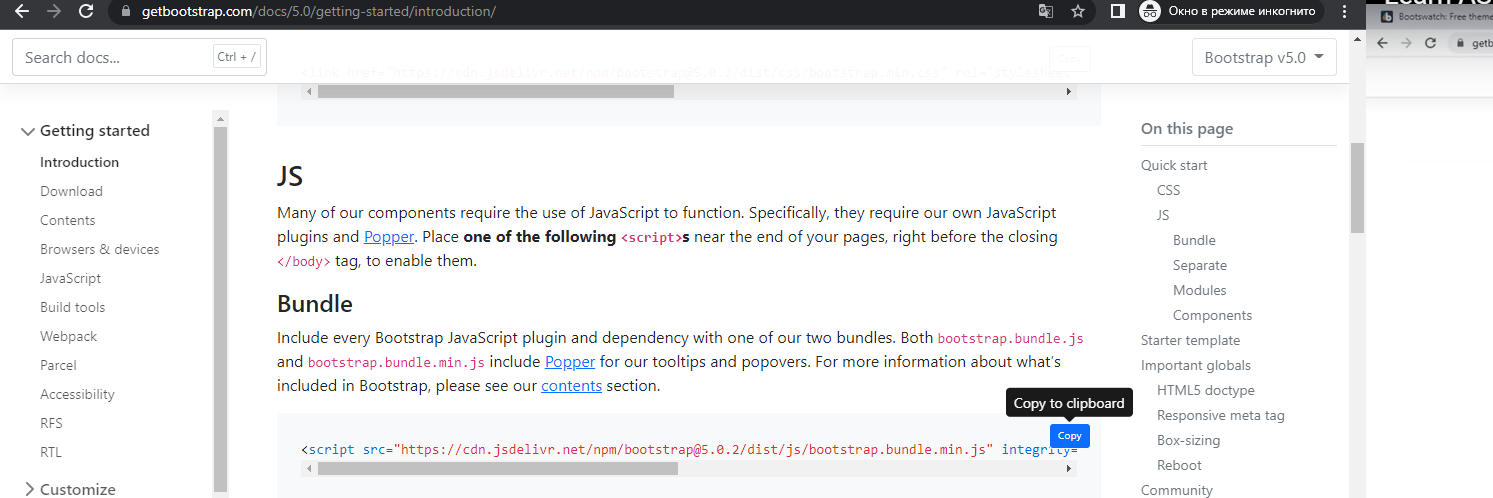
Далее в \_Layout это удаляем или коментируем



А это добавляем



Потом заходим на getbootstrap , кликаем на introduction и в правом верхнем углу выбираем bootstrap V5.0 Находим bundle 🡪 copy

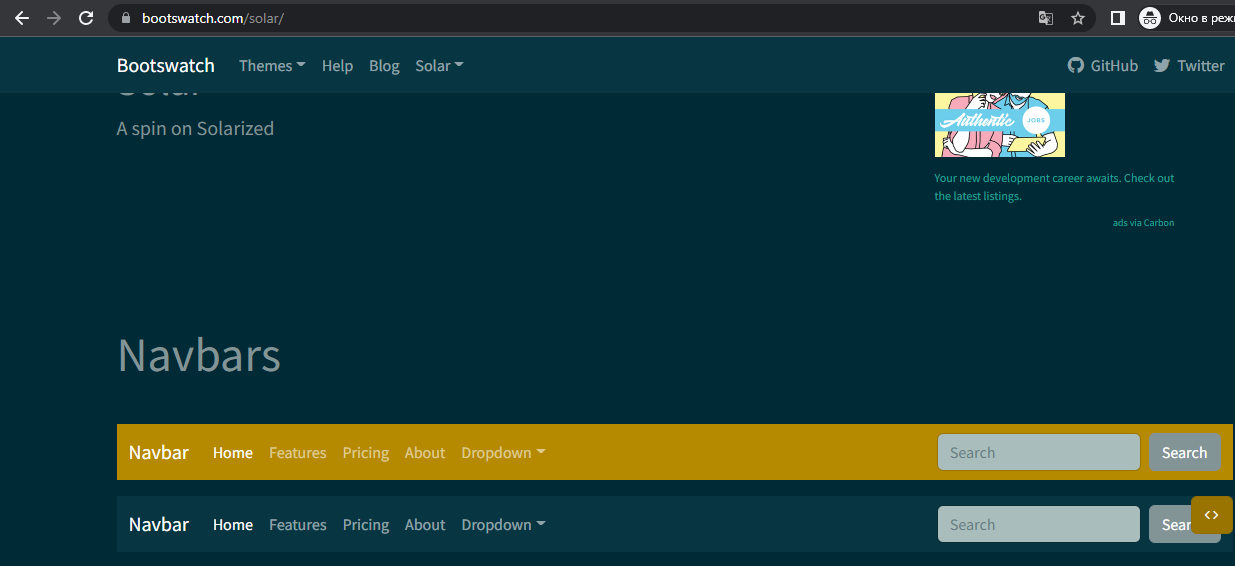


Потом в Лайоуте снизу удаляем скрипт bootstrap.bundle.min.js

И вставляем новый скопированный скрипт

Нажимаем Ctrl + S и запускаем проект и удивляемся красоте

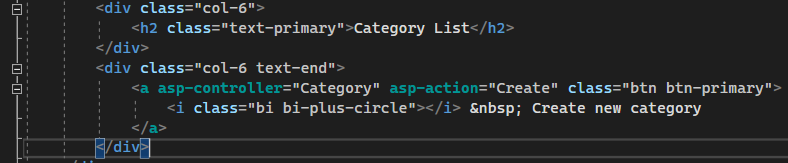
Далее заходим на bootswatch 🡪 Solar theme 🡪 Preview и копируем Navbar , а затем вставляем в проект в \_Layout



Шаг 11

Надо на странице Category/Index создать кнопку Create new Category .

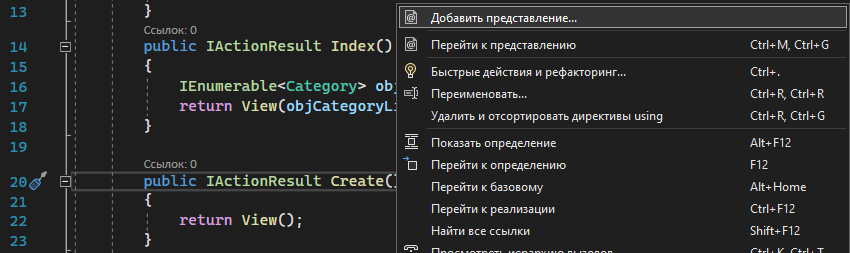
Для этого в Category/Index прописываем следующее



Значение иконки тега<i> берём на сайте бутстрап в разделе icons

Шаг 12

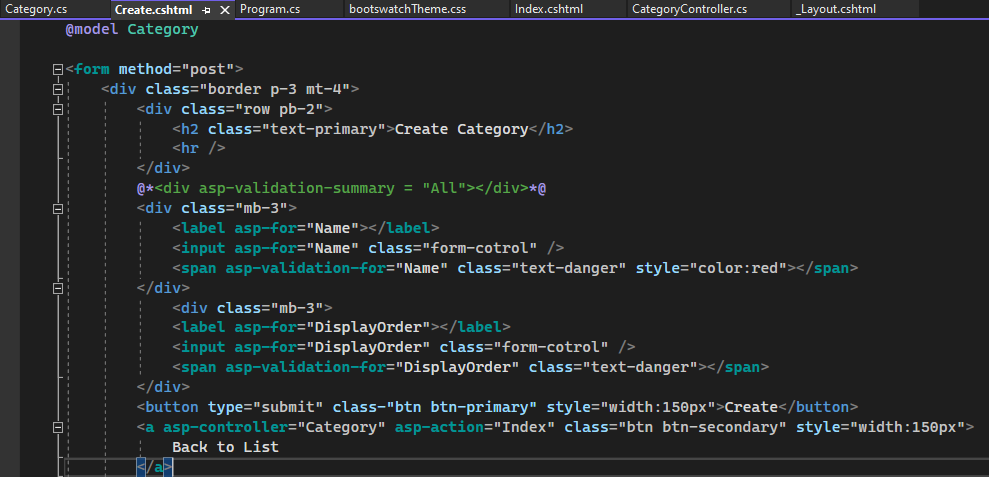
Создаём действие Create (запрос GET) в CategoryController🡪 правый клик по Create 🡪 Добавить представление

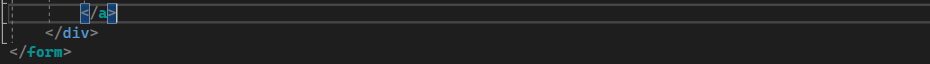


В созданном razor page Create пропишем Model через

@model Category , то есть не обязательно получать модель через действия контроллера

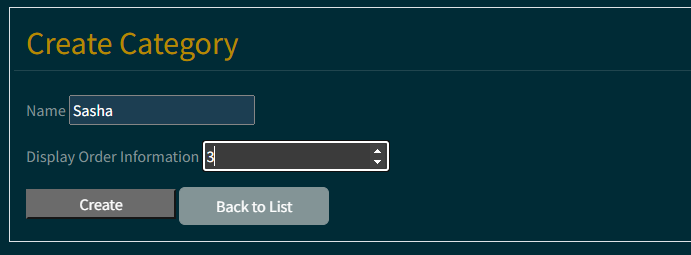
Смотри на Razor page Create



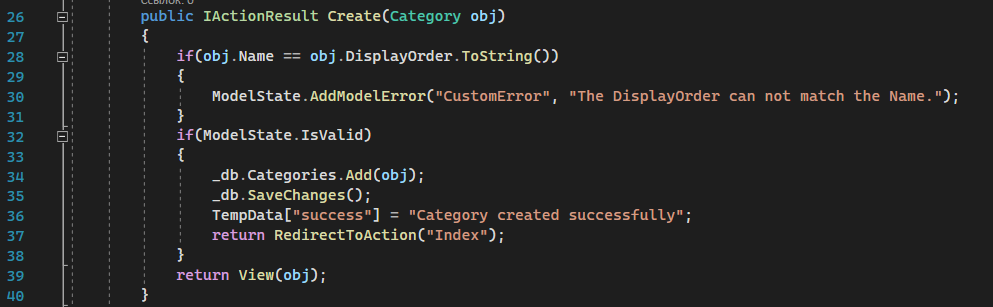


Обрати внимание на form method=”post” – это означает , что мы публикуем данные. Также посмотри на label asp-for и Input asp-for , они работают с полем Name из Model/Category

При запске проекта Category/Create видим



Но если нажмём Create , то ничего не произойдёт, надо прописать кнопку Create через TagHelpers , которые будут вести к новому действию Create в нашем контроллере. Два действия Create отличаются тем, что один без параметров, а другой с ними, таким образом компилятор понимает, какой именно нам нужен. И так прописываем Второе действие Create



Проверка 1 – поле 1 не должно быть равно полю 2 иначе выведи ошибку …

Проверка 2 – проверяет действительна ли модель , то есть она не должна быть равна 0 и не должна быть null. Модель в этом действии получаем из прошлого действия Create , эта модель передаётся в качестве параметра Category obj.

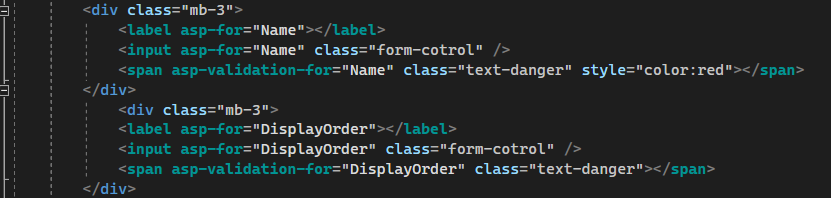
\_db.Categories.Add(obj) – добавит новую запись

\_db.SaveChanges() – сохранит зменения в БД

TempData[] – опишу потом, пока что сообщение об успехе

RedirectToAction – перейти на страницу другого действия

В Razor page Create посмотри на это



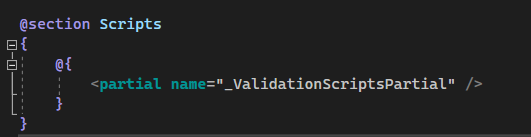
Span asp-validation-for – проверки, в случае каких-нибудь ошибок, всплывёт окно с их содержимым

Эта задокументированная строка выводит все ошибки вверху блока

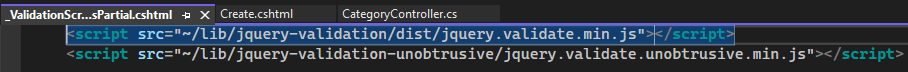


Шаг 13

Теперь при нажатии на кнопку на загруженной странице, страница постоянно обновляется – это значит , что мы бьём по серверу, что не очень хорошо, поэтому надо сделать так, чтоб проверка была на стороне клиента, а не сервера . Для этого на Razor page Create добавляем эту часть кода в конец



То есть мы подключаем частичнй вид, в котором , прописаны js за нас. Этот файл дефолтный с начала создания проекта



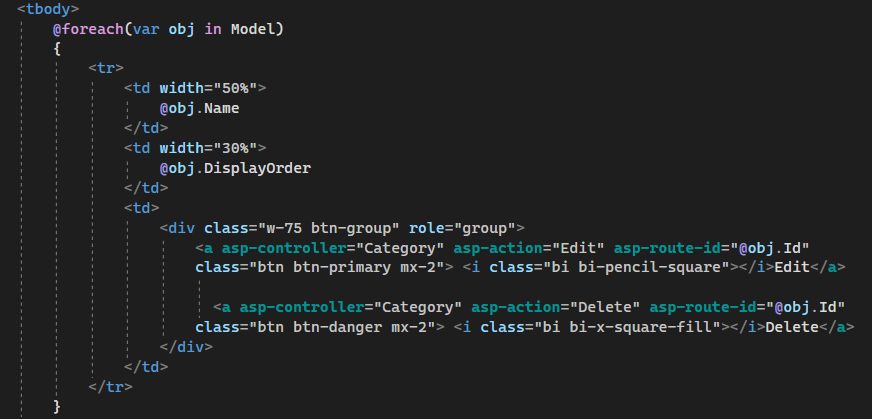
Теперь проверка идёт только на стороне клиента и сервер постоянно не обновляется

[DisplayName(“BlaBla”)] – аннотация , которая отображает имя BlaBla вместо корневого из Model/Category

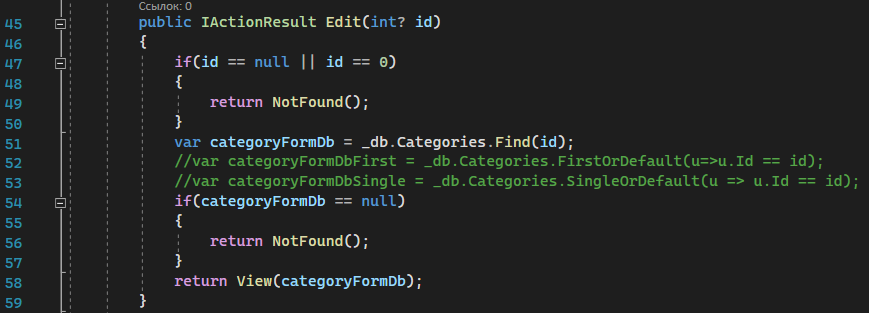
Конец действия Create, переходим к Edit(update)

Шаг 14

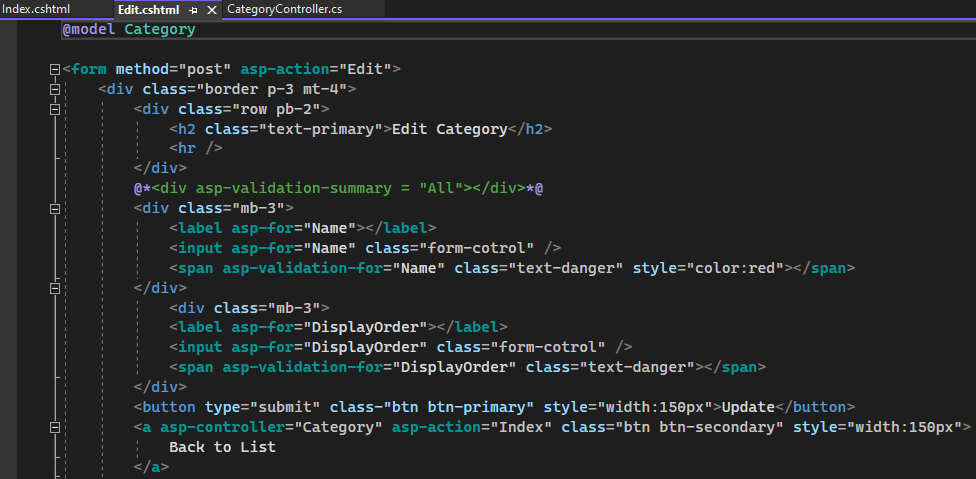
Аналогично в индексе создаём кнопку Edit

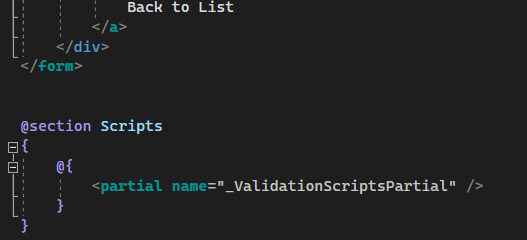


Прописываем логику Edit get

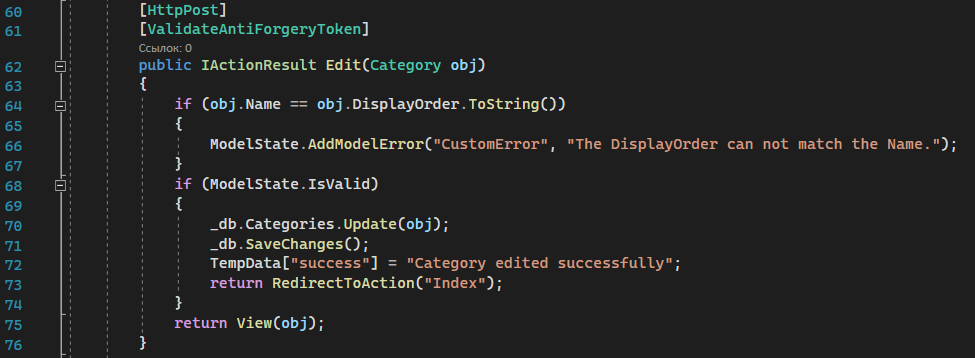


Параметр – лишь Id , ищем элемент БД по индексу, если мы его нашли , то вызываем Razor page Edit и передаём туда в качестве параметра этот найденый элемент





Обрати внимание на form method post Edit – это указывает, что при нажатии на button edit – мы исспользуем второе действие edit в нашем контроллере методом Post(мы его сейчас пропишем)



Add изменили на Update, а так всё понятно

Напоминаю ,что в индексе прописано



А это значит что тут новый тег хелпер asp-route-id , он передаст айди в качестве параметра действию Edit(int ? id)

Это конец Edit

Шаг 15

Делаем Delete

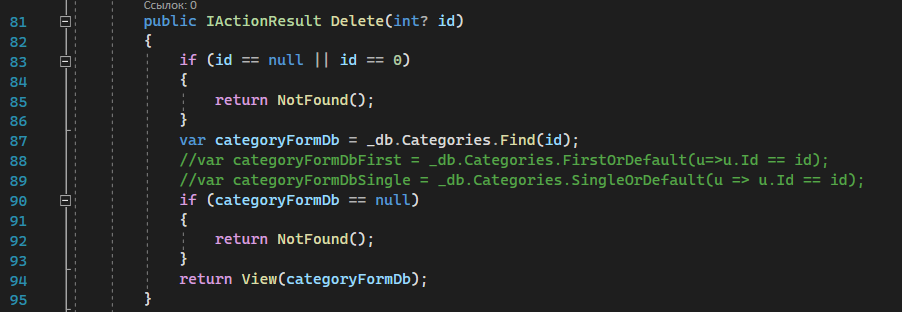
Создаём кнопку в индексе



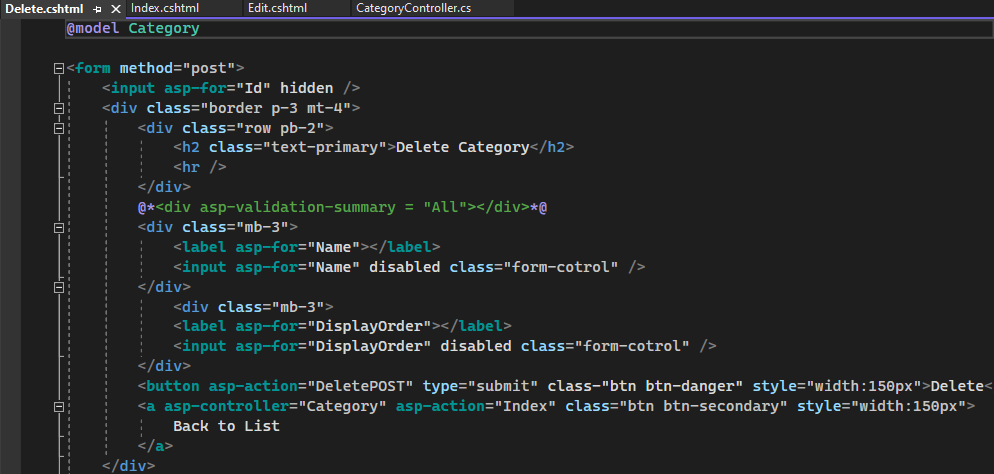
Контроллер действие айди

Айди в ккчетсве парамтра для нашего контроллера

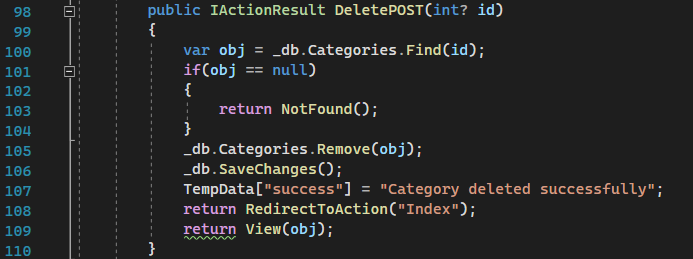
Прописываем действие delete (get) в контроллере



По айди находим нужный элемент и передаём его в качестве модели для razor page delete

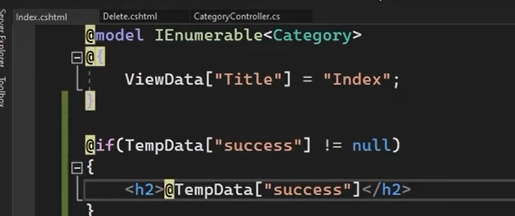


При нажатии на кнопку delete запустится действие deletePOST в контроллере

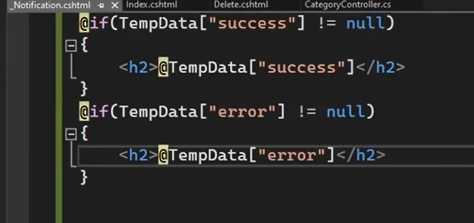


TempData[] – временные данные , отображаются до обновления страницы, один переход и они теряются. Идеально подходит для того, чтоб отобразить сообщение об успехе после действия контроллера.

Так они вставляются на страницу Index, чтоб работали



Второй способ – сощдать частичный вид в shared



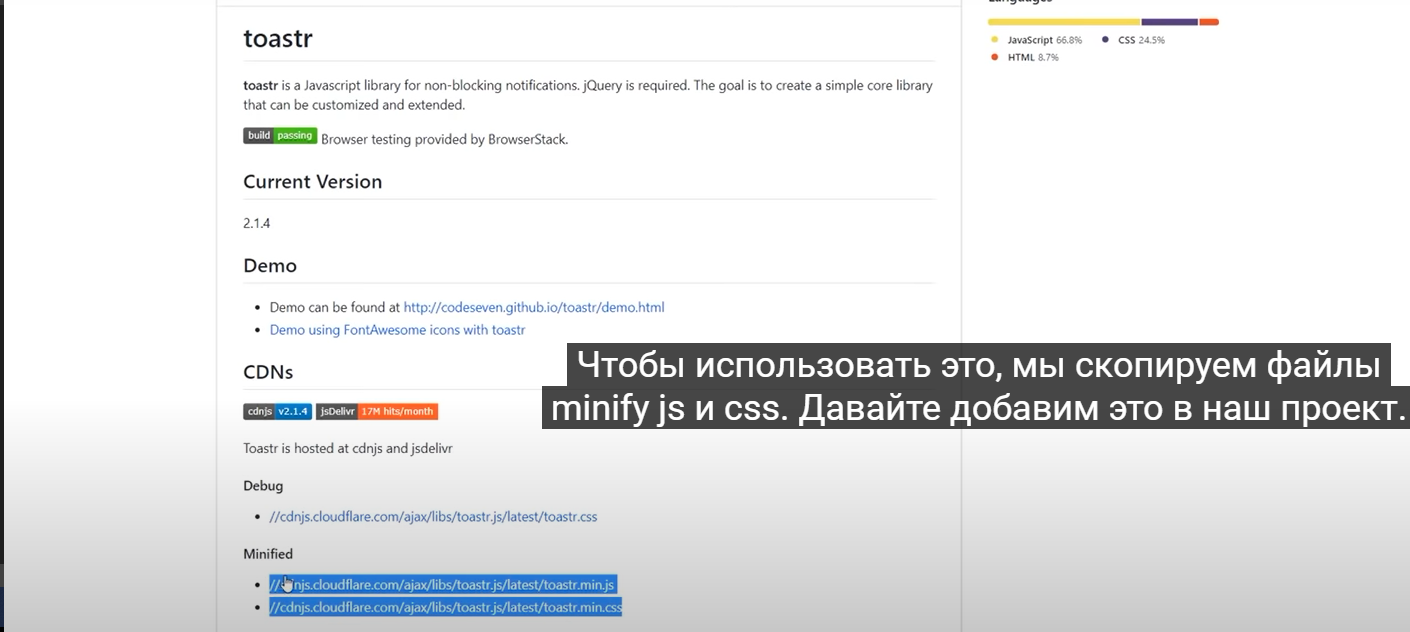
И объявить его на нужной странице например индекс



Шаг 16

Делаем уведомления красивыми и необычными с помощью js и toastr

Заходим на гитхаб тоастр копируем два пути из Minified



Путь css вставляем в \_Layout



А js путь вставляем в частичный вид \_notification

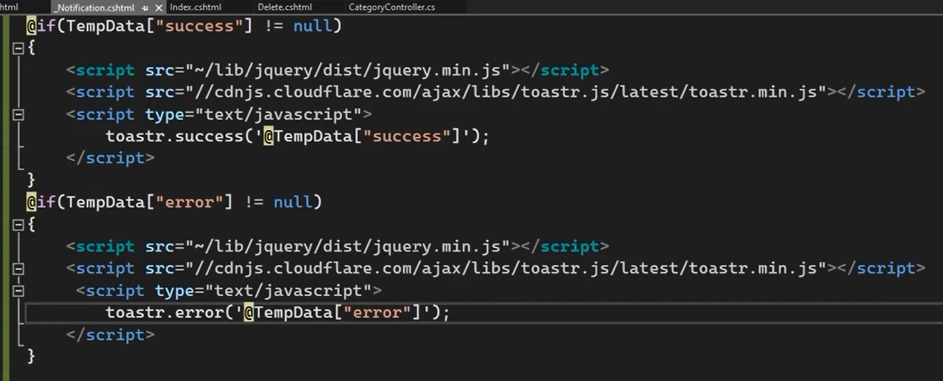


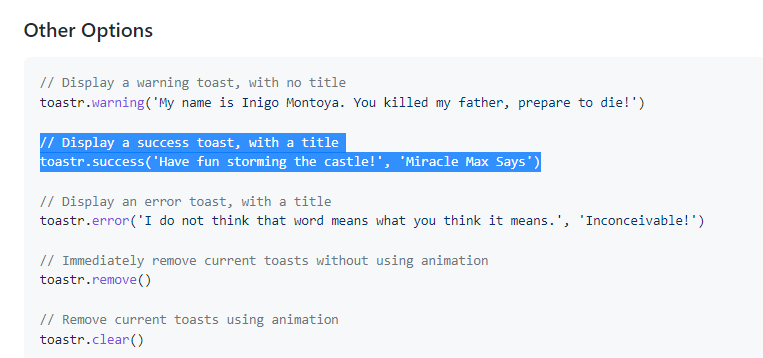


Далее сюда же добавляем jquery из wwwroot/lib/jquery/dist/jquery.min.js



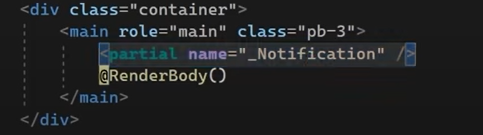
Хер знает, что дальше





Теперь красивые уведомления

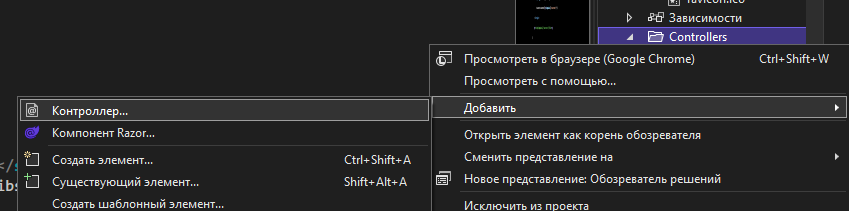
Также есть смысл не вызывать partial view , на какой-то странице, можно его сразу в \_Layout перед RenderBody() сунуть и эти Notification будут работать на весь шаблон



Шаг 17

Сгенерировать автоматический контроллер со всеми действиями сразу(креэйт делит едит…)

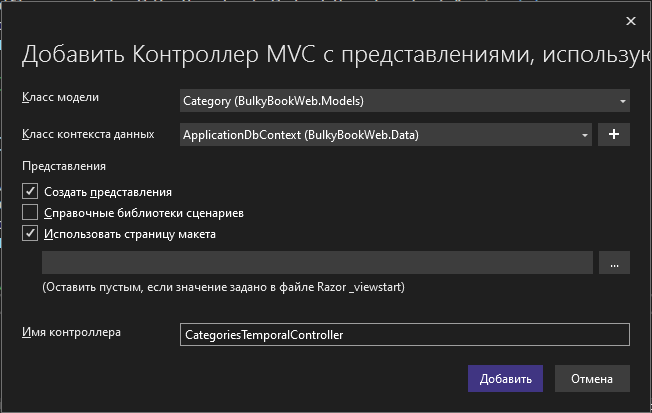
Нажимаю добавить контроллер



Выбираю это



Выбираю основной класс Models/Category и доступный контекст БД, задаю имя



По идее создастся контроллер со всеми этими действиями и готовыми Razor pages , Надо лишь подшаманить , изменить их под себя. Лично у меня наверное нестыковка версий NuGet пакетов , поэтому не прёт.

Шаг последний

Публикация на платформе Azure…

В жопу её

Ссылка на видос

<https://youtu.be/hZ1DASYd9rk>

Заливать будем сюда

