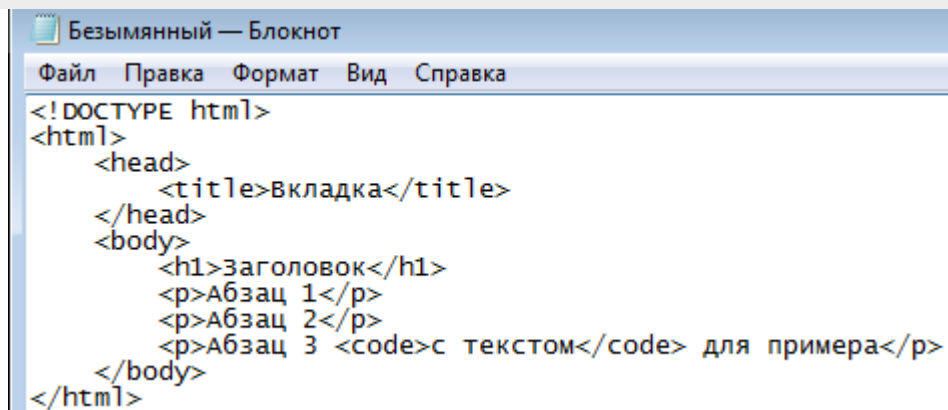


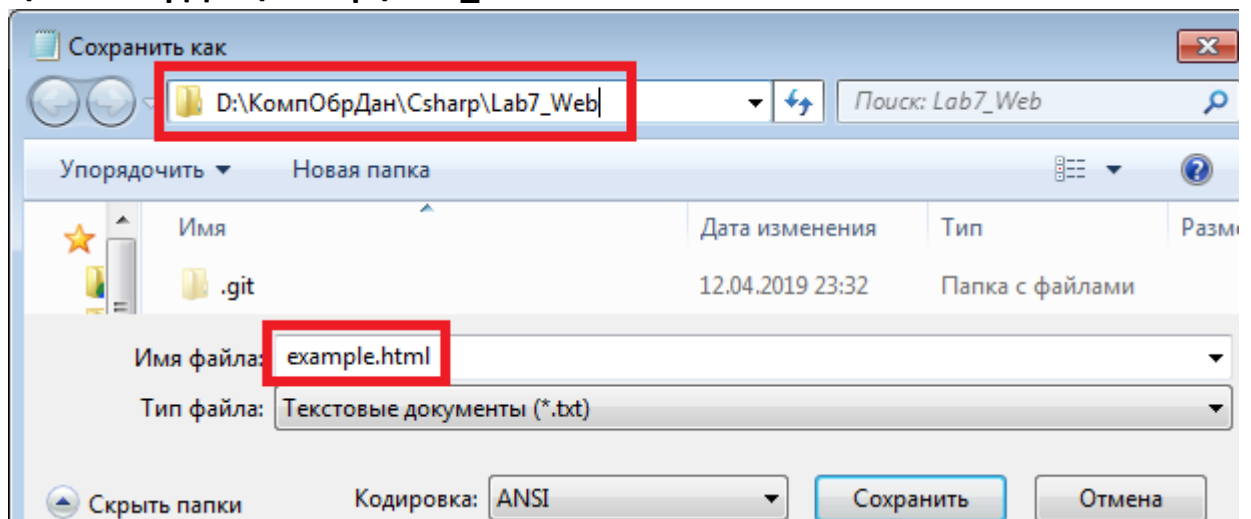
Лабораторная работа № 7 «Web»

1. Для создания веб-страниц используются языки **HTML** (HyperText Markup Language — «язык гипертекстовой разметки») и **CSS** (Cascading Style Sheets — каскадные таблицы стилей). **HTML** отвечает за структуру и содержание страницы, **CSS** — за внешний вид.
2. Открыть **Блокнот**, и в нём ввести следующий текст:

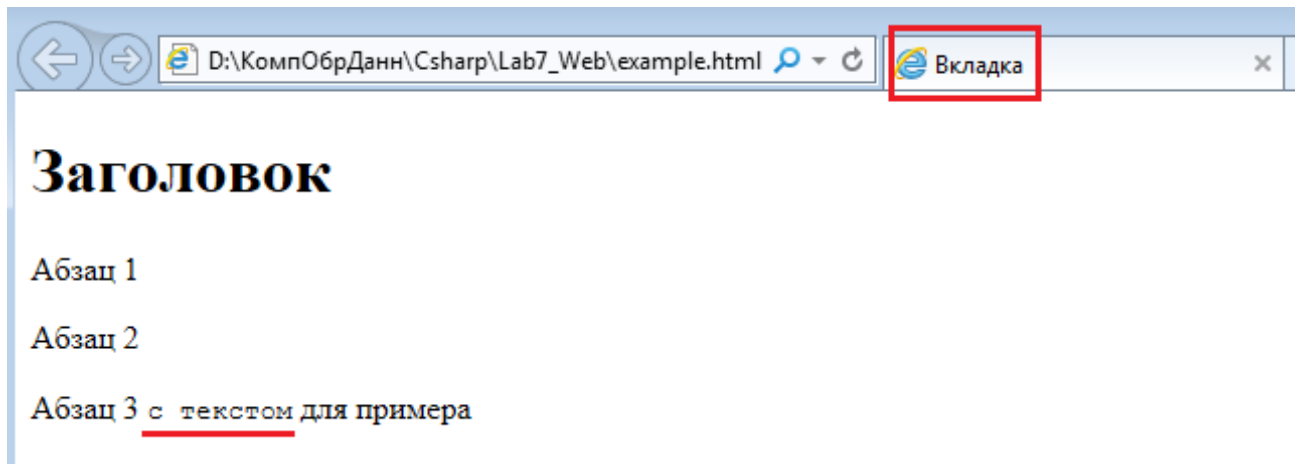
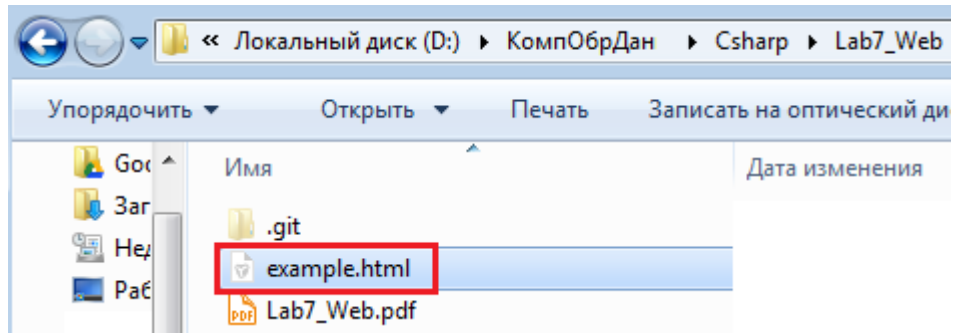
```
<!DOCTYPE html>
<html>
  <head>
    <title>Вкладка</title>
  </head>
  <body>
    <h1>Заголовок</h1>
    <p>Абзац 1</p>
    <p>Абзац 2</p>
    <p>Абзац 3 <code>с текстом</code> для примера</p>
  </body>
</html>
```



3. Сохранить файл под именем **example.html** в папку **D:\КомпОбрДан\Csharp\Lab7_Web**



4. Запустить сохранённый файл. Он откроется в браузере.



5. Элементы в угловых скобках `< >` (например `<head>`, `<body>`, `<h1>`, `<p>` и т.д.) называются тегами.

Каждый тег начинается с символа `<` и заканчивается символом `>`.

Все теги можно разделить на парные и одиночные. Каждый парный тег состоит из двух частей: открывающего тега и закрывающего. Внутри закрывающего тега используется символ `/`, например `<p> </p>`.

У каждого тега своё назначение, например, в тегах `<body> </body>` размещается контент, который должен быть виден на странице, а в тегах `<head> </head>` размещается заголовок страницы и техническая информация. Более подробно о назначении тегов можно прочитать, например, в <http://htmlbook.ru> или <https://html5book.ru> и т.д.

Теги отвечают за разметку текста на странице, а браузер, обнаруживая эту разметку отображает текст в соответствующем этой разметке виде.

6. CSS состоя из множества правил, примерно такого вида:

```
селектор {  
    свойство1: значение1;  
    свойство2: значение2;  
}
```

Количество свойств является очень большим.

Например:

```
body {  
    padding: 0 20px;  
    font-size: 20px;  
    font-family: "Consolas", sans-serif;  
}
```

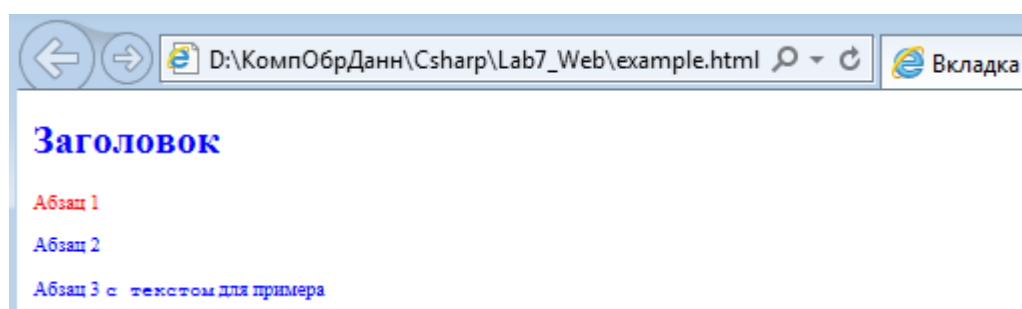
или

```
h1 {  
    color: #618ad2;  
    text-shadow: 2px 2px 0 #cccccc;  
}
```

CSS могут быть расположены как внутри некоторых тегов, так и в отдельном файле. Каждый тег поддерживает только определённый набор свойств и стилей.

7. В **Блокноте** внести изменения в разметку файла **example.html** как в примере ниже, сохранить файл, и открыть его в браузере. Это пример стилей внутри тегов. Очевидно, что если текст большого размера и/или большое число элементов текста требуют своих стилей, то задание стилей внутри тегов становится проблематичным, ориентирование в тексте усложняется, повышается вероятность ошибки.

```
<!DOCTYPE html>  
<html>  
    <head>  
        <title>Вкладка</title>  
    </head>  
    <body style="font-size: 10px; color: blue;">  
        <h1>Заголовок</h1>  
        <p style="color: red;">Абзац 1</p>  
        <p>Абзац 2</p>  
        <p>Абзац 3 <code>с текстом</code> для примера</p>  
    </body>  
</html>
```

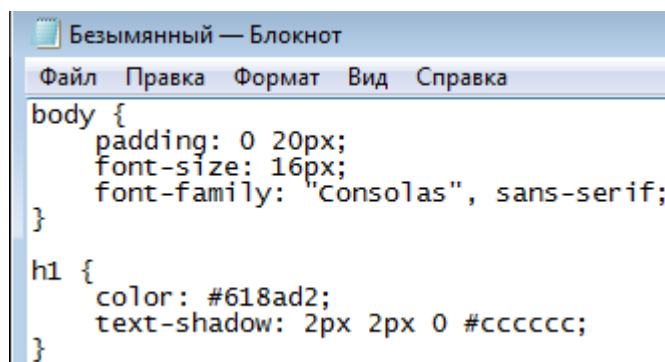


8. Создать в **Блокноте** новый файл, вписать в него код ниже, и сохранить его в папке **D:\КомпОбрДан\Csharp\Lab7_Web** под именем **style.css**

В данном случае в селекторах задаются стили отображения для текста внутри тегов, имена которых соответствуют именам селекторов.

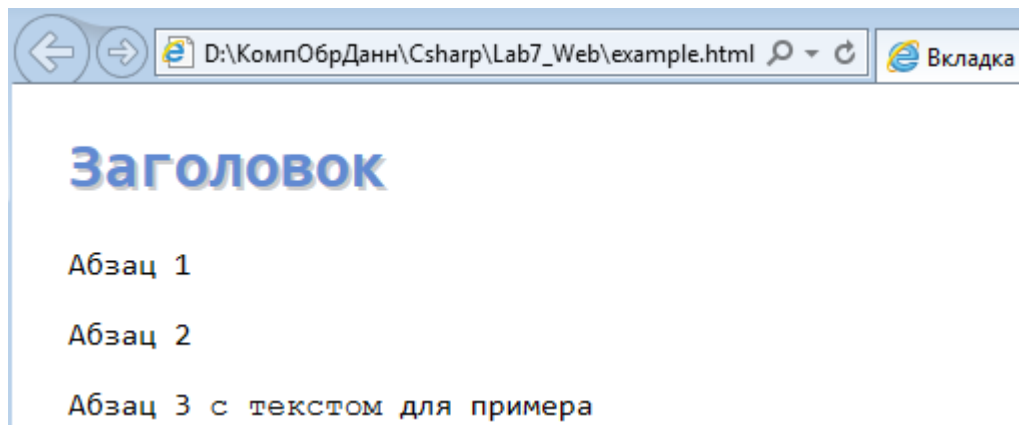
```
body {
    padding: 0 20px;
    font-size: 16px;
    font-family: "Consolas", sans-serif;
}

h1 {
    color: #618ad2;
    text-shadow: 2px 2px 0 #cccccc;
}
```



9. Вернуться в **Блокноте** к файлу **example.html**, и внести в разметку изменения, сделав ссылку на файл со стилями, находящийся в этой же папке. Сохранить изменения, и открыть файл **example.html** в браузере

```
<!DOCTYPE html>
<html>
  <head>
    <title>Вкладка</title>
    <link rel="stylesheet" href="style.css">
  </head>
  <body>
    <h1>Заголовок</h1>
    <p>Абзац 1</p>
    <p>Абзац 2</p>
    <p>Абзац 3 <code>с текстом</code> для примера</p>
  </body>
</html>
```



10. Если необходимо задать стили для произвольного текста, то для требуемых элементов текста назначается соответствующий класс.

В файле **example.html** внести следующие изменения:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Вкладка</title>
    <link rel="stylesheet" href="style.css">
  </head>
  <body>
    <h1>Заголовок</h1>
    <p class="red">Абзац 1</p>
    <p class="s20">Абзац 2</p>
    <p class="other-font">Абзац 3 <code>с текстом</code>
    для примера</p>
  </body>
</html>
```

В файле **style.css** внести следующие изменения:

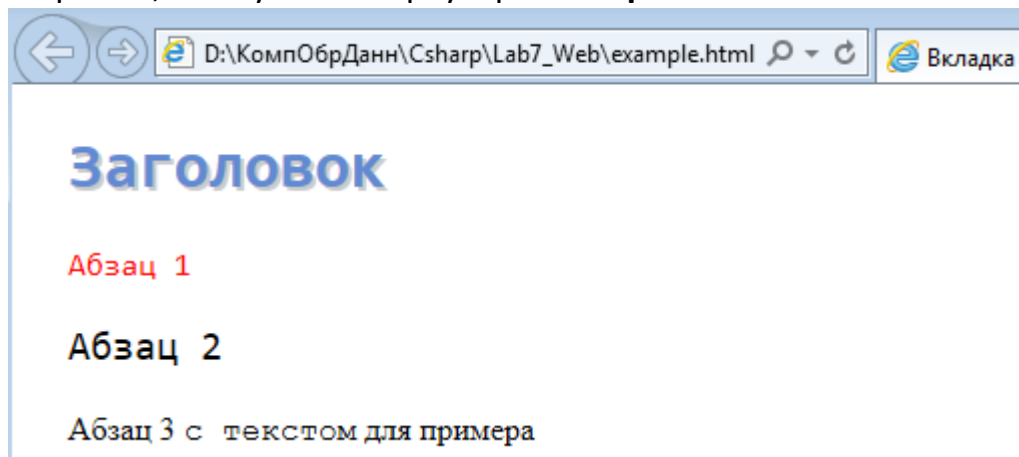
```
body {
  padding: 0 20px;
  font-size: 16px;
  font-family: "Consolas", sans-serif;
}

h1 {
  color: #618ad2;
  text-shadow: 2px 2px 0 #cccccc;
}

.red {
  color: red;
}
```

```
.s20 {  
    font-size: 20px;  
}  
  
.other-font {  
    font-family: "Times";  
}
```

Сохранить файлы, и запустить в браузере **example.html**



11. Файл **example.html** представляет собой неизменяемую (статическую) веб-страницу. Для изменения отображаемой информации необходимо редактировать содержание файла каждый раз, что является проблематичным. Для того, чтобы веб-страница была динамическая (меняющаяся), используется язык **JavaScript**. Код на языке **JavaScript** вставляется в **HTML**-разметку, и позволяет менять отображаемую в браузере информацию не внося изменений в **HTML**-файл.

Чтобы вставить **JavaScript** в **HTML** страницу, используется тег **<script>**, строки между **<script>** и **</script>** исполняются браузером.

Внутри тега **<script>** используется атрибут **type** для определения языка сценариев.

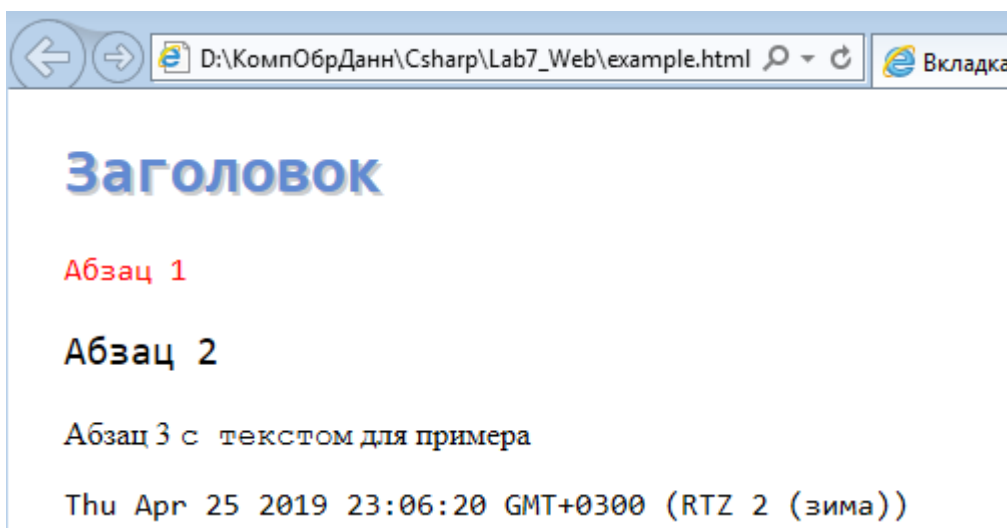
12. В файле **example.html** внести изменения, сохранить, и открыть в браузере. Пример ниже вставляет текущую дату в элемент **<p>**. Браузер заменит содержимое **HTML** элемента с идентификатором **id="demo"** на текущую дату. Если обновлять страницу браузера, то время будет меняться.

```
<!DOCTYPE html>
<html>
  <head>
    <title>Вкладка</title>
    <link rel="stylesheet" href="style.css">
  </head>
  <body>
    <h1>Заголовок</h1>
    <p class="red">Абзац 1</p>
    <p class="s20">Абзац 2</p>
    <p class="other-font">Абзац 3 <code>с текстом</code>
    для примера</p>

    <p id="demo">Абзац 4</p>

    <script type="text/javascript">
      document.getElementById("demo").innerHTML=Date();
    </script>

  </body>
</html>
```



Без тегов **<script>**, браузер будет интерпретировать **"document.getElementById("demo").innerHTML=Date();"** как обычный текст и просто выведет его на страницу.

13. Точно также, как и в случае с **CSS**, код **JavaScript** можно вынести в отдельный файл. Создать в **Блокноте** новый файл, внести в него следующий код, и сохранить его в папке **D:\КомпОбрДан\Csharp\Lab7_Web** под именем **xxx.js**

```
//случайное число между 0 и 1
document.write(Math.random() + "<br />");

//случайное число между 0 и 10
document.write(Math.floor(Math.random()*11));

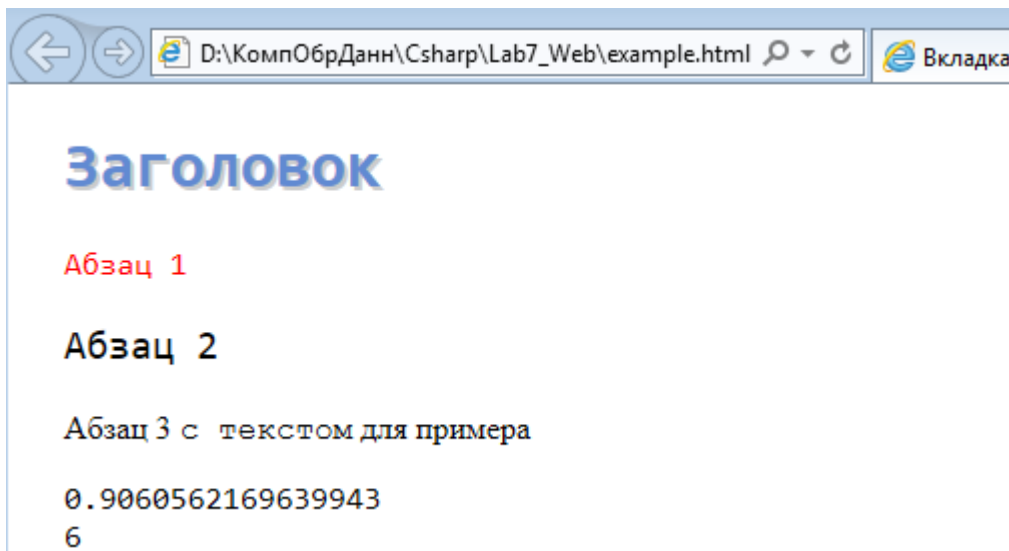
// автоматическое обновление страницы в браузере
// через 1000 миллисекунд (1 секунду)
setTimeout(function(){
    location.reload();
}, 1000);
```

14. В **Блокноте** отредактировать файл **example.html**, и запустить его в браузере. Если обновлять страницу браузера, то результат будет меняться.

```
<!DOCTYPE html>
<html>
  <head>
    <title>Вкладка</title>
    <link rel="stylesheet" href="style.css">
  </head>
  <body>
    <h1>Заголовок</h1>
    <p class="red">Абзац 1</p>
    <p class="s20">Абзац 2</p>
    <p class="other-font">Абзац 3 <code>с текстом</code>
    для примера</p>

    <script type="text/javascript" src="xxx.js">
    </script>

  </body>
</html>
```

15. Общая суть построения веб-страниц примерно такая, как описано выше.

То есть веб-страница состоит из:

- содержание (текст, графика),
- разметка **HTML** (функциональное назначение элементов и их взаимное расположение относительно друг друга),
- стили **CSS** (визуальное оформление текста и графики),
- исполняемый код **JavaScript** (динамическое изменение содержания веб-страницы).

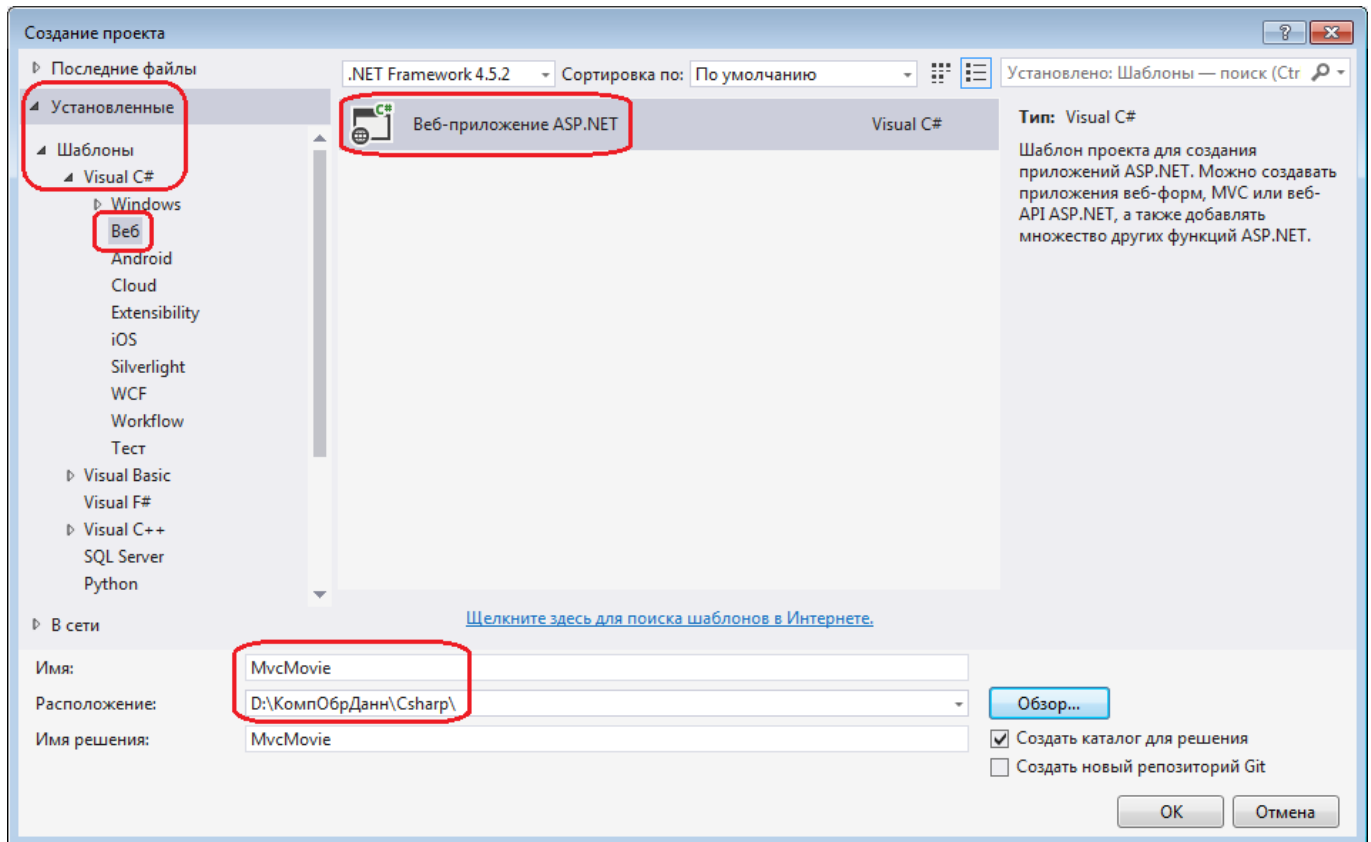
Далее необходимо создать проект в **Visual Studio**.

Visual Studio запускает локальный веб-сервер, динамически генерирует содержимое веб-страниц для веб-клиентов (браузеров) в процессе работы, взаимодействует с базой данных для создания, записи, чтения и редактирования данных при соответствующих запросах.

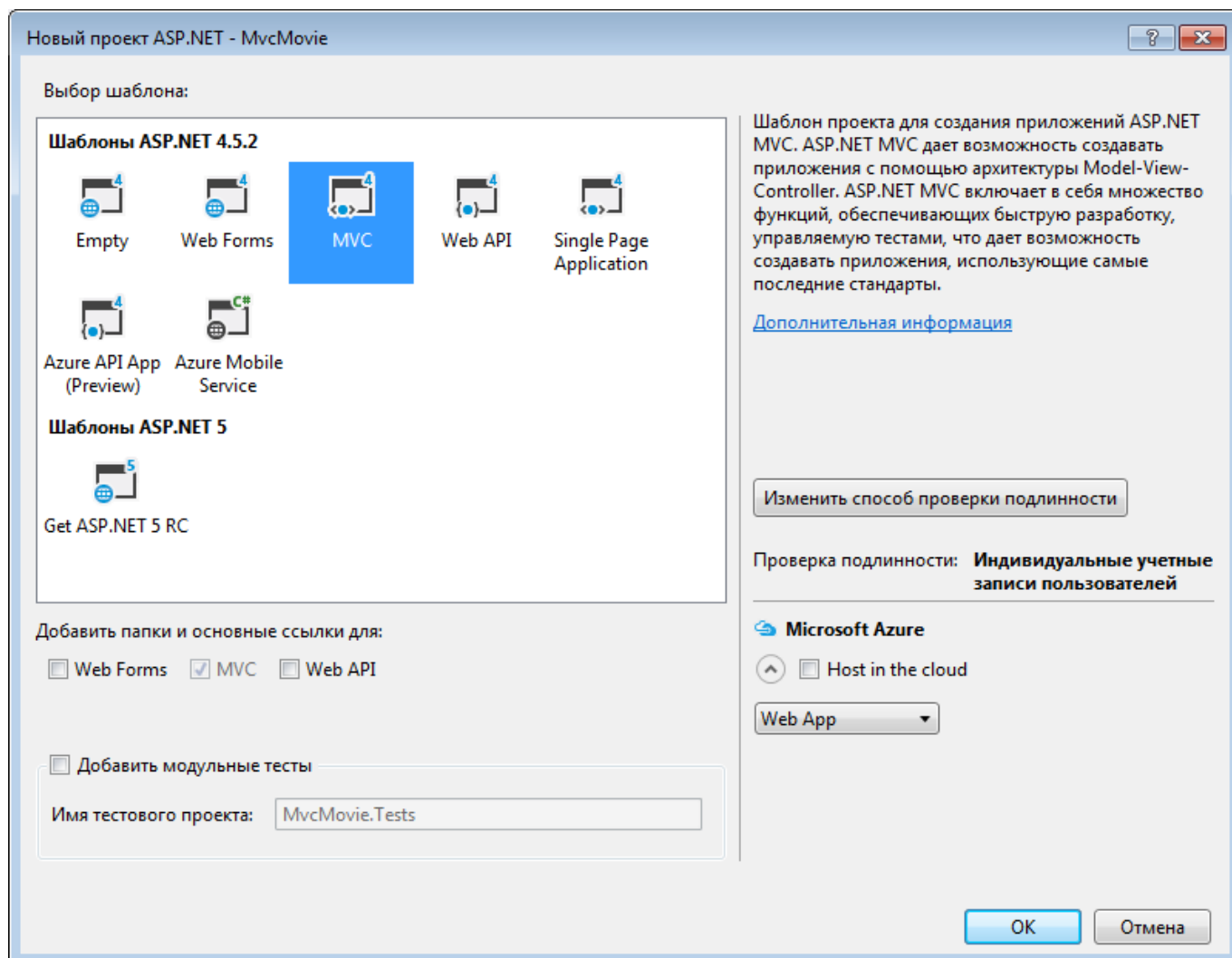
Содержимое сгенерированных веб-страниц также состоит из разметки **HTML**, стилей **CSS** и кода **JavaScript**.

Алгоритм генерации веб-страниц пишется на языке **C#**.

16. Создать новый проект: имя **MvcMovie**, расположение проекта **d:\КомпОбрДан\Csharp\Lab7_Web**.
Шаблон Веб – Веб-приложение ASP.Net – ОК.

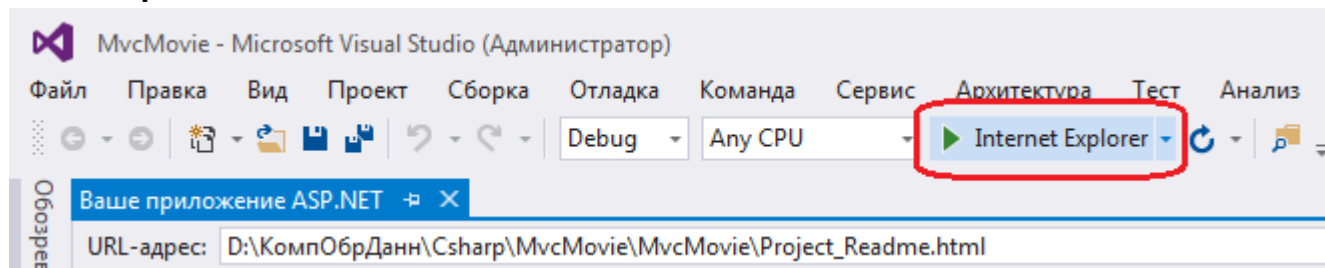


17. В окне Новый проект ASP.NET – выбрать MVC – ОК.

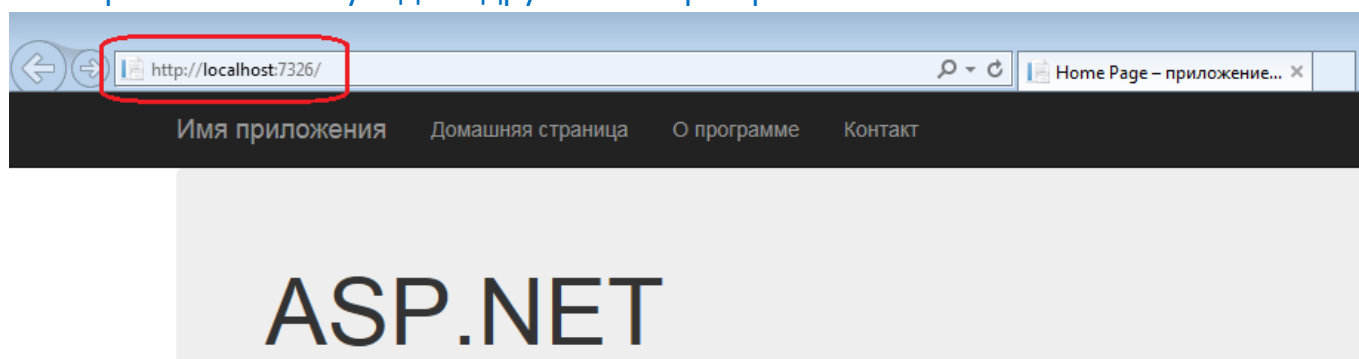


18. Запустить отладку мышью, или клавишей **F5**.

По кнопке выпадающего меню возле кнопки запуска отладки можно выбрать браузер, в котором будет запускаться web-проект, в данном случае это **Internet Explorer**.



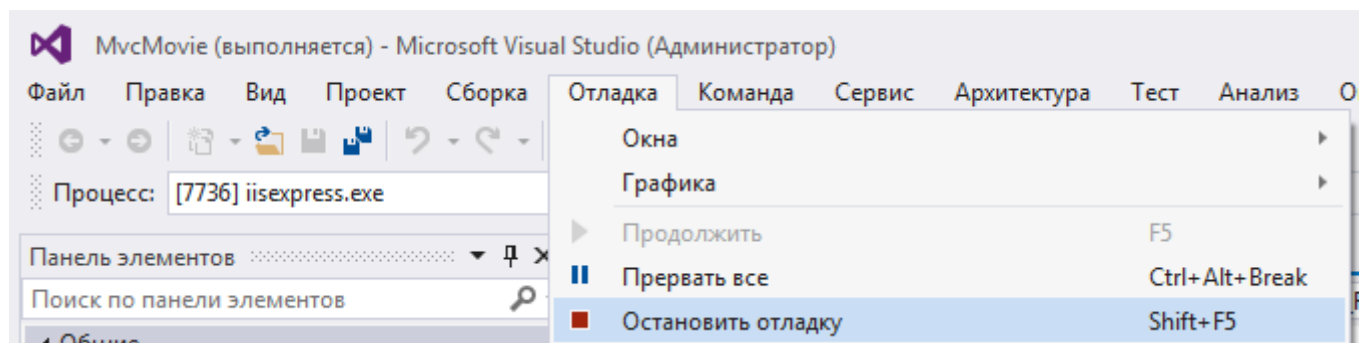
В адресной строке браузера указано **localhost:port** (где **port** - четырёхзначное число). Адрес **localhost** – это ваш собственный локальный компьютер, на котором запускается только что созданное приложение. Когда **Visual Studio** запускает веб-проект, для веб-сервера используется случайный порт. На приведенном ниже изображении номер порта составляет **7326**. При запуске приложения вы увидите другой номер порта.



Этот шаблон по умолчанию имеет страницы **Имя приложения**, **Домашняя страница**, **О программе**, **Контакт**, **Регистрация** и **Выполнить вход**.

Далее необходимо изменить работу этого приложения.

19. Закрыть web-страницу браузера или нажать кнопку **Остановить отладку** или сочетание клавиш **Shift + F5** в **Visual Studio**.



Добавление контроллера

Model-View-Controller (MVC, «Модель-Представление-Контроллер», «Модель-Вид-Контроллер») — схема разделения данных приложения, пользовательского интерфейса и управляющей логики на три отдельных компонента таким образом, что модификация каждого компонента может осуществляться независимо.

Models: классы, которые представляют данные приложения и которые используют логику проверки для обеспечения соблюдения правил для этих данных. **Модель** предоставляет данные и реагирует на команды контроллера, изменяя свое состояние.

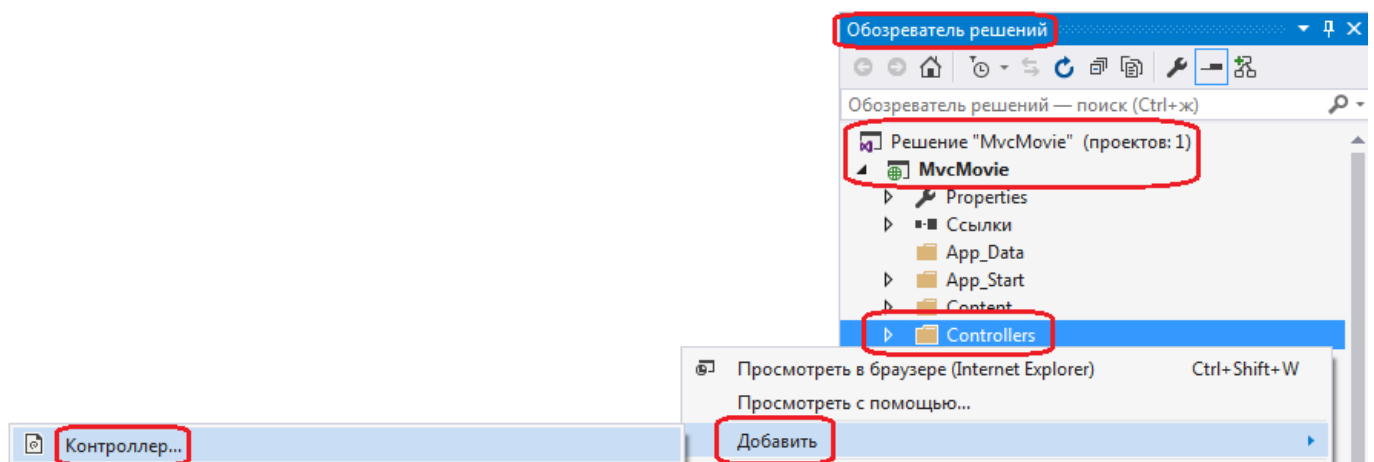
Views: файлы шаблонов, которые ваше приложение использует для динамической генерации HTML-страниц. **Представление** отвечает за отображение данных модели пользователю, реагируя на изменения модели.

Controllers: классы, которые обрабатывают входящие запросы браузера, извлекают данные модели и затем определяют шаблоны представлений, которые возвращают ответ браузеру. **Контроллер** интерпретирует действия пользователя, оповещая модель о необходимости изменений.

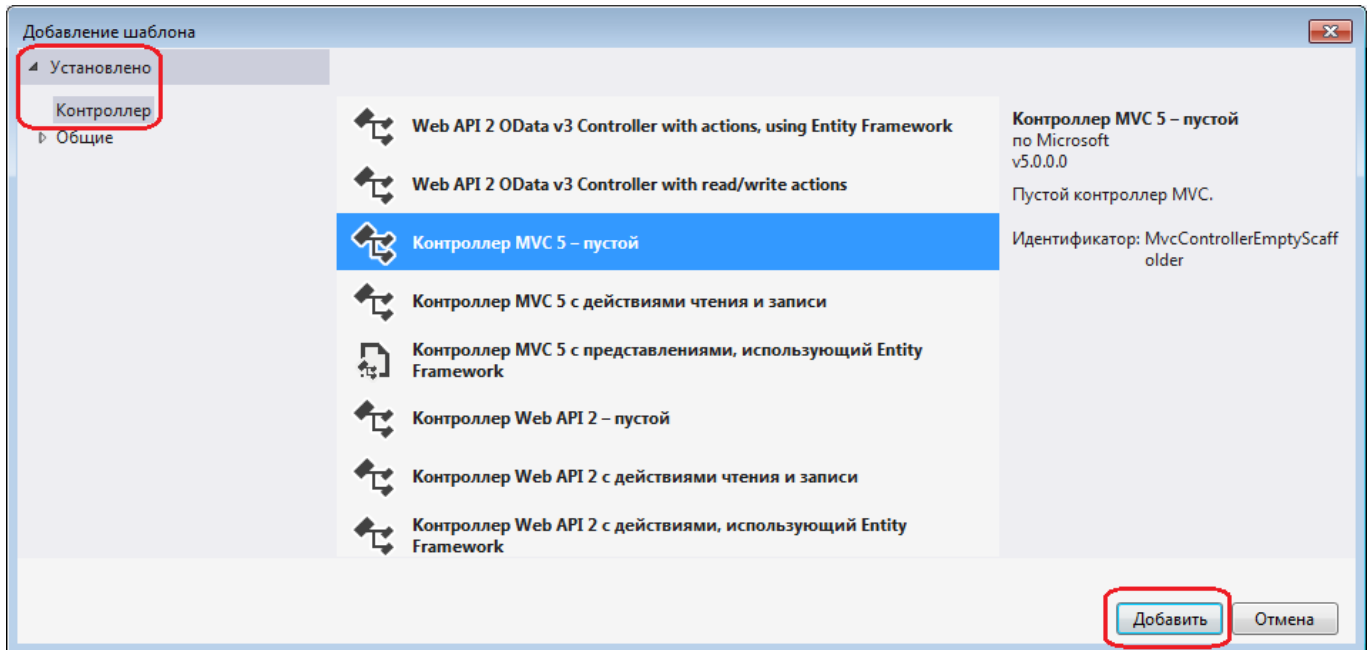
На предыдущем шаге был выбран шаблон MVC по умолчанию, что по умолчанию создаёт контроллеры HomeController, AccountController и ManageController в папке Controllers в Обозревателе решений.

Для начала необходимо создать класс контроллера.

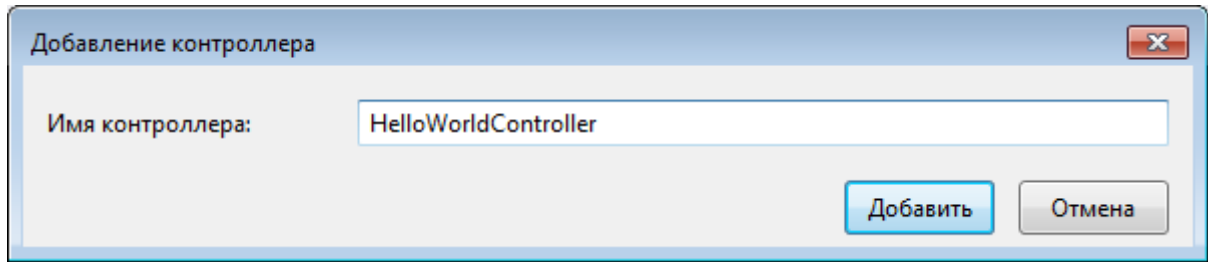
20. В **Обозревателе решений** щелкнуть правой кнопкой мыши папку **Контроллеры** – **Добавить** – **Контроллер**.



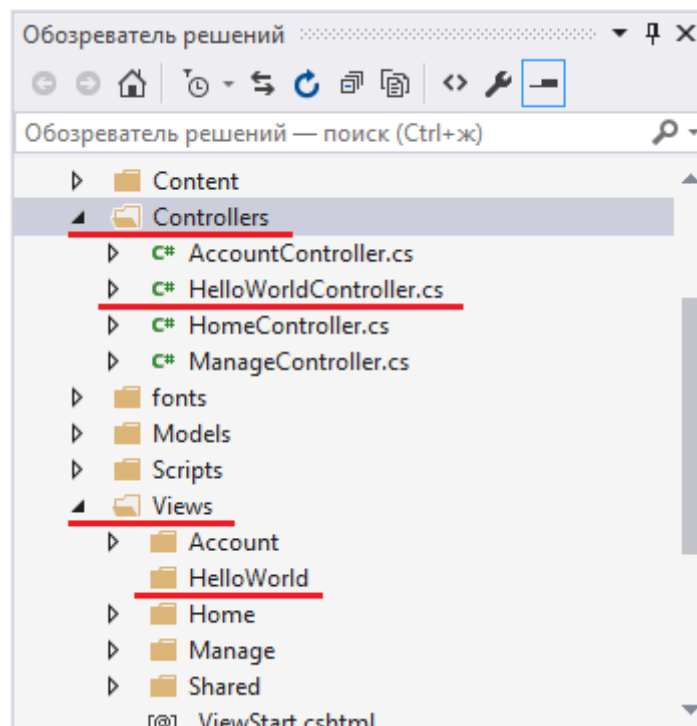
21. В окне **Добавление шаблона** выбрать **Контроллер MVC 5 – пустой** – **Добавить**.



22. В окне **Добавление контроллера** задать имя контроллера **HelloWorldController** – **Добавить**.



В **Обозревателе решений** создадутся новый файл с именем **HelloWorldController.cs** и новая папка **Views\HelloWorld**.



23. В классе **HelloWorldController.cs** заменить содержимое на следующий код:

```
using System.Web;
using System.Web.Mvc;

namespace MvcMovie.Controllers
{
    public class HelloWorldController : Controller
    {
        //
        // GET: /HelloWorld/

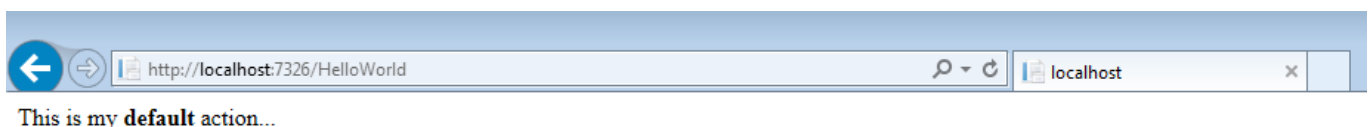
        public string Index()
        {
            return "This is my <b>default</b> action...";
        }

        //
        // GET: /HelloWorld/Welcome/

        public string Welcome()
        {
            return "This is the Welcome action method...";
        }
    }
}
```

В данном контроллере **HelloWorldController** первый метод называется **Index** – это метод, вызываемый по умолчанию при обращении к данному контроллеру.

24. Запустить приложение. В браузере добавить **/HelloWorld** к пути в строке адреса. Например, на рисунке ниже это **http://localhost:7326/HelloWorld**. Страница в браузере отобразит строку «This is my **default** action...» и будет выглядеть следующим образом:



ASP.NET MVC вызывает разные классы контроллеров (и различные методы действий внутри них) в зависимости от входящего URL. Логика маршрутизации URL по умолчанию, используемая ASP.NET MVC, использует такой формат, чтобы определить, какой код вызывать:

/[Controller]/[ActionName]/[Parameters]

Формат маршрутизации устанавливается в файле **App_Start/RouteConfig.cs** и уже прописан в данном файле:

```
public static void RegisterRoutes(RouteCollection routes)
{
    routes.IgnoreRoute("{resource}.axd/{*pathInfo}");

    routes.MapRoute(
        name: "Default",
        url: "{controller}/{action}/{id}",
        defaults: new { controller = "Home", action = "Index", id =
        UrlParameter.Optional }
    );
}
```

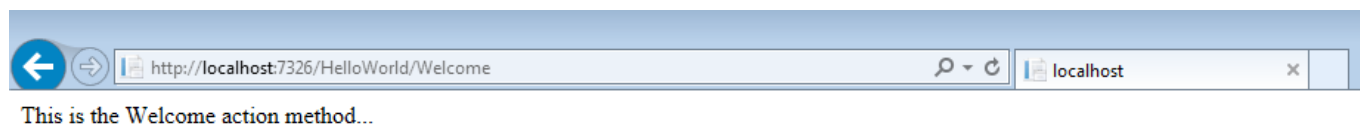
Когда вы запускаете приложение и не добавляете какие-либо сегменты URL-адресов, по умолчанию используется контроллер **Home** и метод действия **Index**, указанный в разделе по умолчанию приведенного выше кода.

Первая часть URL-адреса (**[Controller]**) определяет класс контроллера, который должен выполняться. Таким образом, **/HelloWorld** сопоставляется с классом **HelloWorldController**.

Вторая часть URL-адреса (**[ActionName]**) определяет метод действия для выполняемого класса. Таким образом, **/HelloWorld/Index** приведет к выполнению метода **Index** класса **HelloWorldController**. Так как метод **Index** используется по умолчанию, то его можно явно не указывать.

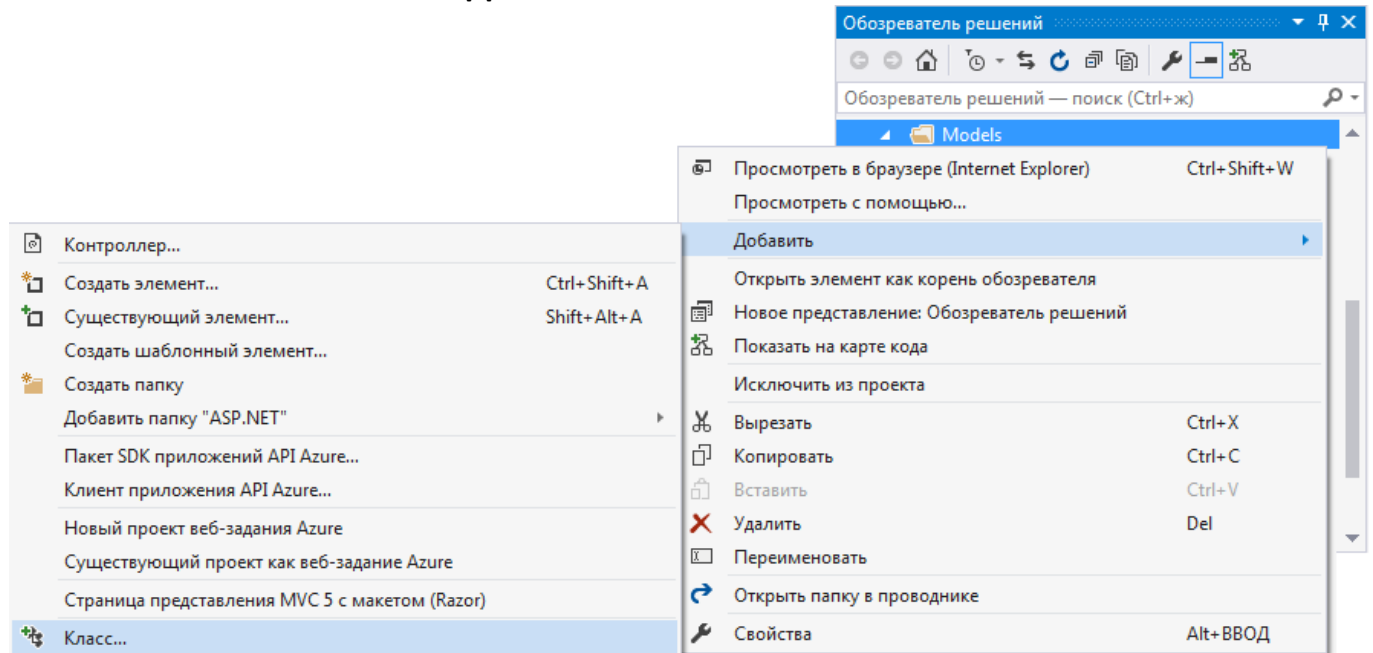
Третья часть URL-адреса (**[Parameters]**) предназначена для маршрутизации данных.

25. В браузере добавить **/Welcome** к пути в строке адреса (**http://localhost:xxxx/HelloWorld/Welcome**). Метод **Welcome** возвращает строку «This is the Welcome action method...».

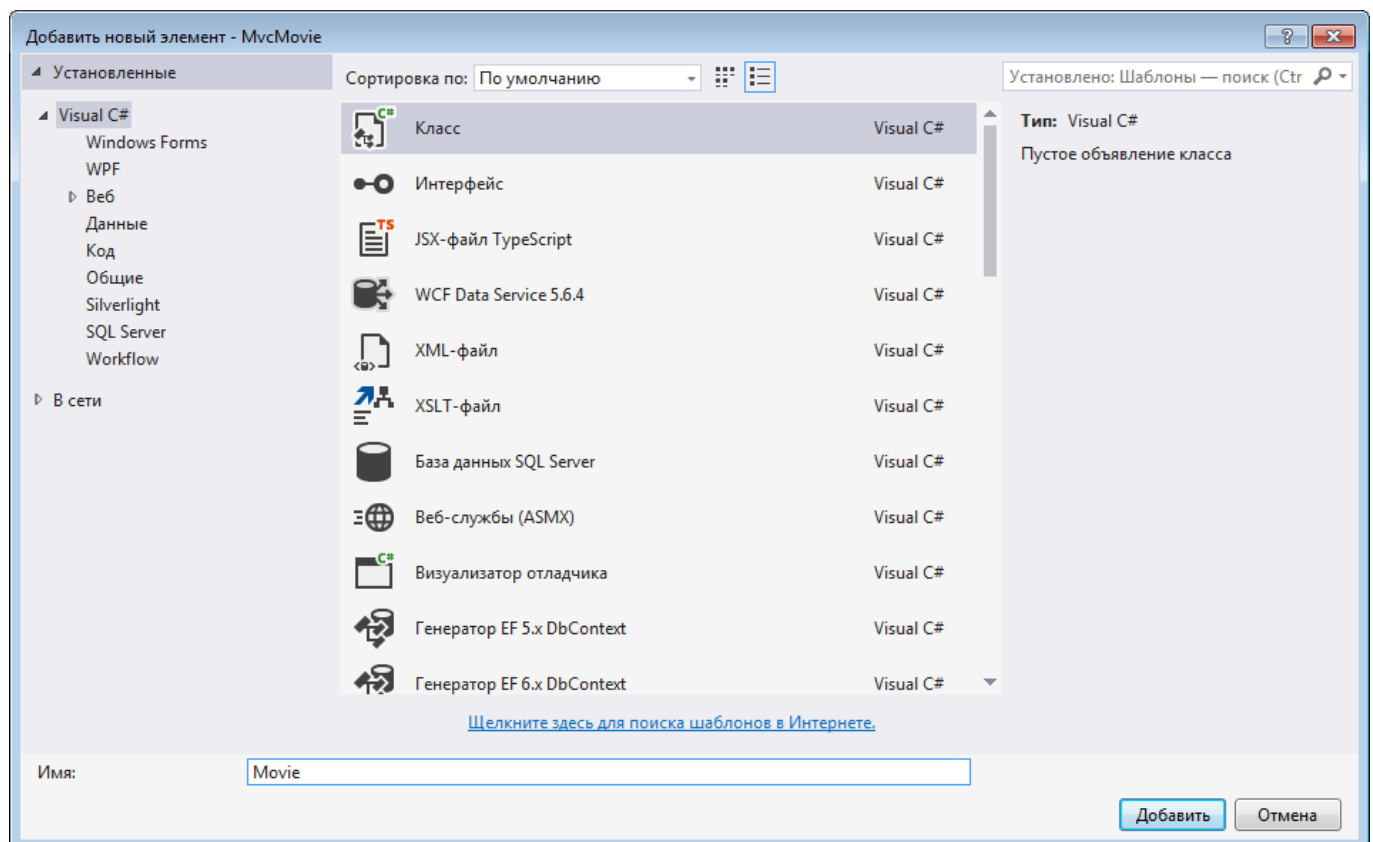


Добавление моделей

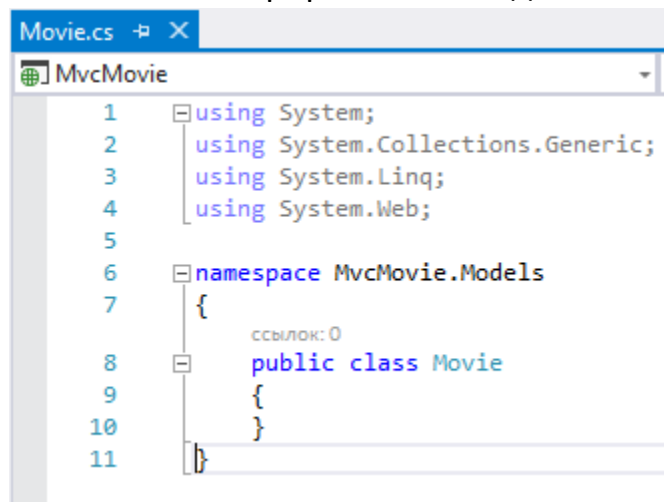
26. Далее необходимо добавить класс модели. В **Обозревателе решений** правый клик по папке **Models** – **Добавить** – **Класс**.



27. Задать имя класса – **Movie**.



28. Изначально автоматически сгенерированный код в классе такой:

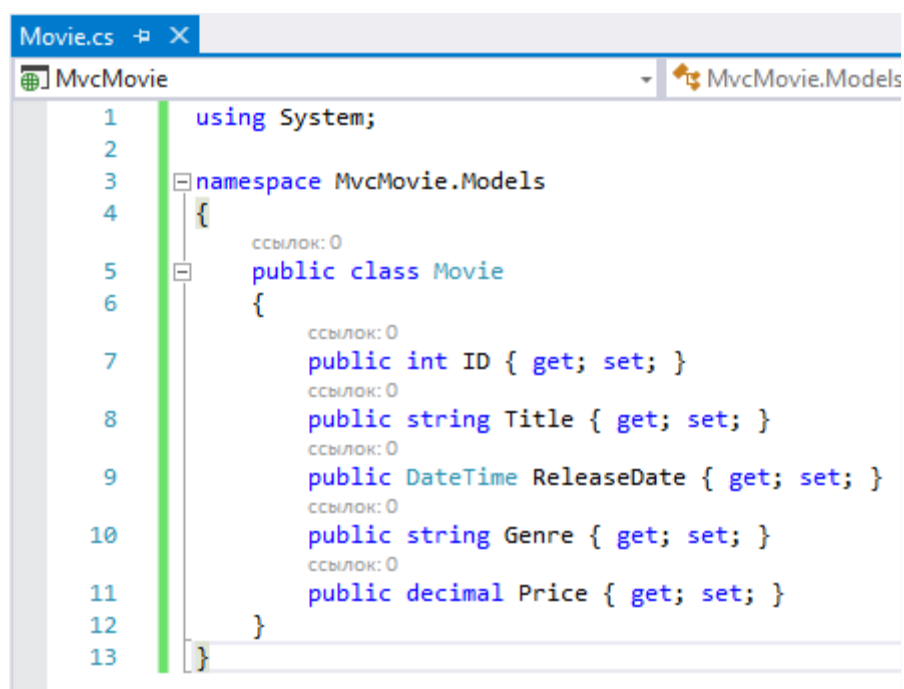


```
1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Web;
5
6 namespace MvcMovie.Models
7 {
8     public class Movie
9     {
10     }
11 }
```

Добавить в класс **Movie** следующие пять параметров (переменных):

```
using System;

namespace MvcMovie.Models
{
    // класс для описания фильма
    public class Movie
    {
        public int ID { get; set; } // порядковый номер фильма в списке
        public string Title { get; set; } // название фильма
        public DateTime ReleaseDate { get; set; } // дата релиза
        public string Genre { get; set; } // жанр фильма
        public decimal Price { get; set; } // цена фильма
    }
}
```



```
1 using System;
2
3 namespace MvcMovie.Models
4 {
5     public class Movie
6     {
7         public int ID { get; set; }
8         public string Title { get; set; }
9         public DateTime ReleaseDate { get; set; }
10        public string Genre { get; set; }
11        public decimal Price { get; set; }
12    }
13 }
```

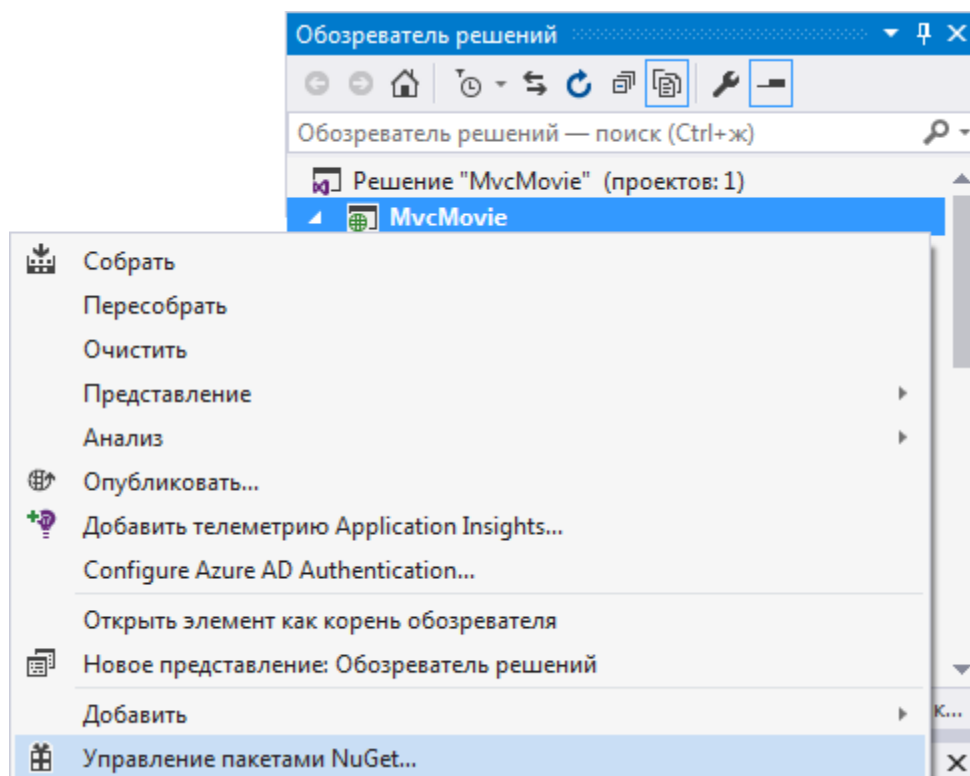
Класс **Movie** будет использоваться для представления фильмов в базе данных. Каждый экземпляр объекта **Movie** будет соответствовать строке в таблице базы данных, и каждый параметр класса **Movie** будет соответствовать столбцу в таблице.

	Movie.Параметр1	Movie.Параметр2	Movie.Параметр3
Объект1 типа Movie			
Объект2 типа Movie			
Объект3 типа Movie			

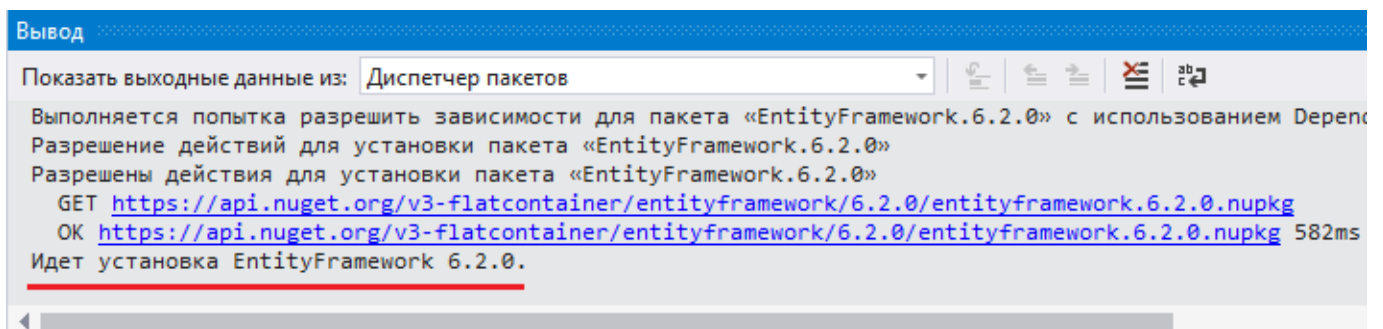
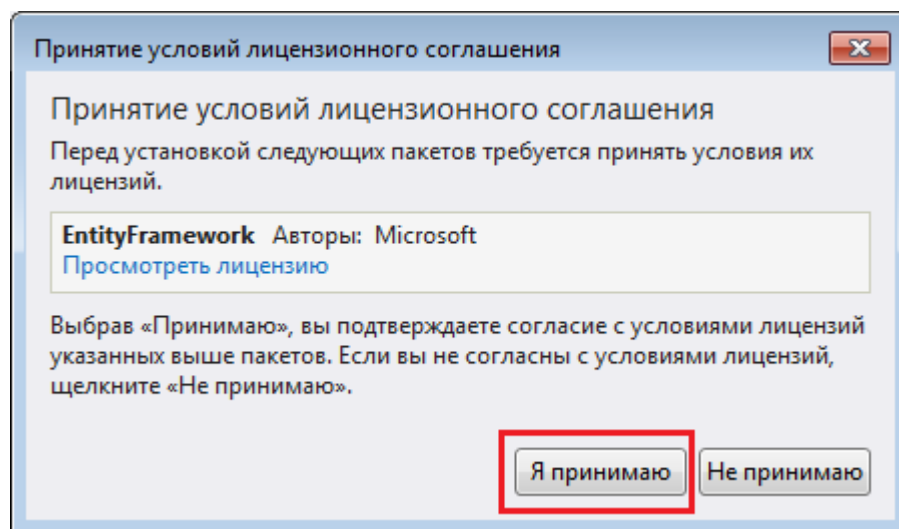
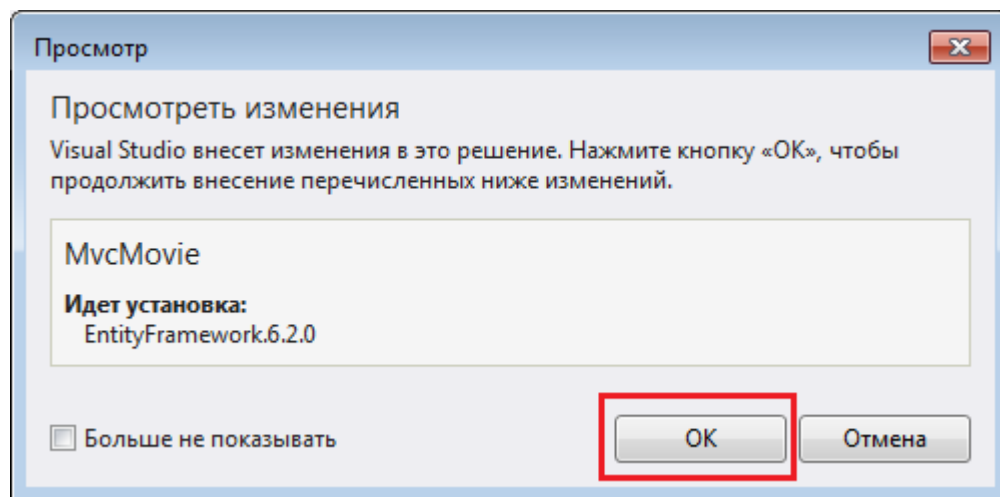
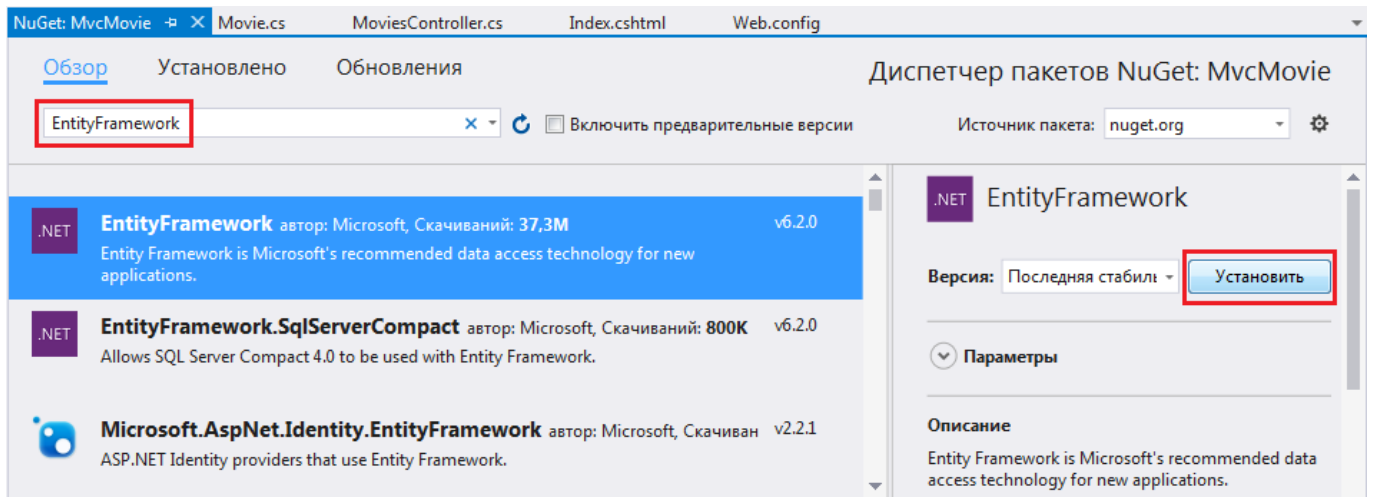
Чтобы использовать **System.Data.Entity** и соответствующий класс, необходимо установить пакет **Entity Framework NuGet**.

Если пакет **Entity Framework** не установлен, то необходимо сделать следующее:

В **Обозревателе решений** кликнуть правой кнопкой мыши на проекте **MvcMovie** – **Управление пакетами NuGet**.

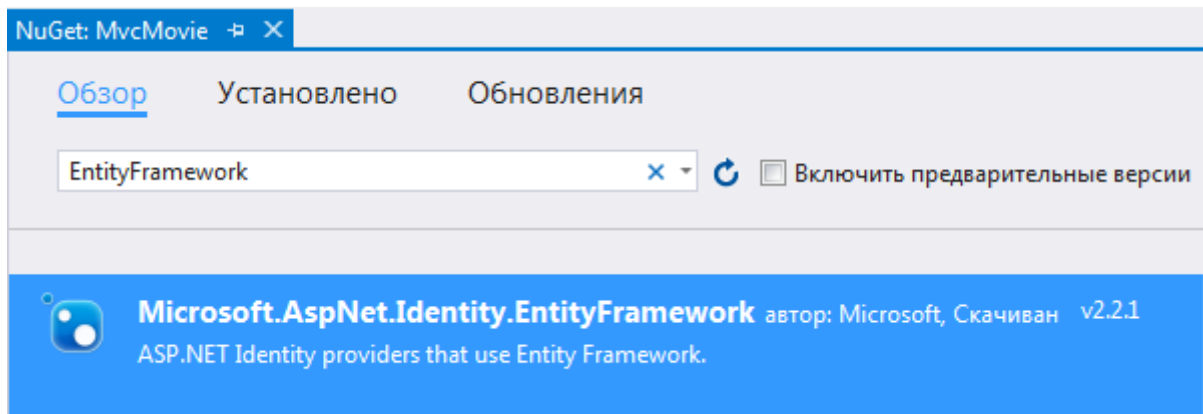


В открывшемся диспетчере пакетов в строке поиска набрать **EntityFramework** – нажать **Enter** – в списке результатов поиска выбрать пакет **EntityFramework** – нажать кнопку **Установить** – **ОК** – принять условия лицензионного соглашения – дождаться окончания установки пакета. Возможно, аналогичным образом потребуется установить пакет **Microsoft.AspNet.Identity.EntityFramework**.



```
Вывод
Показать выходные данные из: Диспетчер пакетов
Выполнение файла сценария «D:\КомпОбрДанн\Csharp\MvcMovie
Выполнение файла сценария «D:\КомпОбрДанн\Csharp\MvcMovie

Type 'get-help EntityFramework' to see all available Ent
"EntityFramework 6.2.0" успешно установлено в MvcMovie
===== Готово =====
```

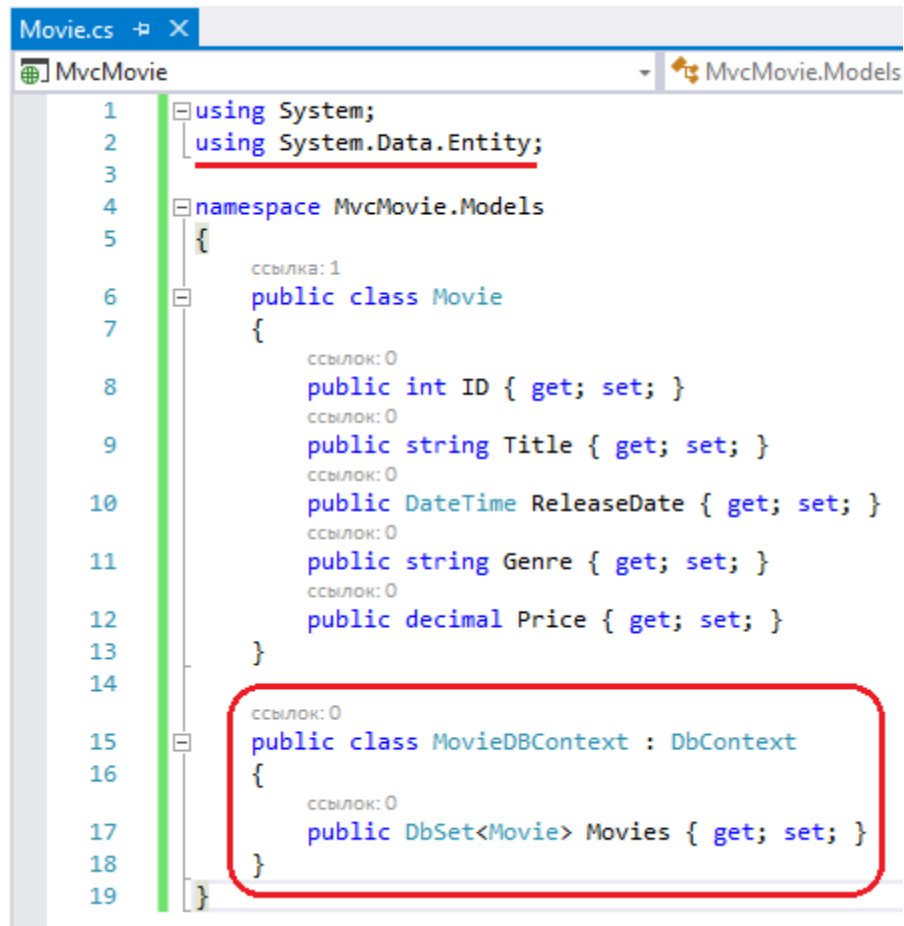


29. В том же файле **Movie.cs** ниже класса **Movie** добавить класс **MovieDbContext**, а в начале файла в раздел **using** добавить класс **System.Data.Entity**:

```
using System;
using System.Data.Entity;

namespace MvcMovie.Models
{
    public class Movie
    {
        public int ID { get; set; }
        public string Title { get; set; }
        public DateTime ReleaseDate { get; set; }
        public string Genre { get; set; }
        public decimal Price { get; set; }
    }

    public class MovieDbContext : DbContext
    {
        // параметр (переменная) для списка фильмов
        public DbSet<Movie> Movies { get; set; }
    }
}
```



```
1  using System;
2  using System.Data.Entity;
3
4  namespace MvcMovie.Models
5  {
6      public class Movie
7      {
8          public int ID { get; set; }
9          public string Title { get; set; }
10         public DateTime ReleaseDate { get; set; }
11         public string Genre { get; set; }
12         public decimal Price { get; set; }
13     }
14
15     public class MovieDbContext : DbContext
16     {
17         public DbSet<Movie> Movies { get; set; }
18     }
19 }
```

Класс **MovieDbContext** представляет контекст базы данных фильма **Entity Framework**, который обрабатывает выборку, хранение и обновление экземпляров класса **Movie** в базе данных, то есть производит операции со списком фильмов. **MovieDbContext** происходит из базового класса **DbContext** предоставляемого платформой **Entity Framework**.

Чтобы иметь возможность ссылаться на **DbContext** и **DbSet**, нужно добавить оператор **using** в верхней части файла: **using System.Data.Entity;**

Таким образом, класс **Movie** предназначен для одного (каждого) конкретного фильма, а класс **MovieDbContext** предназначен для списка всех фильмов (множества объектов типа **Movie**)

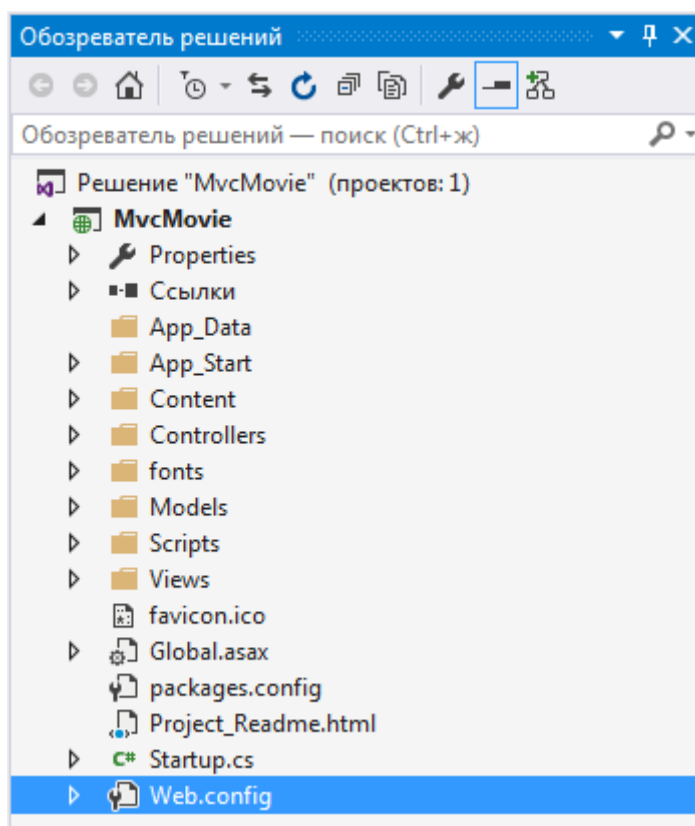
Создание строки подключения (к серверу) и работа с SQL Server LocalDB

Созданный класс **MovieDbContext** обрабатывает задачу подключения к базе данных и сопоставления объектов **Movie** с записями базы данных. По умолчанию **Entity Framework** использует базу данных **LocalDB**, к которой подключается класс **MovieDbContext**. Однако можно явно указать в файле **Web.config** в строке подключения, к какой базе данных необходимо подключаться.

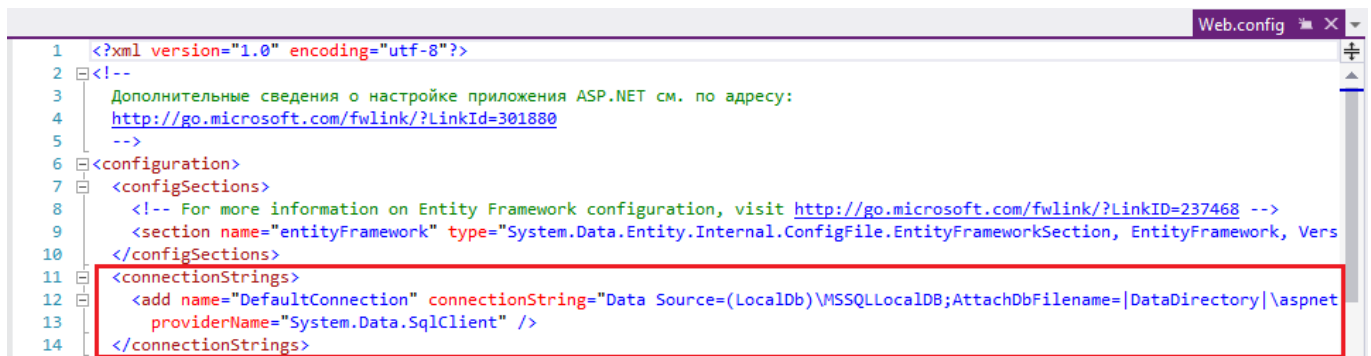
LocalDB - это легкая версия SQL Server Express Database Engine, которая запускается по требованию и запускается в пользовательском режиме. **LocalDB** запускается в специальном режиме выполнения SQL Server Express, что позволяет работать с базами данных в виде файлов **.mdf**. Как правило, файлы базы данных **LocalDB** хранятся в папке **App_Data** веб-проекта.

По умолчанию **Entity Framework** ищет строку подключения с таким же именем, как и класс контекста объекта (**MovieDbContext** для данного проекта).

30. Открыть файл **Web.config**, расположенный в корне приложения **MvcMovie**. (Не файл **Web.config** в папке **Views**)



31. Найти элемент <connectionStrings> в файле Web.config:



```
1 <?xml version="1.0" encoding="utf-8"?>
2 <!--
3     Дополнительные сведения о настройке приложения ASP.NET см. по адресу:
4     http://go.microsoft.com/fwlink/?LinkId=301880
5     -->
6 <configuration>
7   <configSections>
8     <!-- For more information on Entity Framework configuration, visit http://go.microsoft.com/fwlink/?LinkId=237468 -->
9     <section name="entityFramework" type="System.Data.Entity.Internal.ConfigFile.EntityFrameworkSection, EntityFramework, Vers
10    </configSections>
11    <connectionStrings>
12      <add name="DefaultConnection" connectionString="Data Source=(LocalDb)\MSSQLLocalDB;AttachDbFilename=|DataDirectory|\aspnet
13      providerName="System.Data.SqlClient" />
14    </connectionStrings>
```

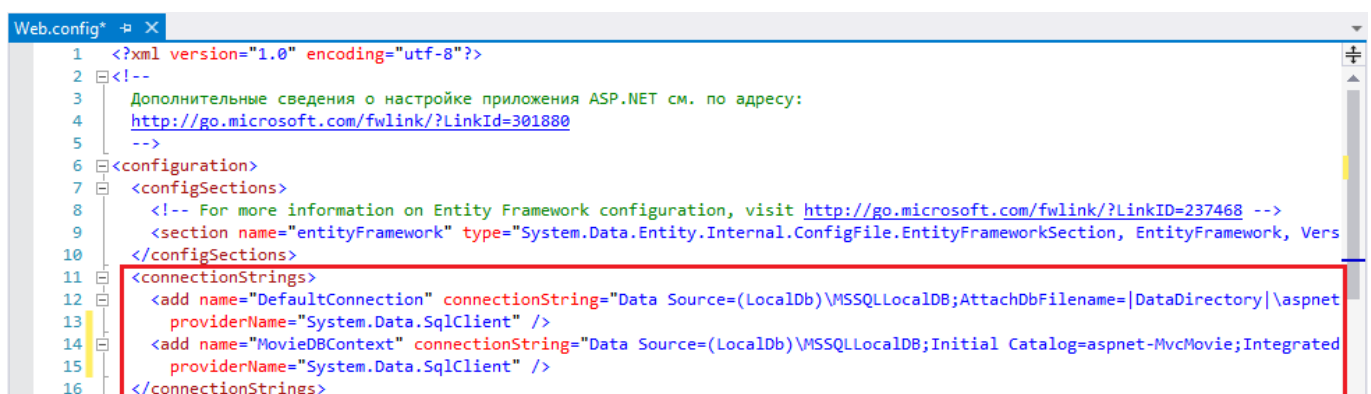
32. Добавить следующую строку в элемент <connectionStrings> в файле Web.config:

```
<add name="MovieDBContext" connectionString="Data
Source=(LocalDb)\MSSQLLocalDB;Initial Catalog=aspnet-MvcMovie;Integrated
Security=SSPI;AttachDbFilename=|DataDirectory|\Movies.mdf"
providerName="System.Data.SqlClient" />
```

P.S. возможно, что потребуется `Data Source=(LocalDb)\v11.0;`

Таким образом, элемент <connectionStrings> примет следующий вид:

```
<connectionStrings>
  <add name="DefaultConnection" connectionString="Data
Source=(LocalDb)\MSSQLLocalDB;Initial Catalog=aspnet-MvcMovie-fefdc1f0-
bd81-4ce9-b712-93a062e01031;Integrated
Security=SSPI;AttachDbFilename=|DataDirectory|\aspnet-MvcMovie-fefdc1f0-
bd81-4ce9-b712-93a062e01031.mdf" providerName="System.Data.SqlClient" />
  <add name="MovieDBContext" connectionString="Data
Source=(LocalDb)\MSSQLLocalDB;Initial Catalog=aspnet-MvcMovie;Integrated
Security=SSPI;AttachDbFilename=|DataDirectory|\Movies.mdf"
providerName="System.Data.SqlClient" />
</connectionStrings>
```



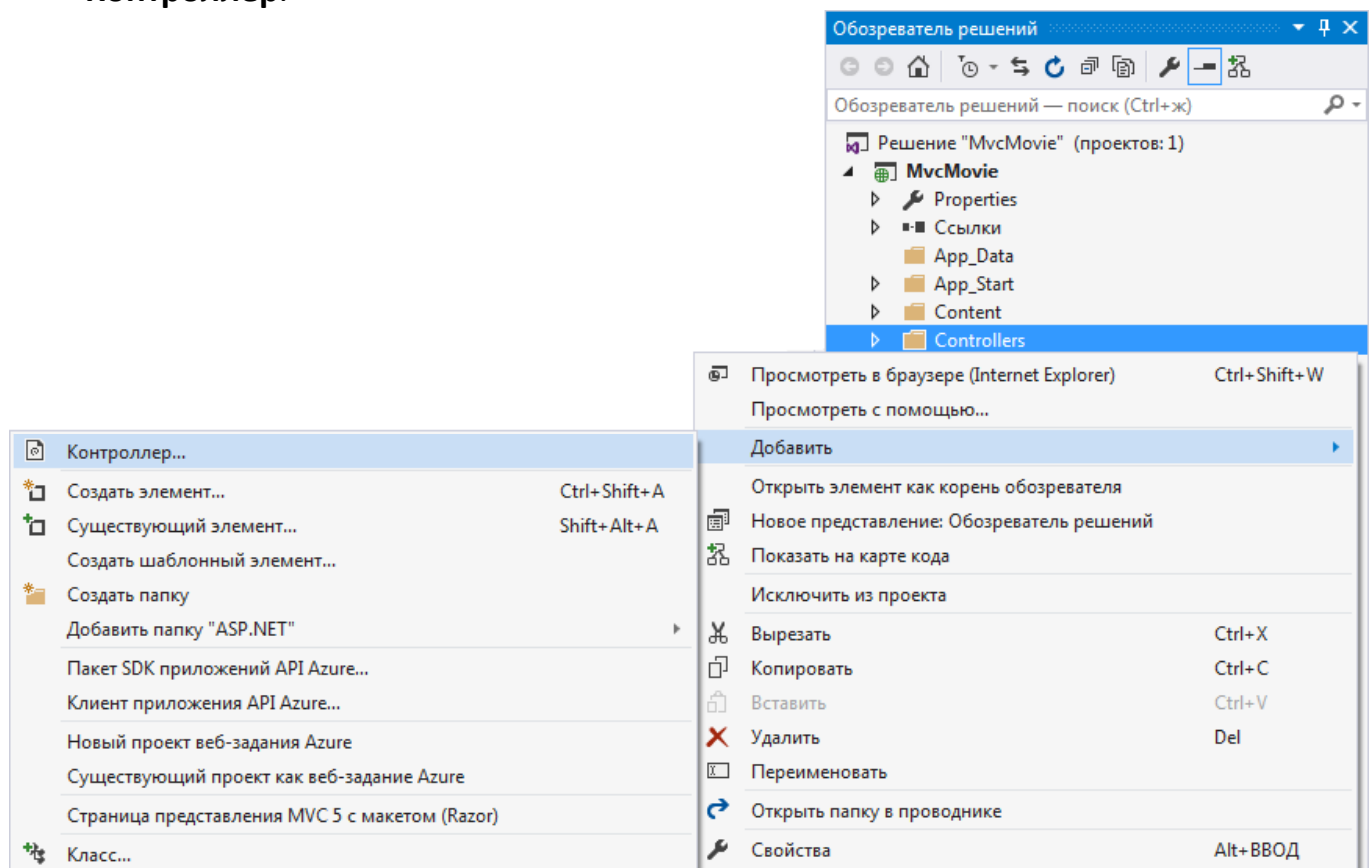
```
Web.config*  X
1 <?xml version="1.0" encoding="utf-8"?>
2 <!--
3     Дополнительные сведения о настройке приложения ASP.NET см. по адресу:
4     http://go.microsoft.com/fwlink/?LinkId=301880
5     -->
6 <configuration>
7   <configSections>
8     <!-- For more information on Entity Framework configuration, visit http://go.microsoft.com/fwlink/?LinkId=237468 -->
9     <section name="entityFramework" type="System.Data.Entity.Internal.ConfigFile.EntityFrameworkSection, EntityFramework, Vers
10    </configSections>
11    <connectionStrings>
12      <add name="DefaultConnection" connectionString="Data Source=(LocalDb)\MSSQLLocalDB;AttachDbFilename=|DataDirectory|\aspnet
13      providerName="System.Data.SqlClient" />
14      <add name="MovieDBContext" connectionString="Data Source=(LocalDb)\MSSQLLocalDB;Initial Catalog=aspnet-MvcMovie;Integrated
15      providerName="System.Data.SqlClient" />
16    </connectionStrings>
```

Две строки подключения очень похожи. Первая строка соединения называется **DefaultConnection** и используется для базы данных членства, чтобы контролировать, кто может получить доступ к приложению. Добавленная строка подключения указывает базу данных **LocalDB** с именем **Movie.mdf**, расположенную в папке **App_Data**. Имя строки подключения должно совпадать с именем класса **DbContext**.

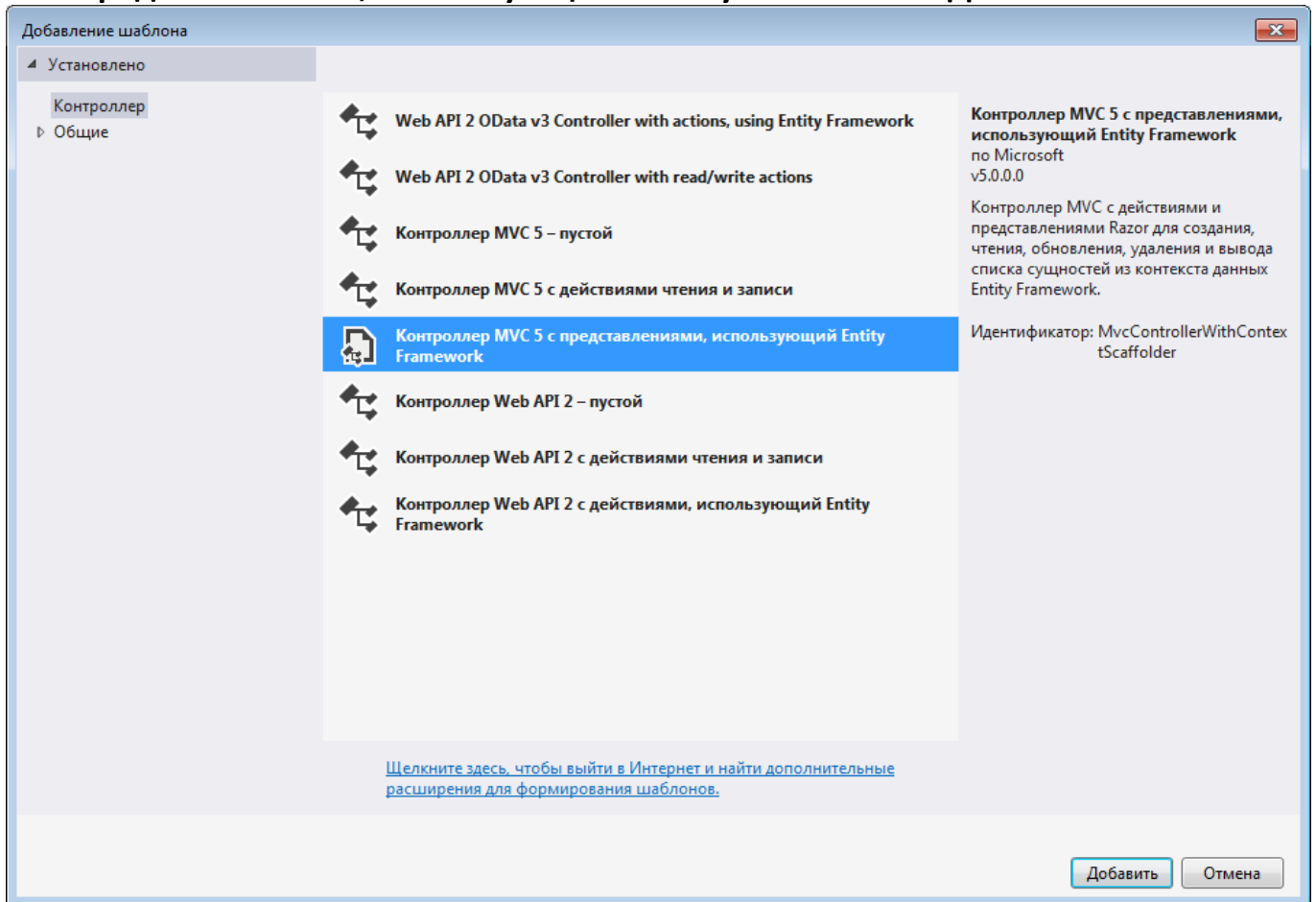
Доступ к данным модели с контроллера

Далее необходимо создать новый класс **MoviesController**, который извлекает из базы данные о фильме и отображает их в браузере, используя шаблон представления.

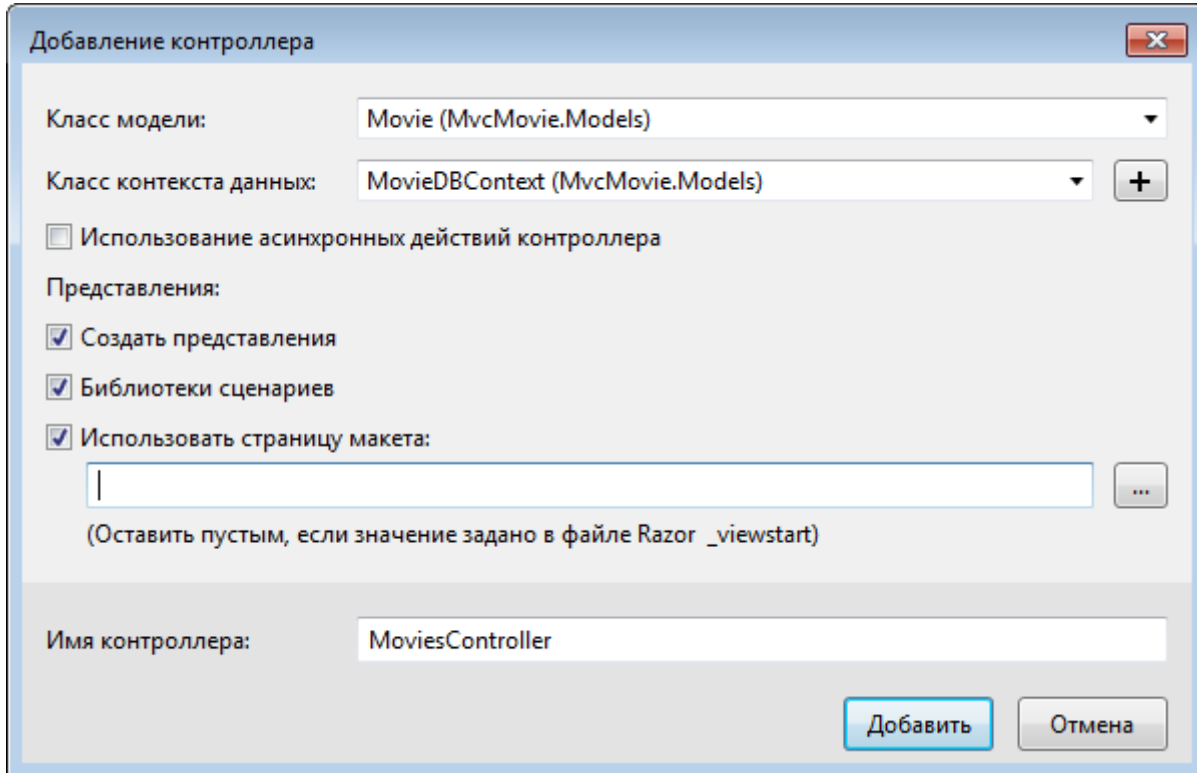
33. Обязательно запустить приложение на компиляцию прежде чем переходить к следующему пункту. Когда веб-страница загрузится в браузере, остановить выполнение программы. Если не скомпилировать приложение, то среда возвратит сообщение об ошибке с добавлением контроллера.
34. В **Обозревателе решений** правый клик на папке **Controllers** – **Добавить – Контроллер**.



35. В появившемся окне **Добавление шаблона** выбрать **Контроллер MVC5 с представлениями, использующими Entity Framework – Добавить.**



36. В появившемся окне **Добавление контроллера** задать следующие параметры:
Имя контроллера – задать **MoviesController**
Класс модели – выбрать **Movie (MvcMovie.Models)**
Класс контекста данных – выбрать **MovieDbContext (MvcMovie.Models)**
Остальные опции оставить по умолчанию.
Нажать **Добавить**.



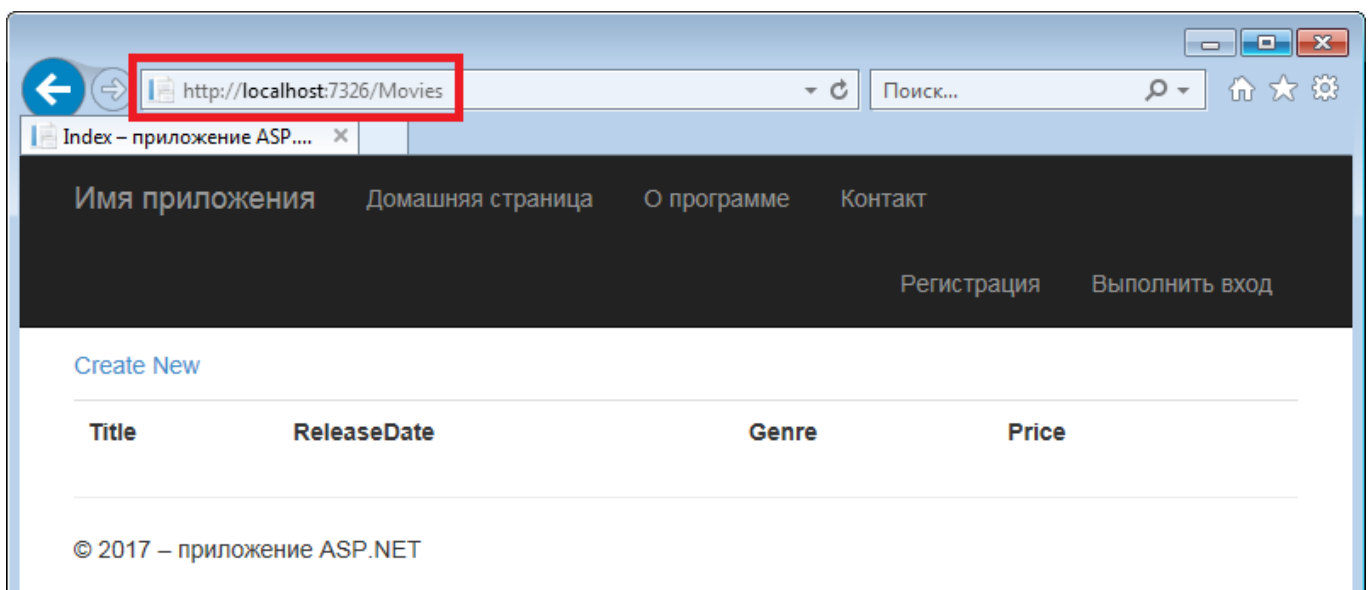
Если получено сообщение об ошибке, то, вероятно, ранее не создано/не скомпилировано приложение, прежде чем добавлять контроллер.

Visual Studio создает следующие файлы и папки:

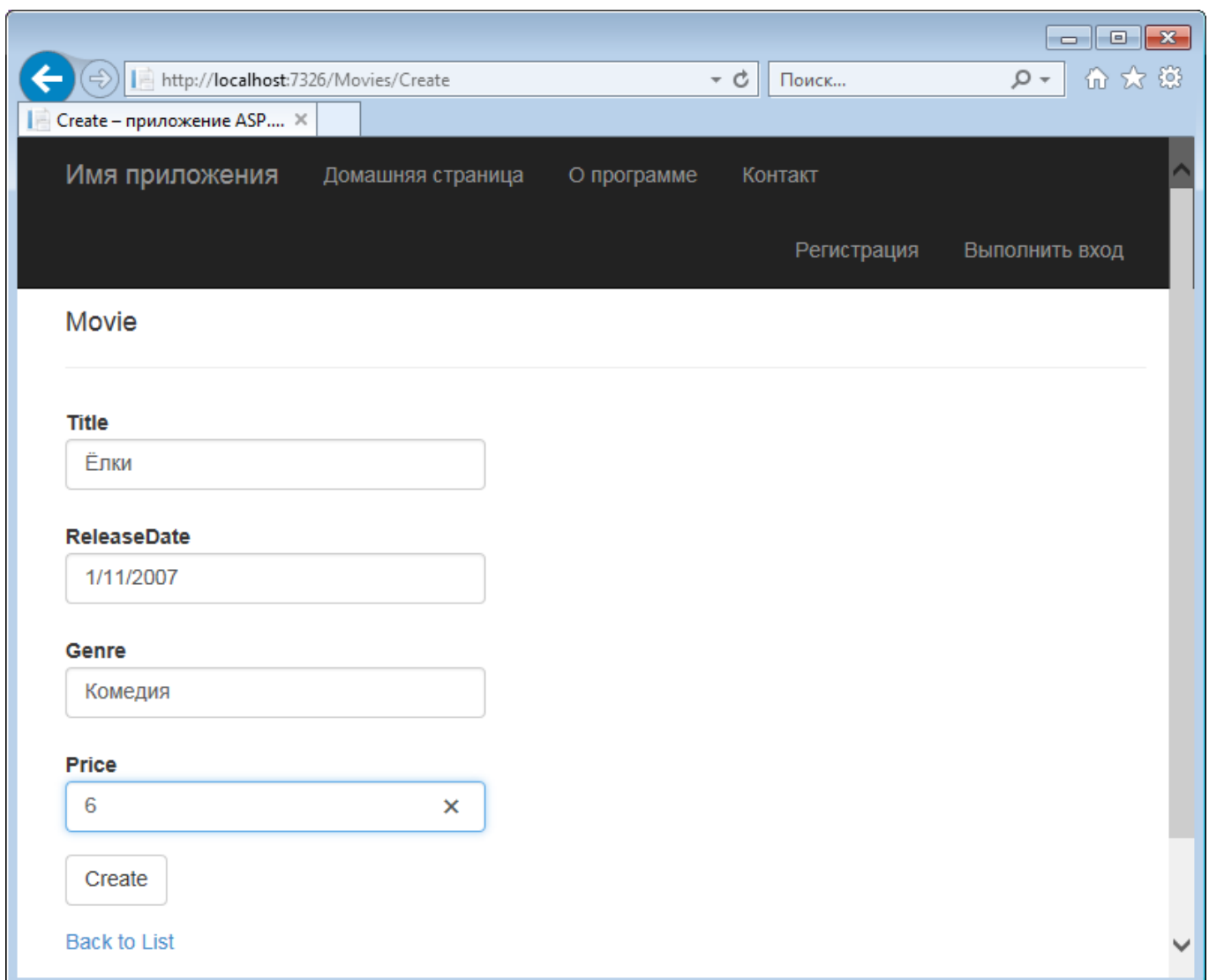
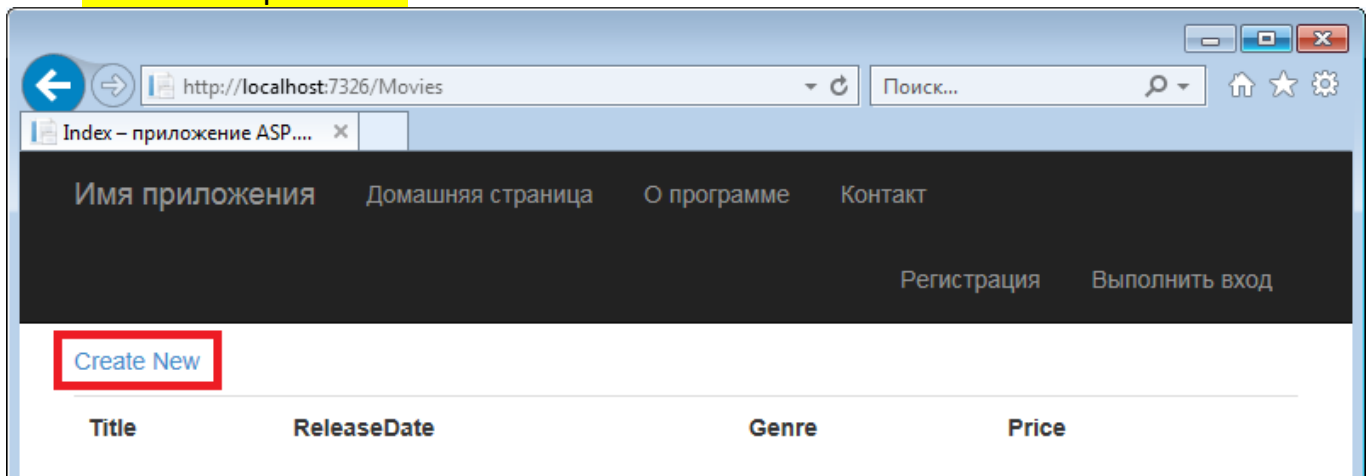
- Файл **MoviesController.cs** в папке **Controllers**.
- Папку **Views\Movies**.
- Файлы **Create.cshtml**, **Delete.cshtml**, **Details.cshtml**, **Edit.cshtml** и **Index.cshtml** в новой папке **Views\Movies**.

37. Снова запустить проект на компиляцию. Когда страница загрузится в браузере, в адресную строку после номера порта (после четырех цифр) добавить **/Movies** и нажать **Enter**. Таким образом адрес будет следующий: <http://localhost:xxxx/Movies> .

Поскольку приложение полагается на маршрутизацию по умолчанию (определенную в файле **App_Start\RouteConfig.cs**), запрос браузера <http://localhost:xxxx/Movies> перенаправляется на метод действия **Index** по умолчанию контроллера **Movies**. Другими словами, запрос браузера <http://localhost:xxxx/Movies> фактически совпадает с запросом браузера <http://localhost:xxxx/Movies/Index>. Результатом является пустой список фильмов, потому что в него ещё ничего не добавлено.



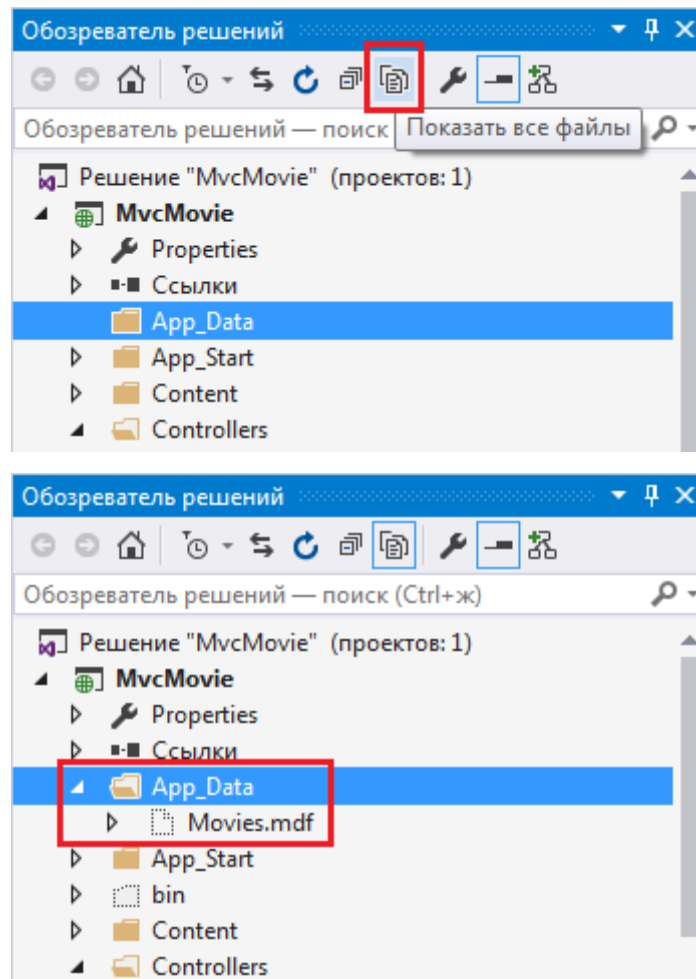
38. Далее нажимая на ссылку **Create New**, можно добавить новый фильм и нажать кнопку **Create**. Для поля Price вводить целочисленные значения (это связано с особенностью реализации неанглийских локалей). **Добавить в базу данных несколько фильмов.**



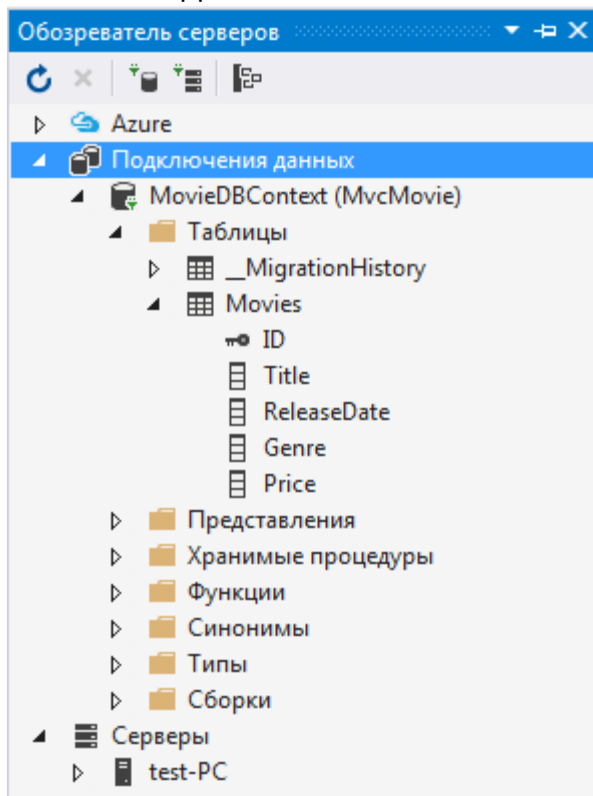
Нажатие кнопки **Create** вызывает отправку формы на сервер, где информация о фильме сохраняется в базе данных. Затем идёт перенаправление на **URL/Movies**, где можно увидеть только что созданный фильм в списке.

Работа с SQL Server LocalDB

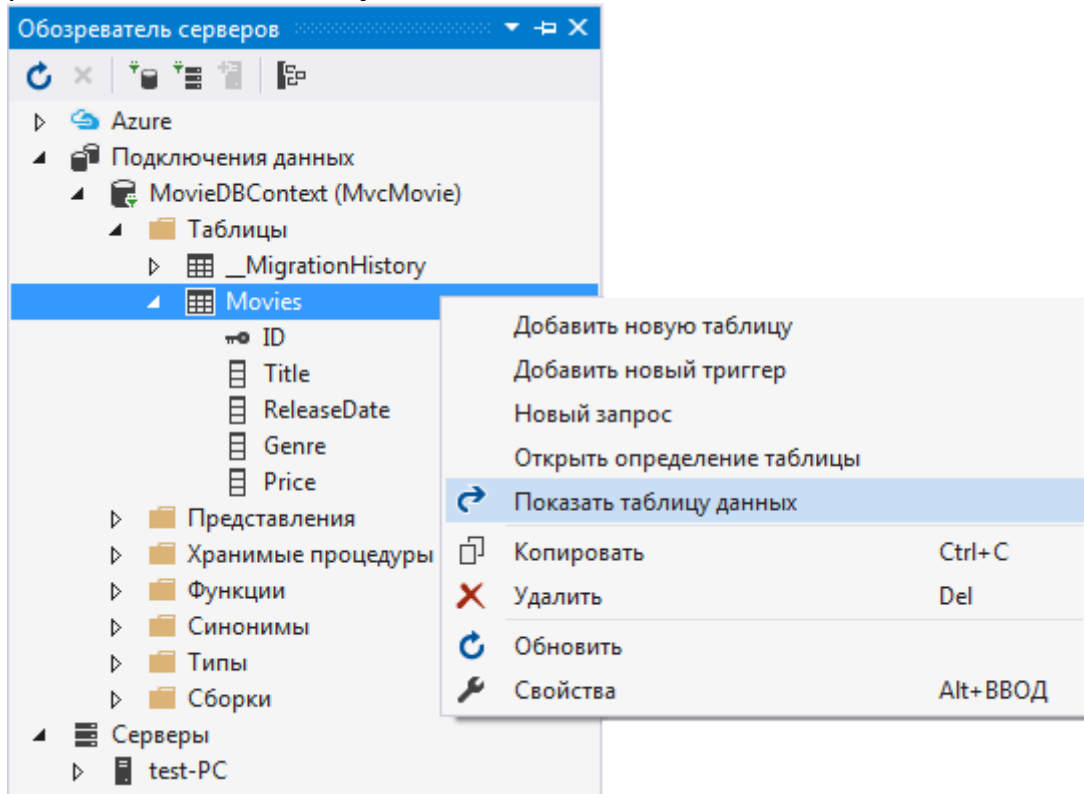
39. **Entity Framework Code First** обнаружил, что строка подключения к базе данных, которая была использована, указывает на базу данных **Movies**, которая еще не существует, поэтому **Code First** автоматически создал базу данных. Можно проверить, что она была создана посмотрев в папке **App_Data**. Если не видно файл **Movies.mdf**, нажать кнопку **Показать все файлы** на панели инструментов **Обозревателя решений**, нажать кнопку **Обновить**, а затем развернуть папку **App_Data**.



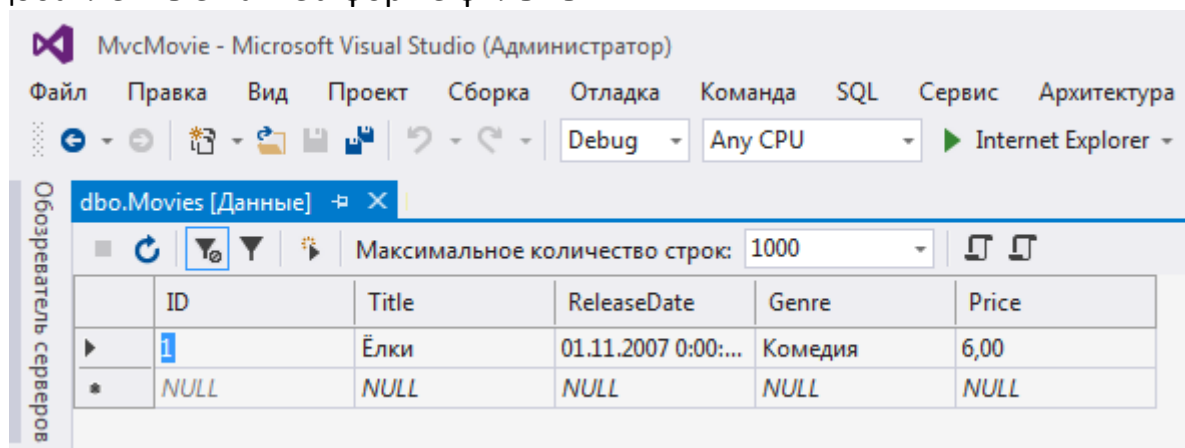
40. Убедиться, что записи добавлены в файл базы данных **Movies.mdf**, можно следующим образом: дважды щелкнуть по файлу **Movies.mdf** – откроется **Обозреватель серверов** – разверните папку **Таблицы** – в ней находится таблица **Movies**, внутри которой показаны столбцы (свойства) созданной таблицы: ID, Title, ReleaseDate, Genre, Price. Обратите внимание на значок ключа рядом со свойством **ID**. По умолчанию EF сделает свойство с именем **ID** основным ключом.



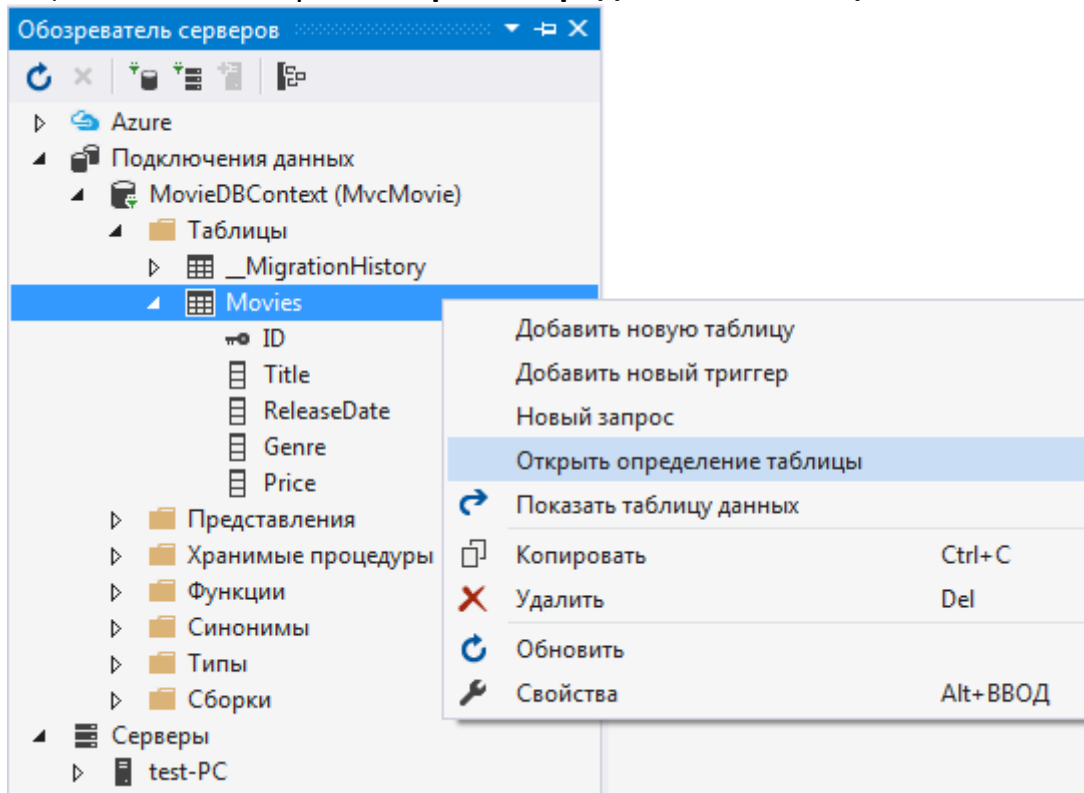
41. В окне **Обозревателя серверов** клик правой кнопкой мыши по таблице **Movies** – выбрать **Показать таблицу данных**.



42. Откроется вкладка с содержанием таблицы **Movies**, в которой можно увидеть добавленные на web-форме фильмы.

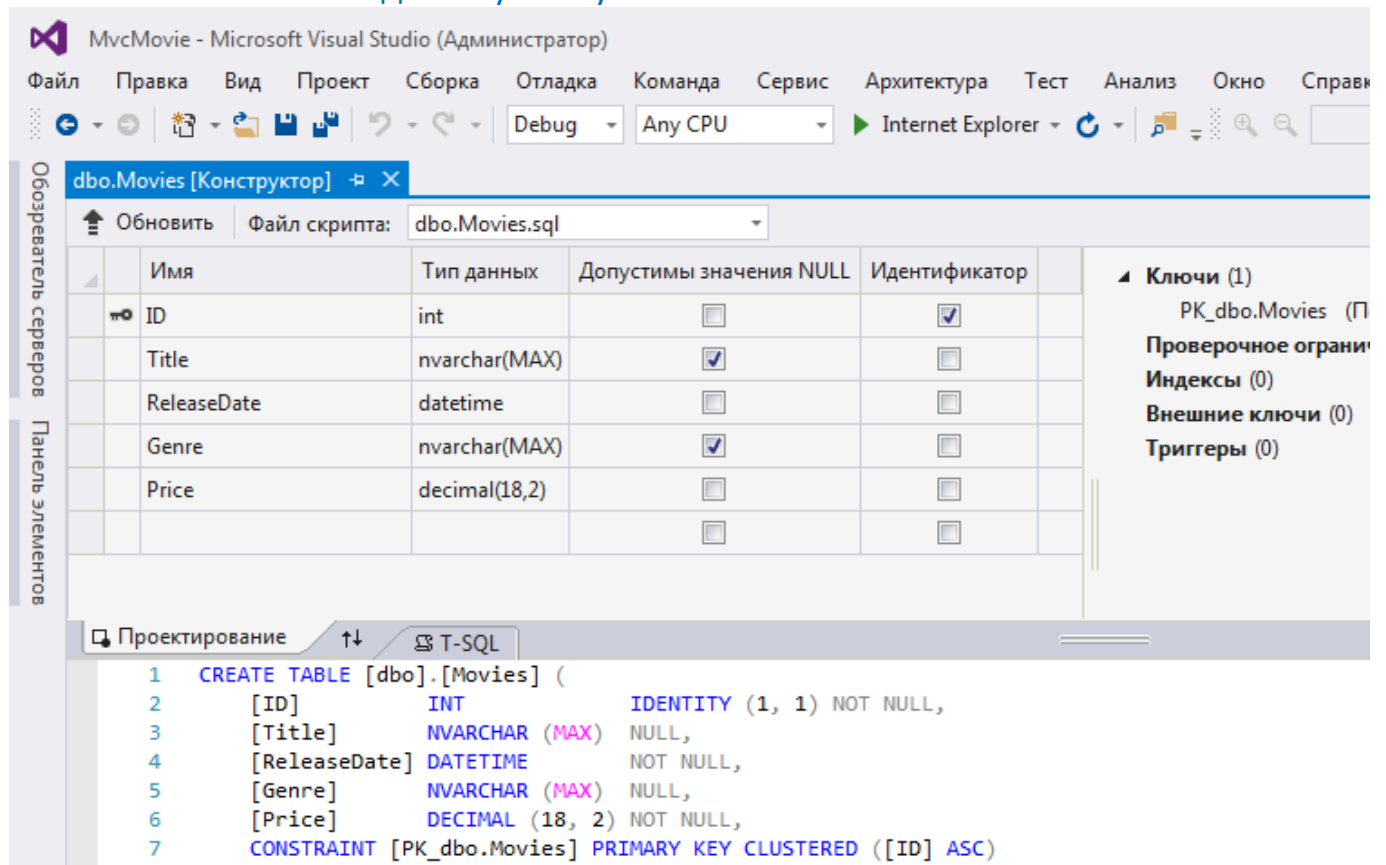


43. Чтобы увидеть структуру таблицы (имена и количество столбцов таблицы, допустимые для данных столбцов типы данных, идентификаторы и т.д.) которая создана **Entity Framework Code First** – кликнуть правой кнопкой мыши по таблице **Movies** – выбрать **Открыть определение таблицы**.



44. Откроется вкладка **Конструктора** со структурой таблицы **Movies**.

Обратите внимание, как схема (структура) таблицы **Movies** сопоставляется с классом **Movie**, который был создан ранее. **Entity Framework Code First** автоматически создаёт эту схему на основе класса **Movie**.



MvcMovie - Microsoft Visual Studio (Администратор)

Файл Правка Вид Проект Сборка Отладка Команда Сервис Архитектура Тест Анализ Окно Справка

Обозреватель серверов Панель элементов

dbo.Movies [Конструктор] X

Обновить Файл скрипта: dbo.Movies.sql

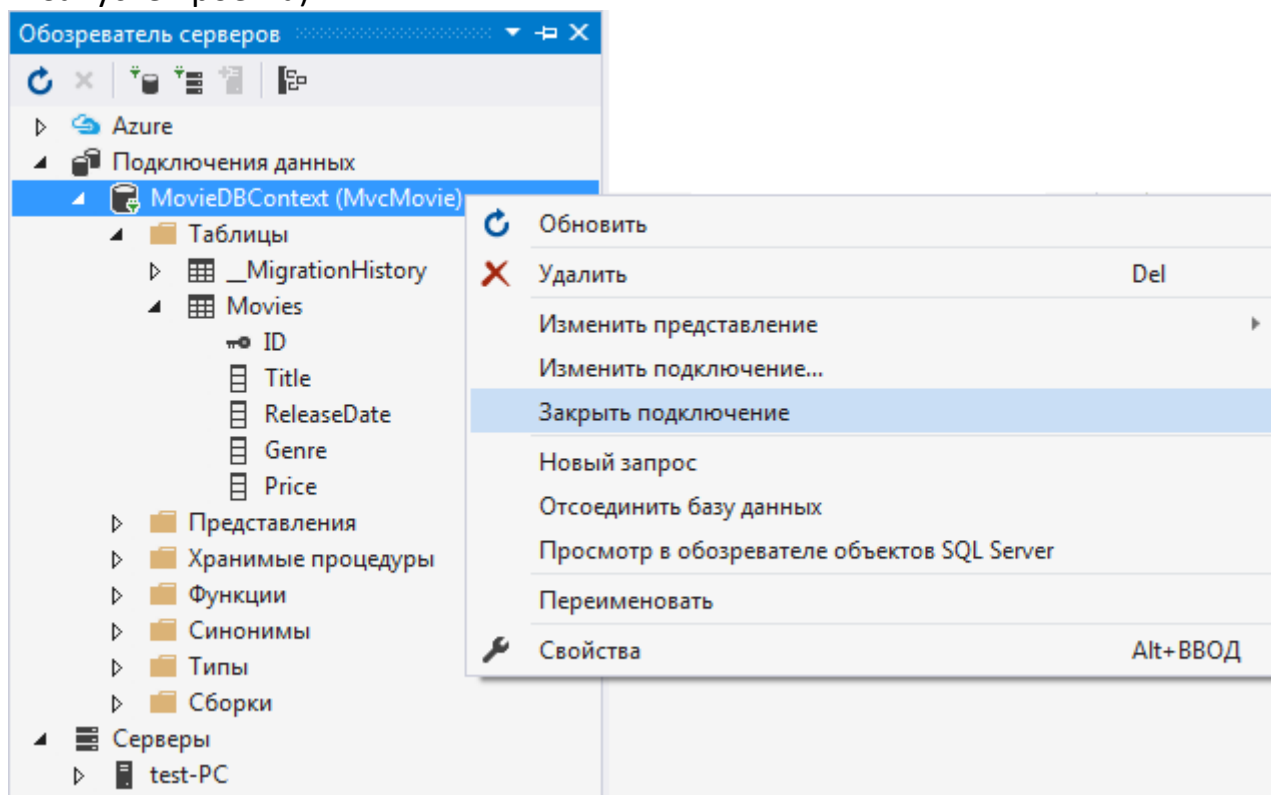
Имя	Тип данных	Допустимы значения NULL	Идентификатор
ID	int	<input type="checkbox"/>	<input checked="" type="checkbox"/>
Title	nvarchar(MAX)	<input checked="" type="checkbox"/>	<input type="checkbox"/>
ReleaseDate	datetime	<input type="checkbox"/>	<input type="checkbox"/>
Genre	nvarchar(MAX)	<input checked="" type="checkbox"/>	<input type="checkbox"/>
Price	decimal(18,2)	<input type="checkbox"/>	<input type="checkbox"/>
		<input type="checkbox"/>	<input type="checkbox"/>

Ключи (1)
PK_dbo.Movies (П)
Проверочное огранич
Индексы (0)
Внешние ключи (0)
Триггеры (0)

Проектирование T-SQL

```
1 CREATE TABLE [dbo].[Movies] (  
2     [ID] INT IDENTITY (1, 1) NOT NULL,  
3     [Title] NVARCHAR (MAX) NULL,  
4     [ReleaseDate] DATETIME NOT NULL,  
5     [Genre] NVARCHAR (MAX) NULL,  
6     [Price] DECIMAL (18, 2) NOT NULL,  
7     CONSTRAINT [PK_dbo.Movies] PRIMARY KEY CLUSTERED ([ID] ASC)
```

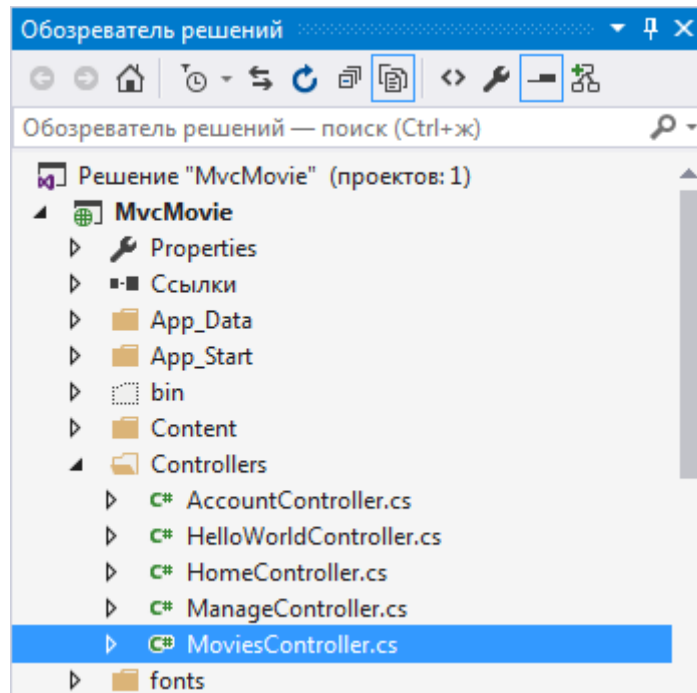
45. По завершении работы с базой данных необходимо закрыть подключение к ней. Для этого в **Обозревателе серверов** кликнуть правой кнопкой мыши по базе данных **MovieDBContext (MvcMovie)** – **Закрыть подключение**. (Если не закрыть подключение, то можно получить сообщение об ошибке при следующем запуске проекта).



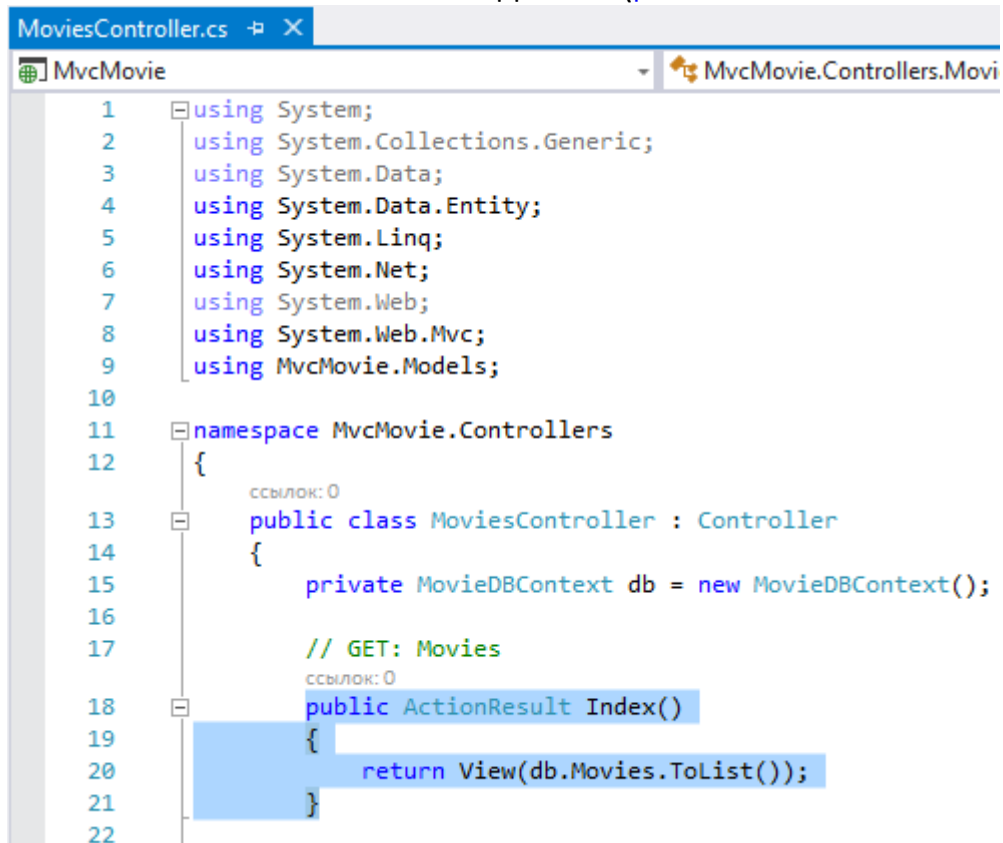
Добавление метода поиска и представления поиска

46. Необходимо добавить функцию поиска к методу **Index** в существующий класс **MoviesController**, которая позволит искать фильмы по жанру или имени. Однако, пользователи не будут менять URL-адрес каждый раз для поиска фильма. Необходимо на web-форму добавить компоненты для поиска.

В **Обозревателе решений** перейти: проект **MvcMovie** – папка **Controllers** – файл **MoviesController.cs**.



47. В классе **MoviesController** найти метод **Index** (`public ActionResult Index()`).

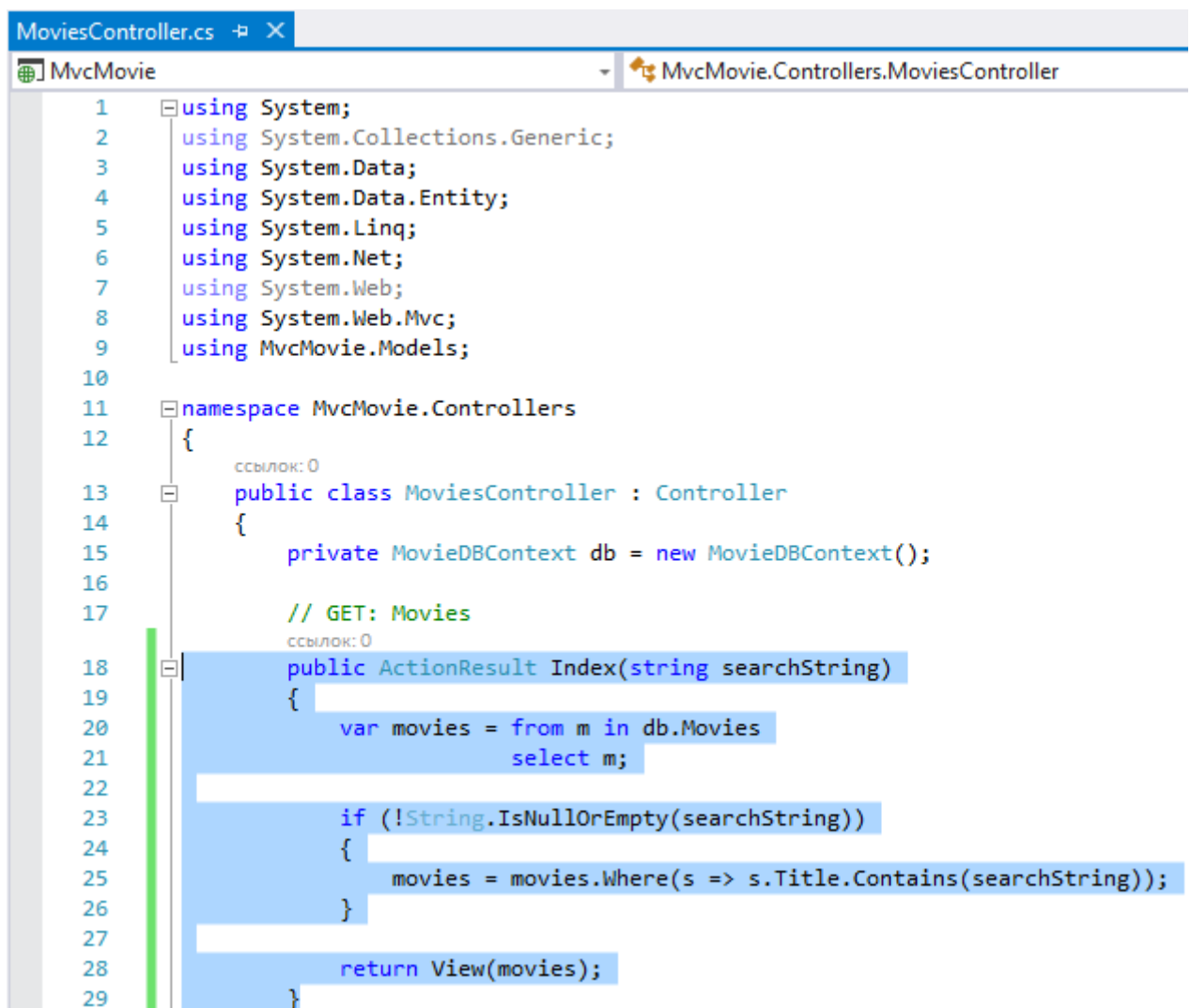


48. Изменить метод **Index** так, чтобы он принимал строковый параметр с именем **searchString** следующим образом:

```
// метод Index принимает в качестве параметра содержимое строки поиска
public ActionResult Index(string searchString)
{
    // Выбрать из источника данных db.Movies все фильмы
    var movies = from m in db.Movies
                  select m;

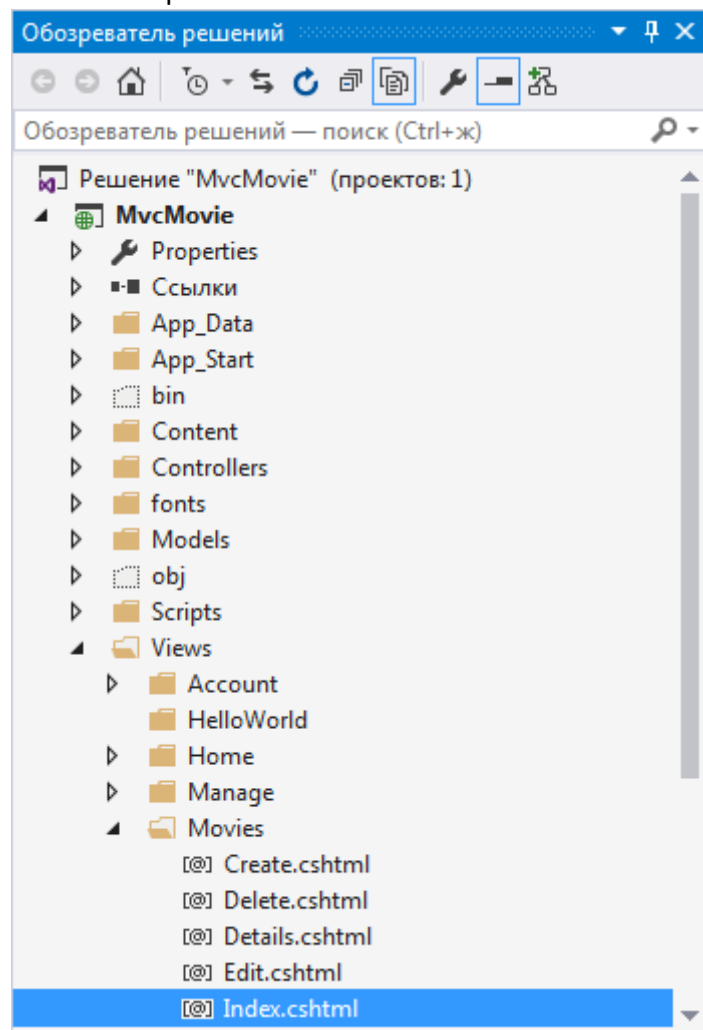
    // Если строка поиска не пустая
    if (!String.IsNullOrEmpty(searchString))
    {
        // выбрать все фильмы в названии которых содержится строка поиска
        movies = movies.Where(s => s.Title.Contains(searchString));
    }

    return View(movies);
}
```



```
1  using System;
2  using System.Collections.Generic;
3  using System.Data;
4  using System.Data.Entity;
5  using System.Linq;
6  using System.Net;
7  using System.Web;
8  using System.Web.Mvc;
9  using MvcMovie.Models;
10
11 namespace MvcMovie.Controllers
12 {
13     public class MoviesController : Controller
14     {
15         private MovieDbContext db = new MovieDbContext();
16
17         // GET: Movies
18         public ActionResult Index(string searchString)
19         {
20             var movies = from m in db.Movies
21                           select m;
22
23             if (!String.IsNullOrEmpty(searchString))
24             {
25                 movies = movies.Where(s => s.Title.Contains(searchString));
26             }
27
28             return View(movies);
29     }
```

49. Открыть файл **Views\Movies\Index.cshtml** и сразу после **@Html.ActionLink("Create New", "Create")** добавить разметку формы, выделенную ниже жёлтым цветом :



```
@model IEnumerable<MvcMovie.Models.Movie>
```

```
@{  
    ViewBag.Title = "Index";  
}
```

```
<h2>Index</h2>
```

```
<p>  
    @Html.ActionLink("Create New", "Create")
```

```
    @using (Html.BeginForm()){  
        <p> Title: @Html.TextBox("SearchString") <br />  
        <input type="submit" value="Filter" /></p>  
    }  
</p>
```

```

Index.cshtml  + X
1  @model IEnumerable<MvcMovie.Models.Movie>
2
3  @{
4      ViewBag.Title = "Index";
5  }
6
7  <h2>Index</h2>
8
9  <p>
10     @Html.ActionLink("Create New", "Create")
11     @using (Html.BeginForm())
12     {
13         <p>
14             Title: @Html.TextBox("SearchString") <br />
15             <input type="submit" value="Filter" />
16         </p>
17     }
18 </p>
19 <table class="table">

```

Html.BeginForm создает открывающий тег **<form>**. **Html.BeginForm** заставляет форму публиковать себя, когда пользователь отправляет форму, нажав кнопку **Filter** (Фильтр).

50. Запустить приложение на компиляцию (или нажать клавишу **F5**), и выполнить фильтрацию по словосочетанию в окне **Title**.

Имя приложения Домашняя страница О программе Контакт Регистрация Выполнить вход

Create New

Title:

Title	ReleaseDate	Genre	Price	
Ёлки	01.11.2007 0:00:00	Комедия	6,00	Edit Details Delete

Добавление поиска по жанру

51. Для поиска фильмов по жанрам изменить метод **Index** в **MvcMovie\Controllers\MoviesController.cs** следующим образом:

```
public ActionResult Index(string movieGenre, string searchString)
{
    var GenreLst = new List<string>();

    var GenreQry = from d in db.Movies
                   orderby d.Genre
                   select d.Genre;

    GenreLst.AddRange(GenreQry.Distinct());
    ViewBag.movieGenre = new SelectList(GenreLst);

    var movies = from m in db.Movies
                 select m;

    if (!String.IsNullOrEmpty(searchString))
    {
        movies = movies.Where(s => s.Title.Contains(searchString));
    }

    if (!string.IsNullOrEmpty(movieGenre))
    {
        movies = movies.Where(x => x.Genre == movieGenre);
    }

    return View(movies);
}
```

```

MoviesController.cs  X
MvcMovie  MvcMovie.Controllers.MoviesController

17 // GET: Movies
18 // ссылка: 0
19 public ActionResult Index(string movieGenre, string searchString)
20 {
21     var GenreLst = new List<string>();
22
23     var GenreQry = from d in db.Movies
24                     orderby d.Genre
25                     select d.Genre;
26
27     GenreLst.AddRange(GenreQry.Distinct());
28     ViewBag.movieGenre = new SelectList(GenreLst);
29
30     var movies = from m in db.Movies
31                   select m;
32
33     if (!String.IsNullOrEmpty(searchString))
34     {
35         movies = movies.Where(s => s.Title.Contains(searchString));
36     }
37
38     if (!string.IsNullOrEmpty(movieGenre))
39     {
40         movies = movies.Where(x => x.Genre == movieGenre);
41     }
42
43     return View(movies);
44 }
```

Эта версия метода **Index** принимает дополнительный параметр, а именно **movieGenre**. Первая строка метода создаёт объект **List** для хранения жанров фильмов из базы данных. Следующие строки представляют собой запрос LINQ, который извлекает все жанры из базы данных.

```
var GenreQry = from d in db.Movies
                orderby d.Genre
                select d.Genre;
```

Далее в коде используется метод **AddRange** универсальной коллекции **List** для добавления в список всех выбранных жанров (модификатор **Distinct** исключает из списка повторяющиеся жанры). Затем происходит сохранение списка жанров в объекте **ViewBag.movieGenre**. Сохранение данных категорий (таких как жанров фильмов) объекта **SelectList** происходит в **ViewBag**, тогда доступ к данным категориям будет предоставлен в раскрывающемся списке.

Следующий код показывает, как проверить параметр **movieGenre**. Если он не пуст, код дополнительно ограничивает запрос видео, чтобы ограничить выбранные фильмы указанным жанром.

```
if (!string.IsNullOrEmpty(movieGenre))
{
    movies = movies.Where(x => x.Genre == movieGenre);
}
```

Добавление разметки в представление Index для поиска по жанру

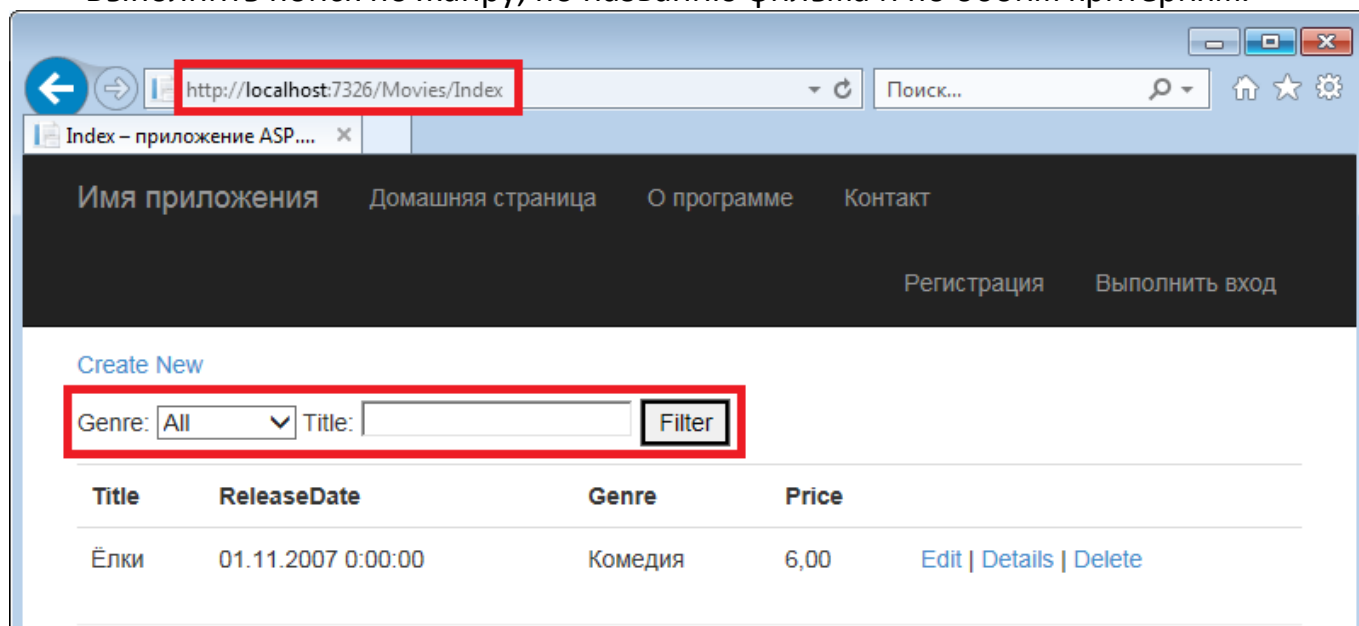
52. Добавить `Html.DropDownList` в файл `Views\Movies\Index.cshtml`, непосредственно перед `TextBox`, так чтобы код принял следующий вид:

```
@model IEnumerable<MvcMovie.Models.Movie>
@{
    ViewBag.Title = "Index";
}
<h2>Index</h2>
<p>
    @Html.ActionLink("Create New", "Create")
    @using (Html.BeginForm("Index", "Movies", FormMethod.Get))
    {
        <p>
            Genre: @Html.DropDownList("movieGenre", "All")
            Title: @Html.TextBox("SearchString")
            <input type="submit" value="Filter" />
        </p>
    }
</p>
<table class="table">
```



```
Index.cshtml  X
1  @model IEnumerable<MvcMovie.Models.Movie>
2
3  @{
4      ViewBag.Title = "Index";
5  }
6
7  <h2>Index</h2>
8
9  <p>
10     @Html.ActionLink("Create New", "Create")
11     @using (Html.BeginForm("Index", "Movies", FormMethod.Get))
12     {
13         <p>
14             Genre: @Html.DropDownList("movieGenre", "All")
15             Title: @Html.TextBox("SearchString")
16             <input type="submit" value="Filter" />
17         </p>
18     }
19 </p>
20 <table class="table">
```

53. Запустить приложение и добавить в строке URL-адреса **/Movies/Index**. Выполнить поиск по жанру, по названию фильма и по обоим критериям.



54. Также можно установить параметр, который будет выбран по умолчанию. Если необходимо в качестве опции по умолчанию жанр **Комедия**, то нужно изменить код в методе **Index** в контроллере в **MvcMovie\Controllers\MoviesController.cs** следующим образом:

```
ViewBag.movieGenre = new SelectList(GenreLst, "Комедия");
```

