|  |  |
| --- | --- |
|  | logo_kmpo  **КОЛЛЕДЖ**  **МНОГОУРОВНЕВОГО**  **ПРОФЕССИОНАЛЬНОГО**  **ОБРАЗОВАНИЯ** |

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ**

**ВЫСШЕГО ОБРАЗОВАНИЯ**

**«РОССИЙСКАЯ АКАДЕМИЯ НАРОДНОГО ХОЗЯЙСТВА И ГОСУДАРСТВЕННОЙ СЛУЖБЫ ПРИ ПРЕЗИДЕНТЕ РОССИЙСКОЙ ФЕДЕРАЦИИ»**

КОЛЛЕДЖ МНОГОУРОВНЕВОГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ

**ОТЧЁТ**

**По дисциплине:** Основы объектно-ориентированного программирования

Выполнила студентка

3 курса

Группы 31КС-20

Колосова Василиса

Специальность 09.02.02

«Компьютерные сети»

Преподаватель

Кукшева Б.А.

**Москва 2022 г**

**Оглавление**

[**Введение.** 2](#_Toc122658269)

[**1.** **Добавление модели в приложение Razor Pages в ASP.NET Core.** 2](#_Toc122658270)

[**2.** **Добавление модели данных.** 2](#_Toc122658271)

[**3.** **Razor Pages, созданные путем формирования шаблонов, в ASP.NET Core** 11](#_Toc122658272)

[**4.** **Работа с базой данных.** 13](#_Toc122658273)

[**5.** **Изменение созданных страниц в приложении ASP.NET Core. Обновление модели** 15](#_Toc122658274)

[**6.** **Добавление поиска в Razor Pages в ASP.NET Core.** 17](#_Toc122658275)

[**7.** **Добавление нового поля на страницу Razor в ASP.NET Core** 23](#_Toc122658276)

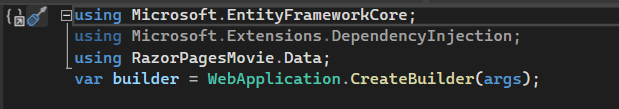
[**Заключение.** 26](#_Toc122658277)

# **Введение.**

В этой серии руководств приводятся основные сведения о создании веб-приложения Razor Pages.

В этом учебнике добавляются классы для управления фильмами в базе данных. Классы моделей приложения используют Entity Framework Core (EF Core) для работы с базой данных. EF Core — это объектно-реляционный сопоставителя (O/RM), упрощающий доступ к данным. Сначала вы записываете классы модели и EF Core создаете базу данных.

# **Добавление модели в приложение Razor Pages в ASP.NET Core.**



**Рисунок 1. Подключение библиотек.**

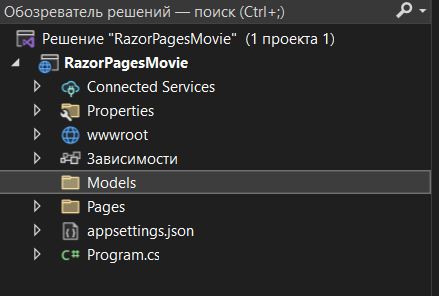
using Microsoft.EntityFrameworkCore;

using Microsoft.Extensions.DependencyInjection;

using RazorPagesMovie.Data;

# **Добавление модели данных.**

В Обозревателе решений щелкните правой кнопкой мыши проект RazorPagesMovie и выберите Добавить> New Folder (Новая папка). Назовите папку Models.



**Рисунок 2. Создание папки “Models”.**

Щелкните правой кнопкой мыши папку Models. Выберите Добавить>Класс. Присвойте классу имя Movie.



**Рисунок 3. Добавление класса Movie.cs.**

Добавьте в класс Movie следующие свойства:

{

public int Id { get; set; }

public string? Title { get; set; }

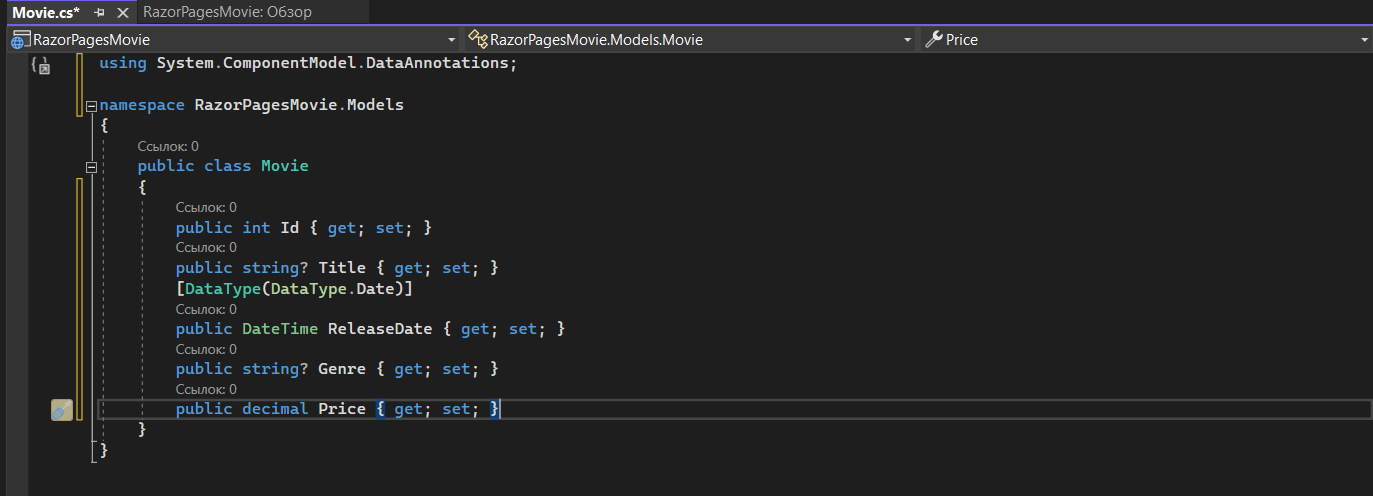
[DataType(DataType.Date)]

public DateTime ReleaseDate { get; set; }

public string? Genre { get; set; }

public decimal Price { get; set; }

}



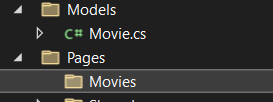
**Рисунок 4. Добавьте в класс Movie следующие свойства.**

Класс Movie содержит:

* Поле ID является обязательным для первичного ключа базы данных
* Атрибут [DataType], указывающий тип данных в свойстве ReleaseDate. С этим атрибутом:
* Пользователю не требуется вводить сведения о времени в поле даты.
* Отображается только дата, а не время.
* Знак вопроса после string указывает, что свойство допускает значение NULL. Дополнительные сведения см. в статье Ссылочные типы, допускающие значение NULL.

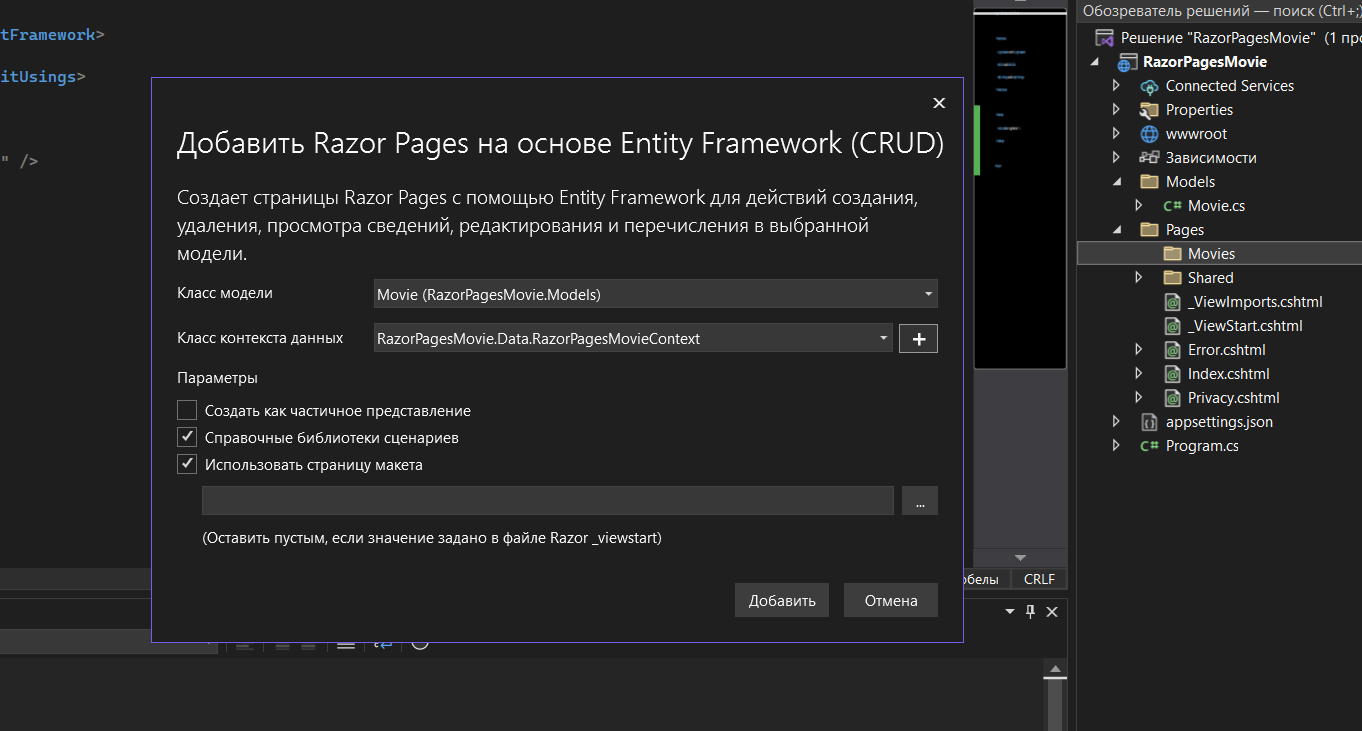
**Создание модели фильма.**

* Создайте папку Pages/Movies.
* Щелкните правой кнопкой мыши папку Pages и выберите Добавить>New Folder (Новая папка).
* Назовите папку Movies.



**Рисунок 5. Создание папку Movies.**

* Щелкните правой кнопкой мыши папку Pages/Movies и выберите Добавить> New Scaffolded Item (Создать шаблонный элемент).
* В диалоговом окне Добавление шаблона щелкните Razor Pages на основе Entity Framework (CRUD)> Добавить.



**Рисунок 6. Создание шаблона.**

В процессе формирования шаблонов создаются указанные ниже файлы.

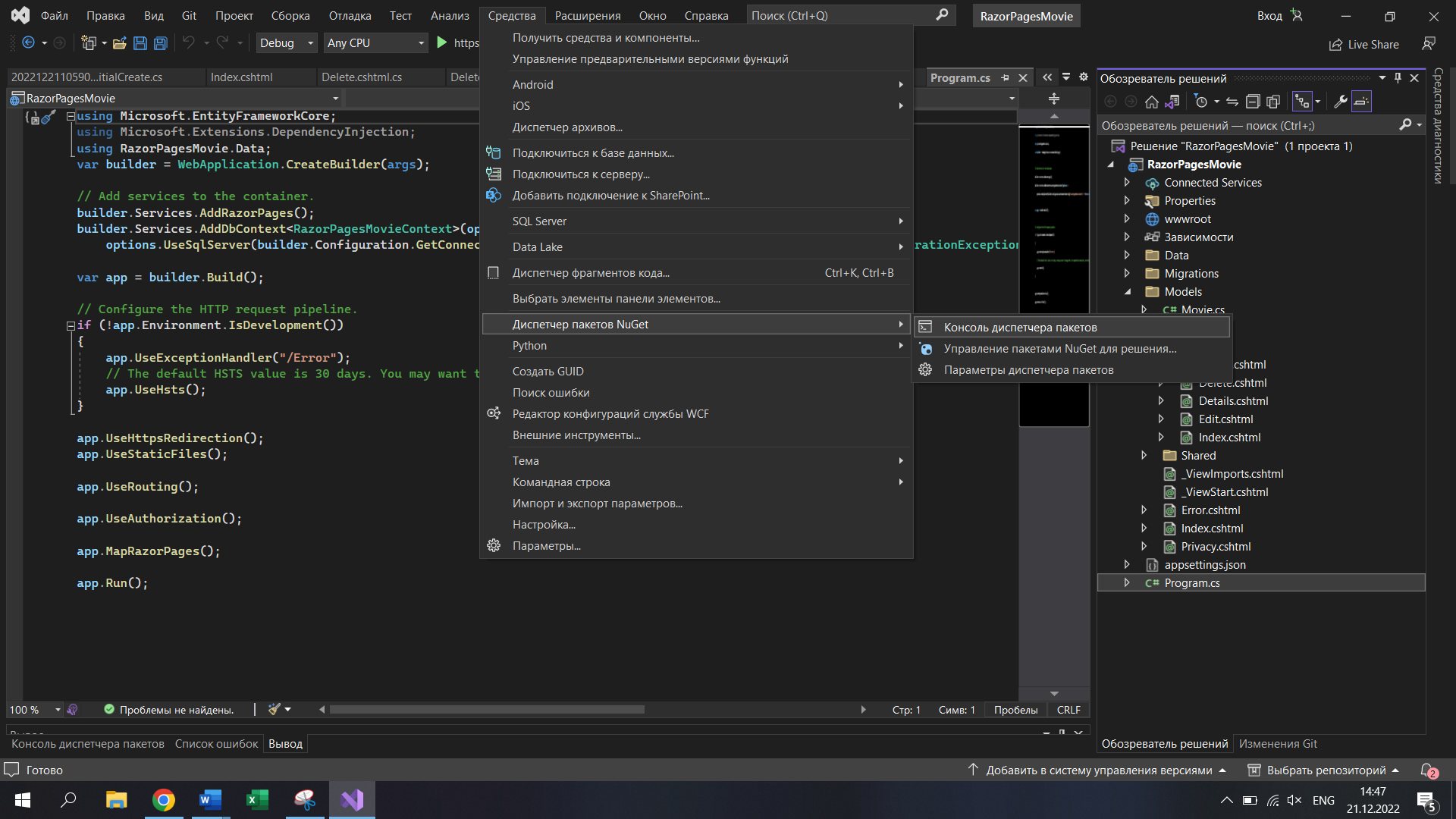
* *Pages/Movies*: Create, Delete, Details, Edit и Index.
* Data/RazorPagesMovieContext.cs

**Создание начальной схемы базы данных с помощью функции миграции EF.**

В этом разделе окно Консоль диспетчера пакетов (PMC) используется для выполнения следующих действий:

* Добавления первоначальной миграции.
* Обновления базы данных с помощью первоначальной миграции.

1. В меню Инструменты выберите Диспетчер пакетов NuGet> Консоль диспетчера пакетов.



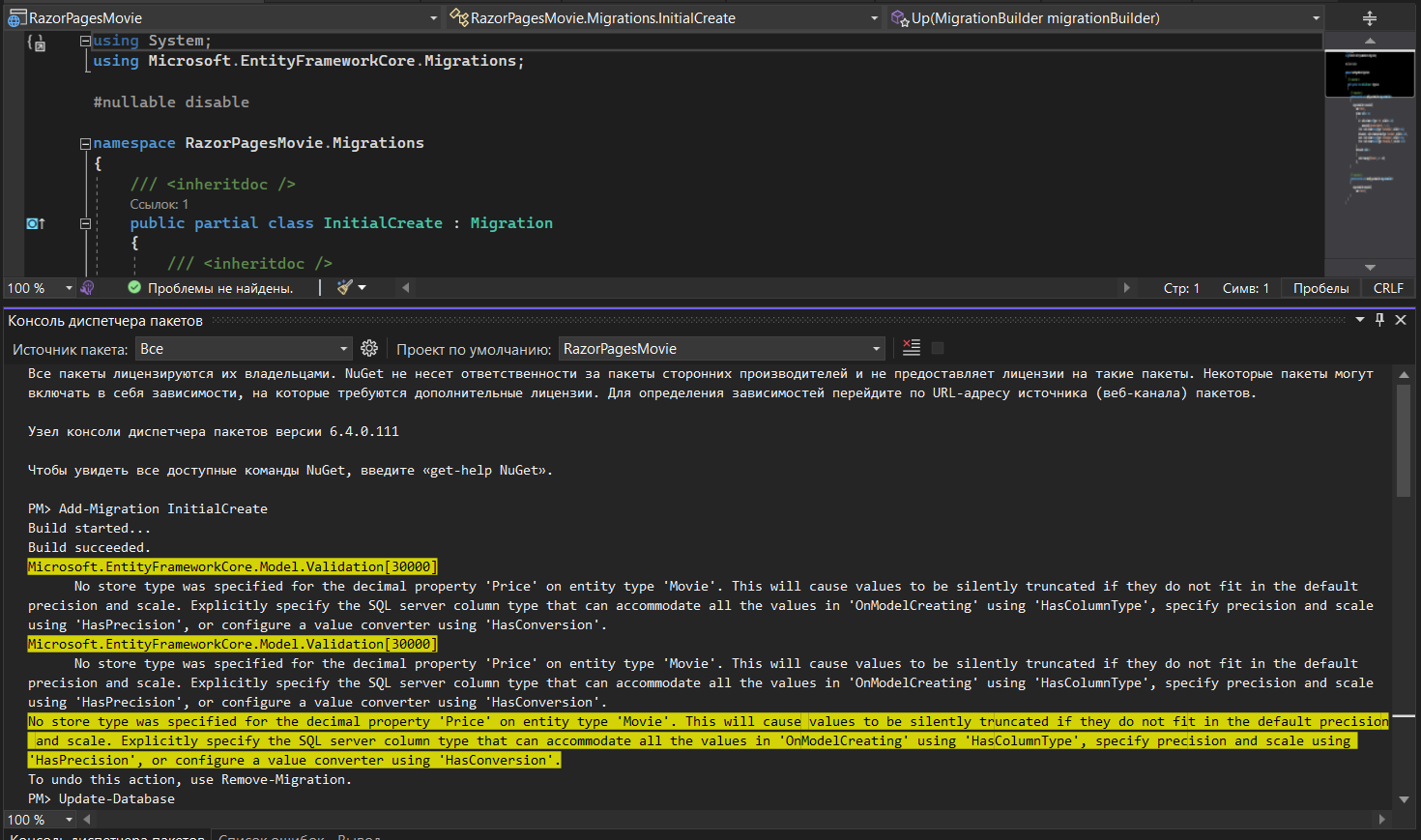
**Рисунок 7. Диспетчер пакетов NuGet> Консоль диспетчера пакетов.**

1. В PMC введите следующие команды:

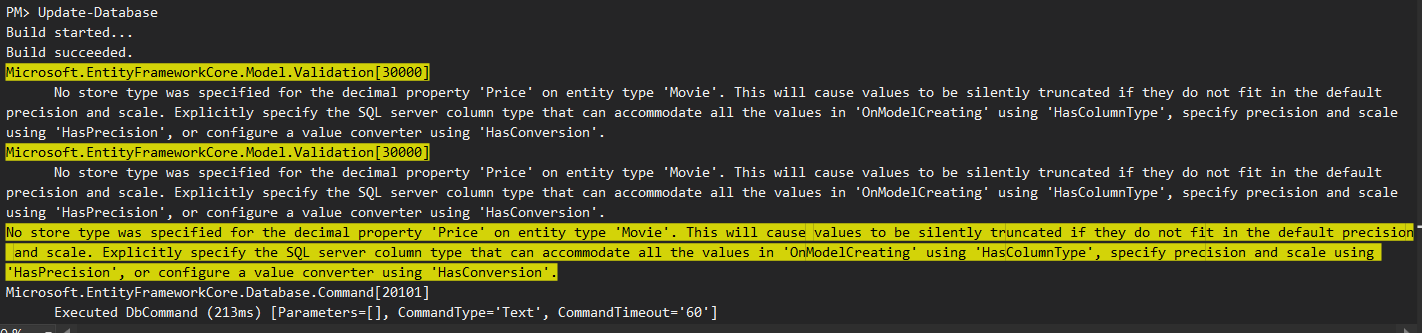
* Add-Migration InitialCreate
* Update-Database

Команда migrations формирует код для создания схемы исходной базы данных. Схема создается на основе модели, указанной в DbContext. Аргумент InitialCreate используется для присвоения имен миграциям. Можно использовать любое имя, однако по соглашению выбирается имя, которое описывает миграцию.

Команда update выполняет метод Up в миграциях, которые не были применены. В этом случае update выполняет метод Up в файле Migrations/<time-stamp>\_InitialCreate.cs, который создает базу данных.



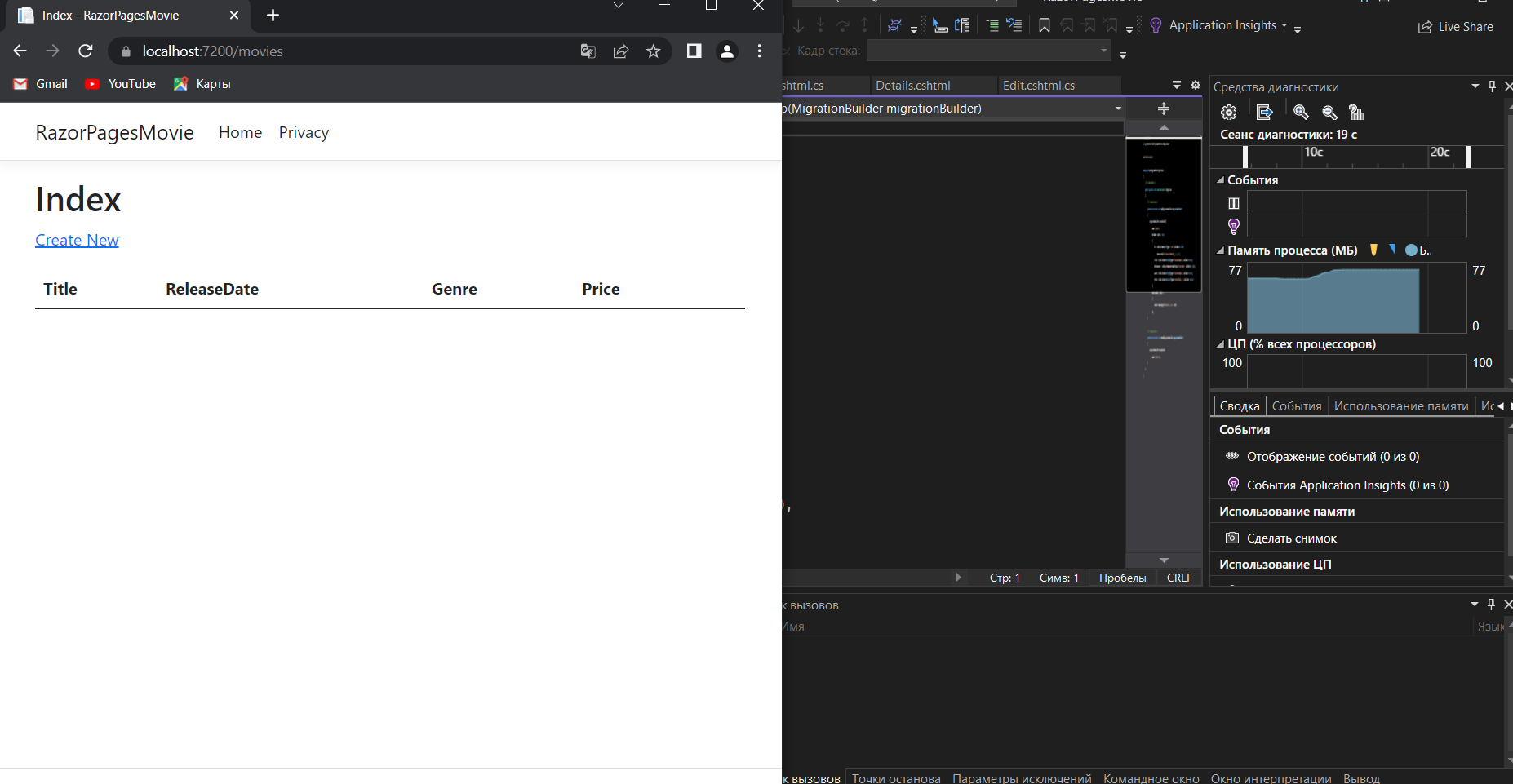
**Рисунок 8. Ввод команды Add** **Migration InitialCreate**



**Рисунок 9. Ввод команды Update-Databased.**

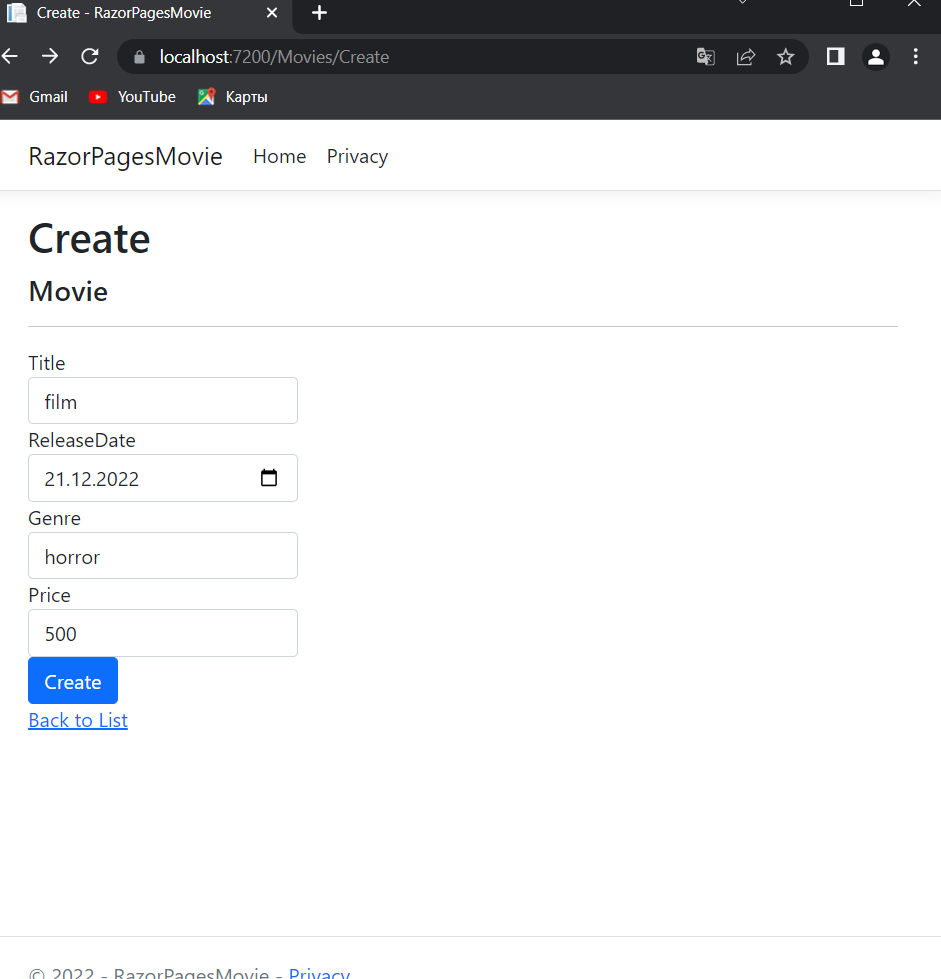
**Тестирование приложения.**

* Запустите приложение и добавьте /Movies к URL-адресу в браузере (<http://localhost:port/movies>).

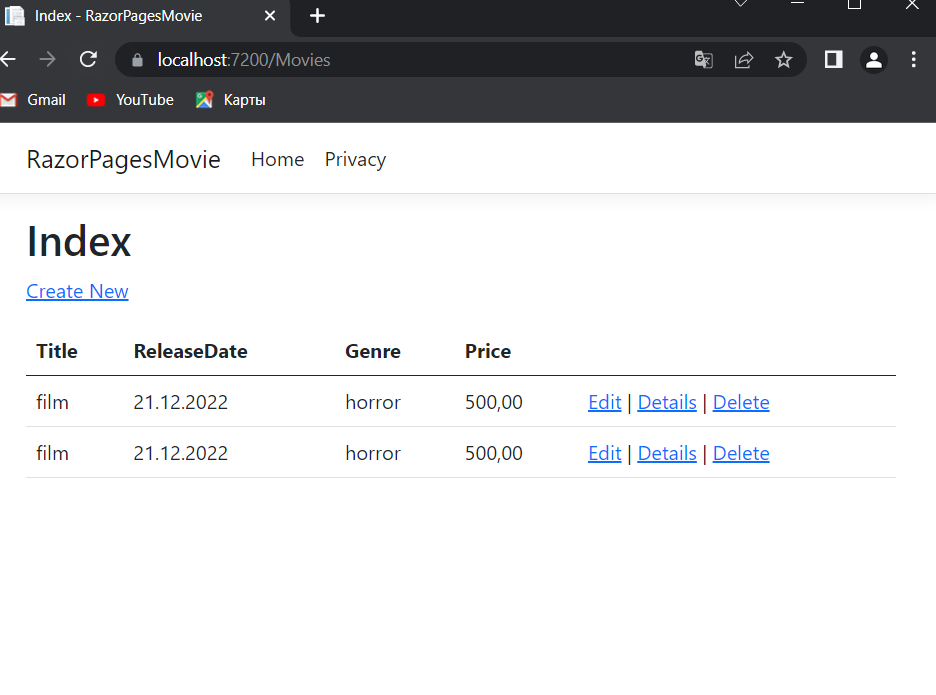


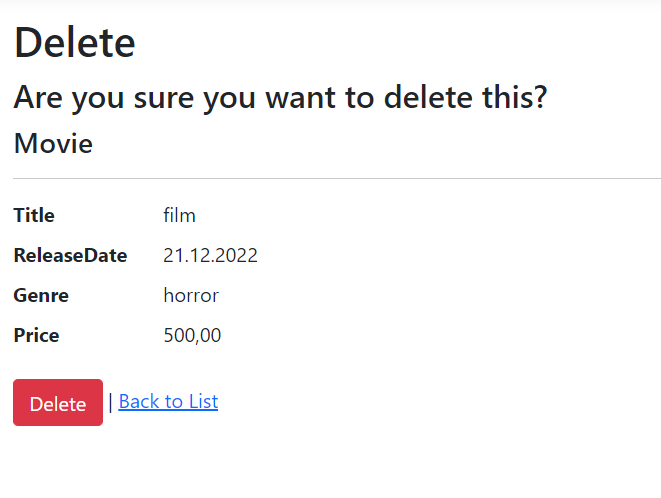
**Рисунок 10. Результат**

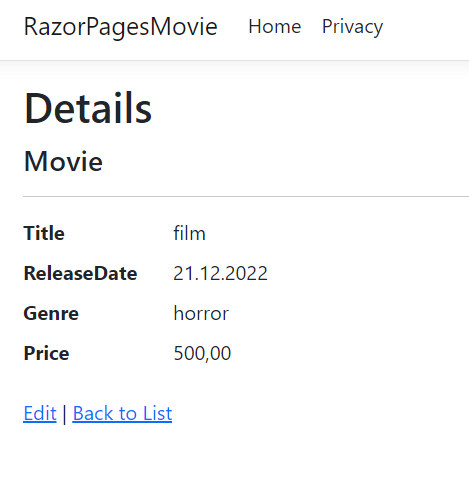
* Протестируйте ссылку Create New (Создать).



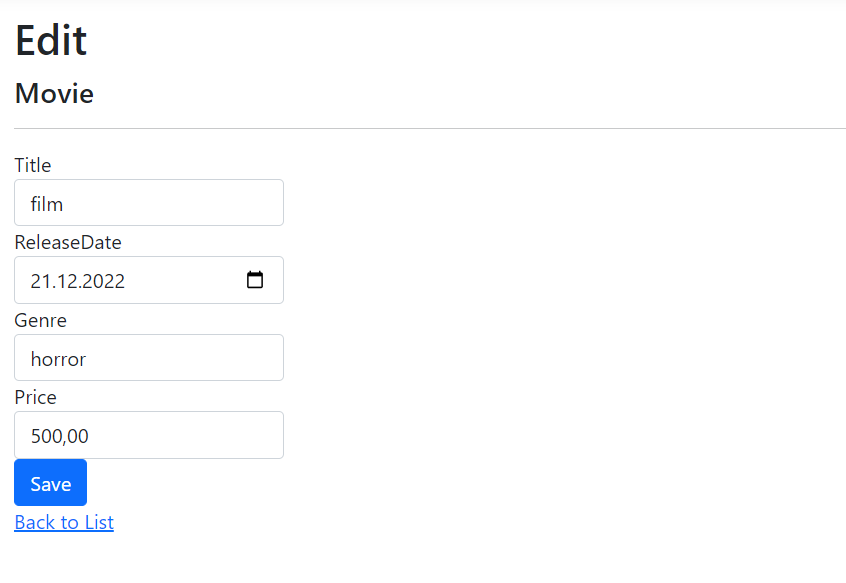
**Рисунок 11. Тестирование ссылки ссылку Create New.**



**Рисунок 12. Тестирование ссылки ссылку Create New.**

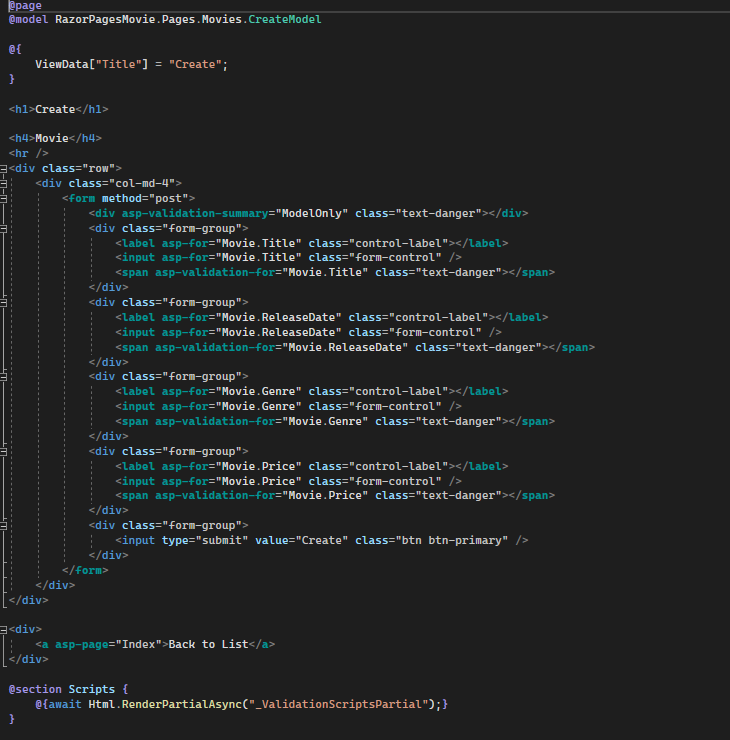


**Рисунок 13. Проверка кнопок Details, Delete.**

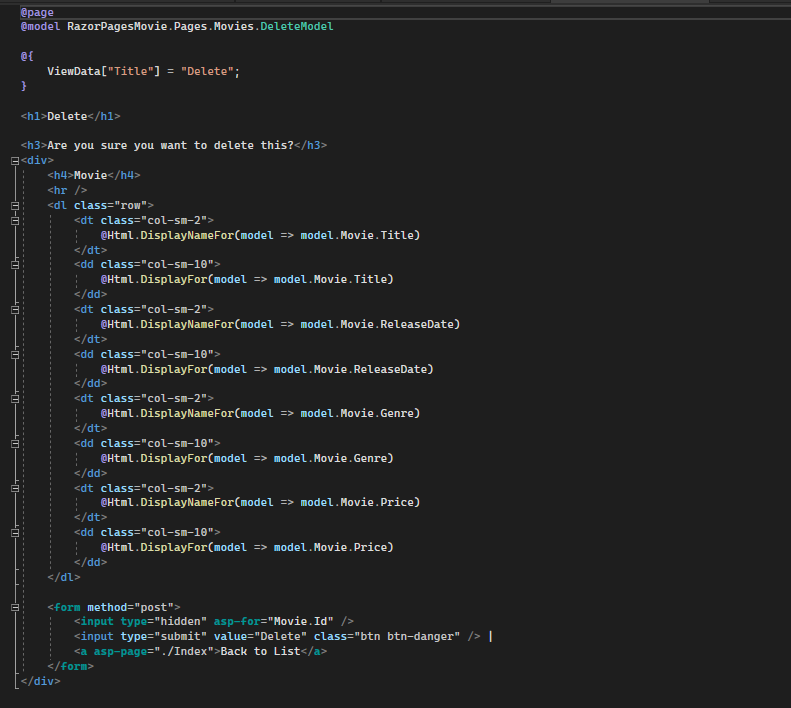


**Рисунок 14. Тестирование кнопки Edit.**

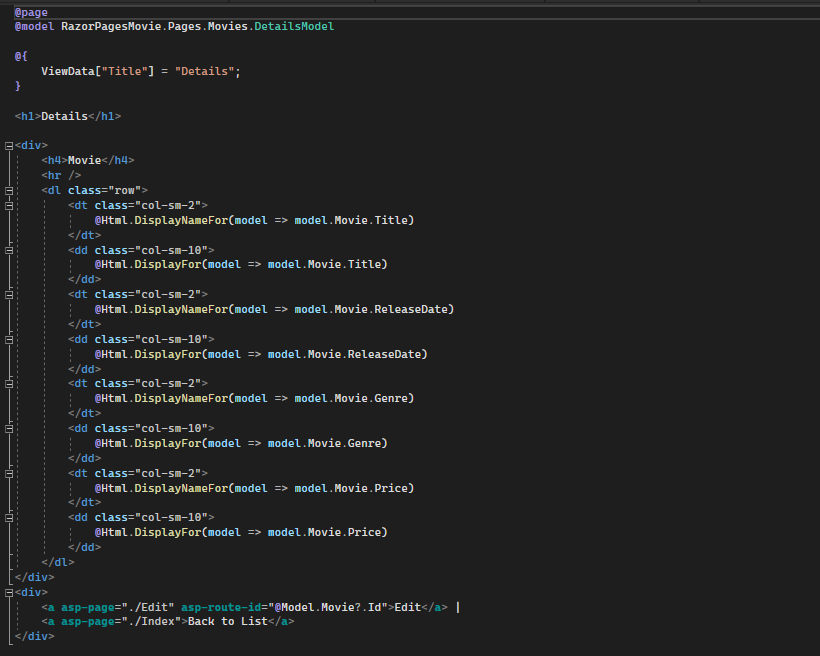
# **Razor Pages, созданные путем формирования шаблонов, в ASP.NET Core**



**Рисунок 15. Код Create.**

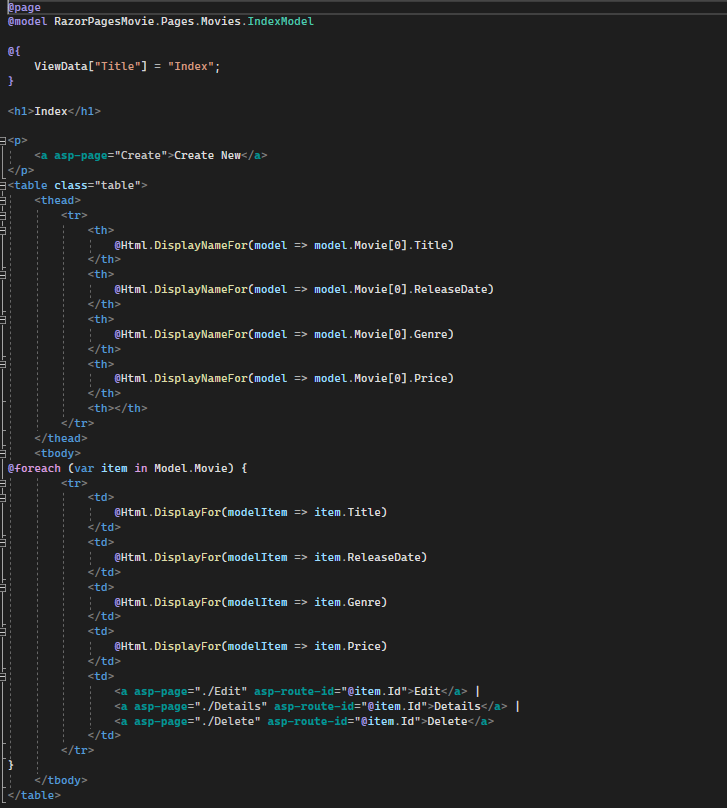


**Рисунок 16. Код Delete.**



**Рисунок 17. Код Details.**

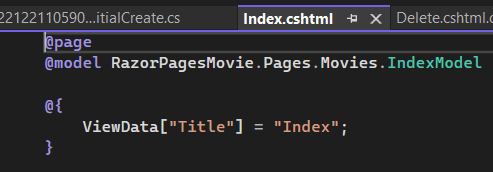
Проверьте страницу Pages/Movies/Index.cshtmlRazor:



**Рисунок 18. Проверка Pages/Movies/Index.cshtmlRazor.**

**ViewData и макет.**

Базовый класс PageModel содержит свойство словаря ViewData. Оно позволяет передать данные в представление. Объекты добавляются в словарь ViewData с помощью шаблона ключ — значение. В приведенном выше примере в словарь ViewData добавляется свойство Title.

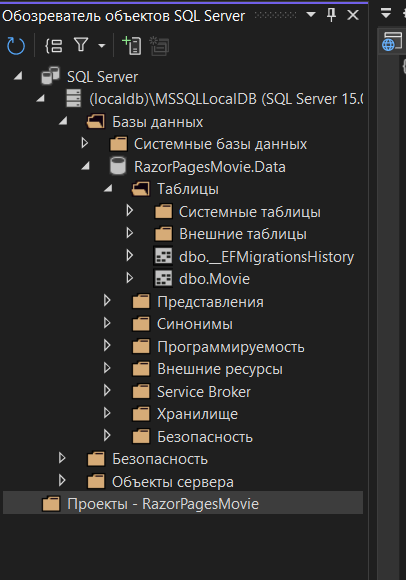


**Рисунок 19. разметка из файла Pages/Movies/Index.cshtml.**

# **Работа с базой данных.**

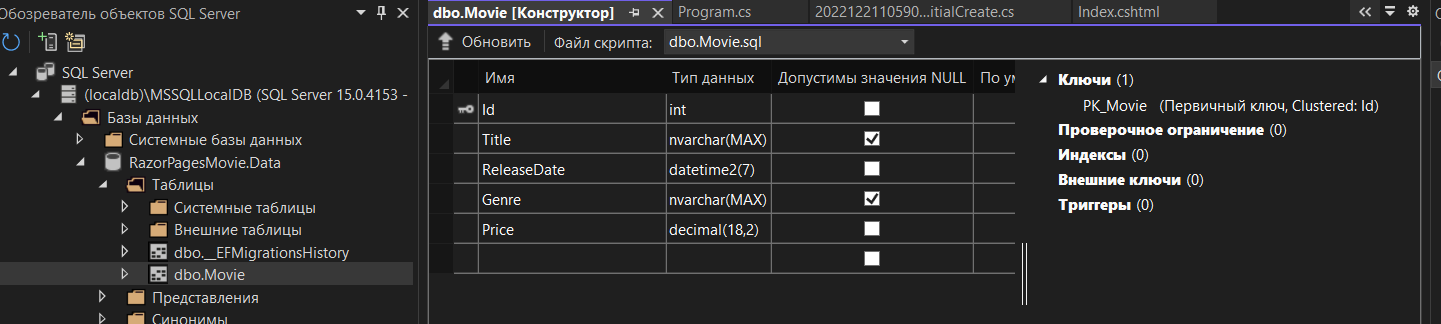
LocalDB — это упрощенная версия ядра СУБД SQL Server Express, предназначенная для разработки программ. LocalDB запускается по запросу в пользовательском режиме, поэтому настройки не слишком сложны. По умолчанию база данных LocalDB создает файлы \*.mdf в каталоге C:\Users\<user>\.

* В меню Вид откройте обозреватель объектов SQL Server (SSOX).



**Рисунок 20. обозреватель объектов SQL Server (SSOX).**

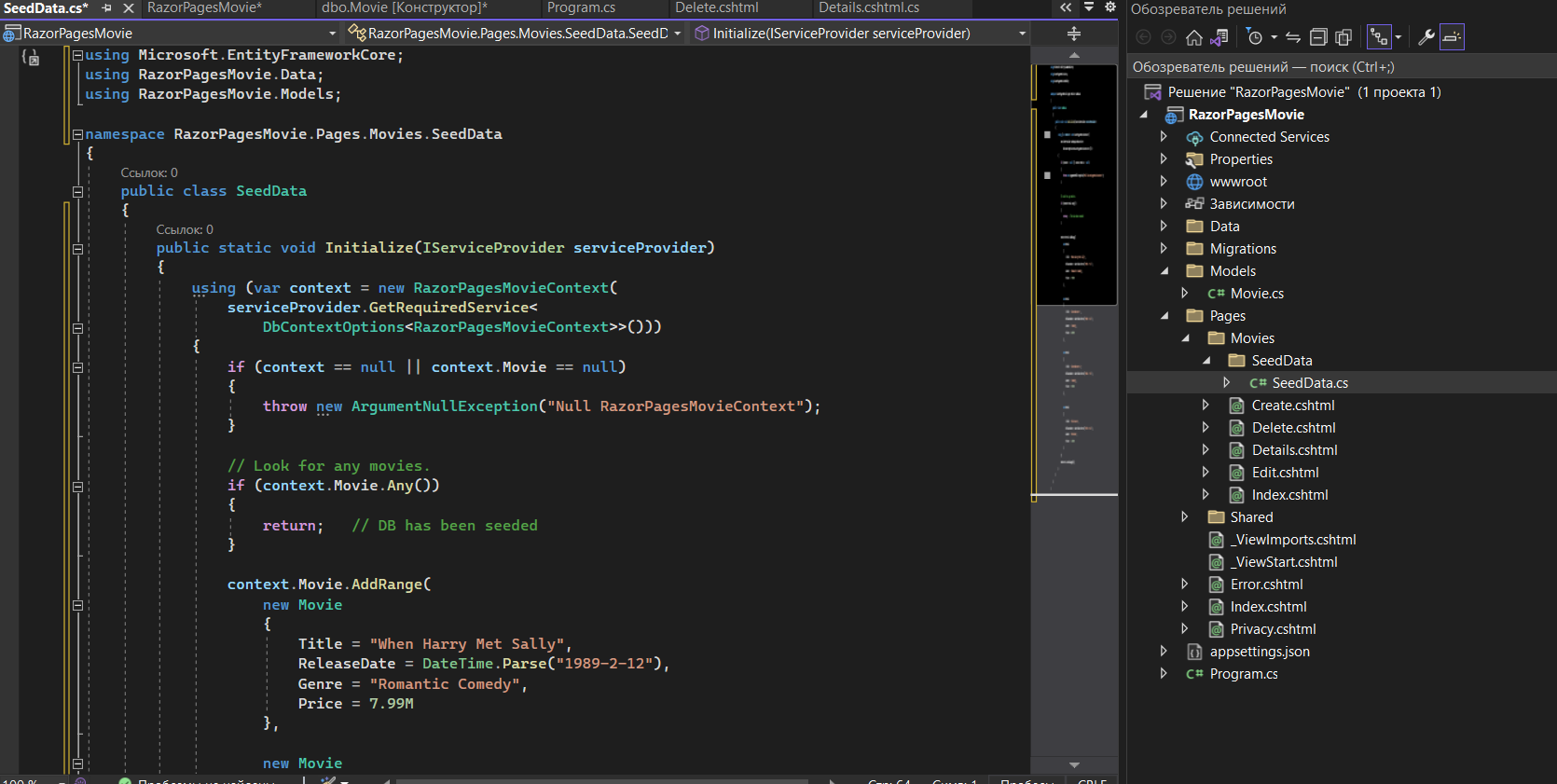
* Щелкните правой кнопкой мыши таблицу Movie и выберите пункт Movie:



Обратите внимание на значок с изображением ключа рядом с ID. По умолчанию EF создает свойство с именем ID для первичного ключа.

**Заполнение базы данных.**

* Создайте класс SeedData в папке SeedData со следующим кодом:



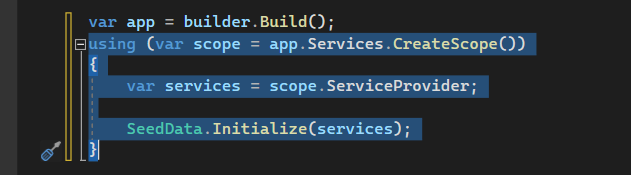
**Рисунок 21. Создание папки и класса SeedData.**

В приведенном выше коде файл Program.cs изменен, чтобы сделать следующее:

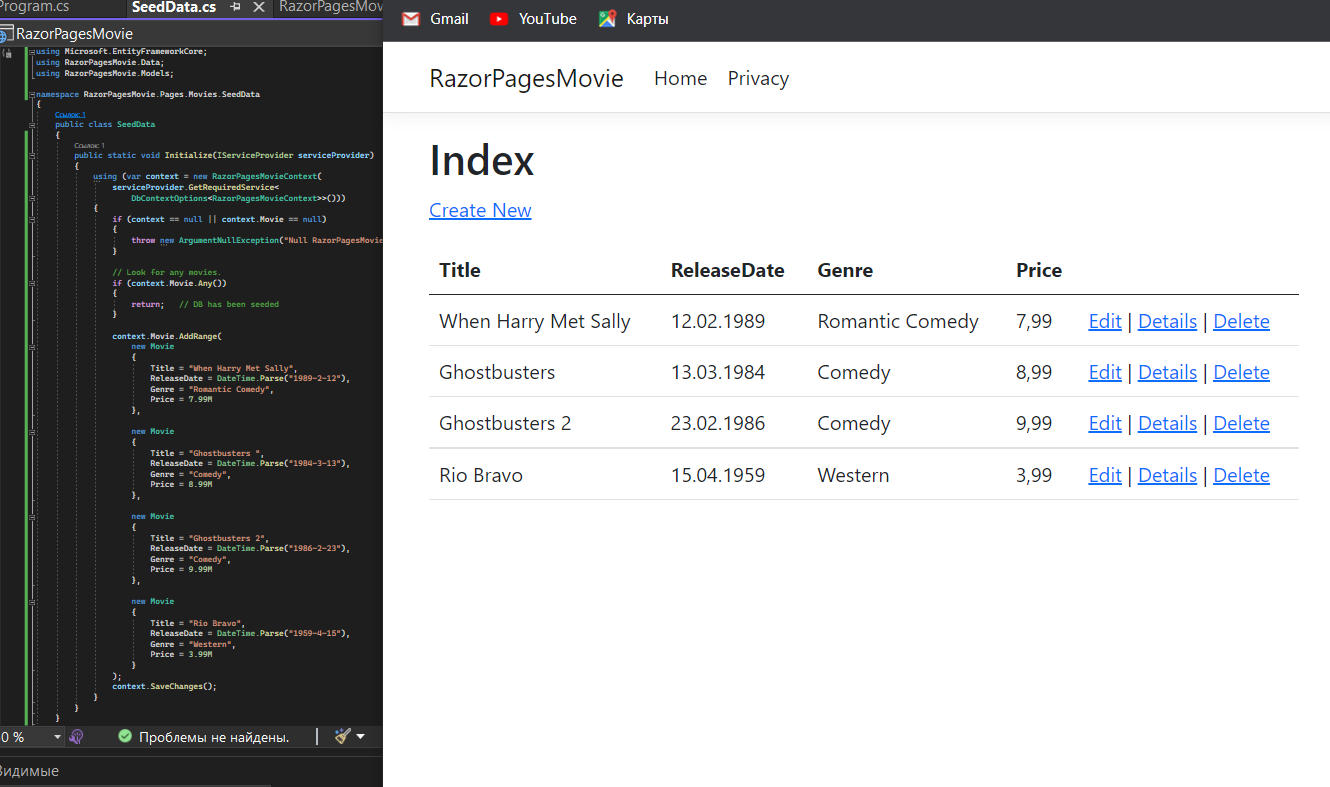
Получение экземпляра контекста базы данных из контейнера внедрения зависимостей (DI).

Вызовите метод seedData.Initialize, передав ему экземпляр контекста базы данных.

Высвобождение контекста после завершения работы метода заполнения. Оператор using гарантирует удаление контекста.

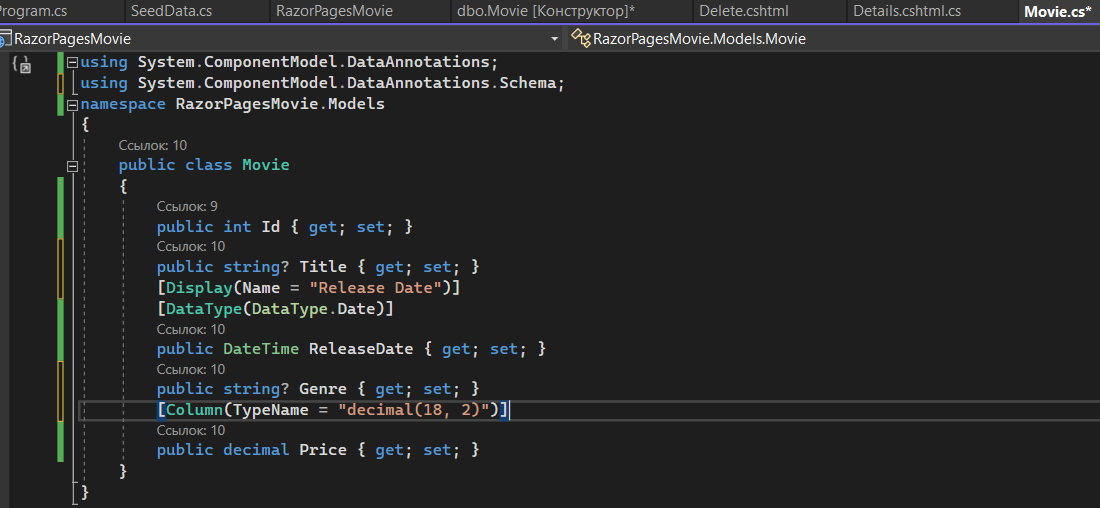


**Рисунок 22. Добавление инициализатора заполнения.**



**Рисунок 23. Тестирование приложения.**

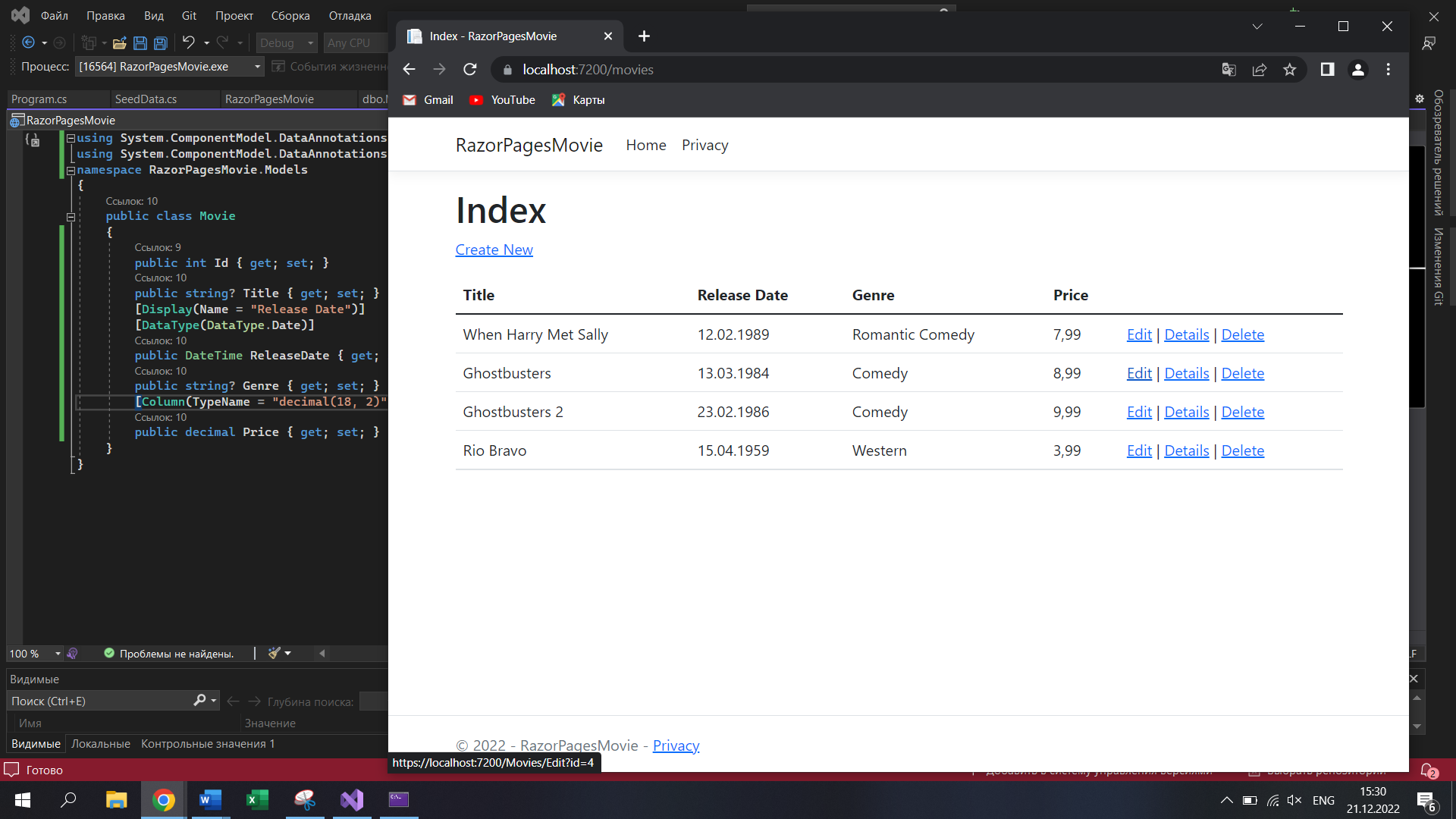
# **Изменение созданных страниц в приложении ASP.NET Core. Обновление модели**



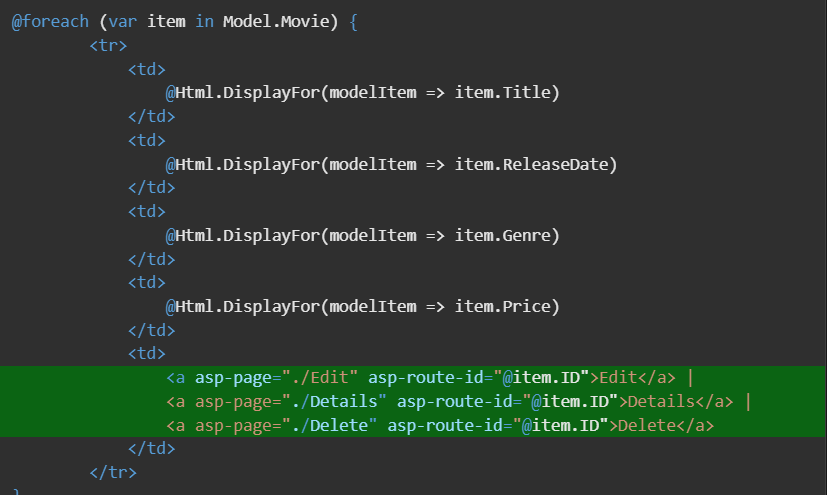
**Рисунок 24. Обновляем файл Models/Movie.cs, используя следующий выделенный код.**

* Заметка к данным [Column(TypeName = "decimal(18, 2)")] позволяет Entity Framework Core корректно сопоставить Price с валютой в базе данных. Дополнительные сведения см. в разделе Типы данных.
* Атрибут [Display] указывает на отображаемое имя поля. В приведенном выше коде Release Date вместо ReleaseDate.
* Атрибут [DataType] указывает тип данных (Date). Сведения о времени, хранящиеся в поле, не отображаются.

Перейдите к Pages/Movies и наведите указатель мыши на ссылку Edit (Изменение), чтобы просмотреть целевой URL-адрес.



**Рисунок 25. Просмотр URL-адреса.**



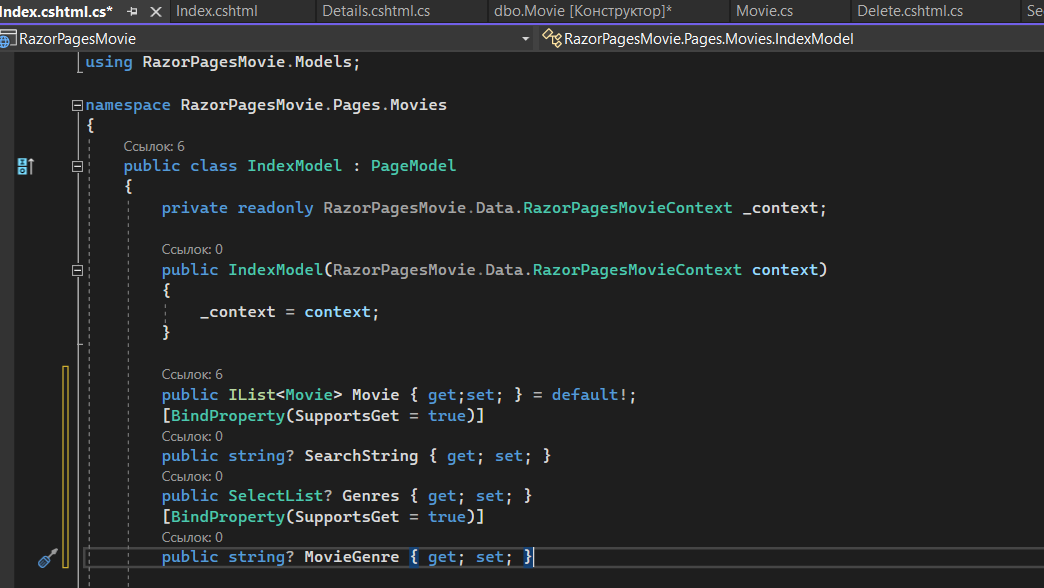
**Рисунок 26. Ссылки Edit, Details и Delete создаются вспомогательной функцией тегов привязки в файле Pages/Movies/Index.cshtml.**

Вспомогательные функции тегов позволяют серверному коду участвовать в создании и отображении HTML-элементов в файлах Razor.

В приведенном выше коде Вспомогательная функция привязки тегов динамически создает значение атрибута HTML href на основе Razor Page (маршрут является относительным), атрибут asp-page и идентификатор маршрута (asp-route-id). Дополнительные сведения см. в разделе Формирование URL-адресов для страниц.

# **Добавление поиска в Razor Pages в ASP.NET Core.**

Добавьте выделенный ниже код в Pages/Movies/Index.cshtml.cs:



**Рисунок 27. Добавляем using Microsoft.AspNetCore.Mvc.Rendering; и [BindProperty(SupportsGet = true)]**

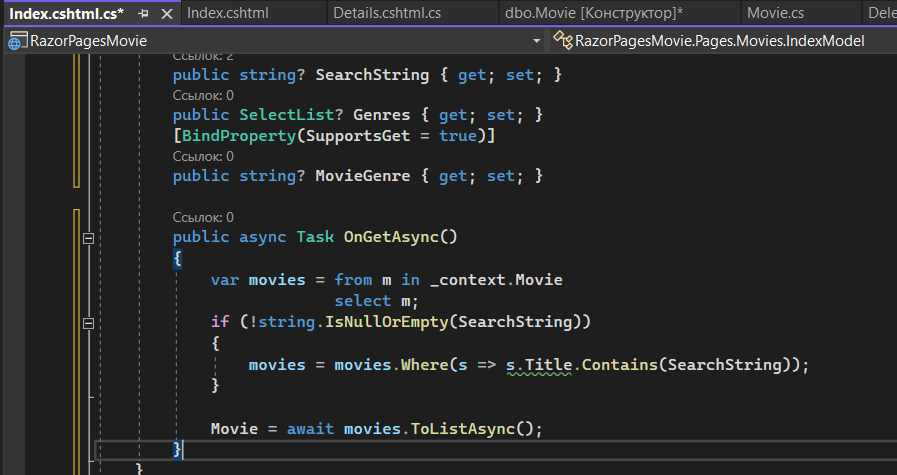
**public string ? SearchString { get; set; }**

**public SelectList ? Genres { get; set; }**

**[BindProperty(SupportsGet = true)]**

**public string ? MovieGenre { get; set; }**

* SearchString: содержит текст, который пользователи вводят в поле поиска. SearchString также имеет атрибут [BindProperty]. [BindProperty] связывает значения из формы и строки запроса с тем же именем, что и у свойства. [BindProperty(SupportsGet = true)] требуется для привязки в запросах HTTP GET.
* Genres: содержит список жанров. Genres дает пользователю возможность выбрать жанр в списке. Для SelectList требуется using Microsoft.AspNetCore.Mvc.Rendering;.
* MovieGenre: содержит конкретный жанр, выбранный пользователем. Например, "Боевик".
* Genres и MovieGenre рассматриваются позднее в этом учебнике.



В первой строке метода OnGetAsync создается запрос LINQ для выбора фильмов:

**// using System.Linq;**

**var movies = from m in \_context.Movie**

**select m;**

Этот запрос только определяется в этой точке и не выполняется для базы данных.

Если свойство SearchString не равно NULL и не пусто, запрос фильмов изменяется для фильтрации по строке поиска:

**if (!string.IsNullOrEmpty(SearchString))**

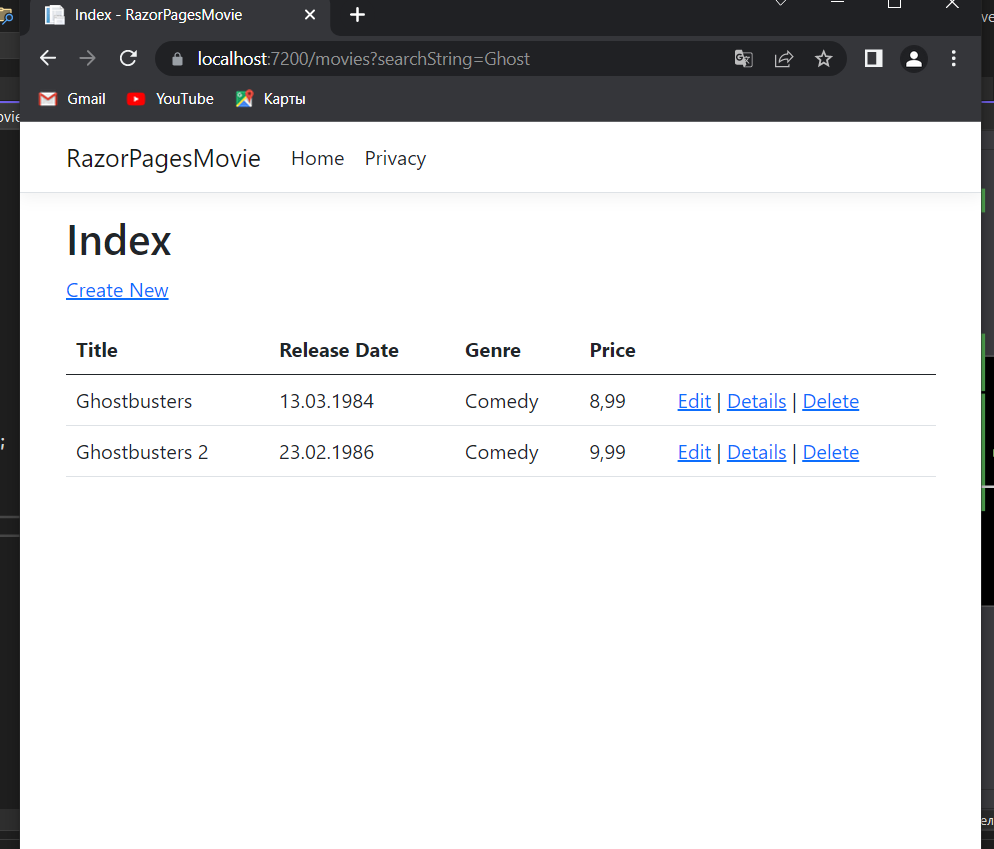
**{**

**movies = movies.Where(s => s.Title.Contains(SearchString));**

**}**

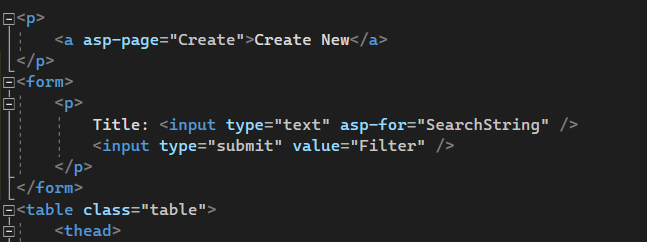
Код s => s.Title.Contains() представляет собой лямбда-выражение. Лямбда-выражения используются в запросах LINQ на основе методов в качестве аргументов стандартных методов операторов запроса, таких как метод Where или Contains. Запросы LINQ не выполняются, если они определяются или изменяются путем вызова метода, например Where, Containsили OrderBy. Вместо этого выполнение запроса откладывается. Вычисление выражения откладывается до тех пор, пока не будет выполнена итерация его реализованного значения или не будет вызван метод ToListAsync.

Перейдите на страницу Movies и добавьте строку запроса, например ?searchString=Ghost, к URL-адресу. Например, https://localhost:5001/Movies?searchString=Ghost. Отображаются отфильтрованные фильмы.



**Рисунок 28. Проверка.**

Откройте файл Pages/Movies/Index.cshtml и добавьте разметку, которая выделена в следующем коде:



<form>

<p>

Title: <input type="text" asp-for="SearchString" />

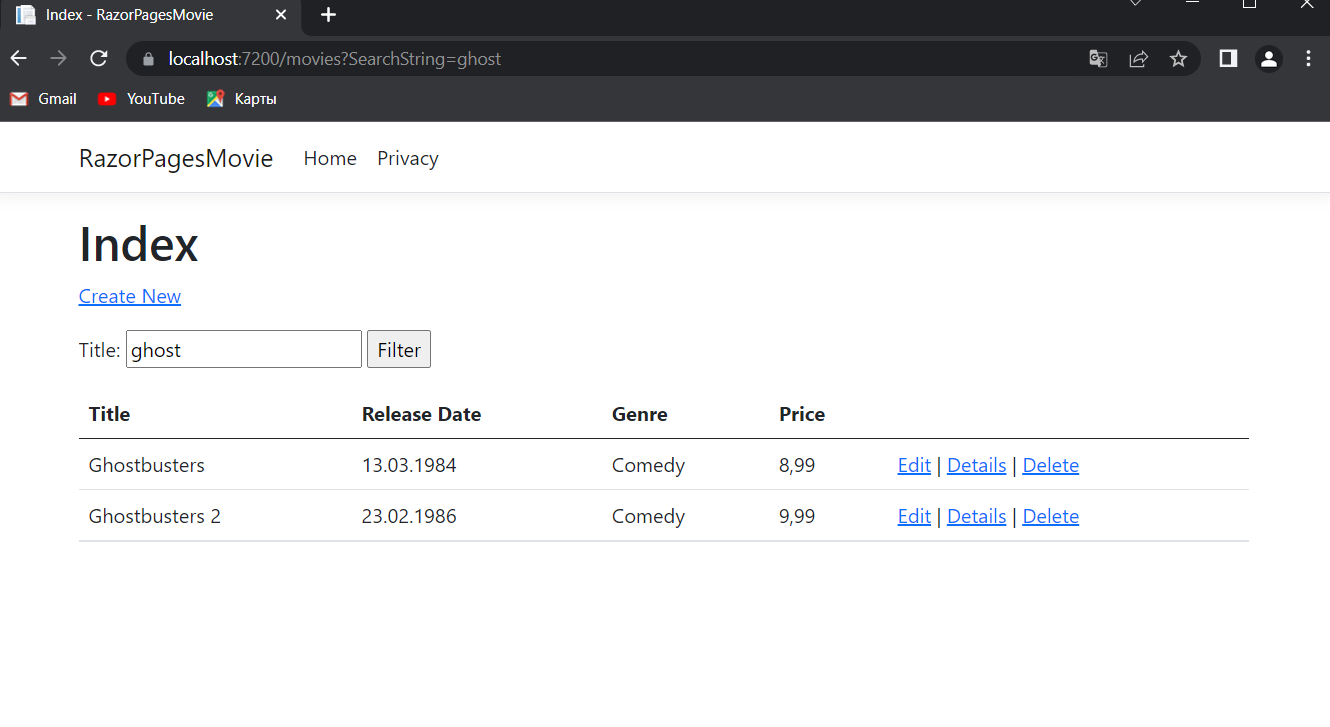
<input type="submit" value="Filter" />

</p>

</form>

Тег HTML <form> использует следующие вспомогательные функции тегов:

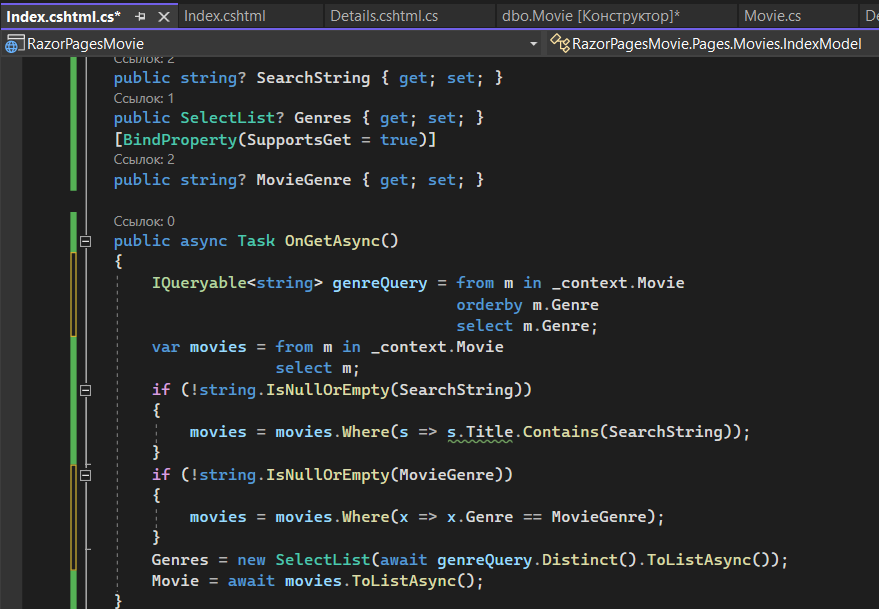
* вспомогательная функция тега форм. При отправке формы строка фильтра отправляется на страницу Pages/Movies/Index в строке запроса.
* Вспомогательная функция тега Input.



**Рисунок 29. Проверка работы фильтров.**

**Поиск по жанру.**

Обновите метод OnGetAsync страницы Index, добавив следующий код:



**Рисунок 30. код, который добавили.**

Следующий код определяет запрос LINQ, который извлекает все жанры из базы данных.

**// Use LINQ to get list of genres.**

**IQueryable<string> genreQuery = from m in \_context.Movie**

**orderby m.Genre**

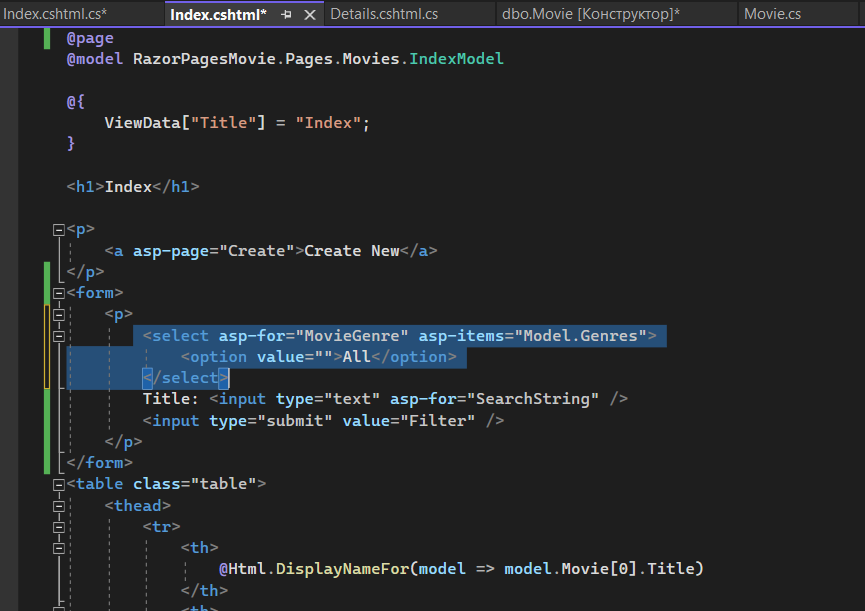
**select m.Genre;**

Список жанров SelectList создается путем проецирования отдельных жанров.

**Genres = new SelectList(await genreQuery.Distinct().ToListAsync());**

**Добавление поиска по жанру на страницу Razor**

Обновите Index.cshtmlэлемент <form>, как показано в следующей разметке:

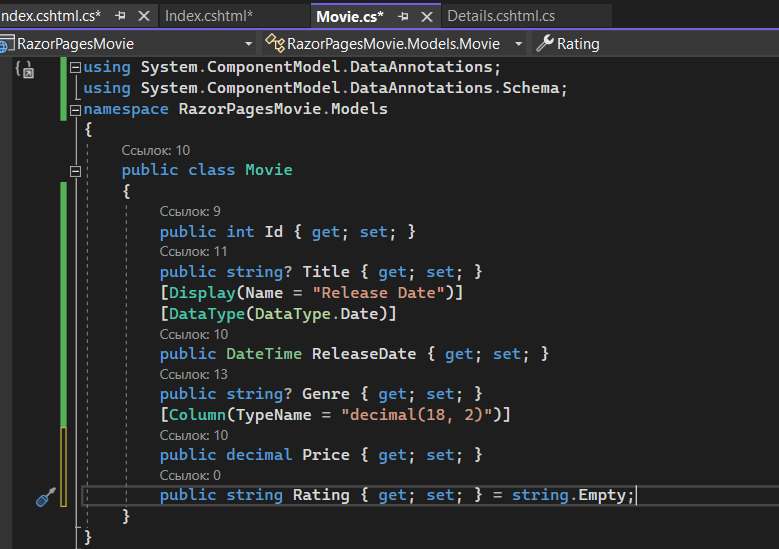


**Рисунок 31. Обновляем Index.cshtmlэлемент <form>, как показано в коде.**

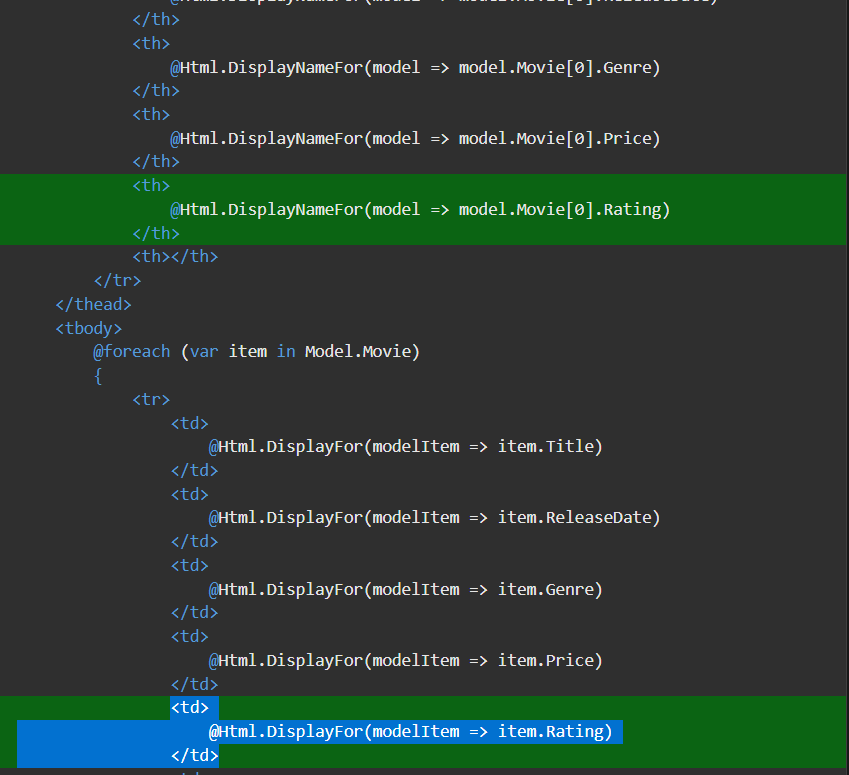
# **Добавление нового поля на страницу Razor в ASP.NET Core**

**Добавление свойства Rating в модель Movie.**

* Откройте файл Models/Movie.cs и добавьте свойство Rating:



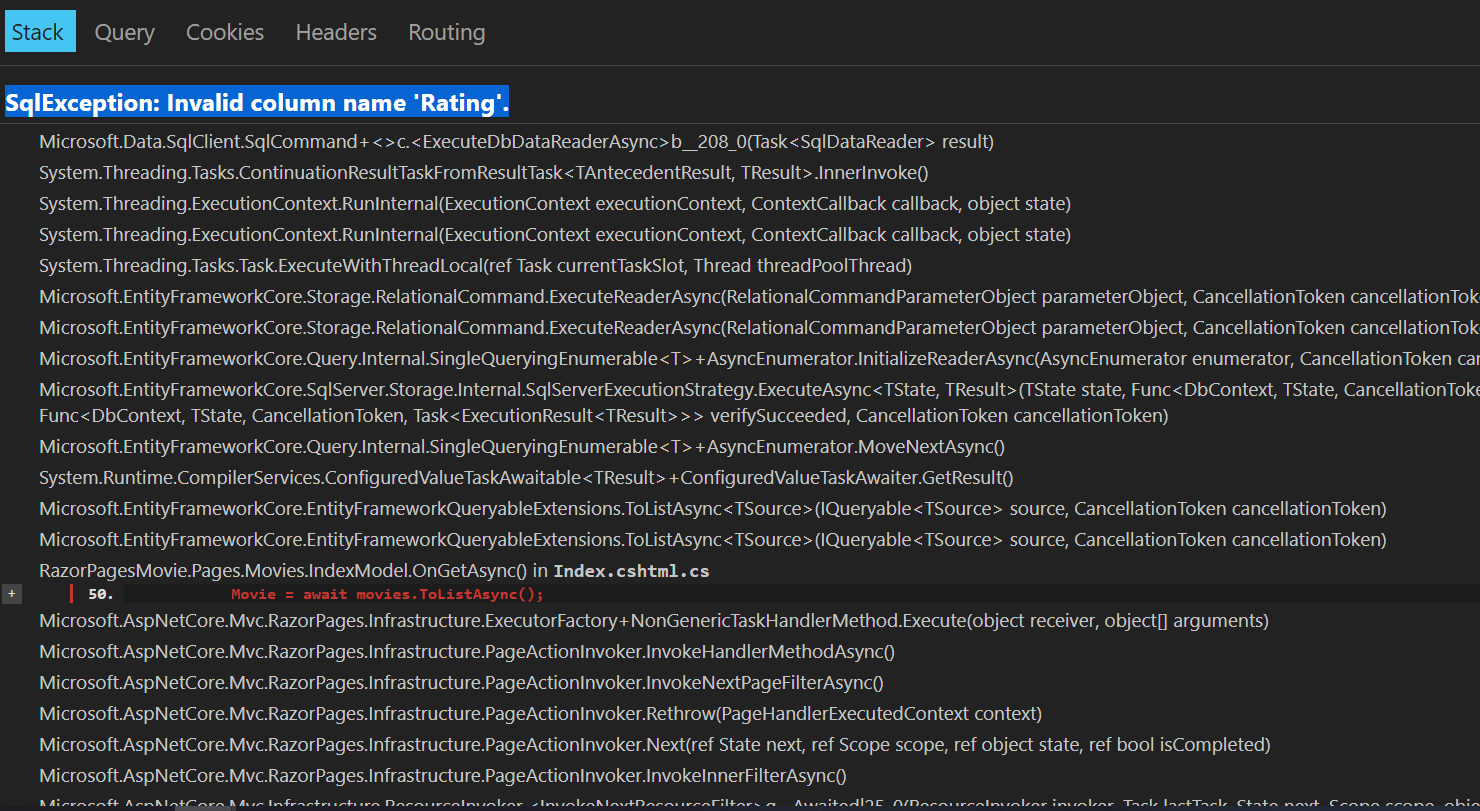
* Отредактируйте Pages/Movies/Index.cshtml и добавьте поле Rating:



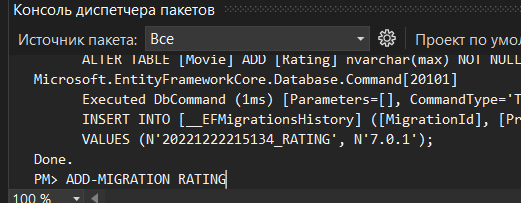
Для работы приложения необходимо обновить базу данных, включив в нее новое поле. При запуске приложения без обновления базы данных возникает SqlException

**SqlException: Invalid column name 'Rating'.**

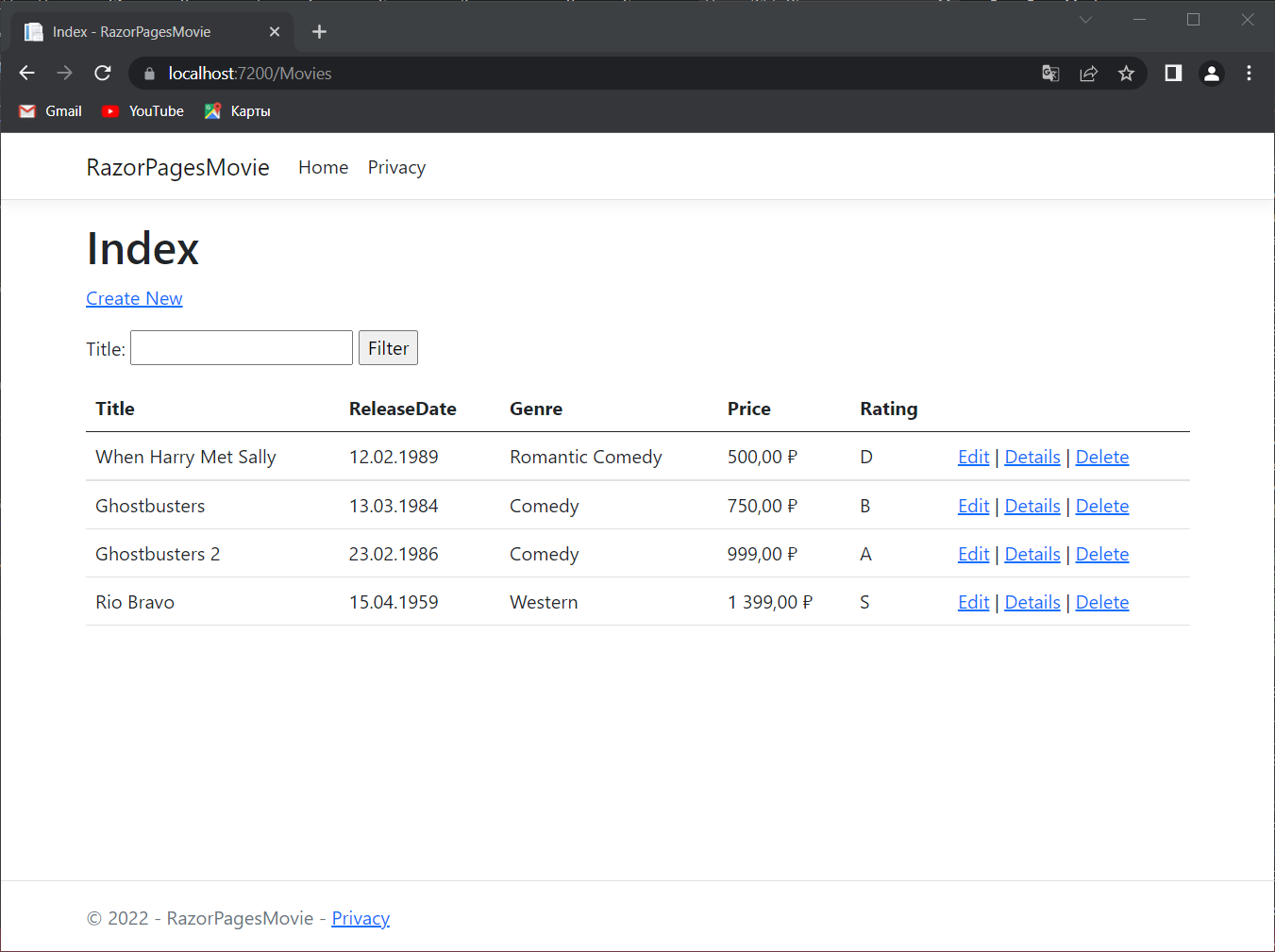
Исключение SqlException связано с тем, что обновленный класс модели Movie отличается от схемы таблицы Movie в базе данных. В таблице базы данных нет столбца Rating**.**



**Рисунок 32.Ошибка.**



**Рисунок 33. Добавление миграции в рейтинг.**



**Рисунок 34. Готово. Успешно выполненная работа.**

# **Заключение.**

Передо мной была поставлена задача Создание веб-приложения Razor Pages с помощью ASP.NET Core и отработать навыки написания программного кода, исправления ошибок и как я считаю задача была выполнена в полном объёме.