Методические указания для выполнения лабораторных работ в среде «Visual Studio ASP.NET» с помощью технологии MVC 5

В рамках данного учебного примера давайте создадим простое web-приложение, работающее с авторами и книгами некой библиотеки и осуществляющее добавление, редактирование и удаление информации о них (рис. 1).

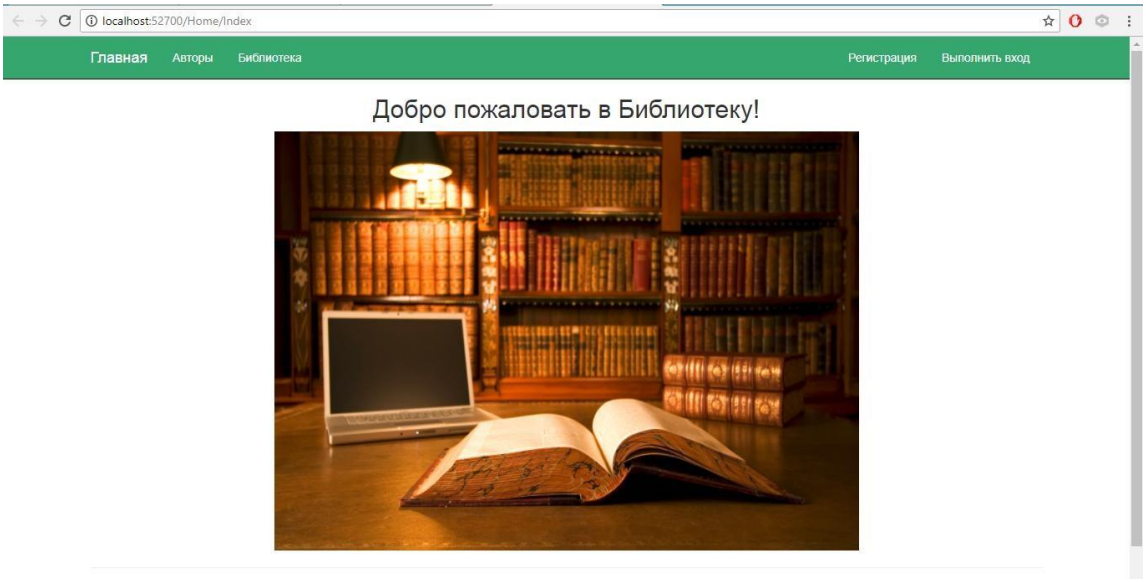


Рис. 1. Внешний вид готового проекта

Этап 1. Создание проекта

Первым делом необходимо создать проект MVC, используя готовый шаблон. Для этого следует открыть среду разработки ***Visual Studio*** (данное пособие адаптировано к версии Visual Studio 2017) и выбрать ***Файл → Создать → Проект***. В окне «Создание проекта» необходимо в дереве шаблонов выбрать ***Visual C# → Веб → Веб-приложение ASP.NET (.NET Framework)***. Этому проекту следует задать имя и расположение в нижних строках (рис. 2). Выбор платформы будет зависеть от того, какие библиотеки установлены на вашем компьютере, и может не совпадать с рассмотренным вариантом. Рекомендуется выбрать платформу 4.6.x либо 4.7.x и нажать «ОК».

В появившемся окне выбираем шаблон MVC, меняем способ проверки подлинности на «Учетные записи отдельных пользователей», остальные поля оставляем без изменения (рис. 3). Нажимаем «ОК». Спустя некоторое время Visual Studio автоматически сгенерирует функционально богатое веб-приложение, которое мы будем адаптировать под поставленные цели (рис. 4).

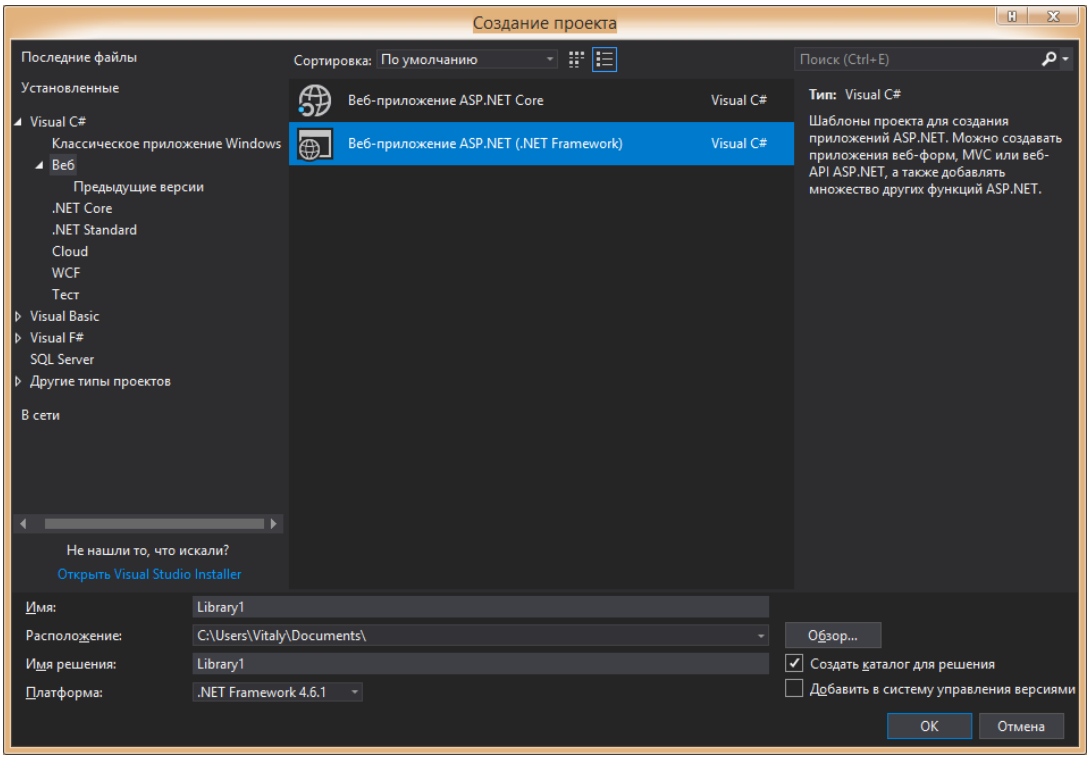

Создание проекта
Последние файлы
Установленные
Visual C#
Классическое приложение Windows
Веб
Предыдущие версии
.NET Core
.NET Standard
Cloud
WCF
Тест
Visual Basic
Visual F#
SQL Server
Другие типы проектов
В сети
Не нашли то, что искали?
Открыть Visual Studio Installer
Сортировка: По умолчанию
Веб-приложение ASP.NET Core Visual C#
Веб-приложение ASP.NET (.NET Framework) Visual C#
Поиск (Ctrl+E)
Тип: Visual C#
Шаблоны проекта для создания приложений ASP.NET. Можно создавать приложения веб-форм, MVC или веб-API ASP.NET, а также добавлять множество других функций ASP.NET.
Имя: Library1
Расположение: C:\Users\Vitaly\Documents\
Имя решения: Library1
Платформа: .NET Framework 4.6.1
Обзор...
Создать каталог для решения
Добавить в систему управления версиями
OK
Отмена

Рис. 2.

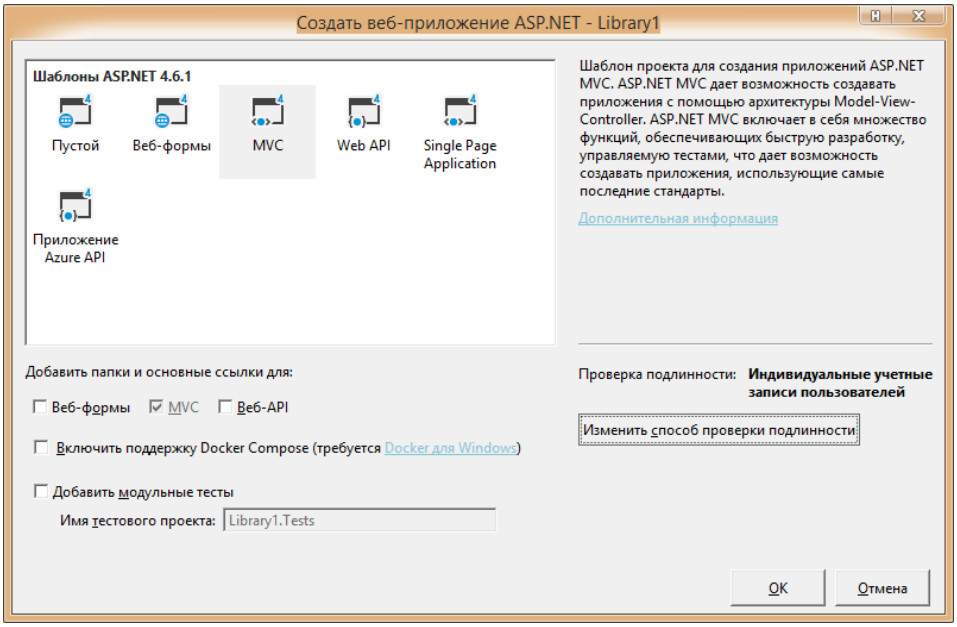

Создать веб-приложение ASP.NET - Library1
Шаблоны ASP.NET 4.6.1
Пустой
Веб-формы
MVC
Web API
Single Page Application
Приложение Azure API
Шаблон проекта для создания приложений ASP.NET MVC. ASP.NET MVC дает возможность создавать приложения с помощью архитектуры Model-View-Controller. ASP.NET MVC включает в себя множество функций, обеспечивающих быструю разработку, управляемую тестами, что дает возможность создавать приложения, использующие самые последние стандарты.
Дополнительная информация
Добавить папки и основные ссылки для:
Веб-формы
MVC
Веб-API
Включить поддержку Docker Compose (требуется Docker для Windows)
Добавить модульные тесты
Имя тестового проекта: Library1.Tests
Проверка подлинности: Индивидуальные учетные записи пользователей
Изменить способ проверки подлинности
OK
Отмена

Рис. 3.

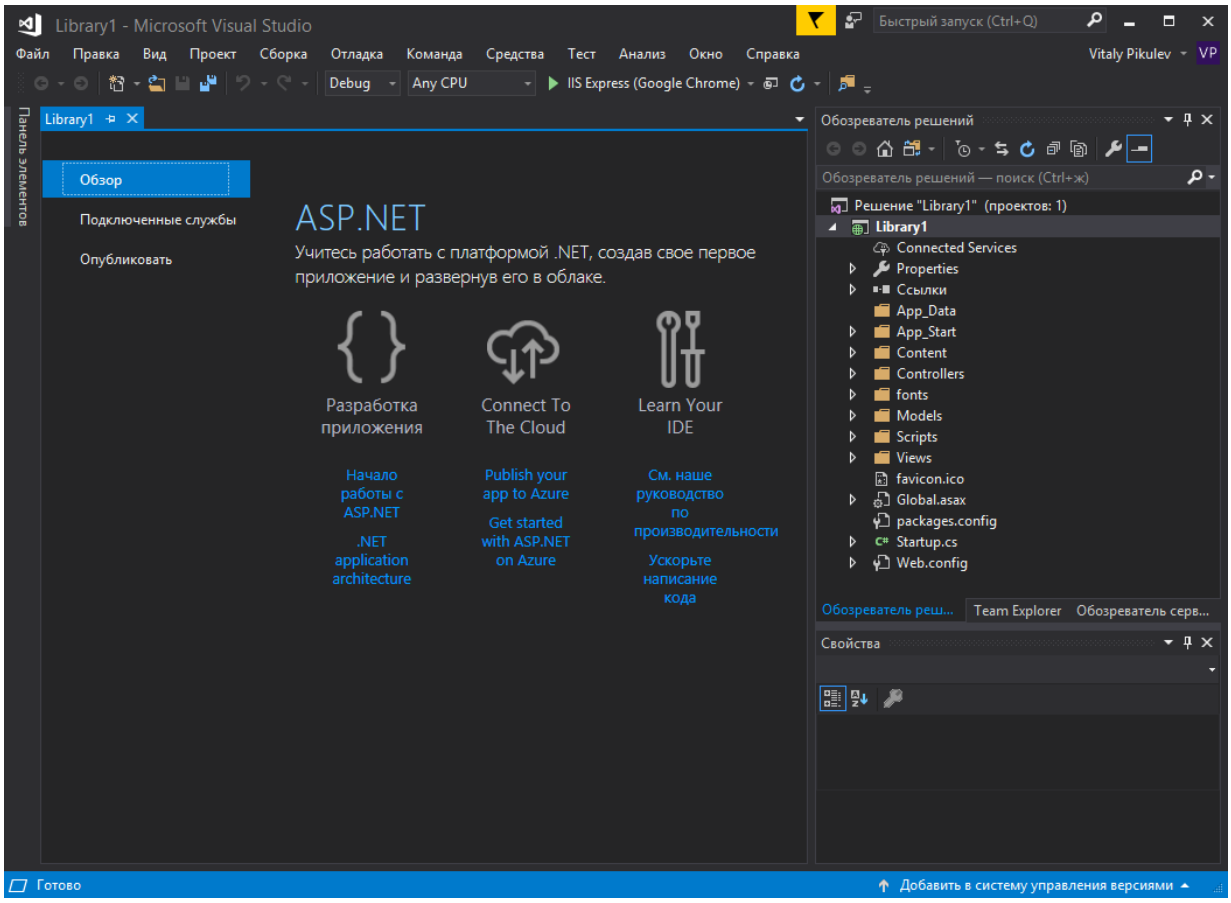


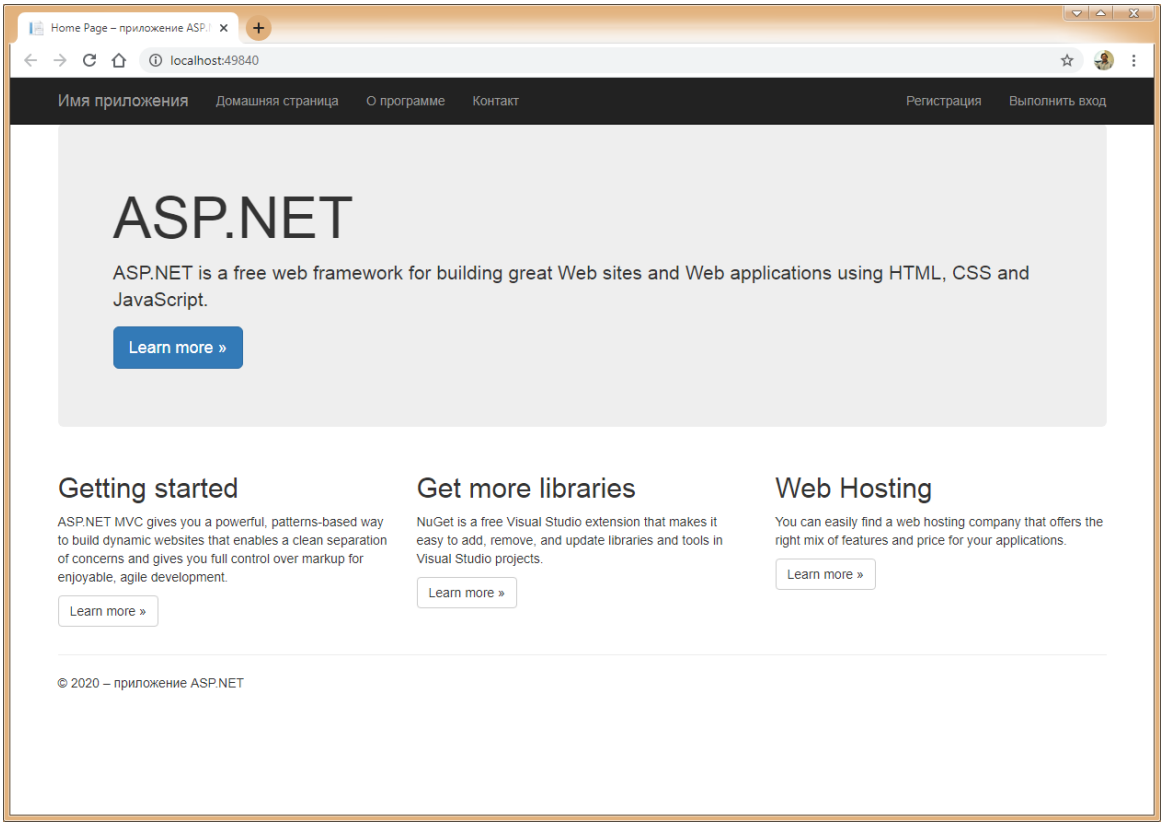
Рис. 4. Сгенерированная структура проекта

В левой части экрана («Обозреватель решений») вы видите дерево проекта, уже содержащее весьма большое количество каталогов и файлов. Подробное описание этой структуры, равно как и самой идеологии MVC будет дано в лекционном курсе по данной учебной дисциплине.

Проведем тестовый запуск проекта, который можно осуществить несколькими способами:

1. Нажать клавишу ***F5***
2. Выбрать пункт меню ***Отладка*** -> ***Начать отладку***
3. Выбрать в верхней панели инструментов кнопку «IIS Express» с указанием браузера, который будет запущен в процессе отладки.

В нашем случае при запуске проекта окно IDE Visual Studio переходит в режим отладки, и стартует браузер Google Chrome (рис. 5). На ваших компьютерах будет подключен браузер по умолчанию, либо вы можете сами выбрать предпочитаемый браузер. Изучите, что может делать приложение в своей исходной конфигурации.



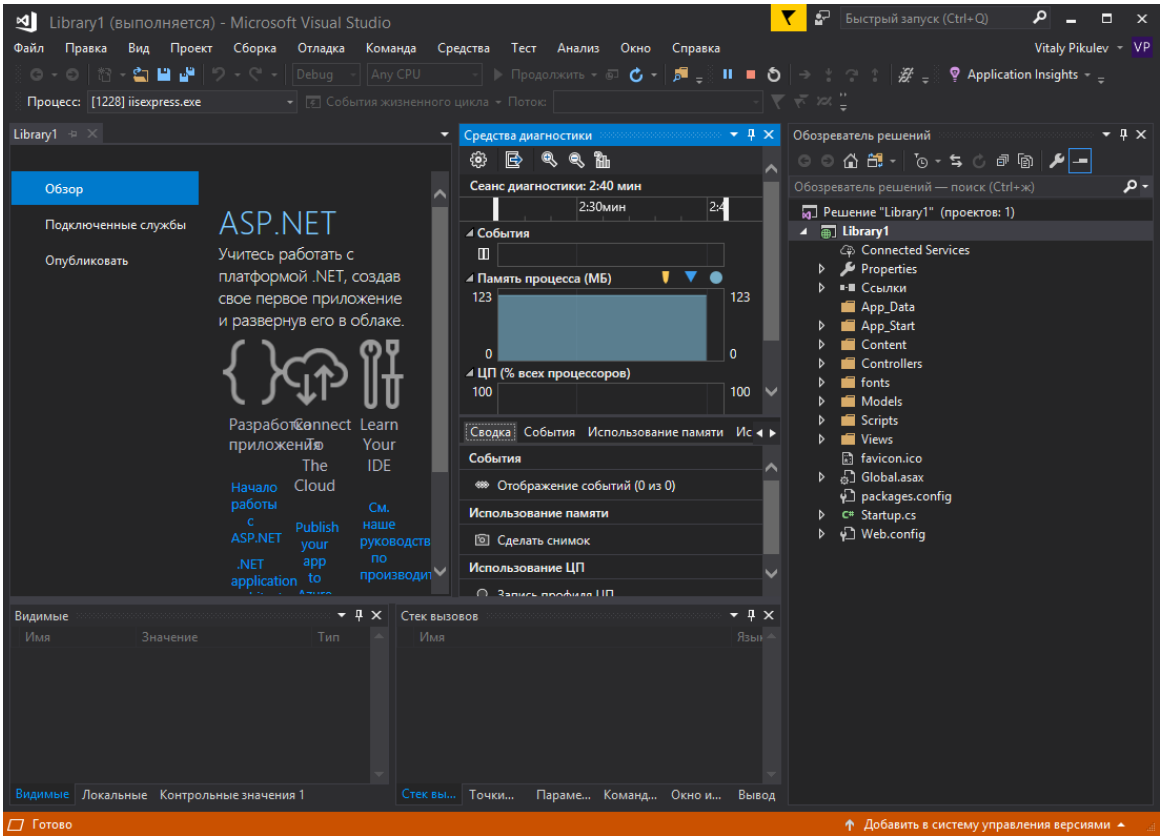


Рис. 5. Внешний вид запущенного проекта

Чтобы начать вносить изменения в проект, необходимо остановить отладку.

Этап 2. Создание классов модели. Технология Code First.

В созданном нами проекте уже существует подключенная к проекту база данных, необходимая для хранения пользовательских аккаунтов. Поэтому создавать новую базу не станем, а будем добавлять необходимые нам таблицы в уже имеющуюся БД.

При создании таблиц будем использовать технологию Code First. То есть, структура базы данных будет создаваться постфактум согласно структуре классов нашего проекта.

Сначала создадим класс *Author*. Для этого в окне «Обозреватель решений» выберем папку *Models*, правой кнопкой мыши вызовем выпадающее меню, в котором выберем пункт ***Добавить → Класс***. Появится диалоговое окно, как на рис. 6.

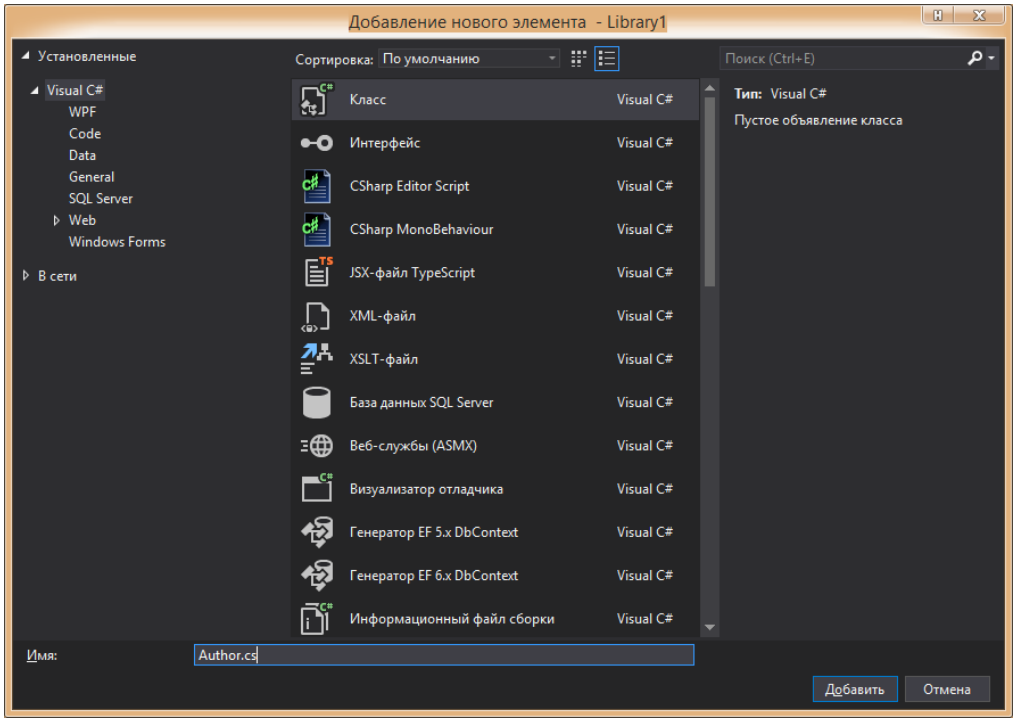


Рис. 6. Создание класса Author.cs

В текст класса *Author* запишем следующее:

```
using System;
using
System.Collections.Generic;
using System.Linq;
using System.Web;

namespace Library1.Models
{
    public class Author
    {
        public int Id { get; set; }
        public string Name { get; set; }
        public string Years { get; set; }
        public ICollection<Book> Books { get; set; }

   public Author()
```

```
            {
                Books = new List<Book>();
            }
        }
    }
```

Мы записали структуру полей и методов класса *Author* в соответствии с принципами ООП, реализованными в языке C#. Важно знать, что данная структура определит поля таблицы *Authors* в базе данных, связанной с проектом. Итератор *ICollection* связывает писателей с книгами (связь «один-ко-многим»), но пока список *Book* пуст.

Аналогичным образом добавим класс *Book* со следующим содержимым:

```
using System;
using
System.Collections.Generic;
using System.Linq;
using System.Web;

namespace Library1.Models
{
    public class Book
    {
        public int Id { get; set; }
        public string Title { get; set; }
        public int? AuthorId { get; set;
}
        public Author Author { get; set;
}
    }
}
```

Здесь у типа поля *AuthorId* стоит вопросительный знак, так как он может принимать значение NULL. Последний метод укажет связь с классом *Author*.

Далее желательно перестроить проект (***Сборка*** -> ***Пересобрать решение***), чтобы подключить к нему новые элементы.

Этап 3. Контроллер и представление. Вывод данных из таблицы базы данных.

Чтобы создать набор страниц для загрузки, редактирования, удаления и просмотра данных, в первую очередь создадим контроллер *AuthorsController*, предназначенный для передачи данных об авторах из БД («от модели») к web-странице (представлению). Для этого в окне «Обозреватель решений» выберем папку *Controllers*, правой кнопкой мыши вызовем выпадающее меню, в котором выберем пункт ***Добавить → Контроллер***. Появится диалоговое окно, как на рис. 7. В нём следует выбрать шаблон для контроллера - «Контроллер MVC 5 с представлениями, использующий Entity Framework». Данный шаблон автоматически сгенерирует методы действия контроллера, а также сформирует представление для каждого из таких действий.

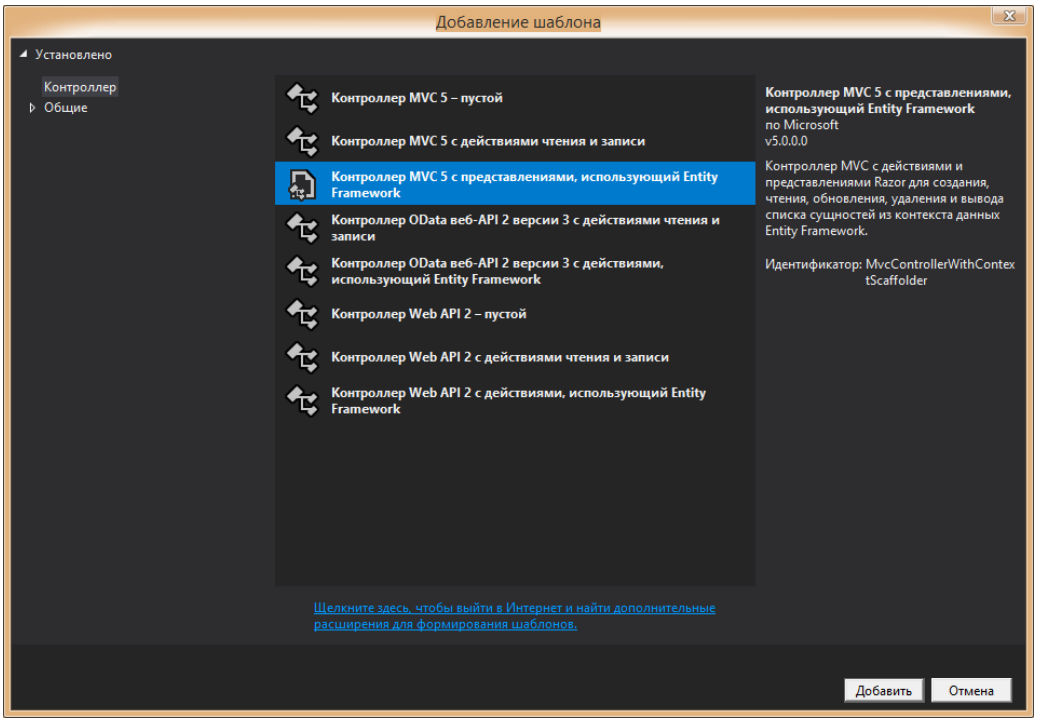


Рис. 7. Добавление шаблона контроллера

В следующем диалоговом окне необходимо выбрать классы модели (Author) и контекста данных (ApplicationDbContext) и ввести имя контроллера (AuthorsController). Следует отметить, что по соглашениям об именовании контроллеров рекомендуется всегда завершать имя словом "Controller". Обязательно включите флажок «Создать представления» (рис. 8). Нажмите кнопку «Добавить».

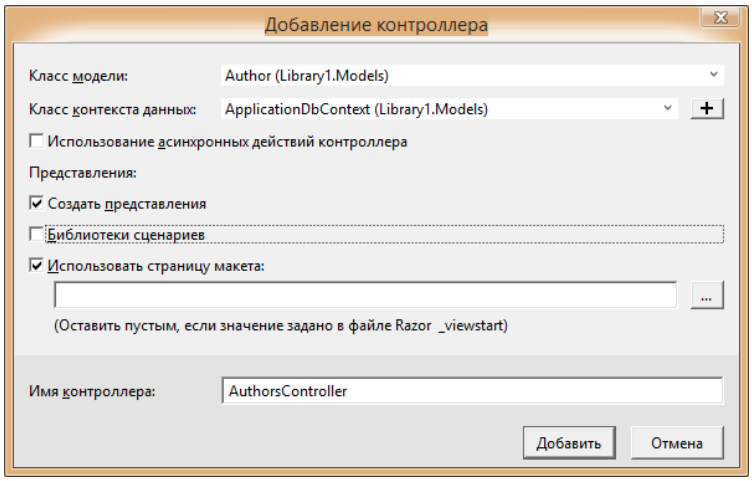


Рис. 8. Добавление контроллера AuthorsController

Visual Studio сгенерировал контроллер *AuthorsController* с настройками по умолчанию, а также папку *Authors* с представлениями для отображения данных (*Index*), редактирования (*Edit*), удаления (*Delete*), просмотра деталей (*Details*) и добавления новых записей (*Create*).

Теперь проведём несколько операций, необходимых для корректной синхронизации базы данных с набором классов проекта, который был изменён нашими предыдущими действиями. Для этого следует провести «миграцию» модели данных следующим образом:

Откройте консоль *NuGet* (***Средства → Диспетчер пакетов NuGet → Консоль диспетчера пакетов***), и в командной строке выполните три следующих команды (рис. 9):

1. **`Enable-Migrations`**
   (выполняется один раз для запуска в проекте механизма миграций, регистр букв имеет значение)
2. **`Add-Migration first`**
   (выполняется всегда, когда возникает ошибка, связанная с необходимостью изменить структуру базы данных, first – любое выбранное вами имя миграции)
3. **`Update-Database`**
   (выполняется всегда вслед за Add-Migration)

Выполнение данных команд не должно приводить к появлению ошибок. В противном случае схема базы данных будет рассогласована со структурой классов, и проект будет неработоспособен.

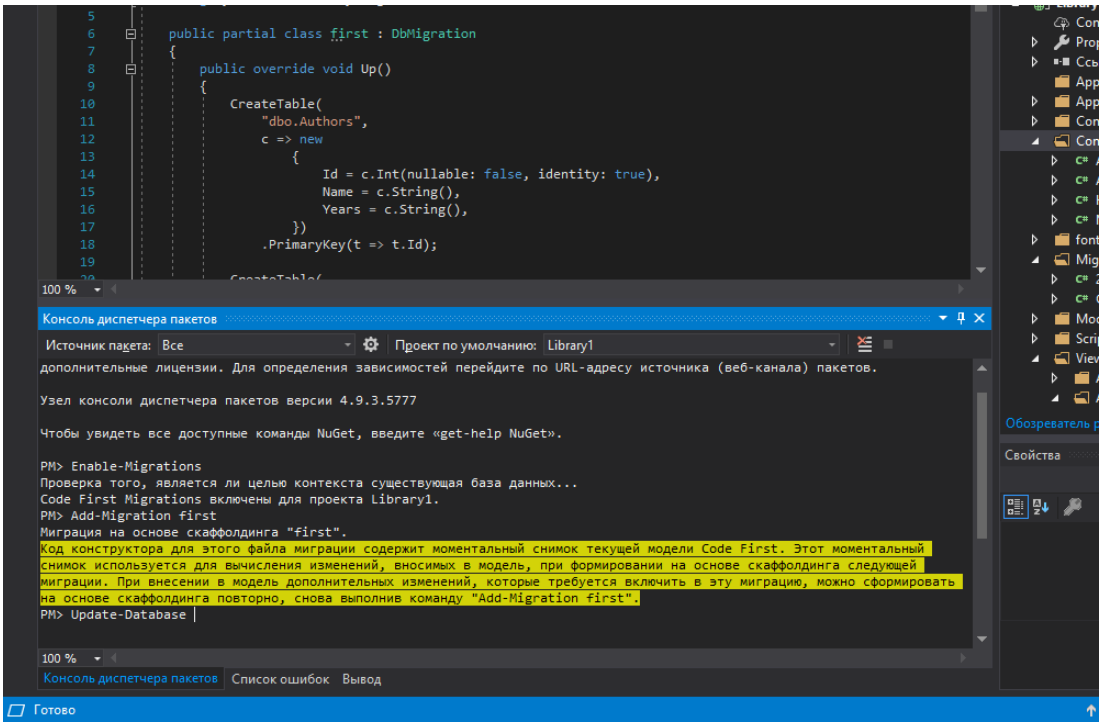


Рис. 9. Консоль *NuGet*

Теперь попробуем запустить приложение, находясь в представлении *Index* контроллера *AuthorsController* (см. рис. 10).

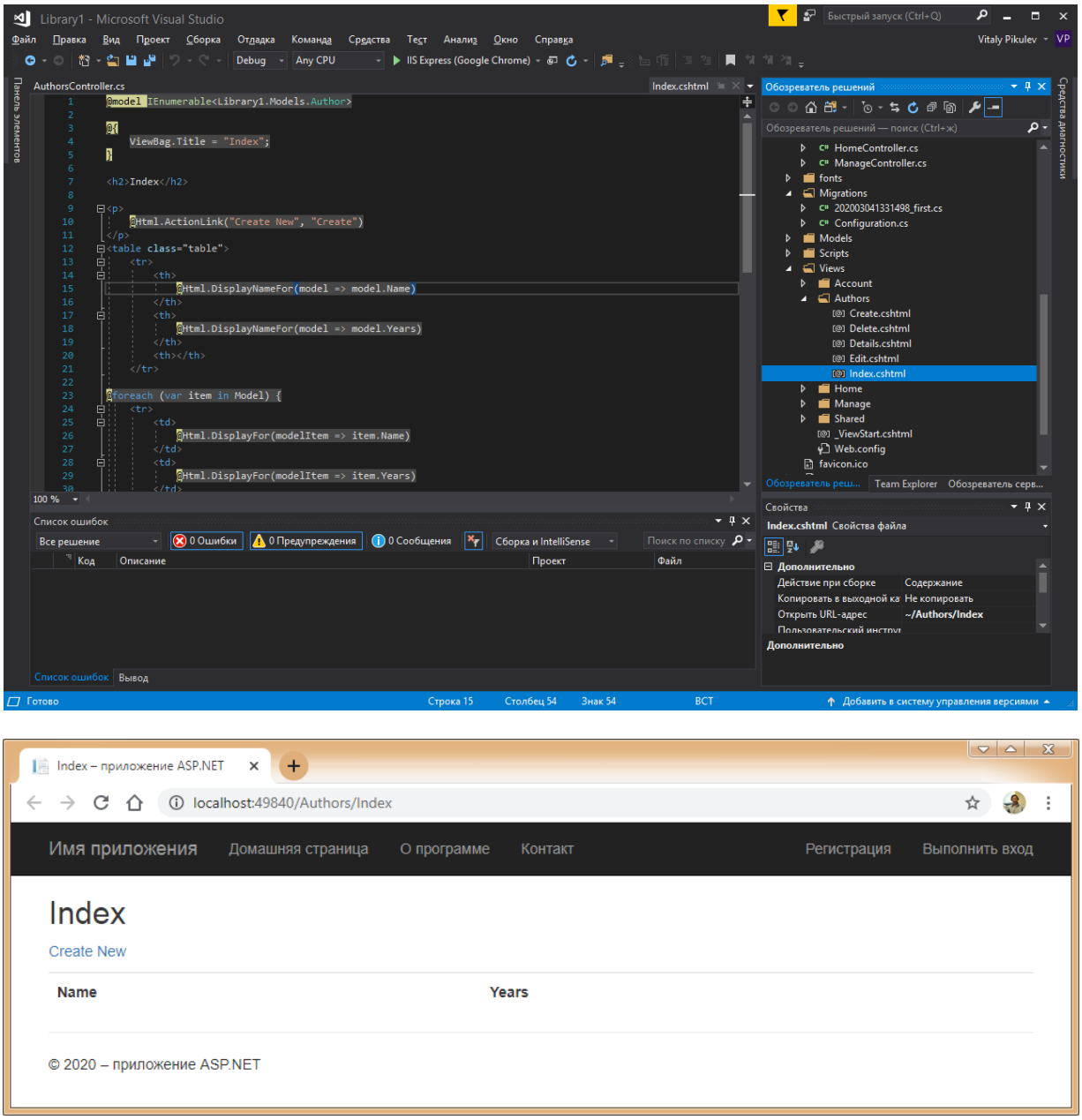


Рис. 13. Представление *Index* контроллера *AuthorsController*

В результате браузер нам открыл страницу, содержащую пустой набор значений, поскольку мы ни одной записи в базу данных не вводили. Давайте введём данные, сначала пользуясь интерфейсом IDE Visual Studio. Для этого необходимо зайти в «Обозреватель серверов», где можно увидеть автоматически созданные для нашего проекта таблицы (рис. 14).

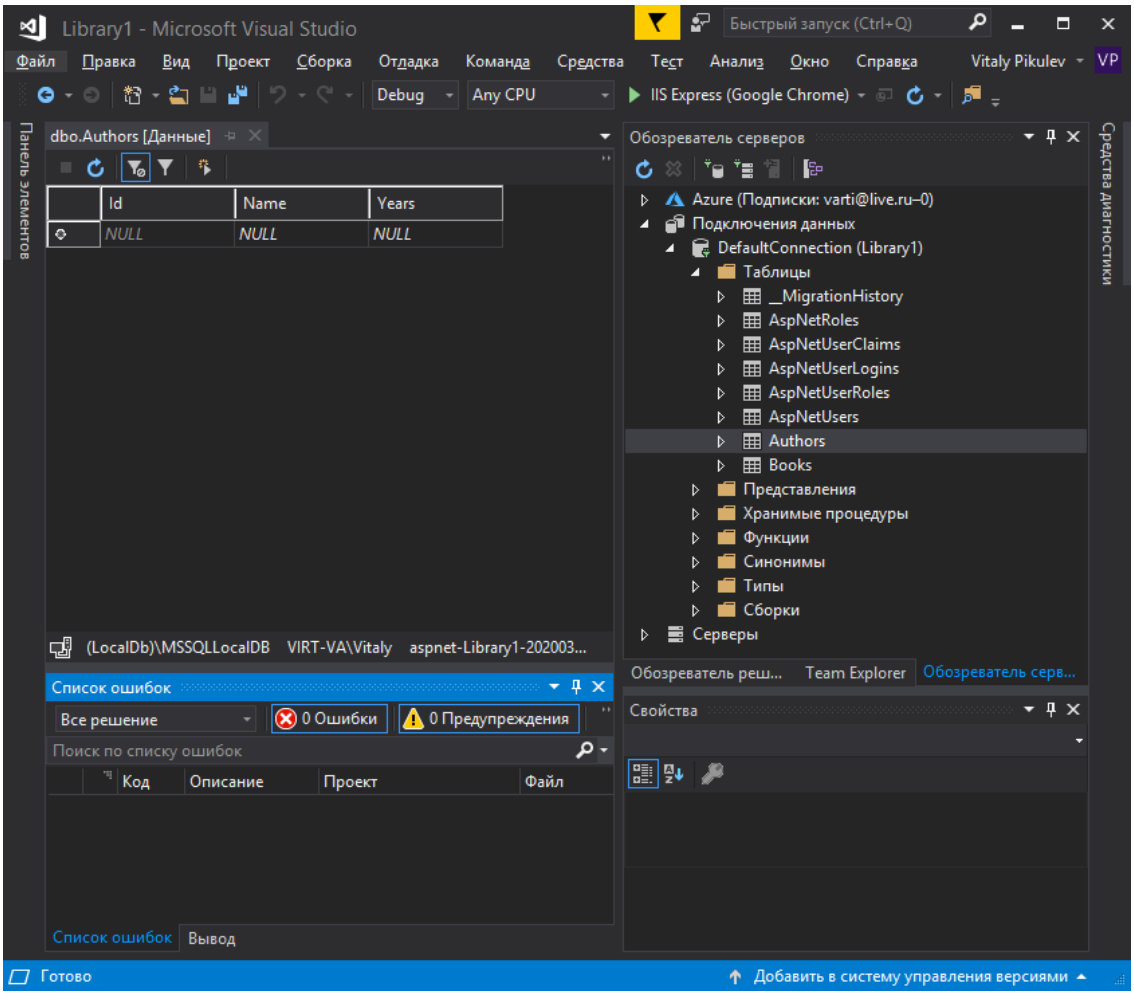


Рис. 14. Обозреватель серверов

Добавляем данные в таблицу Authors. Для этого в окне «Обозреватель серверов» следует щёлкнуть правой кнопкой мыши по таблице и в меню выбрать «Показать таблицу данных» (рис. 15).

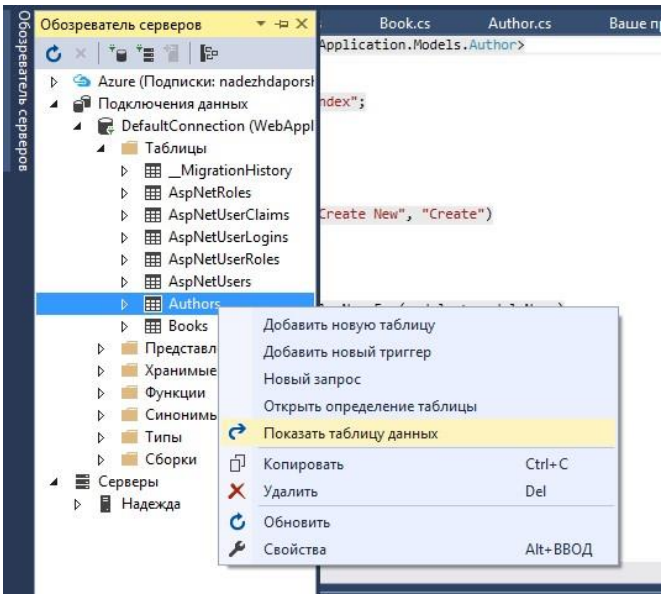


Рис. 15. Выбор таблицы данных

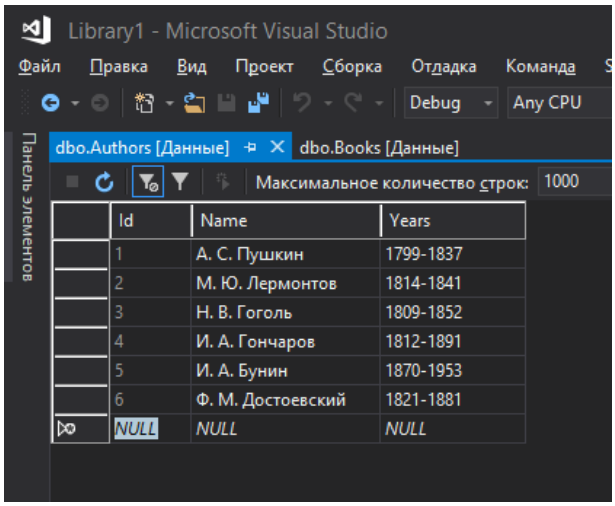


Рис. 16. Добавленные данные в таблицу *Authors*

Аналогичным образом добавим данные в таблицу Books (рис. 17):

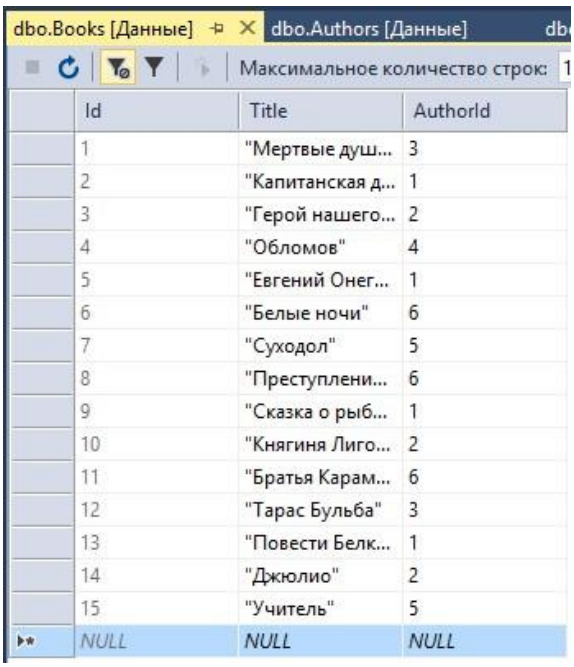


Рис. 17. Добавленные данные в таблицу *Books*

Попробуем запустить проект, также, как и в прошлый раз, находясь в представлении *Index* контроллера *AuthorsController*, и посмотрим, что получилось (Рис. 18).

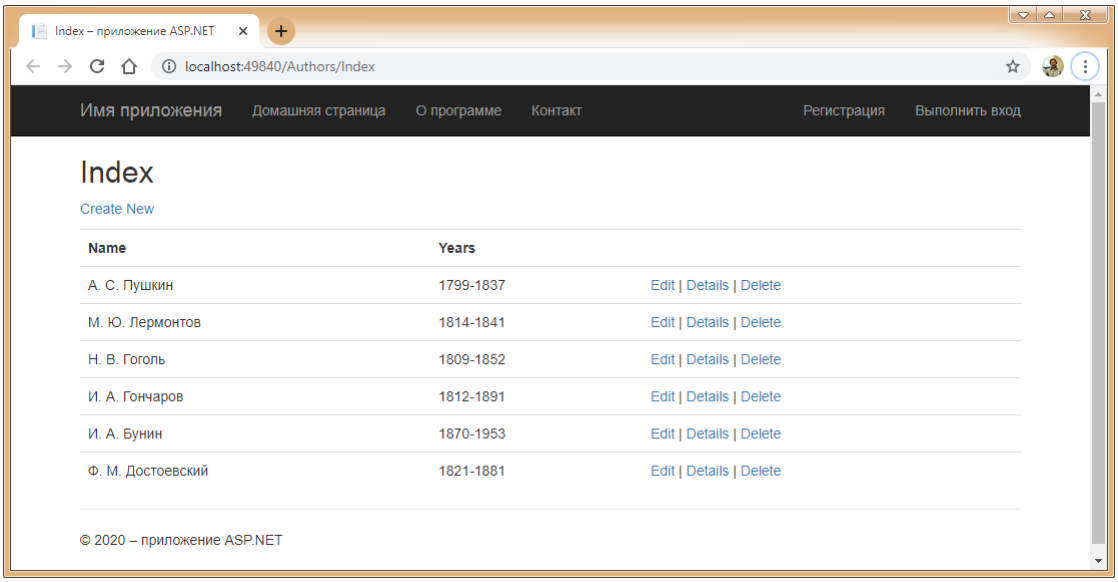


Рис. 18. Внешний вид представления *Index* контроллера *AuthorsController* после ввода данных

Далее займёмся дизайном представления. Попробуем изменить некоторые названия, написанные на латинице, а также удалить со страницы возможность просмотра деталей записей (Details), поскольку для нашего набора данных это совершенно излишне. Вы можете здесь проявить свою творческую фантазию. Простейший вариант подобного редактирования представления *Index* представлен ниже:

```
@model IEnumerable<Library1.Models.Author>

@{
    ViewBag.Title = "Авторы";
}

<h2>Список авторов</h2>

<p>
    @Html.ActionLink("Добавить автора", "Create")
</p>
<table class="table">
    <tr>
        <th style="color: #CCC">
            Имя автора
        </th>
        <th style="color: #CCC">
            Годы жизни
        </th>
        <th></th>
    </tr>

@foreach (var item in Model) {
    <tr>
        <td>
            @Html.DisplayFor(modelItem => item.Name)
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.Years)
        </td>
```

```
        <td>
            @Html.ActionLink("Редактировать", "Edit", new { id=item.Id }) |
            @Html.ActionLink("Удалить", "Delete", new { id=item.Id })
        </td>
    </tr>
}

    </table>
```

Посмотрим на результат работы, запустив приложение повторно (рис. 19):

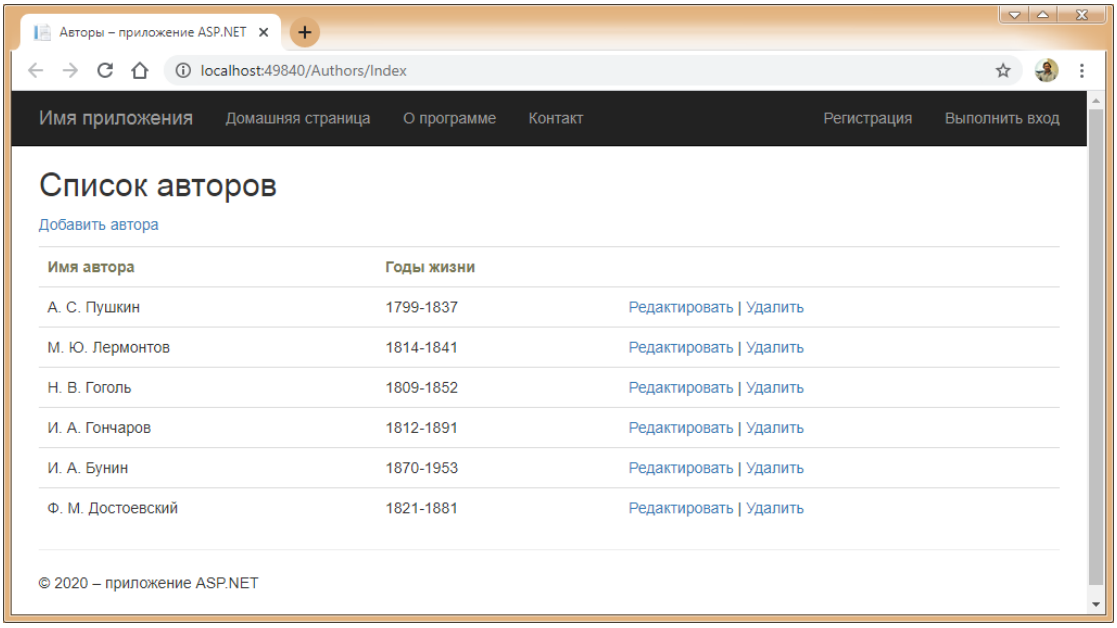


Рис. 19. Отредактированный внешний вид представления *Index* контроллера *AuthorsController*

Потренируемся в добавлении стилевого оформления: преобразуем ссылку «Добавить автора» в кнопку перехода в представление *Create*. Для этого необходимо в представлении *Index* заменить соответствующий код на строку:

```
<a href=@Url.Action("Create", "Authors") class="myButton">Добавить автора</a>
```

Внесение изменений в представление возможно без остановки выполнения проекта, достаточно после внесения изменений обновить страницу в браузере. Пока видимых изменений вы не увидите. Откроем файл стилей проекта (папка *Content*, файл *Site.css*) и пропишем в нём стиль для созданной кнопки:

```
.myButton {
    font-weight: 700;
    color: white;
    text-decoration: none;
    padding: .8em 1em calc(.8em + 3px);
    border-radius: 3px;
    background: rgb(64,199,129);
    box-shadow: 0 -3px rgb(53,167,110) inset;
    transition: 0.2s;
}
    .myBbutton:hover {
        background: rgb(53, 167, 110);
    }
    .myButton:active {
```

```
        background: rgb(33,147,90);
        box-shadow: 0 3px rgb(33,147,90) inset;
      }
```

Должно получиться то, что показано на рис. 20.

Файл *Site.css*, который находится в папке *Content*, предназначен для хранения стилей, используемых во всем проекте. Если после сделанных изменений вы не увидели перемен в стиле вашей кнопки, откройте файл *_Layout.cshtml* из папки *Shared*, который представляет собой шаблон содержания всех страниц проекта и в конце секции <head> пропишите строку:

```
<link href="~/Content/Site.css" rel="stylesheet" />
```

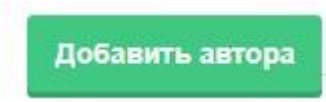


Рис. 20. Результат примененного стилевого решения

Продолжим редактирование файла *Site.css* и добавим стиль для таблицы, содержащей список авторов:

```
table {
    overflow: hidden;
    border: 1px solid #35a76e;
    background: #ffffff;
    width: 70%;
    margin: 5% auto 0;
    border-radius: 5px;
    box-shadow: 0 0 1px rgba(0, 0, 0, 0.2);
}

th, td {
    padding: 18px 28px 18px;
    text-align: center;
}

th {
    padding-top: 22px;
    background: #35a76e;
}

td {
    border-top: 1px solid #e0e0e0;
    border-right: 1px solid #e0e0e0;
}

    td.first, th.first {
        text-align: left
    }

    td.last {
        border-right: none;
    }
```

Посмотрим результат (рис. 21).

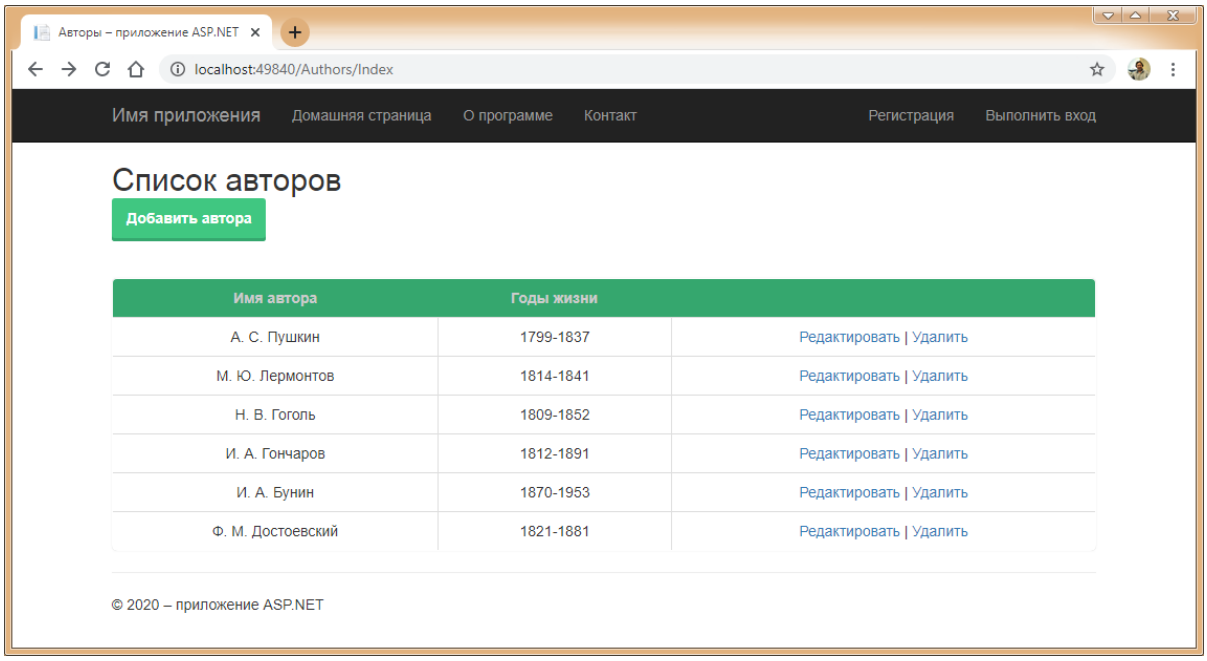


Рис. 21. Отредактированный внешний вид представления Index контроллера AuthorsController

Также предлагаем подредактировать содержимое представления *Create*, чтобы получилось так, как на рис. 22:

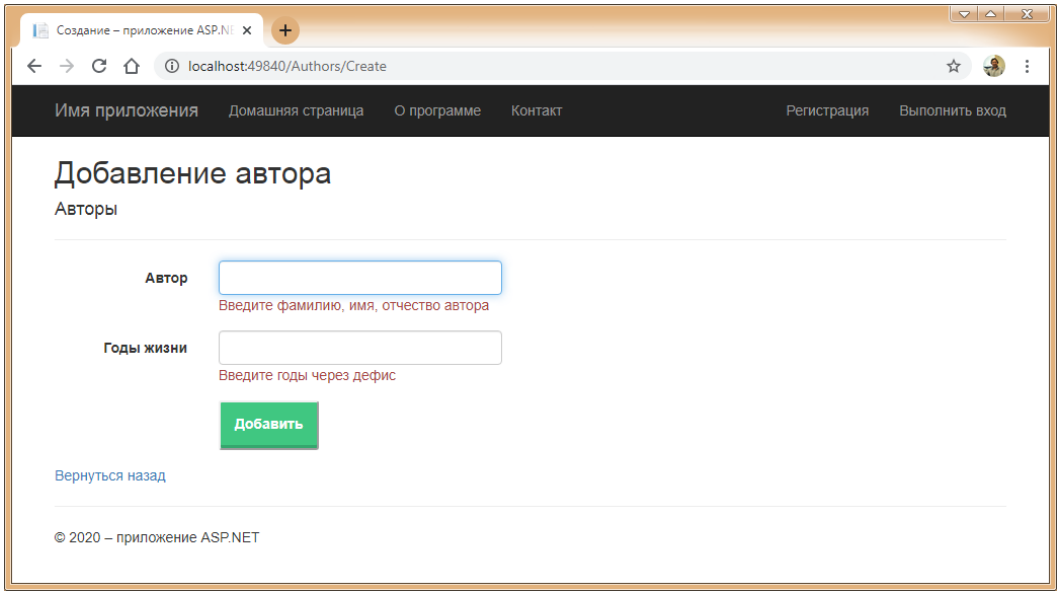


Рис. 22. Внешний вид представления *Create* контроллера *AuthorsController*

Осталось поработать с представлениями *Edit* и *Delete* контроллера *AuthorsController* аналогично *Index* и *Create*. Предлагаем сделать это самостоятельно.

Этап 4. Контроллер и представление. Связь один ко многим.

Создадим контроллер *BooksController*, который будет направлен на вывод информации из БД двух взаимосвязанных таблиц: *Authors* и *Books*. Сделаем это точно так же, как и в предыдущем случае (рис. 23).

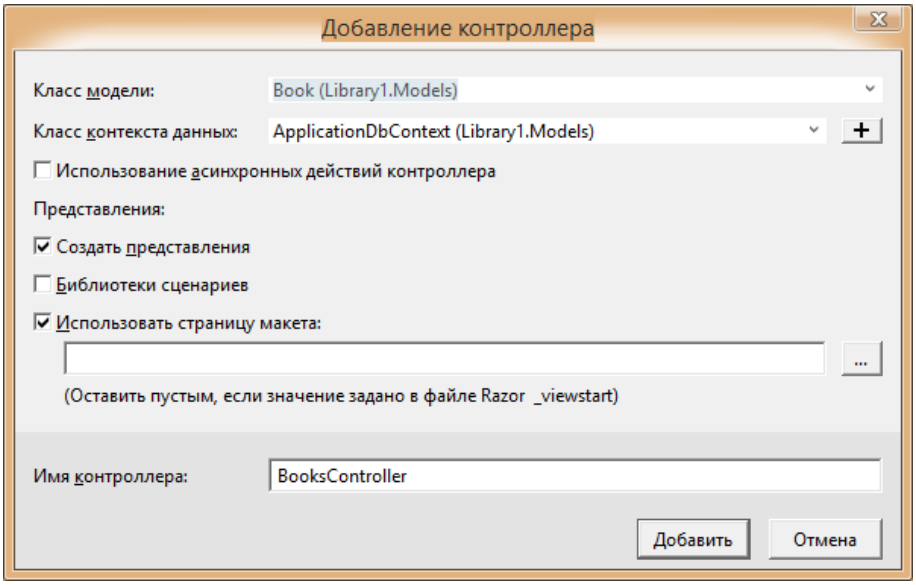


Рис. 23. Добавление контроллера *BooksController*

После создания контроллера запустите проект, открыв представление *Index* контроллера *BooksController*. Как видите, связь между таблицами реализована автоматически. Предлагаем отредактировать созданную форму, приведя её в соответствие с нашим дизайном (рис. 24).

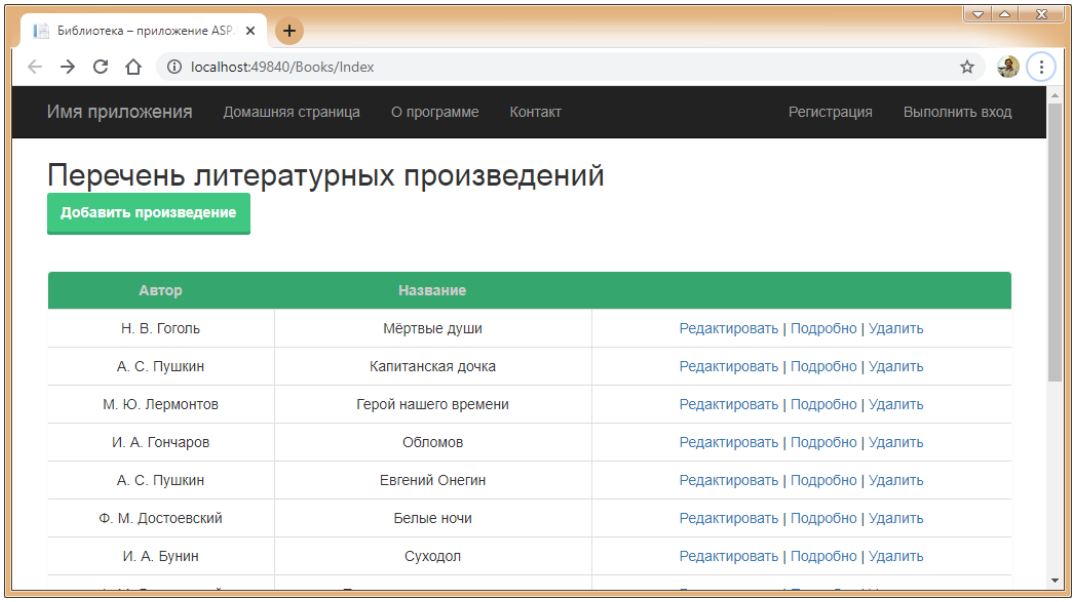


Рис. 24. Внешний вид представления Index контроллера BooksController
со связью таблиц «один ко многим»

Попробуйте ввести новое произведение, воспользовавшись кнопкой «Добавить произведение». Попробуйте удалить либо изменить данные. Всё ли в порядке с формой «Подробно»?

Занятие 5. Меню приложения.

Переход между вкладками меню прописывается в файле *_Layout.cshtml*, который находится в папке *Shared*. Нам необходимо заменить наименования представлений, сформированных Visual Studio, на созданные нами представления. Для этого добавим некоторые строки в файле *_Layout.cshtml* :

```
<li>@Html.ActionLink("Домашняя страница", "Index", "Home")</li>
<li>@Html.ActionLink("О программе", "About", "Home")</li>
<li>@Html.ActionLink("Контакт", "Contact", "Home")</li>
```

На

```
<li>@Html.ActionLink("Главная", "Index", "Home")</li>
<li>@Html.ActionLink("Авторы", "Index", "Authors")</li>
<li>@Html.ActionLink("Список книг", "Index", "Books")</li>
<li>@Html.ActionLink("Контакты", "Contact", "Home")</li>
```

А строку:

```
@Html.ActionLink("Имя приложения", "Index", "Home", new { area = "" }, new {
@class = "navbar-brand" })
```

На

```
@Html.ActionLink("Моя библиотека", "Index", "Home", new { area = "" }, new { @class =
"navbar-brand" })
```

Теперь меню выглядит следующим образом (рис. 25):

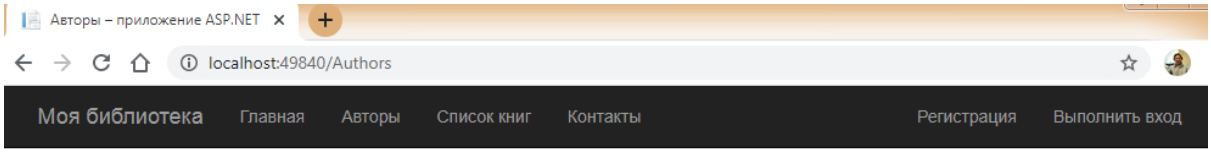


Рис. 25. Меню приложения

Чтобы изменить внешний вид меню, необходимо в файле *Site.css* добавить строки стиля:

```
.navbar {
display: normal;
    background-color: #35a76e;
  }
.navbar-inverse .navbar-brand {
color: white;
}

.navbar-inverse .navbar-nav > li > a {
color: white;
}
.navbar-inverse .navbar-nav > li > a:hover,
.navbar-inverse .navbar-nav > li > a:focus {
color: #000000;
```

```
    background-color: transparent;
}

.navbar-inverse .navbar-brand:hover,
.navbar-inverse .navbar-brand:focus {
color: #000000;
    background-color: transparent;
}
```

Запустим приложение повторно (возможно, потребуется очистка кэша браузера), чтобы посмотреть примененный стиль меню (рис. 26):

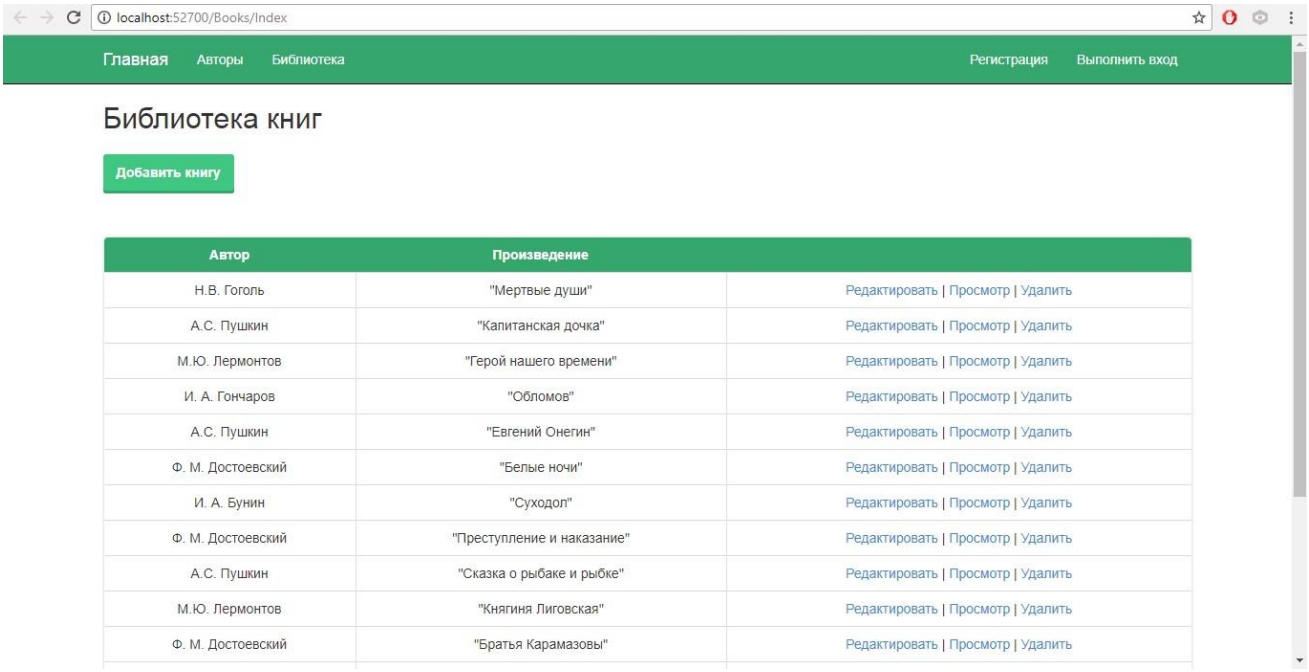


Рис. 26. Отредактированное меню приложения

Этап 6. Главная страница сайта. Добавление изображений.

Для того чтобы добавить картинку на страницу сайта, для начала необходимо поместить ее в папку *Content*:

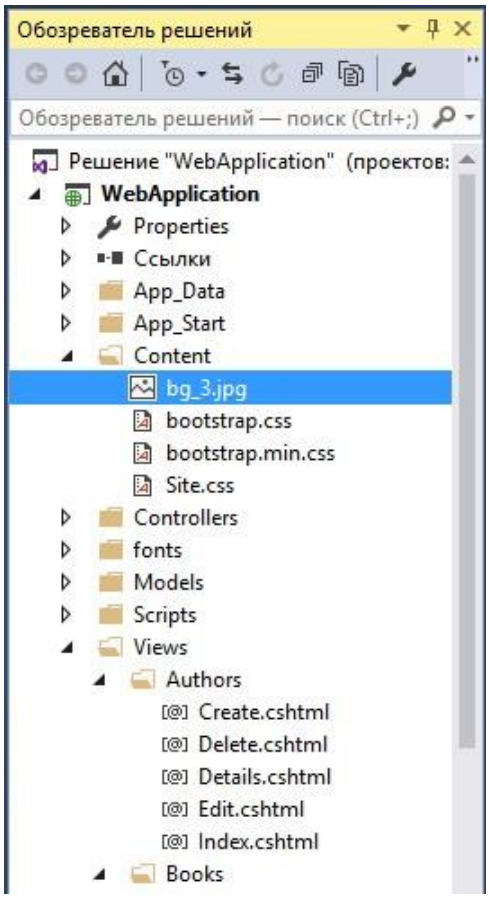


Рис. 27. Добавление картинки в папку *Content*

Если файл не виден в папке, нажмите в обозревателе решений кнопку «Показать все файлы» в верхней строке пиктограмм, щёлкните на появившемся файле картинки правой кнопкой мыши и выберите в выпадающем меню «Добавить в проект». Выполнение проекта при этом должно быть остановлено.

Отредактируйте по своему вкусу главную страницу сайта (представление *Index* контроллера *HomeController*). Для добавления изображение просто перенесите мышкой файл картинки в соответствующее место кода.

```
@model IEnumerable<WebApplication.Models.Book>

@{
    ViewBag.Title = "Библиотека";
} <style>
figure {
        text-align: center;
    }
</style>

<figure>
    <h2>Добро пожаловать в Библиотеку!</h2>
    <img src="~/Content/bg_3.jpg" width="800" height="550">
</figure>
```

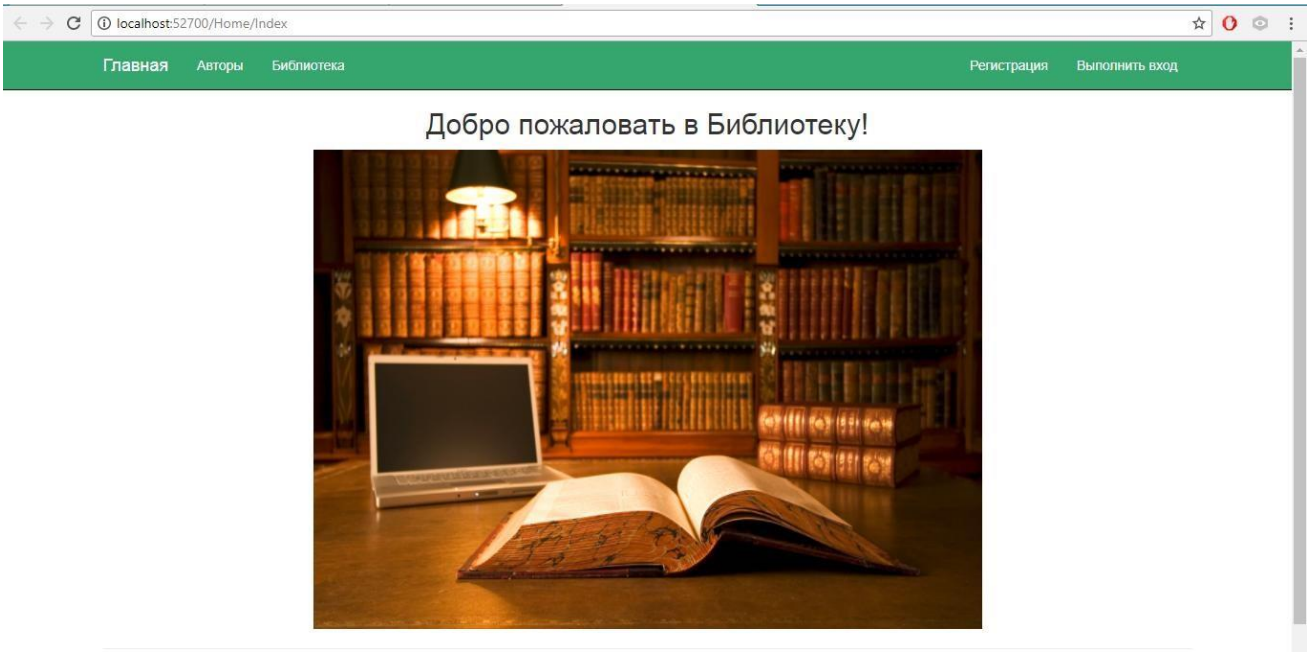


Рис. 28. Стартовая (главная) страница приложения