

Уральский государственный экономический университет
Администрирование информационных систем
Основы WEB программирования
Формализация информации и Big Data

02.03.03 -Математическое обеспечение и администрирование информационных систем,
направленность (профиль) -разработка и администрирование информационных систем

<http://vikchas.ru>

Лабораторная работа.
Тема «Создание работающего проекта
ASP.NET Core MVC с локальной SQL
Server базой данных в Visual Studio 2022»

Часовских Виктор Петрович
доктор технических наук, профессор кафедры
ШИиКМ, ФГБОУ ВО «Уральский государственный
экономический университет

Екатеринбург 2026

В Visual Studio 2022 удобнее использовать [ASP.NET](#) Core MVC на .NET 6/7/8. В руководстве ниже используется современный вариант NET 8.

Необходимо создать простой сайт учета книг.

Функции проекта:

- главная страница;
- раздел “Книги”;
- добавление книги;
- просмотр списка книг;
- редактирование книги;
- удаление книги;
- хранение данных в локальной базе SQL Server;
- использование Entity Framework Core.

Необходимое ПО

1. Visual Studio 2022

2. Рабочая нагрузка:

- [ASP.NET](#) and web development

3. SQL Server Express LocalDB

Обычно устанавливается вместе с Visual Studio.

4. Желательно:

- **SQL Server Management Studio**, сокращенно SSMS

Проверить LocalDB можно через командную строку:

```
sqllocaldb info
```

Если LocalDB установлен, увидите список экземпляров

Создание нового проекта

Откройте **Visual Studio 2022**.

Выберите:

ASP.NET Core Web App (Model-View-Controller)

Project name: StudentBookMvc

Location: выберите папку

Solution name: StudentBookMvc

Нажмите **Далее**.

Настройки:

Framework: .NET 8.0 или .NET 6.0

Authentication type: None

Configure for HTTPS: выключено

Enable Docker: выключено

Нажмите **Далее**

Структура [ASP.NET](#) Core MVC проекта

После создания проекта будут созданы папки:

Controllers

Models

Views

wwwroot

appsettings.json

Program.cs

Кратко:

Папка/файл	Назначение
Controllers	Классы, которые принимают запросы пользователя

Папка/файл	Назначение
Models	Классы данных
Views	HTML-страницы с Razor-синтаксисом
wwwroot	CSS, JavaScript, изображения
appsettings.json	Настройки проекта
Program.cs	Точка запуска и настройка приложения

Установка пакетов Entity Framework Core

Entity Framework Core нужен для работы с базой данных через C#-классы.

Откройте:

Tools -> NuGet Package Manager -> Manage NuGet Packages for Solution

Установите пакеты:

Microsoft.EntityFrameworkCore.SqlServer

Microsoft.EntityFrameworkCore.Tools

Microsoft.EntityFrameworkCore.Design

Или через Package Manager Console:

POWERSHELL

Install-Package Microsoft.EntityFrameworkCore.SqlServer

Install-Package Microsoft.EntityFrameworkCore.Tools

Install-Package Microsoft.EntityFrameworkCore.Design

Если проект на .NET 8, желательно использовать версии пакетов 8.x.

Создание модели данных

В папке **Models** создайте класс:

Book.cs

Код файла Book.cs:

C#

```
using System.ComponentModel.DataAnnotations;
```

```
namespace StudentBookMvc.Models
```

```
{
```

```
    public class Book
```

```
    {
```

```
        public int Id { get; set; }
```

```
        [Required(ErrorMessage = "Введите название книги")]
```

```
        [StringLength(100)]
```

```
        public string Title { get; set; } = string.Empty;
```

```
        [Required(ErrorMessage = "Введите автора")]
```

```
        [StringLength(100)]
```

```
        public string Author { get; set; } = string.Empty;
```

```
        [Range(1, 10000, ErrorMessage = "Количество страниц должно быть  
больше 0")]
```

```
        public int Pages { get; set; }
```

```
        [DataType(DataType.Date)]
```

```
        public DateTime PublishedDate { get; set; }
```

```
    }
```

```
}
```

Что здесь важно:

- Id — первичный ключ;
- Title — название книги;
- Author — автор;
- Pages — количество страниц;
- PublishedDate — дата публикации.

Создание контекста базы данных

Создайте папку:

Data

В ней создайте класс:

ApplicationDbCon .cs

Код:

C#

```
using Microsoft.EntityFrameworkCore;
```

```
using StudentBookMvc.Models;
```

```
namespace StudentBookMvc.Data
```

```
{
```

```
    public class ApplicationDbCon : DbCon
```

```
    {
```

```
        public ApplicationDbCon (DbCon Options<ApplicationDbCon > options)  
            : base(options)
```

```
    {
```

```
    }
```

```
        public DbSet<Book> Books { get; set; }
```

```
}  
}
```

DbCon — это класс, через который приложение работает с базой данных.

Настройка строки подключения

Откройте файл:

appsettings.json

Добавьте строку подключения:

JSON

```
{  
  "ConnectionStrings": {  
    "DefaultConnection":  
    "Server=(localdb)\\MSSQLLocalDB;Database=StudentBookMvcDb;Trusted_Connection=True;MultipleActiveResultSets=true"  
  },  
  
  "Logging": {  
    "LogLevel": {  
      "Default": "Information",  
      "Microsoft.AspNetCore": "Warning"  
    }  
  },  
  
  "AllowedHosts": "*"   
}
```

Пояснение:

Server=(localdb)\\MSSQLLocalDB

означает локальный SQL Server LocalDB.

Database=StudentBookMvcDb

это имя создаваемой базы данных.

Подключение контекста в Program.cs

Откройте файл:

Program.cs

Добавьте пространства имен сверху:

C#

```
using Microsoft.EntityFrameworkCore;
```

```
using StudentBookMvc.Data;
```

Найдите строку:

C#

```
var builder = WebApplication.CreateBuilder(args);
```

Ниже добавьте регистрацию контекста:

C#

```
builder.Services.AddDbContext<ApplicationDbContext>(options =>
```

```
options.UseSqlServer(builder.Configuration.GetConnectionString("DefaultConnection"));
```

Полный пример Program.cs:

C#

```
using Microsoft.EntityFrameworkCore;
```

```
using StudentBookMvc.Data;
```

```
var builder = WebApplication.CreateBuilder(args);
```

```
builder.Services.AddControllersWithViews();
```

```
builder.Services.AddDbContext<ApplicationDbContext>(options =>
```

```
options.UseSqlServer(builder.Configuration.GetConnectionString("DefaultConnection"));
```

```
var app = builder.Build();
```

```
if (!app.Environment.IsDevelopment())
```

```
{
```

```
    app.UseExceptionHandler("/Home/Error");
```

```
    app.UseHsts();
```

```
}
```

```
app.UseHttpsRedirection();
```

```
app.UseStaticFiles();
```

```
app.UseRouting();
```

```
app.UseAuthorization();
```

```
app.MapControllerRoute(
```

```
    name: "default",
```

```
    pattern: "{controller=Home}/{action=Index}/{id?}");
```

```
app.Run();
```

Создание миграции и базы данных

Миграция — это описание изменений в структуре базы данных.

Откройте:

Tools -> NuGet Package Manager -> Package Manager Console

Выполните команду:

POWERSHELL

Add-Migration InitialCreate

Затем создайте базу данных:

POWERSHELL

Update-Database

После успешного выполнения будет создана база:

StudentBookMvcDb

Проверить ее можно через **SQL Server Object Explorer** в Visual Studio:

View -> SQL Server Object Explorer

Или через SSMS.

Создание контроллера BooksController

В папке **Controllers** создайте новый контроллер.

Нажмите правой кнопкой на папку:

Controllers -> Add -> Controller

Выберите:

MVC Controller with views, using Entity Framework

Нажмите **Add**.

Выберите:

Model class: Book

DbCon class: ApplicationDbCon

Controller name: BooksController

Нажмите **Add**.

Visual Studio автоматически создаст:

- контроллер BooksController;
- представления:
 - Index.cshtml;
 - Create.cshtml;
 - Edit.cshtml;
 - Details.cshtml;
 - Delete.cshtml.

Если автоматическая генерация не сработала, можно создать контроллер вручную.

Пример ручного BooksController

Файл:

Controllers/BooksController.cs

Код:

C#

using Microsoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;

using StudentBookMvc.Data;

using StudentBookMvc.Models;

namespace StudentBookMvc.Controllers

```
{
    public class BooksController : Controller
    {
        private readonly ApplicationDbCon _con ;

        public BooksController(ApplicationDbCon con )
        {
            _con = con ;
        }

        public async Task<IActionResult> Index()
        {
            return View(await _con .Books.ToListAsync());
        }

        public async Task<IActionResult> Details(int? id)
        {
            if (id == null)
            {
                return NotFound();
            }

            var book = await _con .Books
                .FirstOrDefaultAsync(m => m.Id == id);

            if (book == null)
            {
                return NotFound();
            }
        }
    }
}
```

```
        return View(book);  
    }
```

```
public IActionResult Create()  
{  
    return View();  
}
```

```
[HttpPost]  
[ValidateAntiForgeryToken]  
public async Task<IActionResult> Create(Book book)  
{  
    if (ModelState.IsValid)  
    {  
        _con .Add(book);  
        await _con .SaveChangesAsync();  
        return RedirectToAction(nameof(Index));  
    }  
  
    return View(book);  
}
```

```
public async Task<IActionResult> Edit(int? id)  
{  
    if (id == null)  
    {  
        return NotFound();  
    }  
}
```

```

var book = await _con .Books.FindAsync(id);

if (book == null)
{
    return NotFound();
}

return View(book);
}

[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> Edit(int id, Book book)
{
    if (id != book.Id)
    {
        return NotFound();
    }

    if (ModelState.IsValid)
    {
        try
        {
            _con .Update(book);
            await _con .SaveChangesAsync();
        }
        catch (DbUpdateConcurrencyException)
        {

```

```

        if (!BookExists(book.Id))
        {
            return NotFound();
        }

        throw;
    }

    return RedirectToAction(nameof(Index));
}

return View(book);
}

public async Task<IActionResult> Delete(int? id)
{
    if (id == null)
    {
        return NotFound();
    }

    var book = await _con .Books
        .FirstOrDefaultAsync(m => m.Id == id);

    if (book == null)
    {
        return NotFound();
    }
}

```

```

        return View(book);
    }

    [HttpPost, ActionName("Delete")]
    [ValidateAntiForgeryToken]
    public async Task<IActionResult> DeleteConfirmed(int id)
    {
        var book = await _con .Books.FindAsync(id);

        if (book != null)
        {
            _con .Books.Remove(book);
            await _con .SaveChangesAsync();
        }

        return RedirectToAction(nameof(Index));
    }

    private bool BookExists(int id)
    {
        return _con .Books.Any(e => e.Id == id);
    }
}

```

Создание представлений вручную

Если Visual Studio не создала представления автоматически, создайте папку:

Views/Books

1 Index.cshtml

Файл:

Views/Books/Index.cshtml

Код:

CSHTML

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

```
@{
```

```
    ViewData["Title"] = "Список книг";
```

```
}
```

```
<h1>Список книг</h1>
```

```
<p>
```

```
    <a asp-action="Create" class="btn btn-primary">Добавить книгу</a>
```

```
</p>
```

```
<table class="table table-bordered table-striped">
```

```
    <thead>
```

```
        <tr>
```

```
            <th>Название</th>
```

```
            <th>Автор</th>
```

```
            <th>Страницы</th>
```

```
            <th>Дата публикации</th>
```

```
            <th></th>
```

```
        </tr>
```

```
    </thead>
```

```

        <tbody>
@foreach (var book in Model)
{
    <tr>
        <td>@book.Title</td>
        <td>@book.Author</td>
        <td>@book.Pages</td>
        <td>@book.PublishedDate.ToShortDateString()</td>
        <td>
            <a asp-action="Edit" asp-route-id="@book.Id">Редактировать</a> |
            <a asp-action="Details" asp-route-id="@book.Id">Подробнее</a> |
            <a asp-action="Delete" asp-route-id="@book.Id">Удалить</a>
        </td>
    </tr>
}
</tbody>
</table>

```

2 Create.cshtml

Файл:

Views/Books/Create.cshtml

Код:

CSHTML

@model StudentBookMvc.Models.Book

```

@{
    ViewData["Title"] = "Добавить книгу";
}

```

```
<h1>Добавить книгу</h1>
```

```
<form asp-action="Create" method="post">
```

```
  <div class="form-group mb-3">
```

```
    <label asp-for="Title" class="control-label">Название</label>
```

```
    <input asp-for="Title" class="form-control" />
```

```
    <span asp-validation-for="Title" class=" -danger"></span>
```

```
  </div>
```

```
  <div class="form-group mb-3">
```

```
    <label asp-for="Author" class="control-label">Автор</label>
```

```
    <input asp-for="Author" class="form-control" />
```

```
    <span asp-validation-for="Author" class=" -danger"></span>
```

```
  </div>
```

```
  <div class="form-group mb-3">
```

```
    <label asp-for="Pages" class="control-label">Количество страниц</label>
```

```
    <input asp-for="Pages" class="form-control" />
```

```
    <span asp-validation-for="Pages" class=" -danger"></span>
```

```
  </div>
```

```
  <div class="form-group mb-3">
```

```
    <label asp-for="PublishedDate" class="control-label">Дата  
публикации</label>
```

```
    <input asp-for="PublishedDate" class="form-control" type="date" />
```

```
    <span asp-validation-for="PublishedDate" class=" -danger"></span>
```

```
  </div>
```

```
<button type="submit" class="btn btn-success">Сохранить</button>
<a asp-action="Index" class="btn btn-secondary">Назад</a>
</form>
```

```
@section Scripts {
    @{await Html.RenderPartialAsync("_ValidationScriptsPartial");}
}
```

3 Edit.cshtml

Файл:

Views/Books/Edit.cshtml

Код:

CSHTML

```
@model StudentBookMvc.Models.Book
```

```
@{
    ViewData["Title"] = "Редактировать книгу";
}
```

```
<h1>Редактировать книгу</h1>
```

```
<form asp-action="Edit" method="post">
```

```
    <input type="hidden" asp-for="Id" />
```

```
    <div class="form-group mb-3">
```

```
        <label asp-for="Title" class="control-label">Название</label>
```

```
        <input asp-for="Title" class="form-control" />
```

```
        <span asp-validation-for="Title" class=" -danger"></span>
```

```
</div>
```

```
<div class="form-group mb-3">
```

```
  <label asp-for="Author" class="control-label">Автор</label>
```

```
  <input asp-for="Author" class="form-control" />
```

```
  <span asp-validation-for="Author" class=" -danger"></span>
```

```
</div>
```

```
<div class="form-group mb-3">
```

```
  <label asp-for="Pages" class="control-label">Количество страниц</label>
```

```
  <input asp-for="Pages" class="form-control" />
```

```
  <span asp-validation-for="Pages" class=" -danger"></span>
```

```
</div>
```

```
<div class="form-group mb-3">
```

```
  <label asp-for="PublishedDate" class="control-label">Дата  
публикации</label>
```

```
  <input asp-for="PublishedDate" class="form-control" type="date" />
```

```
  <span asp-validation-for="PublishedDate" class=" -danger"></span>
```

```
</div>
```

```
<button type="submit" class="btn btn-success">Сохранить</button>
```

```
<a asp-action="Index" class="btn btn-secondary">Назад</a>
```

```
</form>
```

```
@section Scripts {
```

```
  @{await Html.RenderPartialAsync("_ValidationScriptsPartial");}
```

```
}
```

4 Details.cshtml

Файл:

Views/Books/Details.cshtml

Код:

CSHTML

@model StudentBookMvc.Models.Book

@{

 ViewData["Title"] = "Информация о книге";

}

<h1>Информация о книге</h1>

<div>

 <dl class="row">

 <dt class="col-sm-3">Название</dt>

 <dd class="col-sm-9">@Model.Title</dd>

 <dt class="col-sm-3">Автор</dt>

 <dd class="col-sm-9">@Model.Author</dd>

 <dt class="col-sm-3">Страницы</dt>

 <dd class="col-sm-9">@Model.Pages</dd>

 <dt class="col-sm-3">Дата публикации</dt>

 <dd class="col-sm-9">@Model.PublishedDate.ToShortDateString()</dd>

 </dl>

</div>

```
<a asp-action="Edit" asp-route-id="@Model.Id" class="btn btn-warning">Редактировать</a>
```

```
<a asp-action="Index" class="btn btn-secondary">Назад</a>
```

5 Delete.cshtml

Файл:

Views/Books/Delete.cshtml

Код:

CSHTML

```
@model StudentBookMvc.Models.Book
```

```
@{  
    ViewData["Title"] = "Удалить книгу";  
}
```

```
<h1>Удалить книгу</h1>
```

```
<h3>Вы действительно хотите удалить эту книгу?</h3>
```

```
<div>
```

```
    <dl class="row">
```

```
        <dt class="col-sm-3">Название</dt>
```

```
        <dd class="col-sm-9">@Model.Title</dd>
```

```
        <dt class="col-sm-3">Автор</dt>
```

```
        <dd class="col-sm-9">@Model.Author</dd>
```

```

<dt class="col-sm-3">Страницы</dt>
<dd class="col-sm-9">@Model.Pages</dd>

<dt class="col-sm-3">Дата публикации</dt>
<dd class="col-sm-9">@Model.PublishedDate.ToShortDateString()</dd>
</dl>

<form asp-action="Delete" method="post">
  <input type="hidden" asp-for="Id" />
  <button type="submit" class="btn btn-danger">Удалить</button>
  <a asp-action="Index" class="btn btn-secondary">Отмена</a>
</form>
</div>

```

Добавление ссылки в меню сайта

Откройте файл:

Views/Shared/_Layout.cshtml

Найдите меню, где есть ссылки Home и Privacy.

Добавьте ссылку на книги:

CSHTML

```

<li class="nav-item">
  <a class="nav-link -dark" asp-area="" asp-controller="Books" asp-
  action="Index">Книги</a>
</li>

```

Пример фрагмента меню:

CSHTML

```

<ul class="navbar-nav flex-grow-1">
  <li class="nav-item">

```

```
<a class="nav-link -dark" asp-area="" asp-controller="Home" asp-  
action="Index">Главная</a>  
  
</li>  
  
<li class="nav-item">  
  
  <a class="nav-link -dark" asp-area="" asp-controller="Books" asp-  
action="Index">Книги</a>  
  
  </li>  
  
</ul>
```

Запуск проекта

Нажмите:

Ctrl + F5

или кнопку:

Start Without Debugging

Откройте в браузере:

<https://localhost:xxxx/Books>

Вы должны увидеть страницу списка книг.

Нажмите:

Добавить книгу

Введите данные, например:

Название: Война и мир

Автор: Лев Толстой

Страницы: 1225

Дата публикации: 1869-01-01

Нажмите **Сохранить**.

Если запись появилась в списке — проект работает.

Проверка базы данных

В Visual Studio откройте:

View -> SQL Server Object Explorer

Далее:

SQL Server -> (localdb)\MSSQLLocalDB -> Databases -> StudentBookMvcDb -> Tables

Вы должны увидеть таблицу:

dbo.Books

Можно посмотреть данные:

dbo.Books -> View Data

Частые ошибки и решения

Ошибка 1: Add-Migration не распознается

Текст ошибки может быть похож на:

The term 'Add-Migration' is not recognized

Решение:

Установите пакет:

POWERSHELL

Install-Package Microsoft.EntityFrameworkCore.Tools

Проверьте, что в Package Manager Console выбран правильный проект в списке **Default project**.

Ошибка 2: Не найден ApplicationDbContext

Проверьте:

1. Есть ли файл ApplicationDbContext .cs.
2. Правильный ли namespace.
3. Добавлены ли using в Program.cs:

C#

```
using StudentBookMvc.Data;
```

```
using Microsoft.EntityFrameworkCore;
```

Ошибка 3: Не создается база данных

Проверьте строку подключения:

JSON

"DefaultConnection":

"Server=(localdb)\\MSSQLLocalDB;Database=StudentBookMvcDb;Trusted_Connection=True;MultipleActiveResultSets=true"

Затем выполните:

POWERSHELL

Update-Database

Ошибка 4: Ошибка сертификата HTTPS

Для разработки можно выполнить:

BASH

```
dotnet dev-certs https --trust
```

Или временно запускать проект через HTTP-профиль.

Ошибка 5: Ошибка Scaffolded controller

Если автоматическое создание контроллера с представлениями не работает:

1. Проверьте, установлены ли пакеты EF Core.

2. Соберите проект:

Build -> Build Solution

3. Попробуйте снова создать контроллер.

4. Если не помогло — создайте контроллер и Views вручную по примерам выше.

Минимальная схема работы MVC

Model

Модель описывает данные:

C#

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

Controller

Контроллер обрабатывает запрос:

C#

```
public IActionResult Index()
{
    return View();
}
```

View

Представление отображает HTML:

CSHTML

```
<h1>Список книг</h1>
```

Как это связано с SQL

Создадим C#-класс:

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

Entity Framework Core создает таблицу:

SQL

Books

Примерно такую:

SQL

CREATE TABLE Books

```
(
    Id int PRIMARY KEY IDENTITY,
    Title nvarchar(100),
    Author nvarchar(100),
    Pages int,
    PublishedDate datetime2
);
```

Когда вы пишете:

C#

```
_con .Books.Add(book);
await _con .SaveChangesAsync();
```

Entity Framework Core выполняет SQL-запрос на добавление записи.

Практическое задание

Создайте сайт учета книг.

Обязательные требования:

1. Проект [ASP.NET](#) Core MVC.
2. Использование SQL Server LocalDB.
3. Использование Entity Framework Core.
4. Таблица Books.
5. Возможность:
 - добавить книгу;
 - просмотреть список;
 - посмотреть детали;
 - изменить книгу;
 - удалить книгу.
6. На сайте должна быть ссылка “Книги” в меню.

Дополнительное задание:

Добавьте поле:

C#

```
public string Genre { get; set; } = string.Empty;
```

После этого выполните:

POWERSHELL

Add-Migration AddGenreToBook

Update-Database

И добавьте поле Genre на страницы Create, Edit, Details и Index.

Итог

В результате создали полноценный учебный MVC-проект:

- сайт на [ASP.NET](#) Core MVC;
- модель данных Book;

- контекст базы данных ApplicationDbCon ;
- локальную SQL Server базу;
- миграции Entity Framework Core;
- CRUD-операции:
 - ✓ Create;
 - ✓ Read;
 - ✓ Update;
 - ✓ Delete.

Такой проект можно использовать как основу для более сложных систем:

- учет студентов;
- каталог товаров;
- библиотека;
- склад;
- расписание занятий;
- CRM-система;
- интернет-магазин.