**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ «ГРОДНЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ ЯНКИ КУПАЛЫ**

**ФАКУЛЬТЕТ ЭКОНОМИКИ И УПРАВЛЕНИЯ**

**Кафедра математического и информационного обеспечения экономических систем**

Курсовой проект по учебной дисциплине «Объектно-ориентированное проектирование и программирование»

**Информационная система «Библиотека» с применением технологии ASP.NET Core**

**Специальность «**Информационные системы и технологии (в экономике)»

**Автор**

Студентка 2 курса Э.В. Подгайская

**Руководитель**

Доцент И.И. Игнатенко

Гродно, 2023

**РЕФЕРАТ**

**Информационная система «Библиотека»**

Работа содержит 20 страниц, 5 рисунков, 2 использованных источников литературы.

Перечень ключевых слов: БИБЛИОТЕКА, БАЗА ДАННЫХ, ПРОГРАММИРОВАНИЕ, ПРОГРАММА, СИСТЕМА.

Объектом работы являются потенциальные пользователи приложения по просмотру книг.

Предметом курсовой работы служит среда разработки Visual Studio.

Цель данной курсовой работы: разработка приложения на языке программирования C# с использованием технологии ASP.NET Core в среде разработки Visual Studio.

Основные результаты курсовой работы:

1. Разработан интерфейс программы, понятный для пользователя.
2. Спроектирована база данных для работы с продуктами.
3. Разработан набор классов для работы с программой.
4. Написан код программы на языке C#.
5. Выполнены тесты для проверки работоспособности программы.

Работа состоит из введения, 3 глав и списка использованных источников.

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 3](#_Toc150950089)

[1 ПОСТАНОВКА ЗАДАЧИ 4](#_Toc150950090)

[2 ПРОЕКТИРОВАНИЕ СИСТЕМЫ 5](#_Toc150950091)

[2.1 Выбор технологий программирования 5](#_Toc150950092)

[2.2 Классы 6](#_Toc150950093)

[2.3 Представления](#_Toc150950094) 7

2.4 Контроллеры………………………………………………………………...8

[3 ПРОГРММНАЯ РЕАЛИЗАЦИЯ](#_Toc150950095) 10

[3.1 Разработка базы данных 10](#_Toc150950096)

[3.2 Аунтетификация и авторизация](#_Toc150950097) 12

3.3 Тестирование приложения………………………………………………...13

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ](#_Toc150950099) 14

ПРИЛОЖЕНИЕ………………………………………………………………….15

# **ВВЕДЕНИЕ**

В современном информационном обществе доступ к знаниям и культурным ценностям является неотъемлемым фактором прогресса и развития. В этом контексте библиотеки, как уникальные учреждения, играют важную роль в сохранении, организации и обеспечении доступа к разнообразным формам литературы и информации.

В рамках курсовой работы предлагается разработка современной информационной системы, которая будет способствовать повышению производительности библиотеки. Разрабатываемое веб-приложение под названием "Библиотека" предоставит эффективные инструменты для управления и обслуживания библиотечных ресурсов. Оно позволит библиотекарям эффективно вести учет библиотечного фонда, а пользователям - просматривать каталог книг.

Проектируемая информационная система будет полезной для развития библиотечного дела и популяризации чтения. Она предоставит библиотекарям удобный и надежный инструмент для управления ресурсами, что позволит им эффективно организовать работу и обеспечить максимальное удобство для пользователей. Кроме того, привлекательный и интуитивно понятный интерфейс приложения позволит пользователям легко находить интересующие их книги и получать всю необходимую информацию о них.

Данное веб-приложение "Библиотека" будет способствовать автоматизации процессов, которые ранее требовали значительных усилий и времени. Оно поможет библиотекарям в эффективном ведении учета книг, а также обеспечит пользователям простой и удобный способ поиска и доступа к необходимой информации.

Разработка современной информационной системы для библиотеки является важным шагом в развитии библиотечного дела. Она поможет улучшить доступность и эффективность работы библиотеки, а также повысить интерес пользователей к чтению и образованию.

**1 ПОСТАНОВКА ЗАДАЧИ**

**Целью** **курсовой работы** является повышение производительности библиотеки путём проектирования и разработки информационной системы для библиотеки.

**Основными задачами курсовой работы** являются:

* Изучение технологии программирования.
* Проектирование базы данных.
* Создание информационной системы.

**Основные задачи информационной системы**:

* Уменьшение времени выполнения задач библиотеки.
* Исправление ошибок при заполнении данных.

Информационная система должна выполнять следующие **функции**:

* Добавление, редактирование, удаление новых книг/авторов.
* Просмотр пользователем каталога книг.
* Авторизацию и аутентификацию.

**2 ПРОЕКТИРОВАНИЕ СИСТЕМЫ**

**2.1 Выбор технологии программирования**

**ASP.NET Core** (Active Server Pages для .NET) — кроссплатформенная технология для создания веб-приложений на платформе .NET, развиваемая компанией Microsoft. Частью данной платформы является фреймворк **ASP.NET Core MVC**, отличительной особенностью которого является применения **паттерна MVC**.

Концепция паттерна MVC предполагает разделение приложения на три компонента:

* 1. Модель (Model) описывает используемые в приложении данные, методы работы с ними и структуру программы. Она реагирует на команды из контроллера и выдаёт информацию и/или изменяет своё состояние. Она передаёт данные в представление.
  2. Представление (View). Данный компонент визуализирует информацию, которую он получает от модели. View отображает данные на уровне пользовательского интерфейса. Например, в виде таблицы или списка. Представление определяет внешний вид приложения и способы взаимодействия с ним.
  3. Контроллер (Controller) обеспечивает взаимодействие с системой: обрабатывает действия пользователя, проверяет полученную информацию и передает ее модели. Контроллер определяет, как приложение будет реагировать на действия пользователя. Также контроллер может отвечать за фильтрацию данных и авторизацию.

Контроллер

Входящий HTTP-запрос

Модель

Если необходимо обращение к модели

Представление

Передача данных из модели

Генерация нужного представления

**2.2 Классы**

Выполнение приложения начинается с класса **Program.cs**. Приложение в ASP.NET Core представляет собой объект **Microsoft.AspNetCore.Builder.WebApplication**. Этот объект настраивает всю *конфигурацию* приложения (настройки приложения, которые можно изменить, не переделывая полностью код), его *маршруты*, *используемые зависимости* (классы получают объекты из базы данных), *обработку запросов* (приложение получает запрос от клиента и выполняет соответствующие действия для удовлетворения этого запроса). Для создания объекта **WebApplication** необходим специальный класс-строитель — **WebApplicationBuilder**. И в файле **Program.cs** вначале создается данный объект с помощью статического метода **WebApplication.CreateBuilder**. Получив объект **WebApplicationBuilder**, у него вызывается метод **Build()**, который собственно и создает объект **WebApplication**. Для запуска приложения используется метод **Run()**.

В папке *Models* созданы два класса: **Book.cs** и **Author.cs**. Класс **Author.cs** представляет автора и содержит следующие свойства:

1. ***Id*** (тип int) — уникальный идентификатор автора. Он помечен атрибутом **[Key]**, что указывает на то, что это первичный ключ в базе данных.
2. ***Name*** (тип string) — имя автора. Оно помечено атрибутом **[DisplayName("Author")]**, который задает отображаемое имя для этого свойства в пользовательском интерфейсе.

Класс **Book.cs** представляет модель книги. В классе определены следующие свойства и атрибуты:

1. ***Id*** (тип int) — уникальный идентификатор книги. Атрибут **[Key]** указывает, что свойство ***Id*** является первичным ключом для таблицы в базе данных.
2. ***Title*** (тип string) — заголовок книги.
3. ***Genre*** (тип string) — жанр книги.
4. ***Author*** (тип string) — имя автора книги.
5. ***AuthorId*** (тип int) — идентификатор автора. Атрибут **[ForeignKey("Authors")]** указывает, что свойство ***AuthorId*** является внешним ключом, связывающим класс **Book.cs** с классом **Author.cs**. Это означает, что в базе данных будет создана связь между таблицами **dbo.Book** и **dbo.Author** на основе этого внешнего ключа.
6. ***Authors*** (тип Author) — виртуальное свойство, представляющее автора книги. Оно используется для доступа к объекту автора, связанному с данной книгой.

**2.3 Представления**

В ASP.NET Core представления – это файлы с расширением cshtml, которые содержат код пользовательского интерфейса в основном на языке **html**, а также конструкторы **Razor** – специального движка представлений, который позволяет переходить от кода html к коду на языке C#.

Для каждого контроллера в проекте создается подкаталог в папке **Views**, который называется по имени контроллера и который хранит представления, используемые методами данного контроллера. Так, по умолчанию имеется контроллер **HomeController** и для него в папке **Views** есть подкаталог **Home** с представлениями для методов контроллера **HomeController**.

В папке **Views** нужно создать два подкаталога **Book** и **Author**.с такими представлениями как **Create**, **Delete**, **Details**, **Edit**, **Index**.

В начале каждого представления нужно указать на тип модели, например, @model MyLibrary.Models.Author указывает на тип представления Author.

Для валидации формы прописываем в **Create** и **Edit**:

@{await Html.RenderPartialAsync("\_ValidationScriptsPartial");

В представлениях используются HTML-теги в сочетании с классами Bootstrap, чтобы создать стилизованную форму визуального редактирования автора. Bootstrap - это популярный фреймворк CSS, который предоставляет набор готовых стилей и компонентов для создания привлекательного пользовательского интерфейса. Внутри HTML-тегов формы (<form>) и других элементов (<div>, <label>, <input>, <span>, <a>) использованы классы Bootstrap, такие как form-group, form-control, text-danger, btn, mt-2, которые определяют различные аспекты внешнего вида и поведения элементов.

Чтобы применить пользовательские стили и использовать CSS-файлы в представлении, необходимо создать файл с расширением .css и указать путь к этому файлу: <link rel="stylesheet" href="~/@(nameof(MyLibrary)).styles.css" />

В папке **Shared** хранятся общие представления для всех контроллеров. По умолчанию это файлы **\_Layout.cshtml** (используется в качестве мастер-страницы), **Error.cshtml** (использутся для отображения ошибок) и **\_ValidationScripsPartial.cshtml** (частичное представление, которое подключает скрипты валидации формы), **\_LoginPartial.cshtml** (используется для логгирования).

**2.4 Контроллеры**

Контроллеры **AuthorController** и **BookController** используют ApplicationDbContext для взаимодействия с базой данных с помощью Entity Framework Core. Они содержит методы, которые обрабатывают различные HTTP-запросы и выполняют соответствующие операции с данными авторов.

В коде контроллеров используются некоторые **атрибуты**, которые предоставляют дополнительную информацию и указывают поведение методов контроллеров. **[HttpPost]** указывает, что метод контроллера должен обрабатывать только HTTP POST запросы. В контексте CRUD операций, методы, помеченные этим атрибутом, обычно используются для создания или обновления данных. Атрибут **[HttpGet]** указывает, что метод контроллера должен обрабатывать только HTTP GET запросы. Методы с этим атрибутом, как правило, предоставляют доступ к данным для чтения или отображения информации.

Контроллер **AuthorController** отвечает за обработку HTTP-запросов, связанных с операциями CRUD для сущности **Author**. Методы данного контроллера:

* Метод ***Index()*** с атрибутом [HttpGet] возвращает представление, отображающее список всех авторов из базы данных.
* Метод ***Details()*** с атрибутом [HttpGet] принимает идентификатор автора в качестве параметра и возвращает представление, отображающее подробную информацию об авторе с указанным идентификатором.
* Метод ***Create()*** ([HttpGet]) возвращает представление, позволяющее пользователю создать нового автора.
* Метод ***Create()*** ([HttpPost]) принимает данные нового автора в качестве параметра, добавляет его в базу данных и перенаправляет пользователя на метод ***Index()***.
* Метод ***Edit()*** ([HttpGet]) принимает идентификатор автора в качестве параметра и возвращает представление для редактирования информации об авторе с указанным идентификатором.
* Метод ***Edit()*** ([HttpPost]) принимает идентификатор автора и измененные данные в качестве параметров. Он обновляет информацию об авторе в базе данных и перенаправляет пользователя на метод ***Index()***.
* Метод ***Delete()*** ([HttpGet]) принимает идентификатор автора в качестве параметра и возвращает представление для подтверждения удаления автора с указанным идентификатором.
* Метод ***DeleteConfirmed()*** ([HttpPost]) принимает идентификатор автора в качестве параметра. Он удаляет автора из базы данных и перенаправляет пользователя на метод ***Index()***.
* Метод ***AuthorExists()*** выполняет проверку наличия автора в базе данных по заданному идентификатору.

Контроллер **BookController** отвечает за управление операциями CRUD (создание, чтение, обновление и удаление) для сущности **Book**. В контроллере определены следующие методы:

* ***Index()*** отвечает за отображение списка всех книг. Он получает список книг из контекста базы данных **\_context.Books** с загруженными связанными данными авторов (**Include("Authors")**) и передает его в представление **Index.cshtml**.
* ***LoadAuthors()*** загружает список авторов из контекста базы данных **\_context.Authors** и сохраняет его в **ViewBag.Authors** в формате **SelectList**. Это используется для отображения списка авторов в представлениях.
* ***Create()*** отображает форму для создания новой книги. Он вызывает приватный метод ***LoadAuthors()*** для загрузки списка авторов и передает его в представление **Create.cshtml**.
* ***Create(Book model)*** вызывается при отправке формы создания книги. Он добавляет новую книгу в контекст базы данных **\_context.Books**, сохраняет изменения с помощью **\_context.SaveChanges()** и перенаправляет пользователя на метод ***Index***().
* ***Edit(int? id)*** отображает форму редактирования существующей книги.
* ***Edit(Book model)*** вызывается при отправке формы редактирования книги. Он сначала удаляет состояние модели **ModelState.Remove("Authors")**, чтобы избежать ошибки валидации, затем проверяет, что модель model является допустимой. Если она допустима, то обновляет книгу в контексте базы данных **\_context.Books**, сохраняет изменения с помощью **\_context.SaveChanges()** и перенаправляет пользователя на метод ***Index()***.
* ***Delete(int? id)*** отображает форму подтверждения удаления книги. Он сначала проверяет, что идентификатор **id** не равен null, иначе возвращает ***NotFound()***. Затем он вызывает ***LoadAuthors()*** для загрузки списка авторов и находит книгу с указанным идентификатором в контексте базы данных \_context.Books. Книга передается в представление Delete для отображения.
* ***DeleteConfirmed(Book model)*** вызывается при отправке формы подтверждения удаления книги. Он удаляет книгу из контекста базы данных **\_context.Books**, сохраняет изменения с помощью **\_context.SaveChanges()** и перенаправляет пользователя на метод ***Index()***.

**3 ПРОГРАММНАЯ РЕАЛИЗАЦИЯ**

**3.1 Разработка баз данных**

**Entity Framework Core** представляет ORM-решение (Object-Relational Mapping), то есть позволяет автоматически связать обычные классы языка C# с таблицами в базе данных. Работа с базами данным будет выполняться в СУБД **MS SQL Server**, которую нужно предварительно установить.

Для взаимодействия с MS SQL Server через Entity Framework Core необходим пакет **Microsoft.EntityFrameworkCore.SqlServer**. По умолчанию он отсутствует в проекте, поэтому его надо добавить через пакетный менеджер NuGet.

Также для удобства взаимодействия с MS SQL Server можно установить **SSMS** (SQL Server Management Studio). С помощью SSMS можно создавать, изменять и удалять базы данных, таблицы. А также SSMS предоставляет возможность создавать резервные копии баз данных и восстанавливать их в случае сбоев или потери данных.

Чтобы взаимодействовать с базой данных через Entity Framework Core нам нужен контекст данных — класс, унаследованный от класса **Microsoft.EntityFrameworkCore**. Для этого в папке *Data* создан класс **ApplicationDbContext.cs**, который представляет собой контекст базы данных для приложения и наследуется от класса IdentityDbContext. Он обеспечивает доступ к таблицам базы данных, связанным с авторами и книгами, а также предоставляет возможность работы с учетными записями пользователей через наследование от IdentityDbContext.

Через параметр options в конструктор контекста данных будут передаваться настройки контекста:

public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options) :

base(options) { }

В данном классе определены два свойства ***DbSet***:

public DbSet<Author> Authors { get; set; }

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

Данные свойства представляют собой коллекцию объектов, которая сопоставляется с определенной таблицей в базе данных. С помощью этих свойств можно выполнять операции CRUD (создание, чтение, обновление, удаление) с соответствующими данными в базе данных.

Чтобы подключится к базе данных, нужно задать параметры подключения. Для этого в файл appsettings.json необходимо добавить определение строки подключения: "ConnectionStrings": {"DefaultConnection":"Server=DESKTOP-TU7H6VS\\SQLEXPRESS;Database=webapp;Trusted\_Connection=True;MultipleActiveResultSets=true;TrustServerCertificate=True;encrypt=false"}

Имя сервера — **DESKTOP-TU7H6VS\SQLEXPRESS**, имя базы данных — **webapp**.

**Миграции** в Entity Framework Core представляют собой способ управления изменениями схемы базы данных вместе с соответствующими изменениями моделей приложения. Миграции позволяют автоматически применять эти изменения моделей к базе данных, включая создание, изменение и удаление таблиц, добавление и удаление столбцов и другие операции изменения схемы базы данных. Чтобы создать миграцию нужно в командной строке Package Manager Console прописать add-migration initial, затем для применения миграции — update-database. После запуска последней команды будут применены все непримененные миграции в порядке их создания к базе данных, что включает в себя создание таблиц, изменение существующих таблиц, добавление или удаление столбцов.

**3.2 Аунтетификация и авторизация**

В файле Program.cs приведенный ниже код выполняет настройку авторизации на основе ролей в приложении:

builder.Services.AddDefaultIdentity<IdentityUser>(options => options.SignIn.RequireConfirmedAccount = true) /\* Добавляет службу идентификации по умолчанию, используя класс IdentityUser в качестве модели пользователя. Опция SignIn.RequireConfirmedAccount устанавливает требование подтверждения аккаунта перед входом. \*/

.AddRoles<IdentityRole>() /\* Добавляет поддержку ролей в приложении, используя класс IdentityRole для представления ролей пользователей. \*/

.AddEntityFrameworkStores<ApplicationDbContext>() /\* Указывает, что данные об идентификации и ролях будут храниться в ApplicationDbContext, который представляет базу данных приложения. \*/

Для проверки авторизации на основе ролей в Razor страницах, контроллерах или действиях контроллера нужно использовать конструкцию @if (User.IsInRole("Admin")), где "Admin" - это имя роли, к членам которой ограничивается доступ. В представлении Razor код внутри блока @if будет выполняться только для пользователей, принадлежащих роли "Admin".

Например, если пользователь является членом роли "Admin", то будет отображен заголовок страницы с помощью кода <h3>@ViewData["Title"]</h3>. Если пользователь не является членом роли "Admin", то этот код не будет выполнен и заголовок страницы не будет отображен.

**3.3 Тестирование приложения**

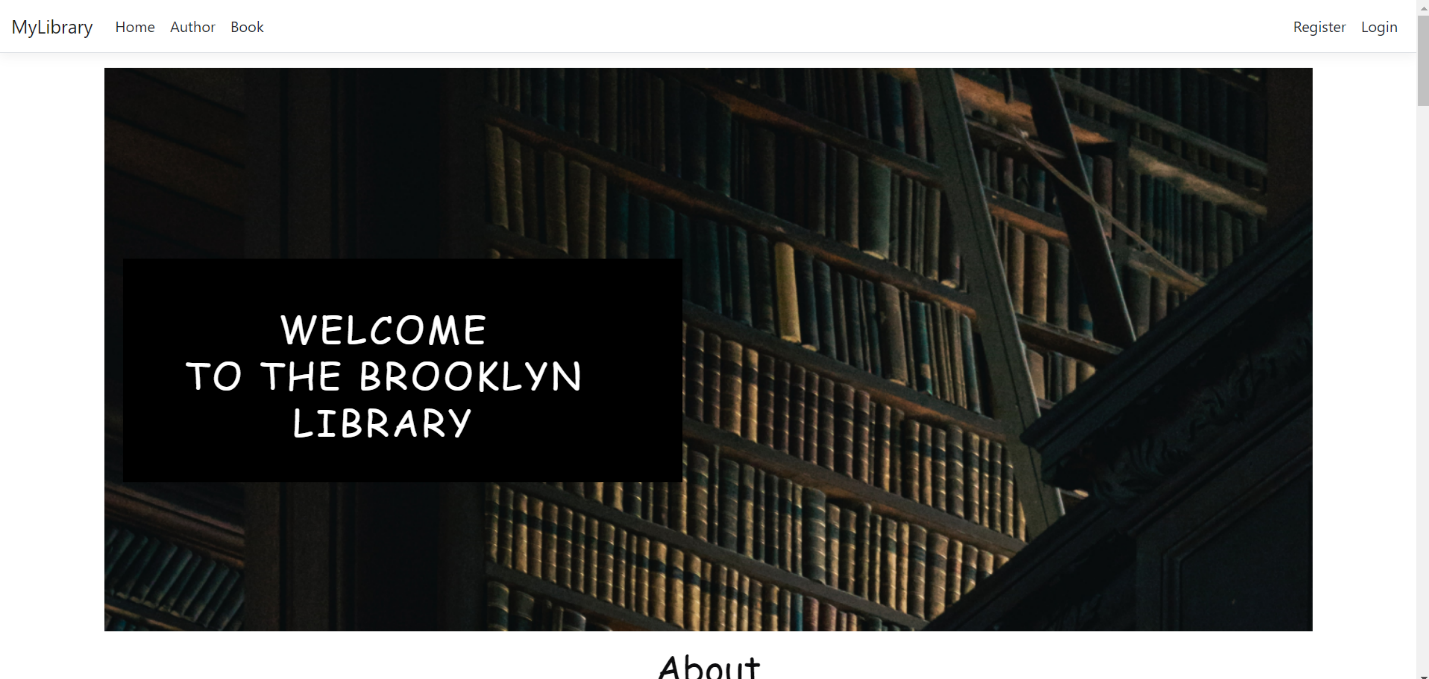
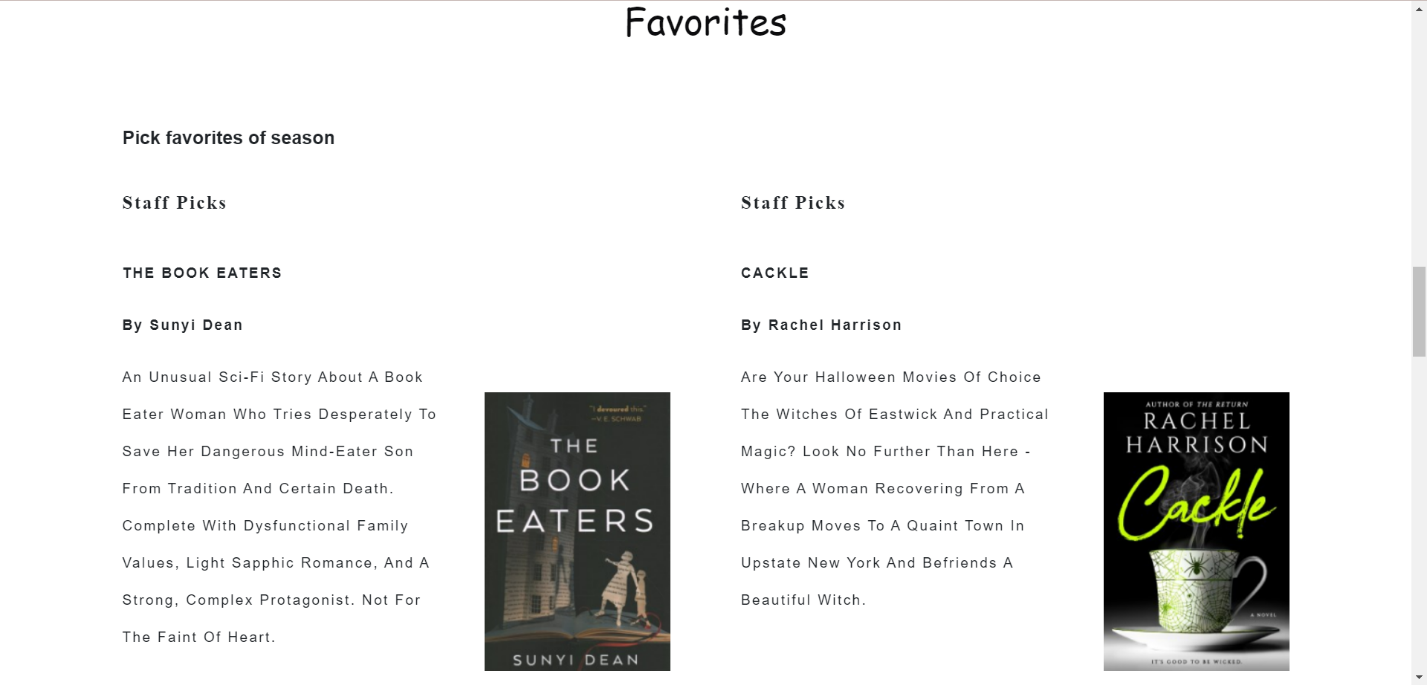
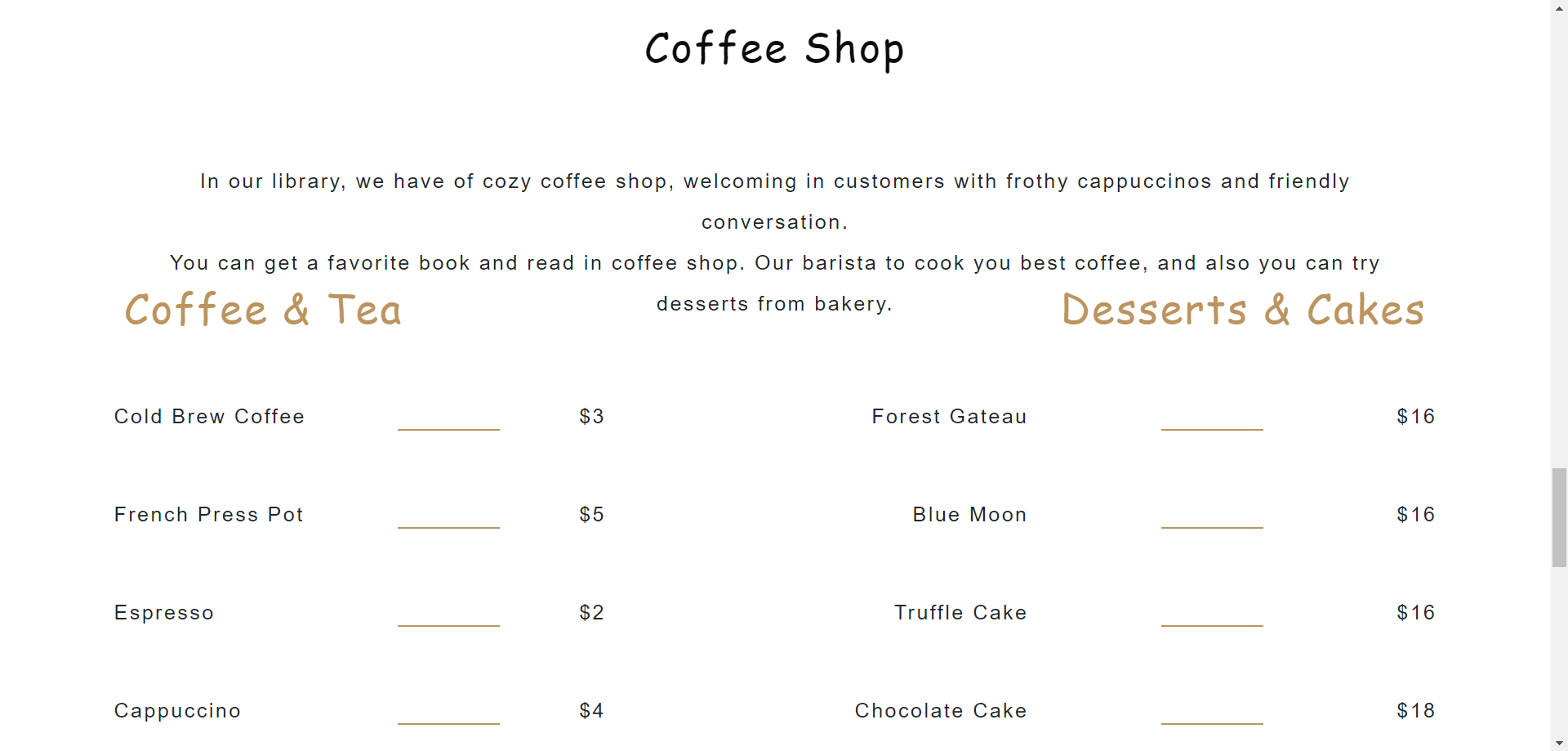
Главная страница (рис. 3.3.1):

Рисунок 3.3.1 – Главная страница

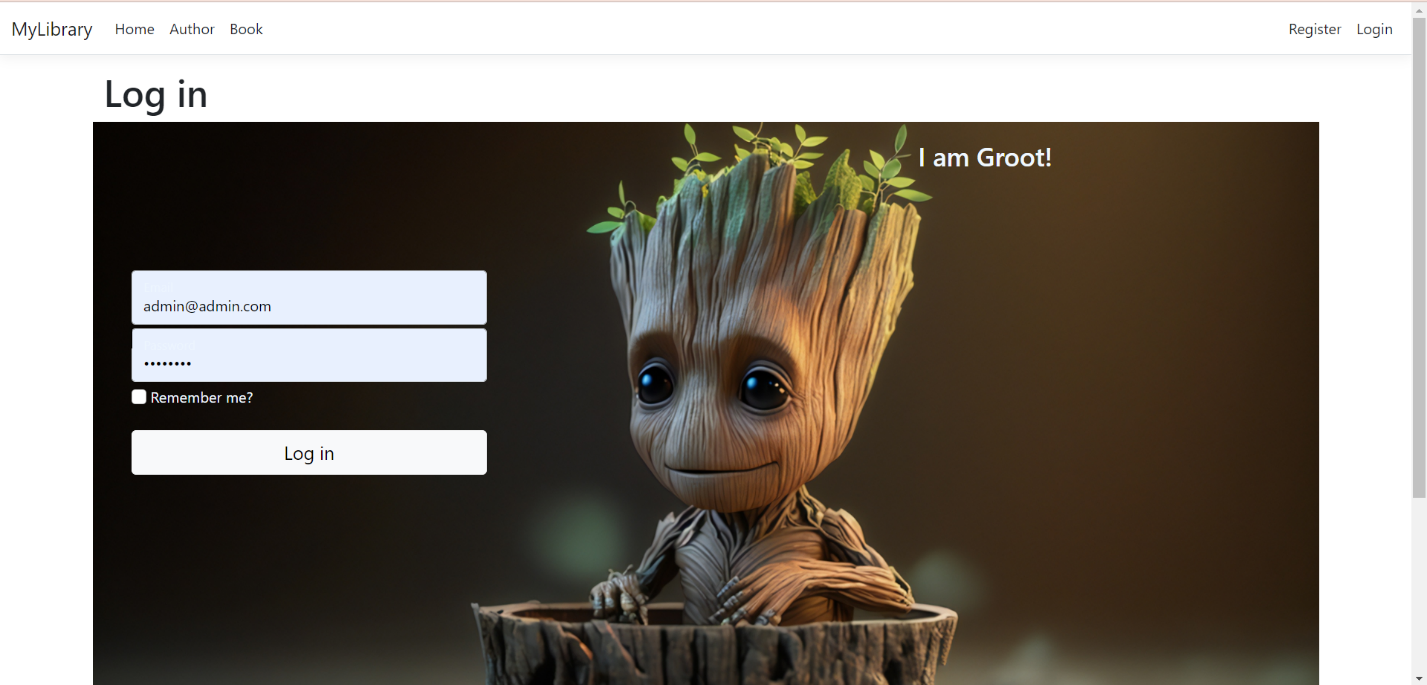
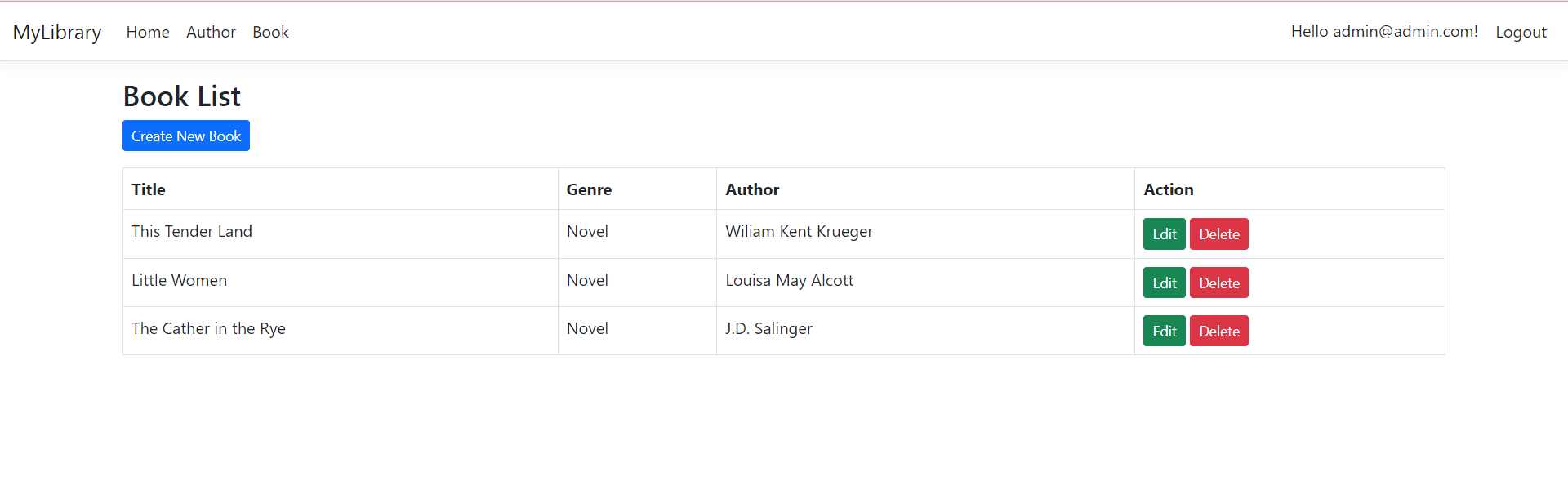
Страница логгирования (рис. 3.3.2):

Рисунок 3.3.2 – Страница логгирования

После успешной аутентификации в систему с использованием почтового адреса и пароля администратора, открывается виртуальный каталог книг (рис. 3.3.3) и список авторов на другой странице (рис. 3.3.4). Администратор может вносить поправки в фонд.

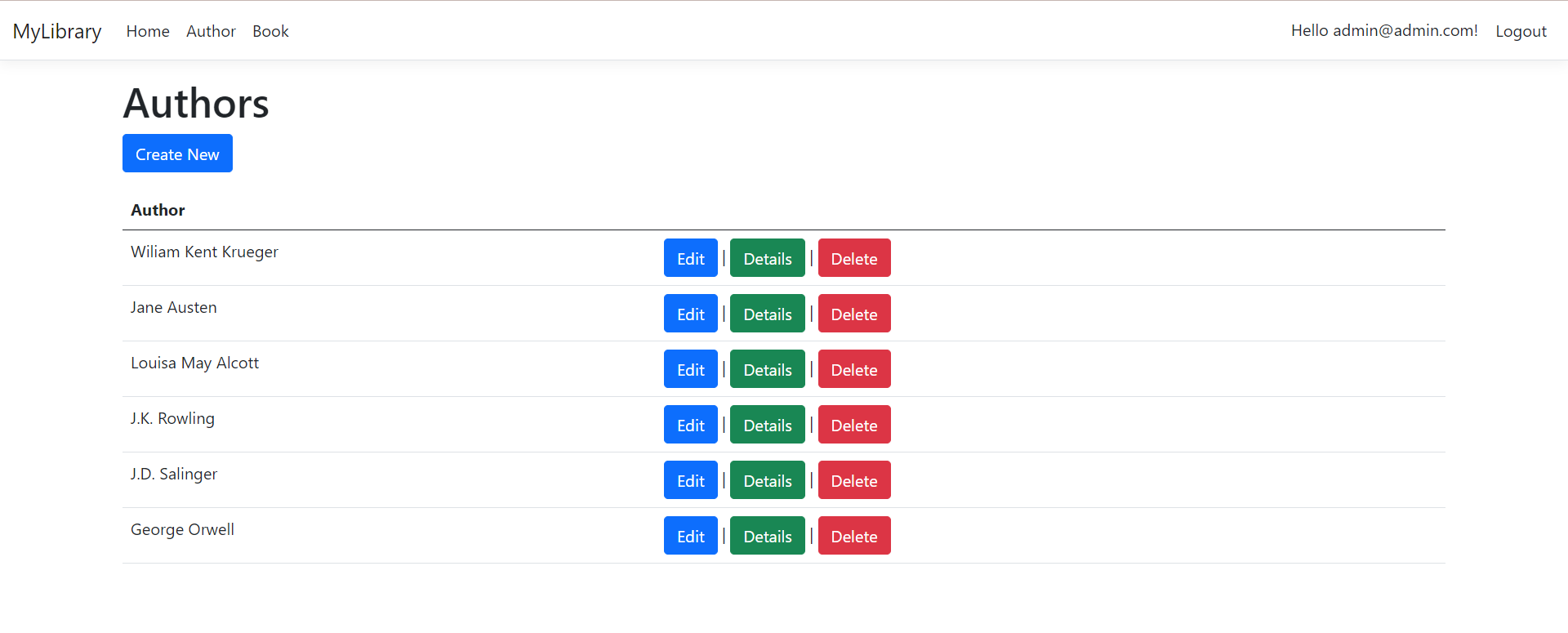
Рисунок 3.3.3 – Фонд библиотеки

Рисунок 3.3.4 – Список авторов

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. C# documentation [Электронный ресурс]. – Электронные данные. – Режим доступа: https://www.learn.microsoft.com/en-us/dotnet/csharp/. — Дата доступа: 12.12.2023

2. Metanit.com [Электронный ресурс]. – Электронные данные. – Режим доступа: https://www.metanit.com/sharp/general.php. — Дата доступа: 26.11.2023

(Приложение 2.2)

**Program.cs**

using Microsoft.AspNetCore.Identity;

using Microsoft.EntityFrameworkCore;

using MyLibrary.Data;

namespace MyLibrary

{

public class Program

{

public static async Task Main(string[] args)

{

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.

var connectionString = builder.Configuration.GetConnectionString("DefaultConnection") ?? throw new InvalidOperationException("Connection string 'DefaultConnection' not found.");

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

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

builder.Services.AddDatabaseDeveloperPageExceptionFilter();

builder.Services.AddDefaultIdentity<IdentityUser>(options => options.SignIn.RequireConfirmedAccount = true)

.AddRoles<IdentityRole>()

.AddEntityFrameworkStores<ApplicationDbContext>();

builder.Services.AddControllersWithViews();

var app = builder.Build();

// Configure the HTTP request pipeline.

if (app.Environment.IsDevelopment())

{

app.UseMigrationsEndPoint();

}

else

{

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.MapRazorPages();

app.Run();

}

}

}

|  |  |
| --- | --- |
| **Author.cs** | **Book.cs** |
| using System.ComponentModel.DataAnnotations;  using System.ComponentModel;  namespace MyLibrary.Models  {  public class Author  {  [Key]  public int Id { get; set; }  [DisplayName("Author")]  public string Name { get; set; }  }  } | using System.ComponentModel.DataAnnotations.Schema;  using System.ComponentModel.DataAnnotations;  namespace MyLibrary.Models  {  public class Book  {  [Key]  public int Id { get; set; }  public string Title { get; set; }  public string Genre { get; set; }  public string Author { get; set; }  [ForeignKey("Authors")]  public int AuthorId { get; set; } public virtual Author Authors {get;set;}  }  } |

Приложение (2.3)

**Views**

@model MyLibrary.Models.Author

@{

ViewData["Title"] = "Create";

}

<h1>Create</h1>

<h4>Author</h4>

<hr />

<div class="row">

<div class="col-md-4">

<**form** **asp-action**="Create">

<**div** **asp-validation-summary**="ModelOnly" class="text-danger"></**div**>

<div class="form-group">

<**label** **asp-for**="Name" class="control-label"></**label**>

<**input** **asp-for**="Name" class="form-control" />

<**span** **asp-validation-for**="Name" class="text-danger"></**span**>

</div>

<div class="form-group mt-2">

<input type="submit" value="Create" class="btn btn-primary" />

</div>

</**form**>

</div>

</div>

<div>

<**a** **asp-action**="Index">Back to List</**a**>

</div>

@section Scripts {

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

}

@model MyLibrary.Models.Author

@{

ViewData["Title"] = "Delete";

}

<h1>Delete</h1>

<h3>Are you sure you want to delete this?</h3>

<div>

<h4>Author</h4>

<hr />

<dl class="row">

<dt class = "col-sm-2">

@Html.DisplayNameFor(model => model.Name)

</dt>

<dd class = "col-sm-10">

@Html.DisplayFor(model => model.Name)

</dd>

</dl>

<**form** **asp-action**="Delete">

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

<input type="submit" value="Delete" class="btn btn-danger mt-2" /> |

<**a** **asp-action**="Index">Back to List</**a**>

</**form**>

</div>

@model Book

@{

ViewData["Title"] = "Update Book";

}

<h3>@ViewData["Title"]</h3>

<hr />

<**form** **asp-action**="Edit">

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

<div class="container">

<div class="mb-2">

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

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

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

</div>

<div class="mb-2">

<**label** **asp-for**="Genre" class="control-label"></**label**>

<**input** **asp-for**="Genre" class="form-control" />

<**span** **asp-validation-for**="Genre" class="text-danger"></**span**>

</div>

<div class="mb-2">

<label class="control-label">Author</label>

<**select** **asp-for**="AuthorId" class="form-control" **asp-items**="ViewBag.Authors">

<**option**>-Select Author-</**option**>

</**select**>

</div>

<div>

<input type="submit" value="Update" class="btn btn-success btn-sm" />

</div>

</div>

</**form**>

(Приложение 2.4)

**Controllers**

**AuthorController.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using Microsoft.AspNetCore.Mvc;

using Microsoft.AspNetCore.Mvc.Rendering;

using Microsoft.EntityFrameworkCore;

using MyLibrary.Data;

using MyLibrary.Models;

namespace MyLibrary.Controllers

{

public class AuthorController : Controller

{

private readonly ApplicationDbContext \_context;

public AuthorController(ApplicationDbContext context)

{

\_context = context;

}

// GET: Author

public async Task<IActionResult> Index()

{

return View(await \_context.Authors.ToListAsync());

}

// GET: Author/Details/5

public async Task<IActionResult> Details(int? id)

{

if (id == null)

{

return NotFound();

}

var author = await \_context.Authors

.FirstOrDefaultAsync(m => m.Id == id);

if (author == null)

{

return NotFound();

}

return View(author);

}

// GET: Author/Create

public IActionResult Create()

{

return View();

}

// POST: Author/Create

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Create([Bind("Id,Name")] Author author)

{

if (ModelState.IsValid)

{

\_context.Add(author);

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

return View(author);

}

// GET: Author/Edit/5

public async Task<IActionResult> Edit(int? id)

{

if (id == null)

{

return NotFound();

}

var author = await \_context.Authors.FindAsync(id);

if (author == null)

{

return NotFound();

}

return View(author);

}

// POST: Author/Edit/5

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Edit(int id, [Bind("Id,Name")] Author author)

{

if (id != author.Id)

{

return NotFound();

}

if (ModelState.IsValid)

{

try

{

\_context.Update(author);

await \_context.SaveChangesAsync();

}

catch (DbUpdateConcurrencyException)

{

if (!AuthorExists(author.Id))

{

return NotFound();

}

else

{

throw;

}

}

return RedirectToAction(nameof(Index));

}

return View(author);

}

// GET: Author/Delete/5

public async Task<IActionResult> Delete(int? id)

{

if (id == null)

{

return NotFound();

}

var author = await \_context.Authors

.FirstOrDefaultAsync(m => m.Id == id);

if (author == null)

{

return NotFound();

}

return View(author);

}

// POST: Author/Delete/5

[HttpPost, ActionName("Delete")]

[ValidateAntiForgeryToken]

public async Task<IActionResult> DeleteConfirmed(int id)

{

var author = await \_context.Authors.FindAsync(id);

if (author != null)

{

\_context.Authors.Remove(author);

}

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

private bool AuthorExists(int id)

{

return \_context.Authors.Any(e => e.Id == id);

}

}

}

(Приложение 3.1)

**ApplicationDbContext.cs**

using Microsoft.AspNetCore.Identity.EntityFrameworkCore;

using Microsoft.EntityFrameworkCore;

using MyLibrary.Models;

namespace MyLibrary.Data

{

public class ApplicationDbContext : IdentityDbContext

{

public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options)

: base(options)

{

}

public DbSet<Author> Authors { get; set; }

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

}

}