**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ**

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ

ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГООБРАЗОВАНИЯ

**«ТЮМЕНСКИЙ ИНДУСТРИАЛЬНЫЙ УНИВЕРСИТЕТ»**

**ВЫСШАЯ ШКОЛА ЦИФРОВЫХ ТЕХНОЛОГИЙ**

Кафедра математики и прикладных информационных технологий

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

к курсовому проекту

по дисциплине «Методы и средства проектирования компьютерных приложений»

на тему: «Разработка автоматизированной информационной системы “Электронная библиотека”»

**Выполнил:**

студент 2 курса группы НТм-23-1

направления 09.04.01 «Информатика и вычислительная техника»

Менщикова Д.А.

**Руководитель:**

канд. пед. наук, доцент

Спирин И.С.

2025

**ТЕХНИЧЕСКОЕ ЗАДАНИЕ**

Разработка автоматизированной библиотечно-информационной системы (АБИС) для управления библиотечным каталогом на основе Blazor и современных веб-технологий. Основными функциями системы являются хранение, обработка и предоставление информации о книгах, авторах и пользователях, а также автоматизация процессов поиска, выдачи и учета литературы.

Для реализации АБИС «Электронная библиотека» с удобным веб-интерфейсом, обеспечивающим эффективное управление библиотечными ресурсами, потребуется:

* провести анализ предметной области и существующих решений в области АБИС;
* спроектировать модель базы данных;
* разработать клиентское веб-приложение на основе Blazor с интуитивно понятным интерфейсом в стиле современных приложений;
* реализовать функционал для управления книгами, авторами и пользователями (добавление, редактирование, удаление); поиска и выдачи книг; отслеживания задолженностей.

Система должна обеспечивать удобное взаимодействие с пользователями и автоматизацию ключевых библиотечных процессов.

**АННОТАЦИЯ**

**Цель** – разработка автоматизированной библиотечно-информационной системы (АБИС) для эффективного управления библиотечными ресурсами, обеспечения удобного поиска и доступа к материалам, а также автоматизации ключевых библиотечных процессов.

**Объект работы** – автоматизированная библиотечно-информационная система.

**Предмет работы** – методы и инструменты разработки АБИС с интуитивно понятным интерфейсом и широкими функциональными возможностями.

**Информационная база исследования** – существующие решения в области электронных библиотек, современные технологии программирования, системы управления базами данных (СУБД) и веб-разработки.

**Результат:** прототип автоматизированной системы «Электронная библиотека», реализованный с помощью Blazor на .NET 8.

**Курсовой проект состоит** из введения, трёх глав, заключения, списка использованных источников и приложения.

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 5](#_Toc196277396)

[1 Анализ предметной области 7](#_Toc196277397)

[1.1 Общая характеристика 7](#_Toc196277398)

[1.2 Анализ существующих решений 7](#_Toc196277399)

[1.3 Требования к разрабатываемой системе 10](#_Toc196277400)

[2 Реализация автоматизированной библиотечно-информационной системы 12](#_Toc196277401)

[2.1 Выбор технологий и инструментов 12](#_Toc196277402)

[2.2 Архитектура системы 15](#_Toc196277403)

[3 Руководство пользователя 22](#_Toc196277404)

[3.1 Руководство пользователя – администратора 23](#_Toc196277405)

[3.2 Руководство пользователя – библиотекаря 26](#_Toc196277406)

[3.3 Руководство пользователя – читателя 29](#_Toc196277407)

[ЗАКЛЮЧЕНИЕ 32](#_Toc196277408)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 33](#_Toc196277409)

[ПРИЛОЖЕНИЕ А 34](#_Toc196277410)

[ПРИЛОЖЕНИЕ Б 40](#_Toc196277411)

[ПРИЛОЖЕНИЕ В 51](#_Toc196277412)

**ВВЕДЕНИЕ**

Современное общество характеризуется стремительным развитием информационных технологий, которые проникают во все сферы человеческой деятельности, включая образование, науку и культуру. Внедрение автоматизированных библиотечно-информационных систем становится ключевым направлением модернизации библиотечной деятельности в этих условиях. Современные АБИС позволяют оптимизировать работу с библиотечными фондами, обеспечивая эффективное управление как традиционными бумажными, так и электронными ресурсами.

Актуальность темы курсового проекта обусловлена возрастающей потребностью в автоматизации процессов управления библиотечными фондами, особенно в условиях роста объема цифровых публикаций и необходимости обеспечения удаленного доступа к ним. Традиционные библиотечные системы уже не справляются с современными требованиями к скорости, удобству и функциональности, что делает разработку новой автоматизированной библиотечной информационной системы востребованной задачей.

Целью курсового проекта является разработка программного обеспечения для эффективного управления библиотечными ресурсами, обеспечивающего пользователям удобный поиск и доступ к материалам, а также автоматизацию ключевых процессов, таких как учет книг, обработка запросов и формирование отчетности.

Объектом исследования выступают автоматизированные библиотечно-информационные системы, а предметом исследования – методы и средства разработки автоматизированной библиотечной информационной системы с удобным интерфейсом и широкими функциональными возможностями.

Практическая значимость работы заключается в том, что внедрение разработанной системы позволит оптимизировать работу библиотек, повысить доступность образовательных и научных ресурсов, а также сократить временные затраты на обслуживание пользователей.

Для достижения поставленной цели необходимо решить следующие задачи:

* провести анализ существующих решений автоматизации библиотечно-информационных систем;
* определить функциональные и технические требования к системе;
* разработать архитектуру и базу данных системы;
* реализовать основные модули (учет книг, управление пользователями, поиск и выдача материалов);
* протестировать работоспособность системы и оценить ее эффективность.

Курсовой проект выполняется с использованием современных технологий программирования, включая языки высокого уровня, системы управления базами данных и веб-разработку. В результате будет создан прототип системы, готовый к дальнейшему развитию и внедрению.

**1 Анализ предметной области**

**1.1 Общая характеристика**

Автоматизированная библиотечная информационная система предназначена для учета и управления традиционными библиотечными фондами, состоящими из бумажных книг, журналов и других печатных материалов. Система обеспечивает автоматизацию ключевых библиотечных процессов, упрощая работу сотрудников и улучшая обслуживание читателей.

Основные функции системы:

* учет пользователей – регистрация читателей и ведение их личных карточек;
* каталогизация фондов – внесение и систематизация данных о книгах (автор, название, ISBN, жанр, место хранения и т. д.);
* учет выдачи и возврата – фиксация выданных экземпляров, контроль сроков возврата, напоминания о задолженностях;
* поиск – быстрый поиск книг в каталоге;
* работа с инвентаризацией – учет поступлений, списаний и перемещений книг внутри библиотеки.

АБИС представляет собой современное решение для автоматизации работы с бумажными фондами. Она помогает сократить рутинные операции, минимизировать ошибки в учете и предоставить читателям удобный сервис.

**1.2 Анализ существующих решений**

На рынке представлено множество Автоматизированная библиотечная информационная система. Были рассмотрены следующие решения:

***1.2.1 ИРБИС***

*Разработчик:* государственная публичная научно-техническая библиотека (ГПНТБ) России.

*Тип:* локальная/сетевая система (есть веб-интерфейс).

*Основные пользователи:* научные, вузовские и крупные публичные библиотеки.

Особенности:

* поддержка RUSMARC и других библиотечных стандартов;
* гибкая система каталогизации (книги, периодика, электронные ресурсы);
* встроенный Z39.50-сервер для интеграции с другими каталогами;
* возможность создания сводных электронных каталогов;
* поддержка штрихкодирования и RFID.

Преимущества:

* полная локализация под российские стандарты;
* возможность масштабирования (от небольших библиотек до крупных сетей);
* доступна бесплатная версия (ИРБИС64).

Недостатки:

* устаревший интерфейс (в классической версии);
* требует обучения персонала из-за сложной настройки;
* нет полноценной облачной версии (требует серверного развертывания).

***1.2.2 MARK-SQL***

*Разработчик:* ГПНТБ России.

*Тип:* сетевая система на базе Microsoft SQL Server.

*Основные пользователи:* крупные библиотеки, корпоративные системы.

Особенности:

* работа с большими массивами данных (подходит для библиотек с миллионами записей);
* поддержка межбиблиотечного абонемента (МБА);
* интеграция с 1С для бухгалтерского учета;
* глубокая аналитика (статистика посещений, выдачи).

Преимущества:

* высокая производительность за счет SQL-сервера;
* поддержка электронной книговыдачи (включая временный доступ).

Недостатки:

* дорогое внедрение (требует лицензий Microsoft SQL Server);
* сложность администрирования;
* ограниченная мобильность (нет мобильного приложения).

***1.2.3 1С:Библиотека***

*Разработчик:* фирма 1С (на базе платформы 1С:Предприятие).

*Тип:* Локальная/облачная система.

*Основные пользователи:* Школьные, муниципальные и небольшие вузовские библиотеки.

Особенности:

* полная интеграция с 1С:Бухгалтерией (учет подписки, финансы).
* простой интерфейс, похожий на 1С.
* поддержка штрих-кодов и RFID.
* учет печатных и электронных ресурсов.

Преимущества:

* автоматизированная отчетность (включая статистику для Минкульта);
* гибкость настройки (можно дорабатывать под нужды библиотеки).

Недостатки:

* менее мощная каталогизация, чем у ИРБИС или MARK-SQL;
* зависимость от платформы 1С (лицензирование);
* не подходит для крупных научных библиотек.

Существующие системы решают базовые задачи, но часто страдают от архаичности, сложности или недостаточной гибкости. Разрабатываемая АИС «Электронная библиотека» должна устранить эти недостатки, предложив современное, простое в использовании и адаптируемое решение.

**1.3 Требования к разрабатываемой системе**

На основании проведенного анализа существующих решений можно выделить следующие ключевые проблемы:

* сложные процессы добавления и редактирования материалов.
* сложная навигация и архаичный дизайн,
* отсутствие адаптивности для мобильных устройств,
* не интуитивный процесс поиска и фильтрации контента,
* высокие требования к квалификации персонала.

Исходя из вышеперечисленных проблем были сформулированы актуальные требования к разрабатываемой системе.

Функциональные:

* управление пользователями и ролями;
* управление книгами, авторами и жанрами;
* поиск по различным критериям (автор, название, жанр, ISBN);
* возможность учета выдачи и возврата книг,
* возможность читателям заранее забронировать необходимую книгу.

На основе сформулированных функциональных требований была построена диаграмма вариантов использования системы (рисунок 1.1).

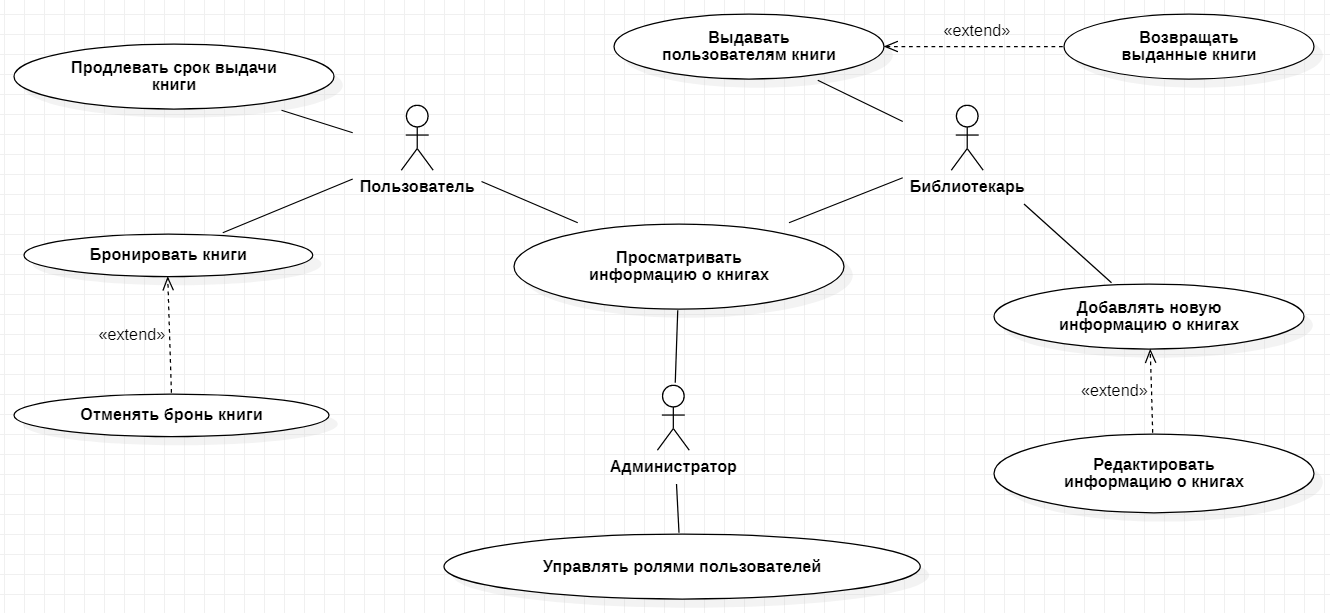


Рисунок 1.1 – Диаграмма вариантов использования

Нефункциональные требования к разрабатываемой системе:

* простота и интуитивность интерфейса,
* масштабируемость и поддержка большого объема данных,
* защита данных (авторизация).

Разрабатываемая система должна устранить основные недостатки существующих решений, предложив современную, удобную и функциональную платформу для работы библиотек. Особое внимание будет уделено улучшению пользовательского опыта, упрощению процессов администрирования и обеспечению высокой производительности и надежности.

Реализация указанных требований позволит создать конкурентноспособный продукт, отвечающий потребностям современных библиотек и их пользователей.

Система должна обеспечивать удобный доступ к ресурсам, гибкое управление контентом и безопасность данных, устраняя недостатки существующих аналогов.

**2 Реализация автоматизированной библиотечно-информационной системы**

**2.1 Выбор технологий и инструментов**

Для реализации системы в качестве основного инструмента была выбрана платформа Blazor.

Blazor представляет собой современную платформу для разработки веб-приложений, основанную на языке C# и платформе .NET. Выбор Blazor обусловлен рядом преимуществ, делающих его оптимальным решением для данного проекта:

* Использование единого языка программирования.

Blazor позволяет разрабатывать фронтенд и бэкенд на едином языке программирования – C#. Это существенно упрощает разработку, повышает производительность и снижает издержки на изучение инструмента.

* Высокая производительность и безопасность.

Blazor WebAssembly обеспечивает высокую производительность благодаря исполнению кода непосредственно в браузере, используя WebAssembly. Это позволяет достичь скорости, сравнимой с нативными приложениями, сохраняя при этом высокий уровень безопасности благодаря изоляции исполняемого кода.

* Удобство интеграции с серверной частью.

Blazor Server предоставляет удобную модель двусторонней связи с сервером через SignalR, что упрощает взаимодействие с серверными службами и позволяет легко интегрировать функциональность, связанную с управлением библиотечными ресурсами и пользователями.

* Широкий спектр инструментов и библиотек.

Платформа Blazor тесно интегрирована с экосистемой .NET, предоставляющей многочисленные готовые библиотеки и инструменты, что позволяет ускорить разработку и повысить качество итогового продукта.

* Совместимость с популярными инструментами разработки.

Blazor поддерживает Visual Studio и Visual Studio Code, предлагая мощные средства отладки и рефакторинга, что делает процесс разработки комфортным и эффективным.

Выбор языка программирования C# и платформы .NET продиктован следующими факторами:

* Продвинутые возможности языка программирования.

C# обладает богатым набором возможностей, таких как строгая типизация, LINQ, лямбда-выражения и асинхронное программирование, что существенно упрощает разработку сложных функциональных элементов системы.

* Поддержка современной архитектуры.

.NET Core и .NET 8 предоставляют отличную поддержку микросервисной архитектуры, контейнеризации и кроссплатформенности, что важно для гибкости и масштабируемости приложения.

* Богатая экосистема библиотек и инструментов.

.NET имеет обширную коллекцию библиотек и инструментов, таких как Entity Framework Core, для работы с базами данных, Identity для управления пользователями и аутентификации, а также разнообразные библиотеки для валидации данных, кэширования и работы с внешними API.

* Простота миграции и поддержки существующего кода.

Использование C# и .NET позволяет легко интегрировать существующие решения и компоненты, а также облегчить процесс технической поддержки и развития системы в будущем.

SQL Server был выбран в качестве системы управления базами данных ввиду следующих преимуществ:

* Надёжность и стабильность.

SQL Server известен своей надежностью и устойчивостью к сбоям, что критично для долгосрочной эксплуатации информационной системы.

* Высокопроизводительная работа с большими объемами данных.

SQL Server демонстрирует высокие показатели производительности при работе с большими массивами данных, что актуально для проектов, связанных с управлением крупными библиотечными фондами и большими объемами информации.

* Комплексные возможности администрирования и безопасности.

SQL Server предоставляет развитые инструменты для мониторинга, резервного копирования и восстановления данных, а также мощные механизмы обеспечения безопасности и конфиденциальности информации.

* Хорошая интеграция с .NET.

SQL Server отлично интегрируется с платформой .NET, обеспечивая удобное и эффективное взаимодействие с базой данных через такие инструменты, как Entity Framework Core, что упрощает разработку и эксплуатацию приложения.

* Поддержка расширенных функций.

SQL Server поддерживает такие важные функции, как полнотекстовый поиск, геопространственные данные, машинное обучение и аналитика, что открывает возможности для дальнейшего развития и расширения функционала системы.

Таким образом, выбор указанных технологий и инструментов обеспечивает надежную, производительную и масштабируемую основу для реализации автоматизированной библиотечно-информационной системы, удовлетворяя требованиям текущего этапа разработки и предусматривая потенциал для будущего развития проекта.

**2.2 Архитектура системы**

Электронная библиотека представляет собой информационную систему, предназначенную для автоматизации процессов управления библиотечным каталогом, учета книг, авторов, пользователей, а также для организации и мониторинга займов и резервирований.

Опираясь на функциональные требования, был использован функционал авторизации, встроенный в Blazor, с делением пользователей на роли: администратор, библиотекарь, пользователь (читатель).

Неавторизованные пользователи имеют доступ к авторизации и регистрации. При регистрации новому аккаунту автоматически назначается роль пользователя (читателя). Авторизованным пользователям доступен функционал в соответствии с их ролью.

Для всех ролей доступен просмотр информации о книгах, жанрах и авторах. Администратору доступно управление ролями аккаунтов. Читателю доступна возможность забронировать книгу и отслеживать свои бронирования и книги, выданные на чтение. Библиотекарь доступна возможность управлять информацией о книгах, авторах и жанрах (создавать, редактировать, удалять) и управлять выдачей и возвратом книг пользователям.

Система организована в виде набора взаимосвязанных модулей, каждый из которых решает свою отдельную задачу:

1. ***Модель данных (Data Models).***

Представляет собой набор классов, отражающих сущности предметной области:

* Автор (Author),
* Книга (Book),
* Пользователь (ApplicationUser),
* Заём (BookLoan),
* Резервирование (BookReservation),
* Жанр (Genre),
* Ассоциация Автор – Книга (BookAuthor),
* Ассоциация Книга – Жанр (BookGenre).

Ниже приведен код модели данных класса Book:

public class Book

{

public int Id { get; set; }

public string Title { get; set; }

public string ISBN { get; set; }

public string? Publisher { get; set; }

public int Year { get; set; }

public int Pages { get; set; }

public string? Language { get; set; }

public string? Description { get; set; }

public int TotalCopies { get; set; }

public int AvailableCopies { get; set; }

public List<BookAuthor> BookAuthors { get; set; } = new();

public List<BookGenre> BookGenres { get; set; } = new();

public List<BookLoan> BookLoans { get; set; } = new();

public List<BookReservation> BookReservations { get; set; } = new();

}

1. ***Хранилище данных (Database Context)***

Класс ApplicationDbContext управляет взаимодействием с базой данных, обеспечивая связь моделей данных с реляционной моделью базы данных. Здесь определяются отношения между сущностями и специфические настройки (например, уникальные индексы, связи и ограничения).

Ниже приведен код описания связей для класса BookLoan из класса ApplicationDbContext:

builder.Entity<BookLoan>(entity =>

{

entity.HasOne(bl => bl.User)

.WithMany(u => u.BookLoans)

.HasForeignKey(bl => bl.UserId)

.OnDelete(DeleteBehavior.Cascade);

entity.HasOne(bl => bl.Book)

.WithMany(b => b.BookLoans)

.HasForeignKey(bl => bl.BookId)

.OnDelete(DeleteBehavior.Cascade);

});

На рисунке 2.1 представлена диаграмма сущностей и связей реляционной базы данных системы на основе классов модели данных.

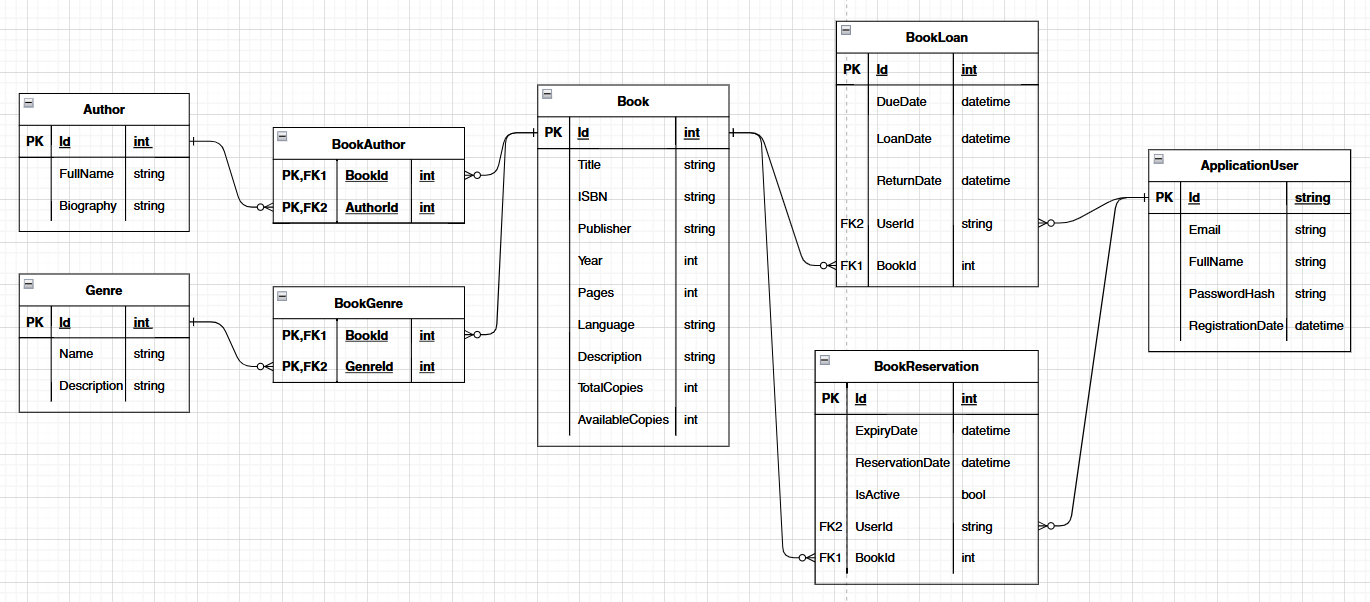


Рисунок 2.1 – ER-диаграмма реляционной базы данных

1. ***Сервисы (Services Layer)***

Сервисы слоя отвечают за реализацию бизнес-логики и координацию взаимодействия с уровнем данных (хранилищами). Основная задача каждого сервиса заключается в обработке конкретных типов данных и выполнении связанных с ними операций, таких как чтение, сохранение, обновление и удаление данных. Такой подход позволяет разделить ответственность между разными модулями и упростить поддержку и развитие проекта.

Каждый сервис ориентирован на конкретные типы сущностей и выполняет узкий круг функций, касающихся именно этих сущностей:

* UserService – управление пользователями и их ролями;
* BookService – работа с книгами, добавление, изменение, удаление, поиск книг, присвоение жанров и авторов;
* AuthorService – управление авторами, добавление, изменение, удаление и поиск авторов;
* GenreService – работа с жанрами, добавление, изменение, удаление и поиск жанров;
* BookLoanService – управление займами книг, выдача, возврат, продление займов, а также проверка состояния займа;
* BookReservationService – организация и отмена резервирования книг, контроль активности резервирования.

Ниже приведен код реализации метода добавления новой книги из BookService:

public async Task AddBookAsync(Book book, List<int> authorIds, List<int> genreIds)

{

if (book == null) throw new ArgumentNullException(nameof(book));

if (await \_context.Books.AnyAsync(b => b.ISBN == book.ISBN))

throw new InvalidOperationException("Книга с таким ISBN уже существует");

await \_context.Books.AddAsync(book);

await \_context.SaveChangesAsync();

foreach (var authorId in authorIds)

{

if (!await \_context.Authors.AnyAsync(a => a.Id == authorId))

continue;

\_context.BookAuthors.Add(new BookAuthor

{

BookId = book.Id,

AuthorId = authorId

});

}

foreach (var genreId in genreIds)

{

if (!await \_context.Genres.AnyAsync(g => g.Id == genreId))

continue;

\_context.BookGenres.Add(new BookGenre

{

BookId = book.Id,

GenreId = genreId

});

}

await \_context.SaveChangesAsync();

}

Все методы, реализуемые в каждом из сервисов, разработаны с применением концепции асинхронного программирования. Это означает, что выполнение запросов к базе данных и другие ресурсоемкие операции выполняются параллельно основному потоку исполнения программы, не блокируя остальные процессы. Библиотека электронных ресурсов, особенно в условиях интенсивного обращения большого числа пользователей, предъявляет высокие требования к быстродействию. Асинхронные методы позволяют сократить задержки и ускорить обработку запросов, особенно при взаимодействии с СУБД, сетью или сторонними системами. Параллельное исполнение запросов позволяет увеличить пропускную способность системы, распределяя нагрузку равномерно между различными потоками. Это создает основу для дальнейшего горизонтального масштабирования, например, путём распределения нагрузки между несколькими серверами. Благодаря асинхронности, интерфейс остаётся быстрым и отзывчивым, независимо от сложности выполняемых операций. Пользователь не сталкивается с заметными паузами при выполнении запросов, что создаёт положительный пользовательский опыт. Асинхронные механизмы снижают вероятность блокировки основного потока выполнения программы в случае возникновения временных сбоев сети или замедления ответа от внешних служб. Приложение продолжает функционировать, не завися от единственного потока исполнения.

Сервисы слоя Services Layer играют ключевую роль в обеспечении нормального функционирования электронной библиотеки. Их четкое разделение по сферам ответственности способствует эффективной разработке и легкой поддержке системы. Разделение на отдельные модули помогает контролировать и обрабатывать большие объемы данных, поддерживать высокий уровень производительности и обеспечивать устойчивую работоспособность приложения.

1. ***API-интерфейс (Blazor UI)***

Веб-интерфейс реализован с использованием технологии Blazor Server, предоставляющей удобный и интуитивно понятный интерфейс взаимодействия с пользователем. Отличительной особенностью Blazor Server является полное отсутствие явного использования JavaScript на стороне браузера: взаимодействие с сервером происходит исключительно средствами C#, а передача данных осуществляется через двунаправленные соединения SignalR, что обеспечивает быструю и плавную работу без необходимости перезагрузки страницы.

Ниже приведен код Account.razor, реализующий интерфейс профиля пользователя:

@page "/account"

@attribute [Authorize]

@rendermode InteractiveServer

@inject AuthenticationStateProvider AuthProvider

@inject IUserService UserService

@inject IBookLoanService BookLoanService

@inject NavigationManager Navigation

<PageTitle>Мой аккаунт</PageTitle>

<AuthorizeView Roles="Admin">

<Authorized>

<AdminDashboard />

</Authorized>

</AuthorizeView>

<AuthorizeView Roles="Librarian">

<Authorized>

<LibrarianDashboard />

</Authorized>

</AuthorizeView>

<AuthorizeView Roles="User">

<Authorized>

<UserDashboard />

</Authorized>

</AuthorizeView>

@code {

private string? userId;

protected override async Task OnInitializedAsync()

{

var authState = await AuthProvider.GetAuthenticationStateAsync();

userId = authState.User.FindFirst(ClaimTypes.NameIdentifier)?.Value;

}

}

Страница пользователя меняется в зависимости от роли: для читателей – отображение списка взятых книг и списка забронированных книг и выполнение действий с ними, для библиотекаря – отслеживание выданных книг и списка забронированных книг, реализация выдачи и возврата книги, для администратора – переход на страницу управления ролей.

Основные особенности архитектуры:

* Используется Entity Framework Core, что обеспечивает единый механизм доступа к данным и управление транзакциями.
* Бизнес-логика отделена от хранения данных и представления, что способствует простоте сопровождения и модификации отдельных компонентов системы.
* Доступ к базе данных организован через специализированные сервисы, что облегчает тестирование и модификацию системы.
* Модули и сервисы спроектированы таким образом, чтобы минимизировать дублирование кода и обеспечить возможность повторного использования.

Данная архитектура позволяет создать стабильную, удобную и эффективную систему управления библиотечным каталогом, удовлетворяющую потребности широкого круга пользователей.

**3 Руководство пользователя**

При запуске программы пользователь увидит экран, приведенный на рисунке 3.1

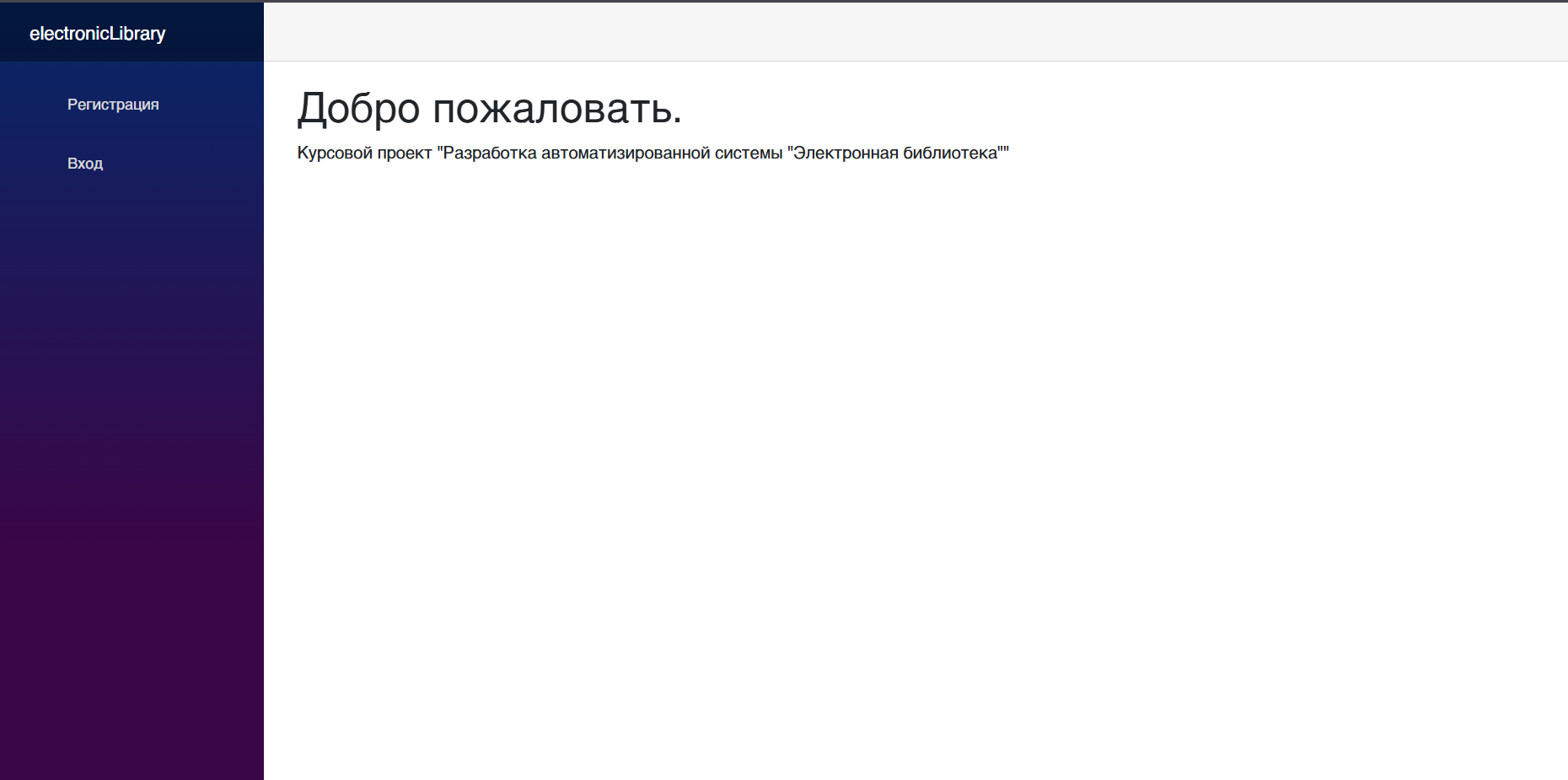


Рисунок 3.1 – Старт программы

Для дальнейшей работы в системе необходима авторизация через вкладку «Вход» (рисунок 3.2).

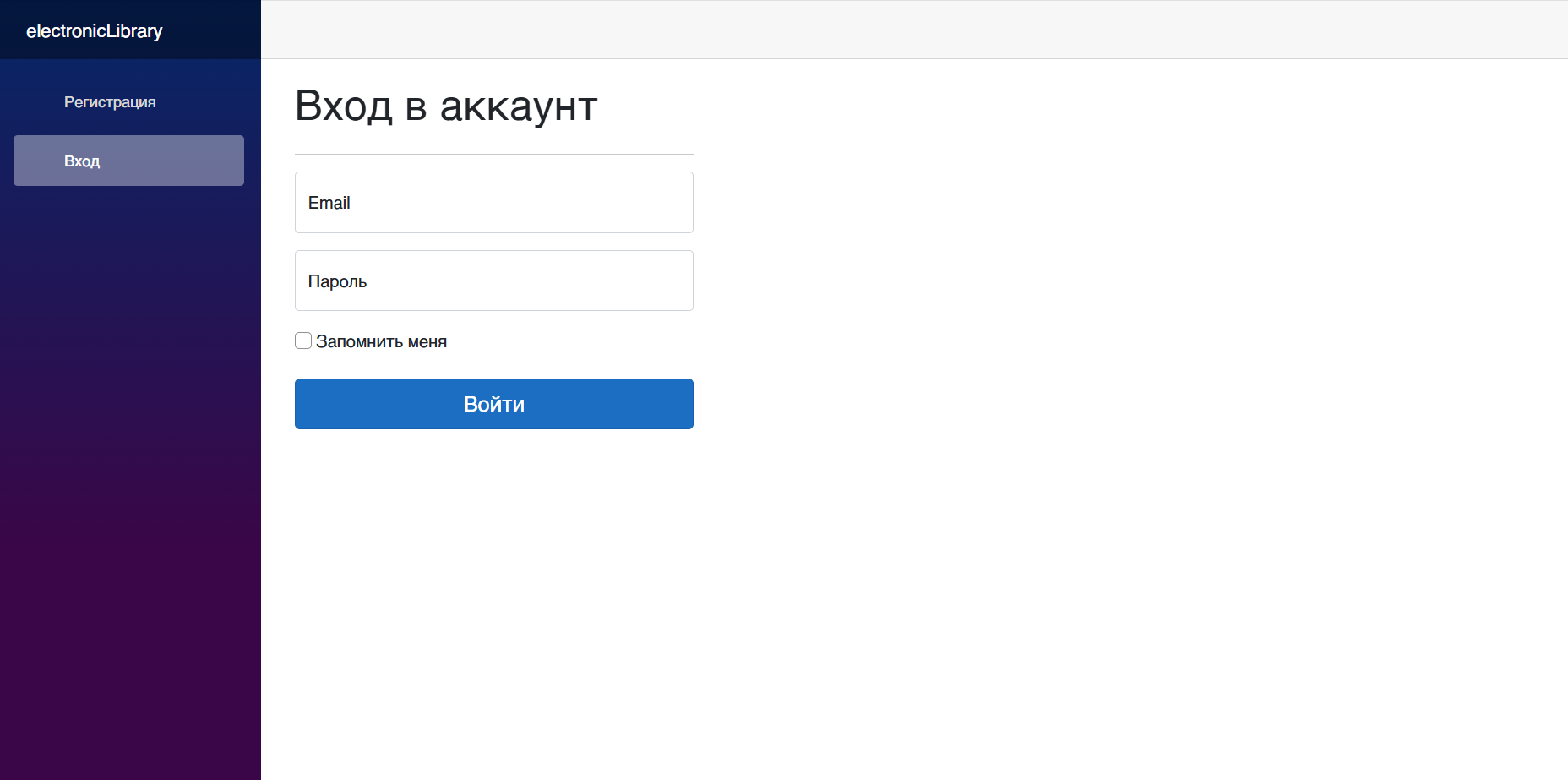


Рисунок 3.2 – Вход в существующий аккаунт

Если у пользователя нет аккаунта, он может его создать во вкладке «Регистрация» (рисунок 3.3).

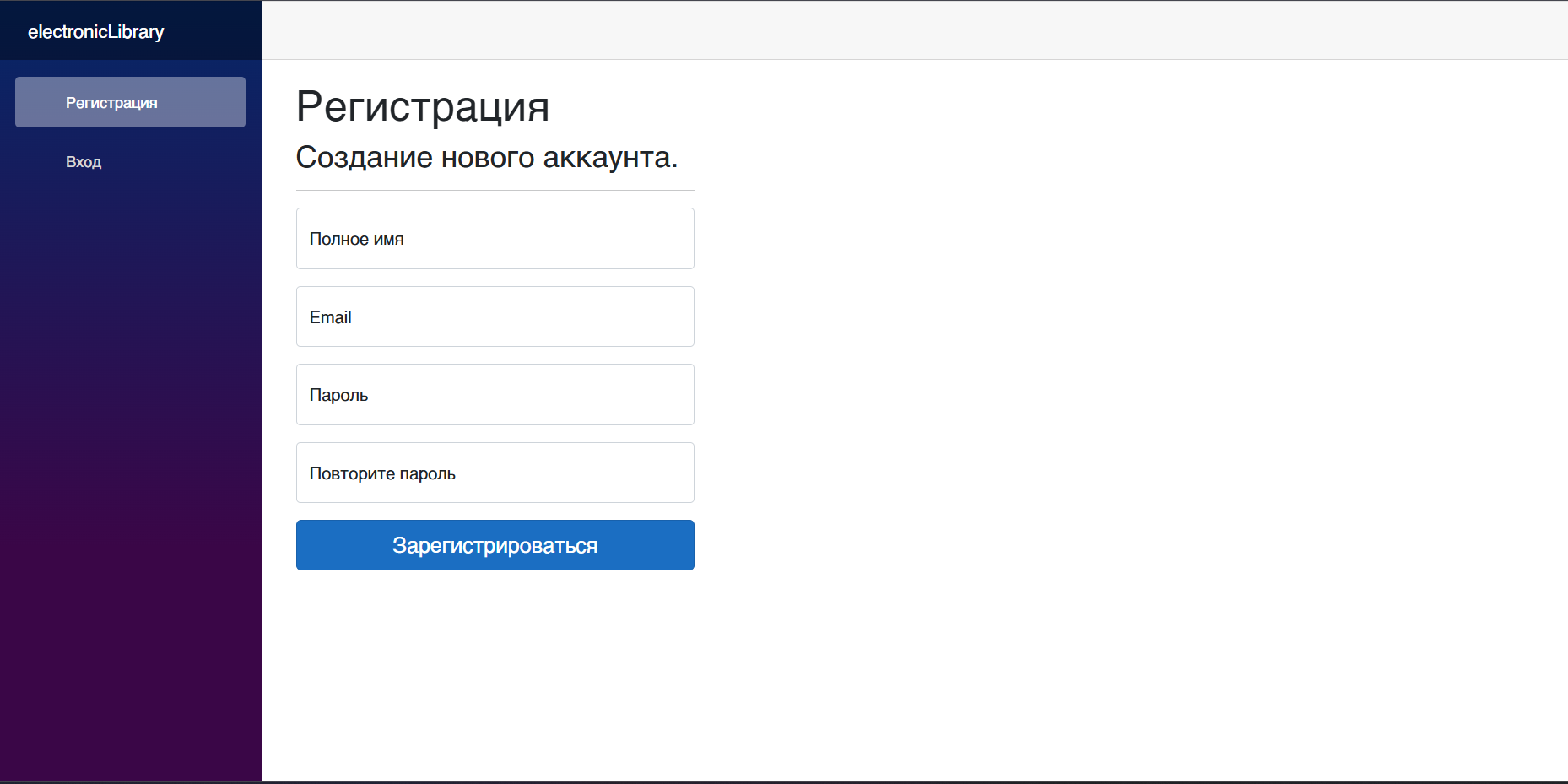


Рисунок 3.3 – Регистрация нового аккаунта

**3.1 Руководство пользователя – администратора**

При авторизации аккаунта с ролью администратора пользователю становится доступен просмотр информации о книгах, авторах и жанрах (рисунок 3.4)

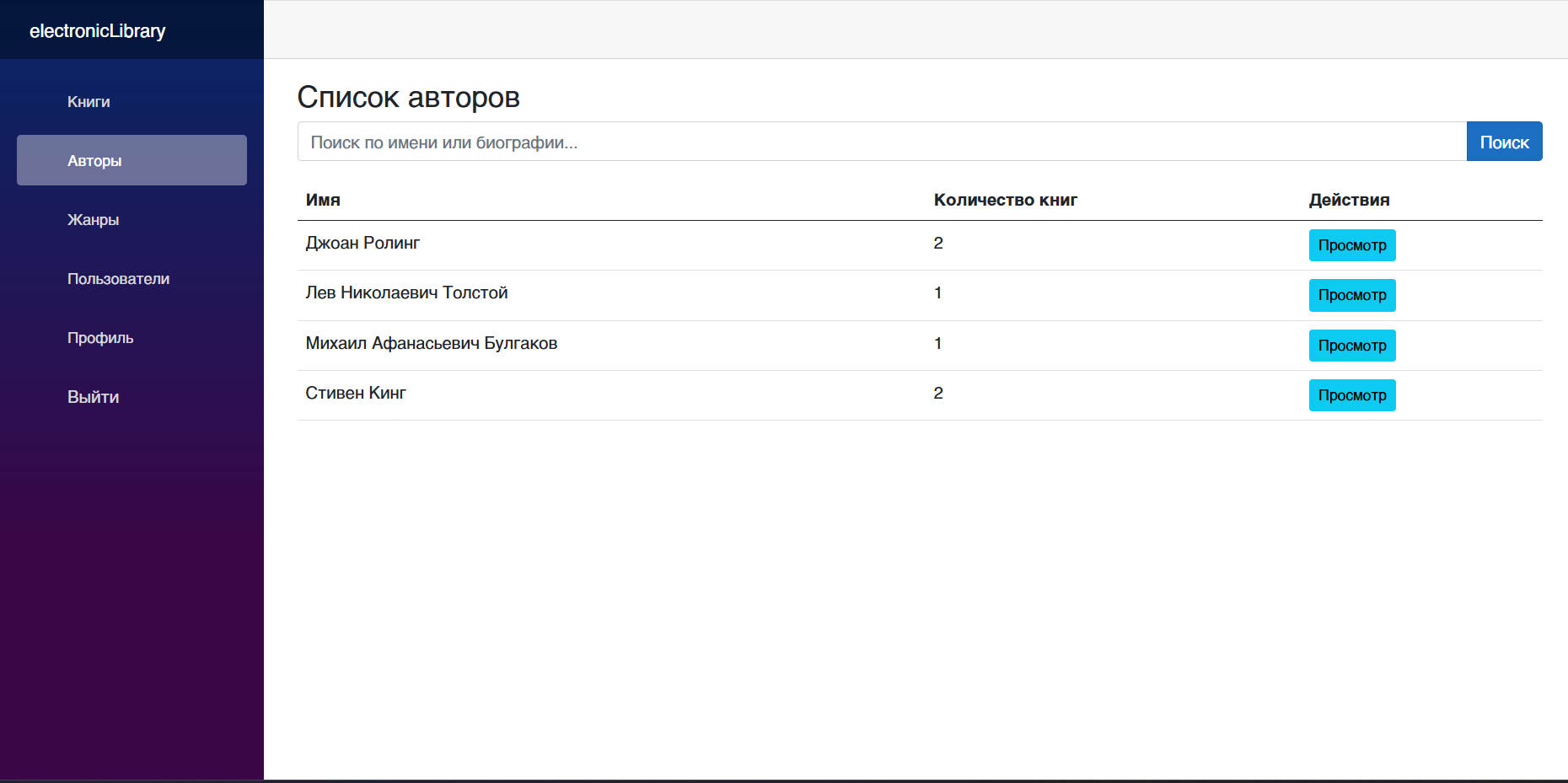


Рисунок 3.4 – Просмотр информации об авторах

На рисунке 3.5 показана карточка автора, содержащая подробную информацию, которая открывается при нажатии на кнопку «Просмотр» в строке с именем автора (аналогично для книг и жанров).

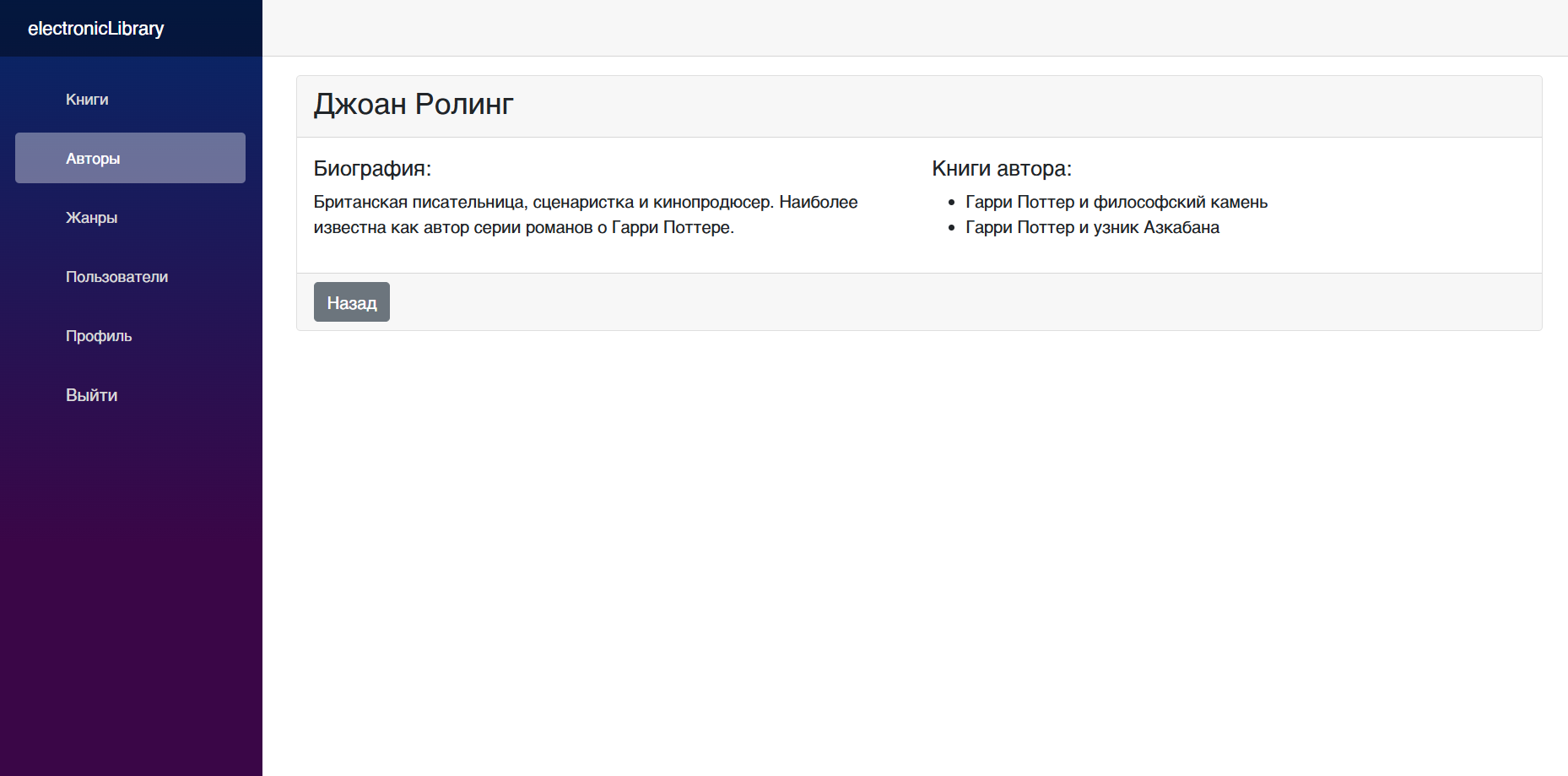


Рисунок 3.5 – Просмотр карточки автора

Также доступен поиск определенного автора по имени (рисунок 3.6).

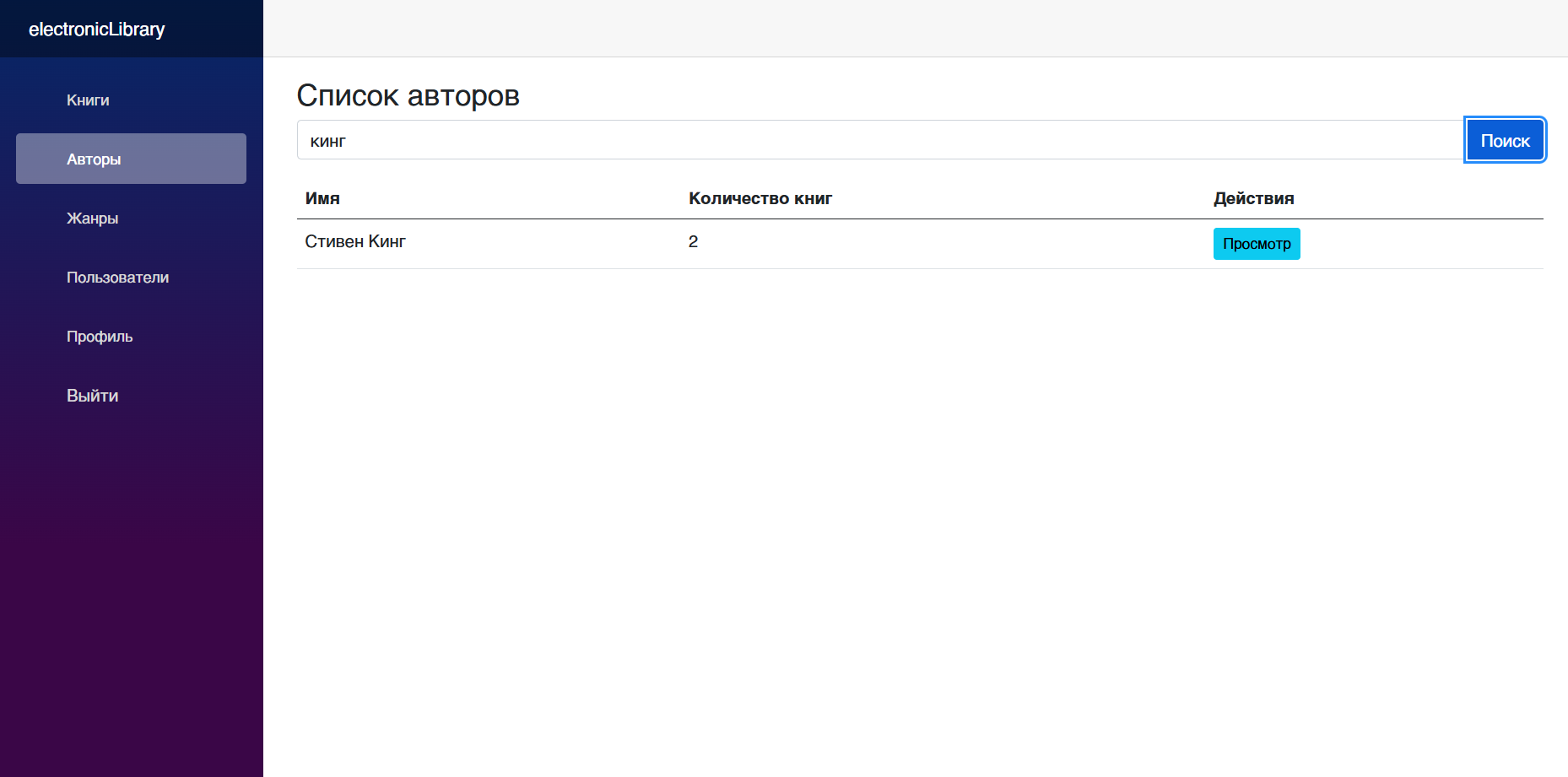


Рисунок 3.6 – Поиск автора «Кинг»

Основной функцией администратора является управление пользователями (рисунок 3.7).

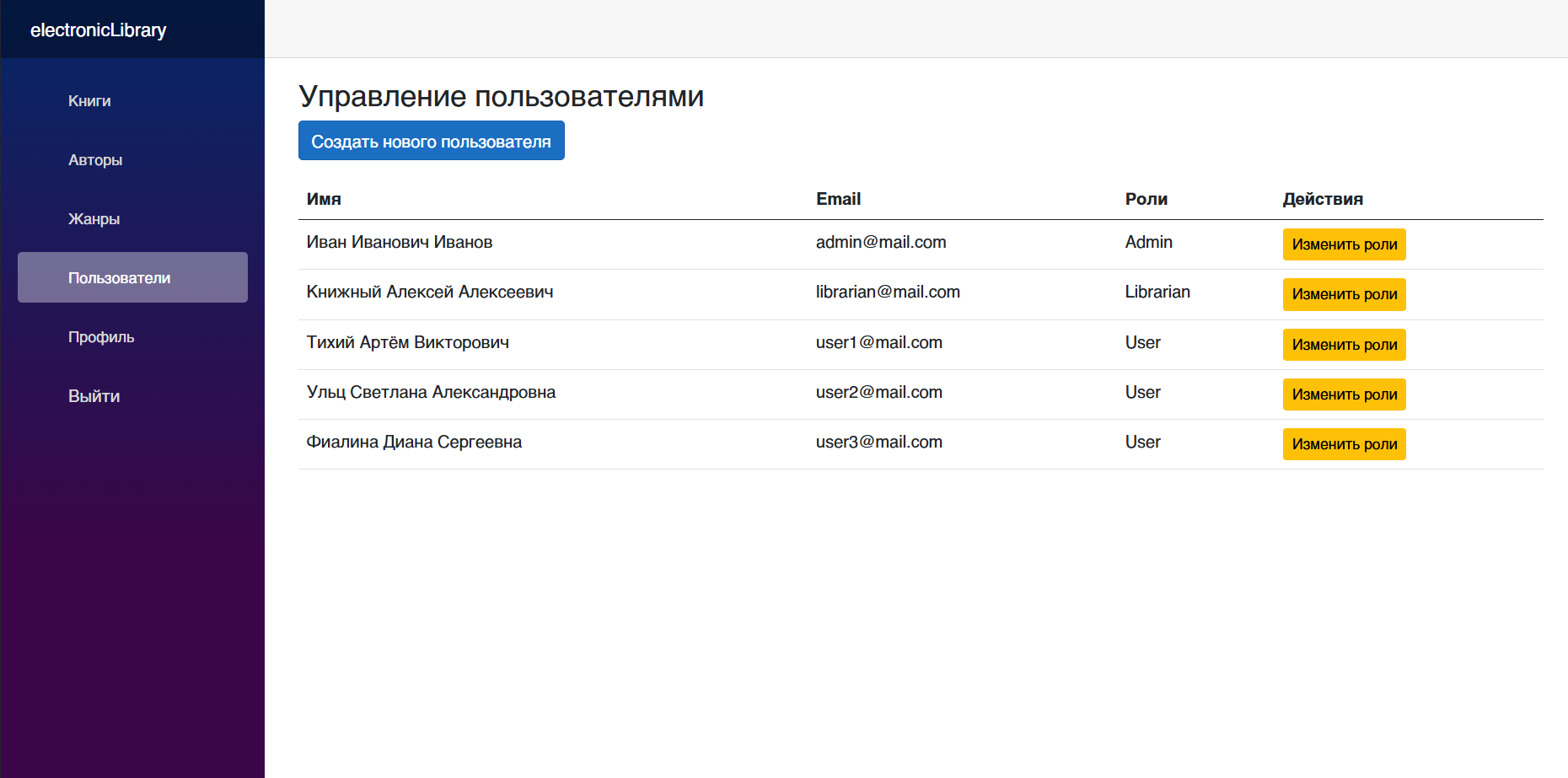


Рисунок 3.7 – Панель управления пользователями

Администратор может создать нового пользователя (рисунок 3.8). или изменить роль у существующих пользователей (рисунок 3.9).

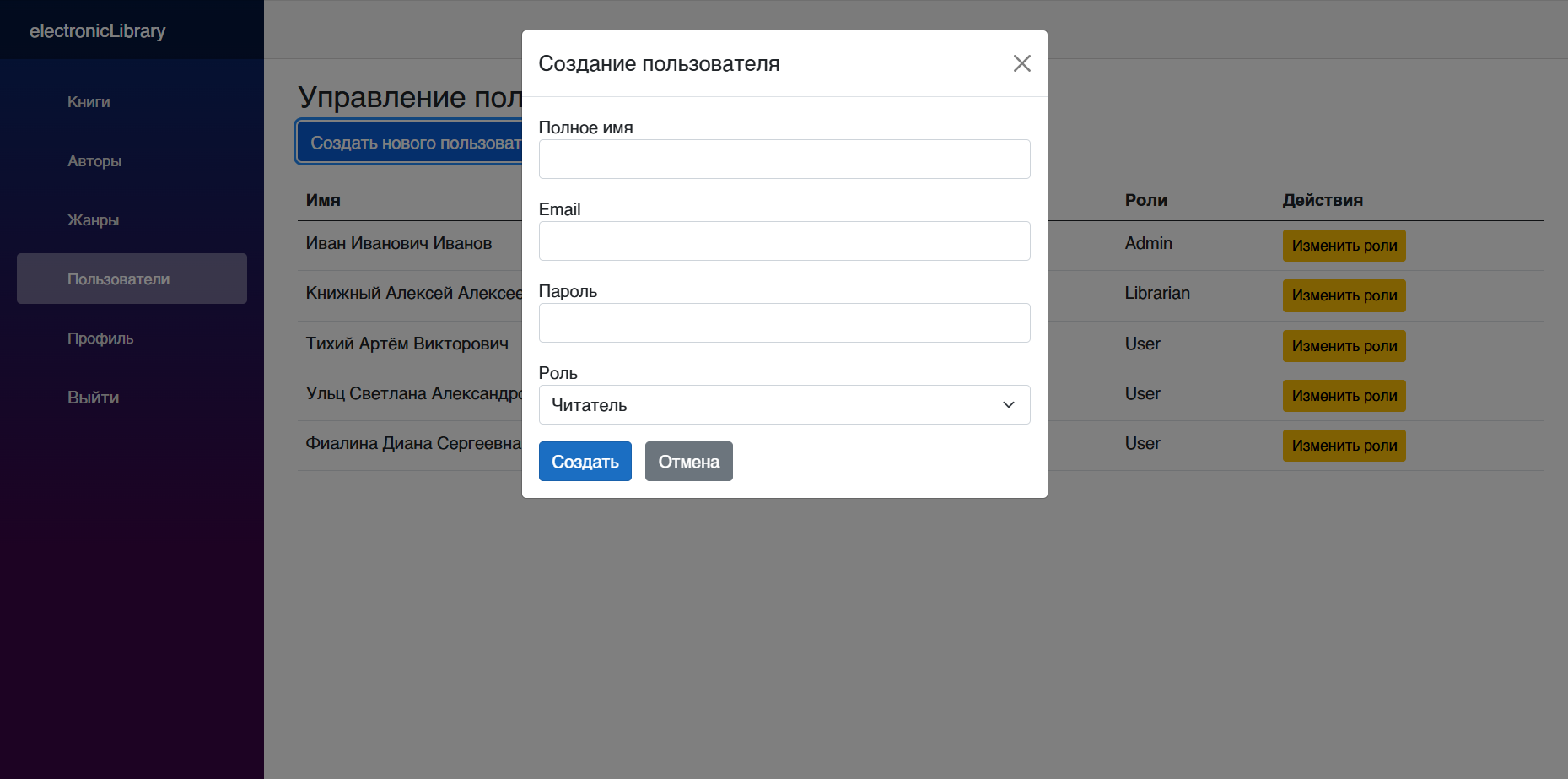


Рисунок 3.8 – Создание нового пользователя

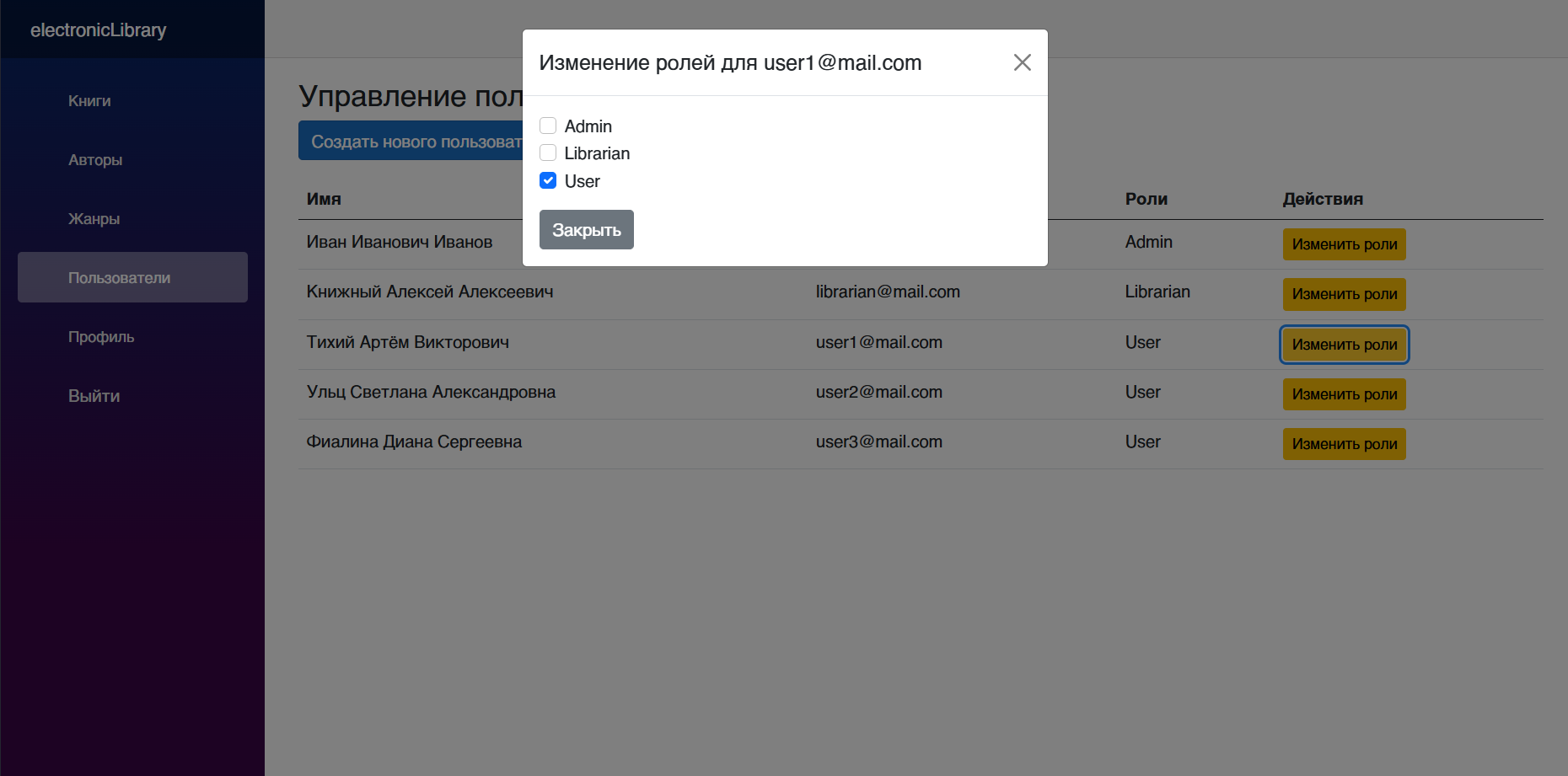


Рисунок 3.9 – Изменение роли существующего пользователя

**3.2 Руководство пользователя – библиотекаря**

При авторизации аккаунта с ролью «библиотекарь» пользователю становится доступно управление информацией о книгах, авторах и жанрах (рисунок 3.10).

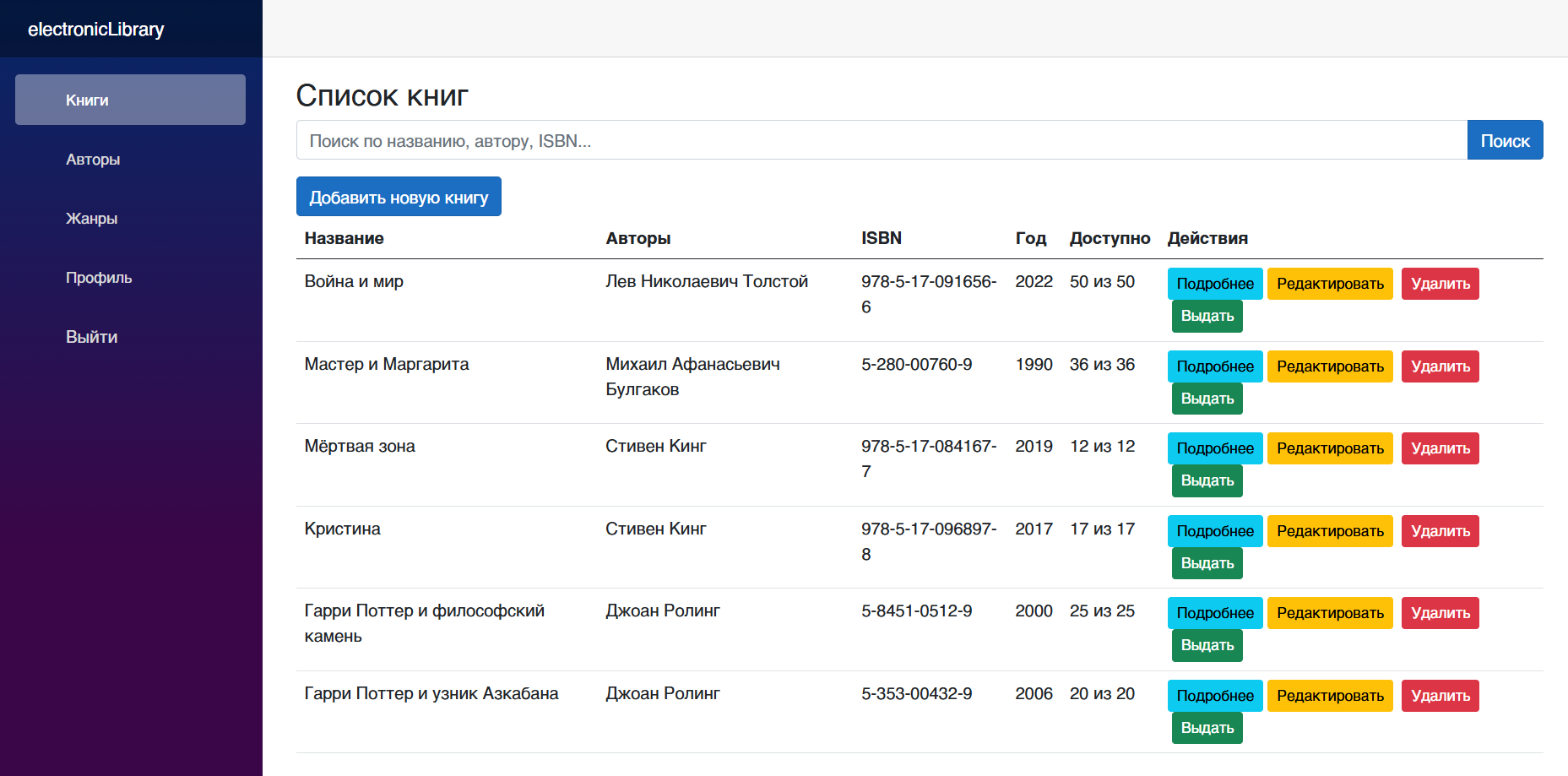


Рисунок 3.10 – Список книг для библиотекаря

Библиотекарь также как и остальные пользователи может просматривать подробную информацию (рисунок 3.11).

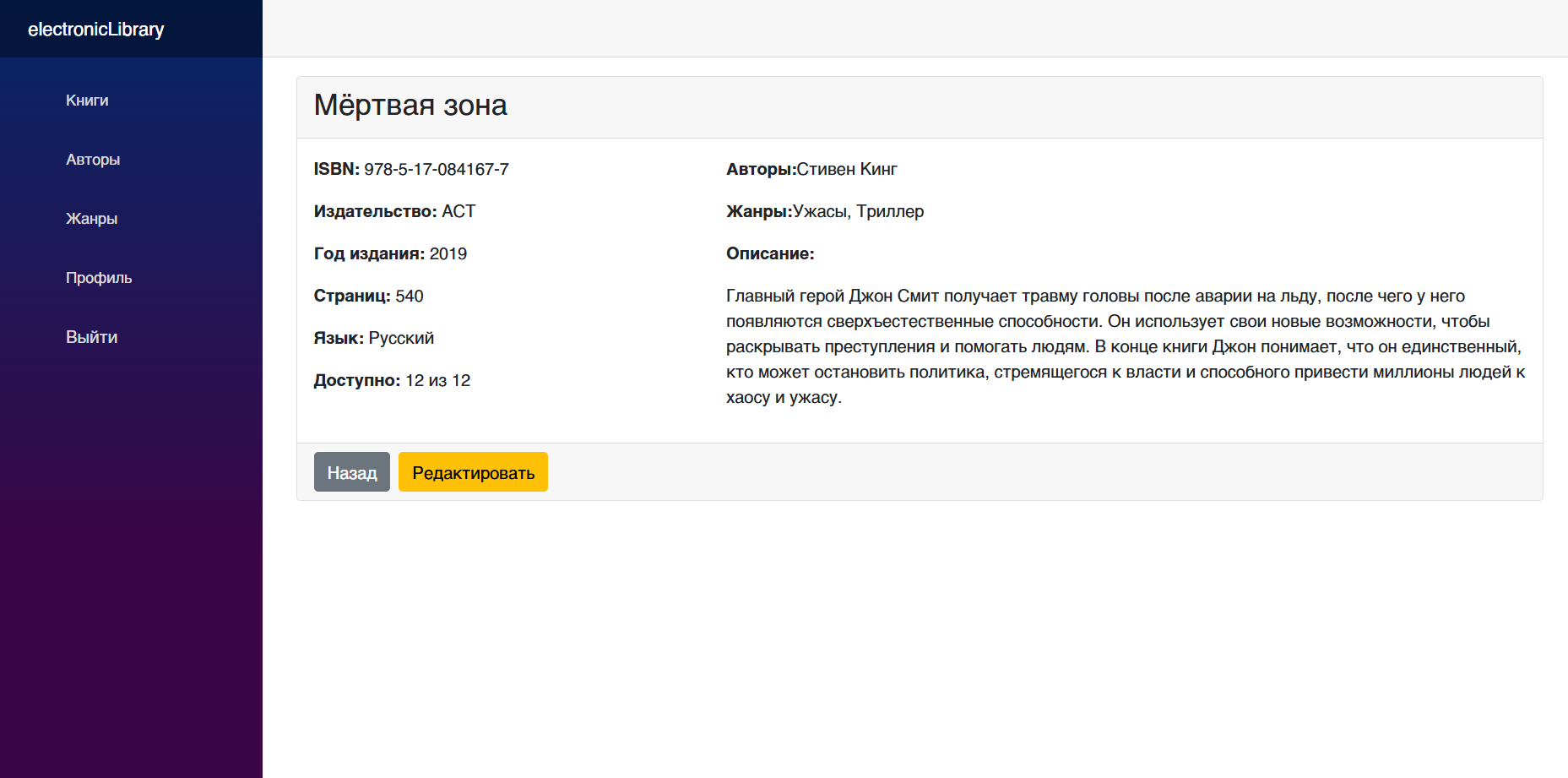


Рисунок 3.11 – Подробная информация о книге

Из карточки книги, также как и из списка всех книг можно перейти на страницу редактирования текущей книги (рисунок 3.12).

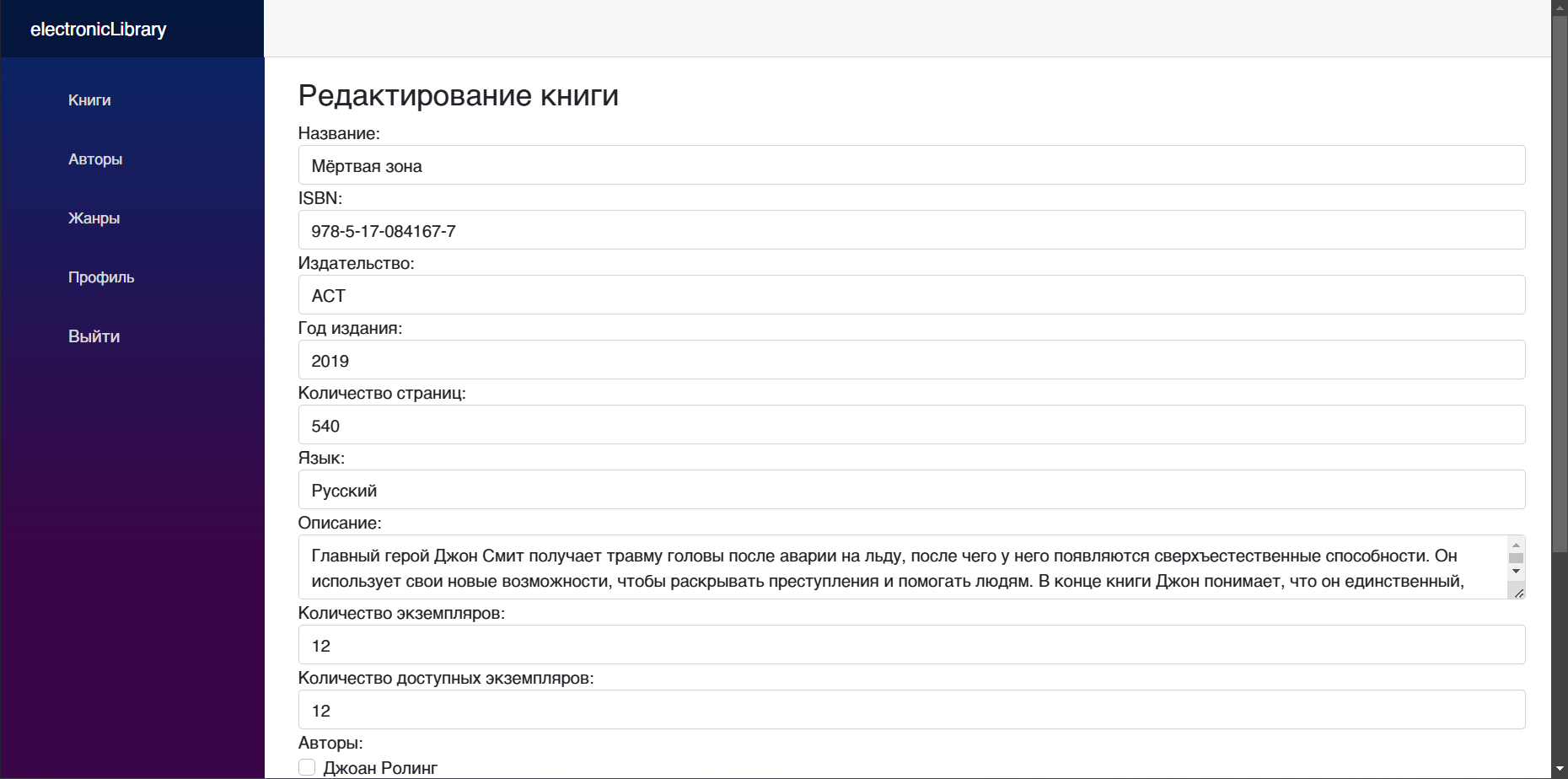


Рисунок 3.12 – Редактирование информации о книге

У пользователя с ролью «библиотекарь» есть возможность создать добавить новую книгу в библиотеку. Форма добавления новой книги аналогична форме редактирования, перейти на нее можно из списка всех книг (рисунок 3.10) при нажатии на кнопку «Добавить новую книгу».

Библиотекарь может выдавать книги определенным пользователям. При нажатии на кнопку выдать откроется окно выдачи книги (рисунок 3.13).

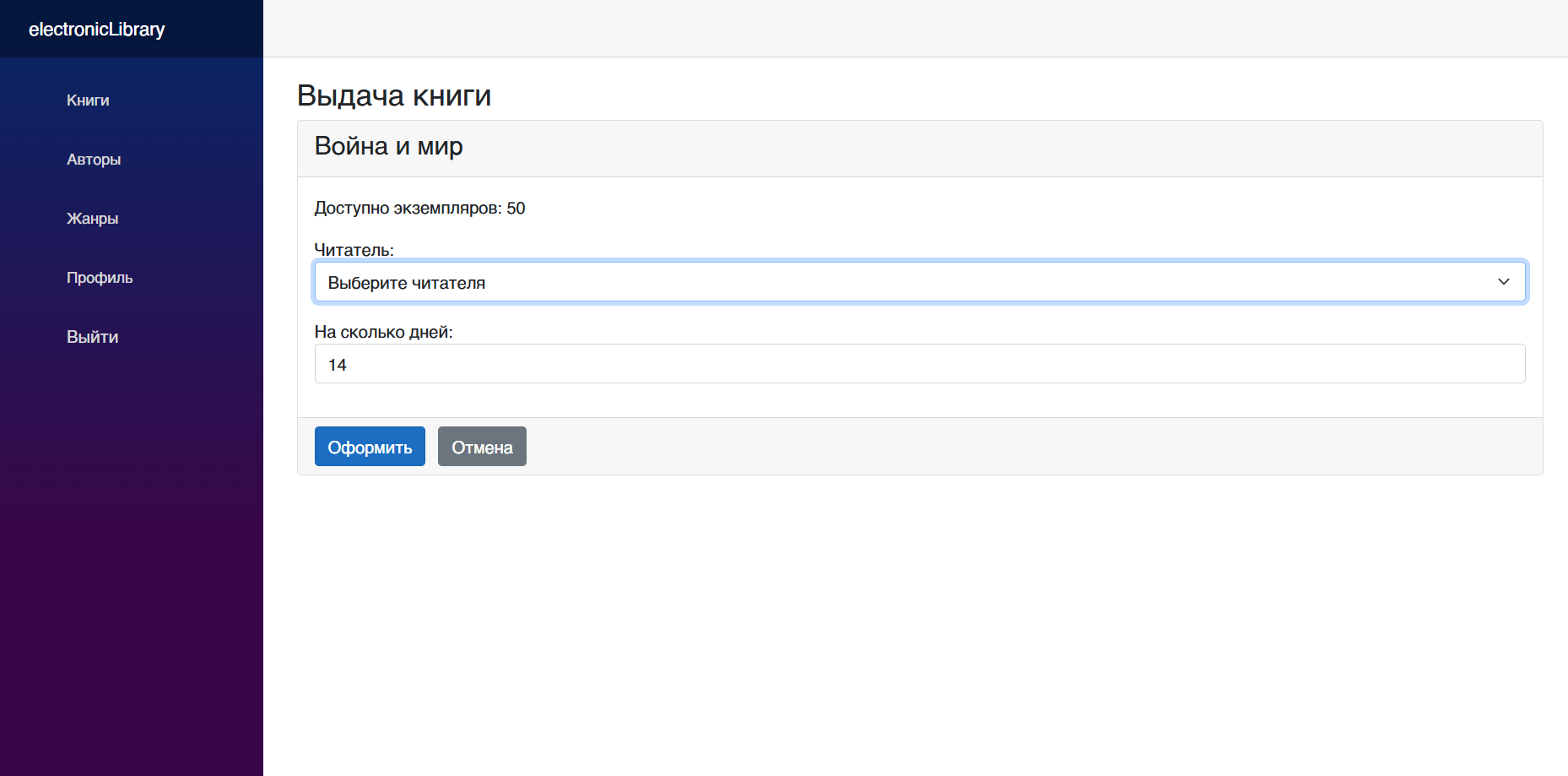


Рисунок 3.13 – Форма выдачи книги пользователю

При переходе на вкладку профиль библиотекарю становятся доступны списки выданных книг и забронированных книг. Данные списки можно отфильтровать по пользователю, а для списка выданных книг можно отследить просроченные (рисунок 3.14).

Забронированные книги можно выдать пользователю либо отменить бронь книги.

Взятые книги можно вернуть в библиотеку или продлить срок нахождения книги у пользователя.

На рисунке 3.15 приведены списки активных броней и активных займов для пользователя.

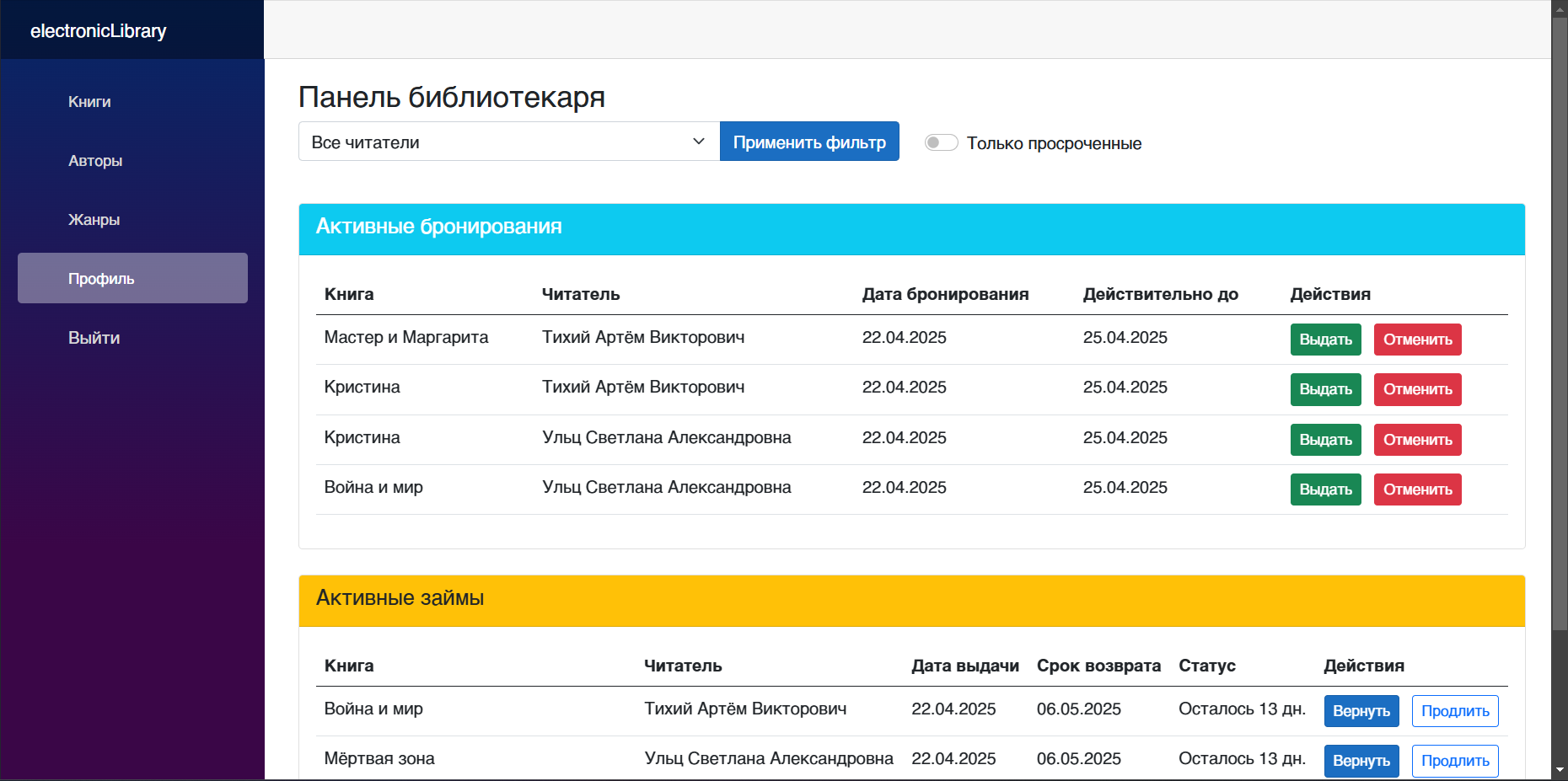


Рисунок 3.14 – Панель библиотекаря

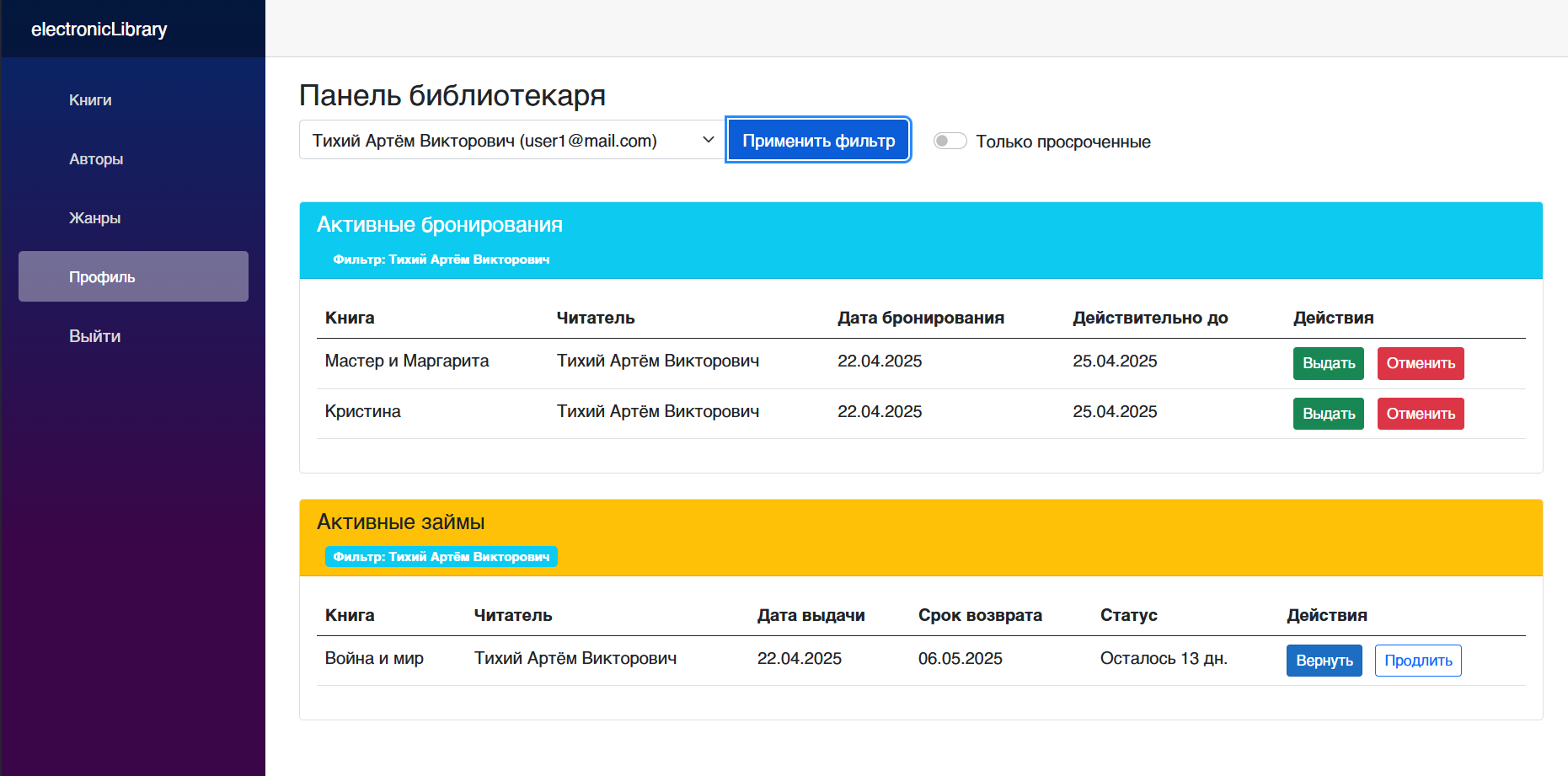


Рисунок 3.15 – Панель библиотекаря с примененным фильтром

**3.3 Руководство пользователя – читателя**

Также как и администратор и библиотекарь, читатель может просматривать информацию о книгах, жанрах и авторах. Отличительной функцией для читателя является бронирование книги (рисунок 3.16).

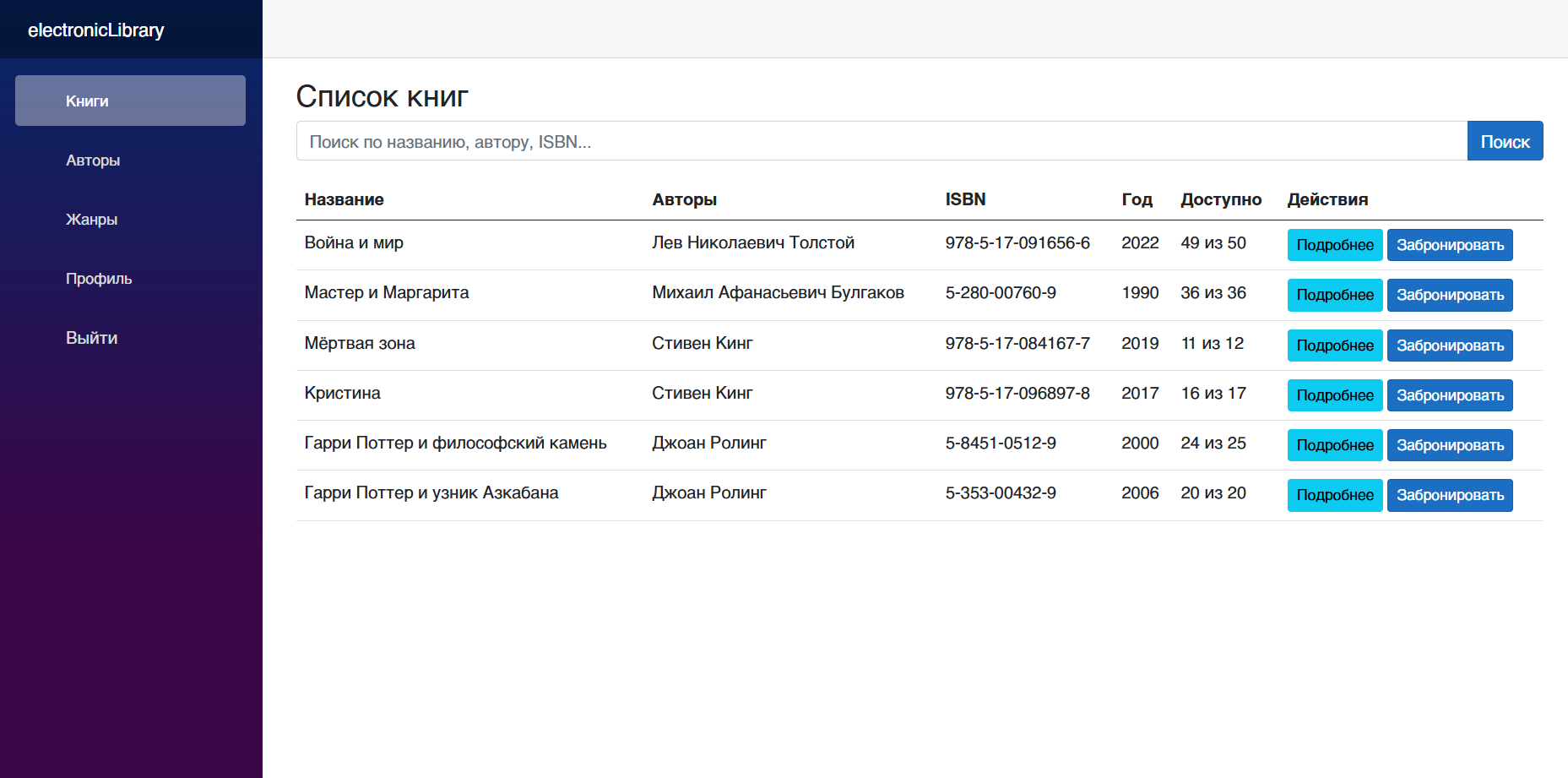


Рисунок 3.16 – Список книг с возможностью бронирования книги

При нажатии на кнопку «Забронировать» откроется оно бронирования книги, где можно указать срок действия брони, который не может быть больше 7 дней (рисунок 3.17).

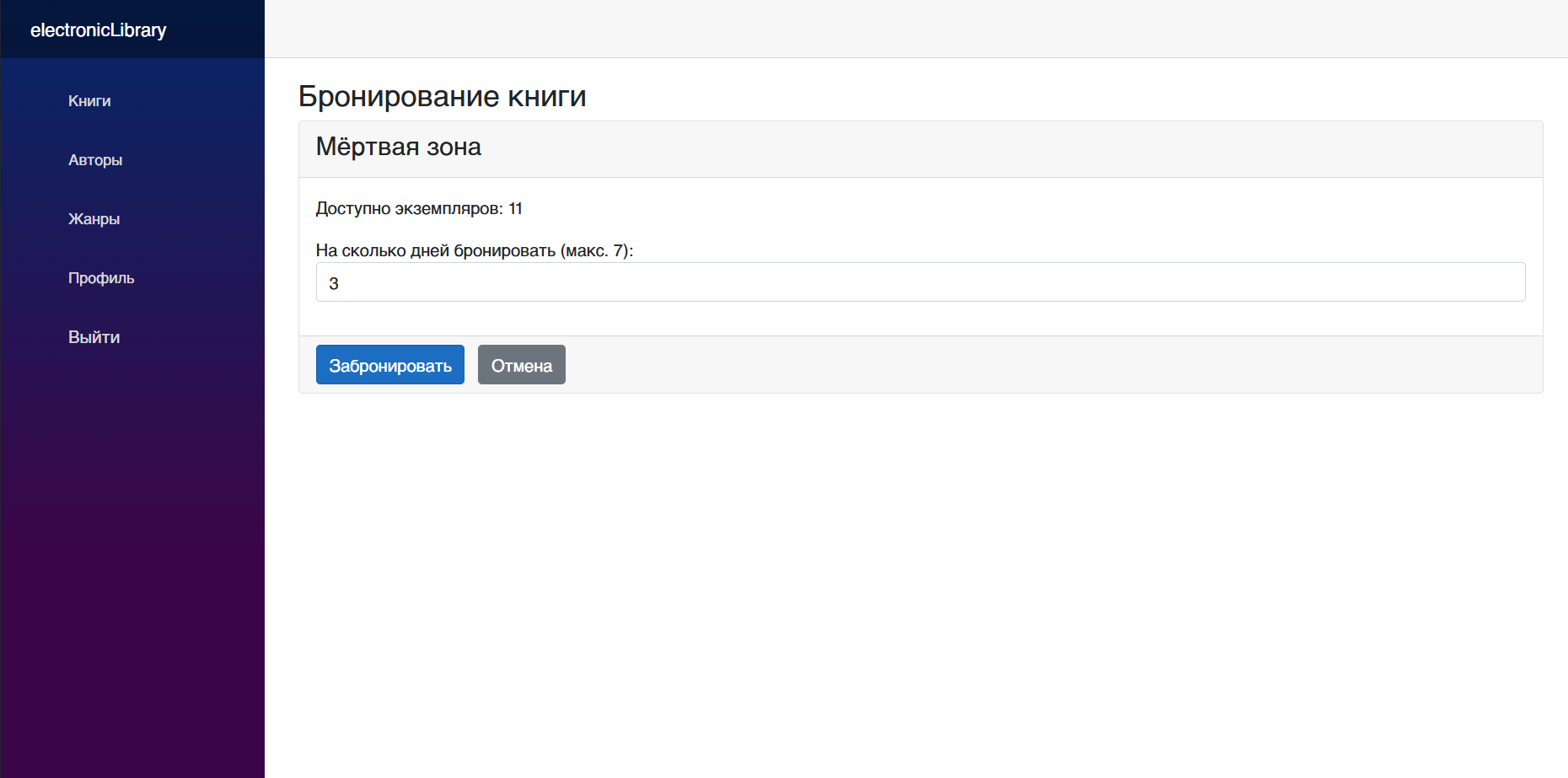


Рисунок 3.17 – Форма бронирования книги

В профиле пользователь может увидеть свои взятые и забронированные книги. У взятых книг допускается продление срока. Активные брони можно отменить (рисунок 3.18).

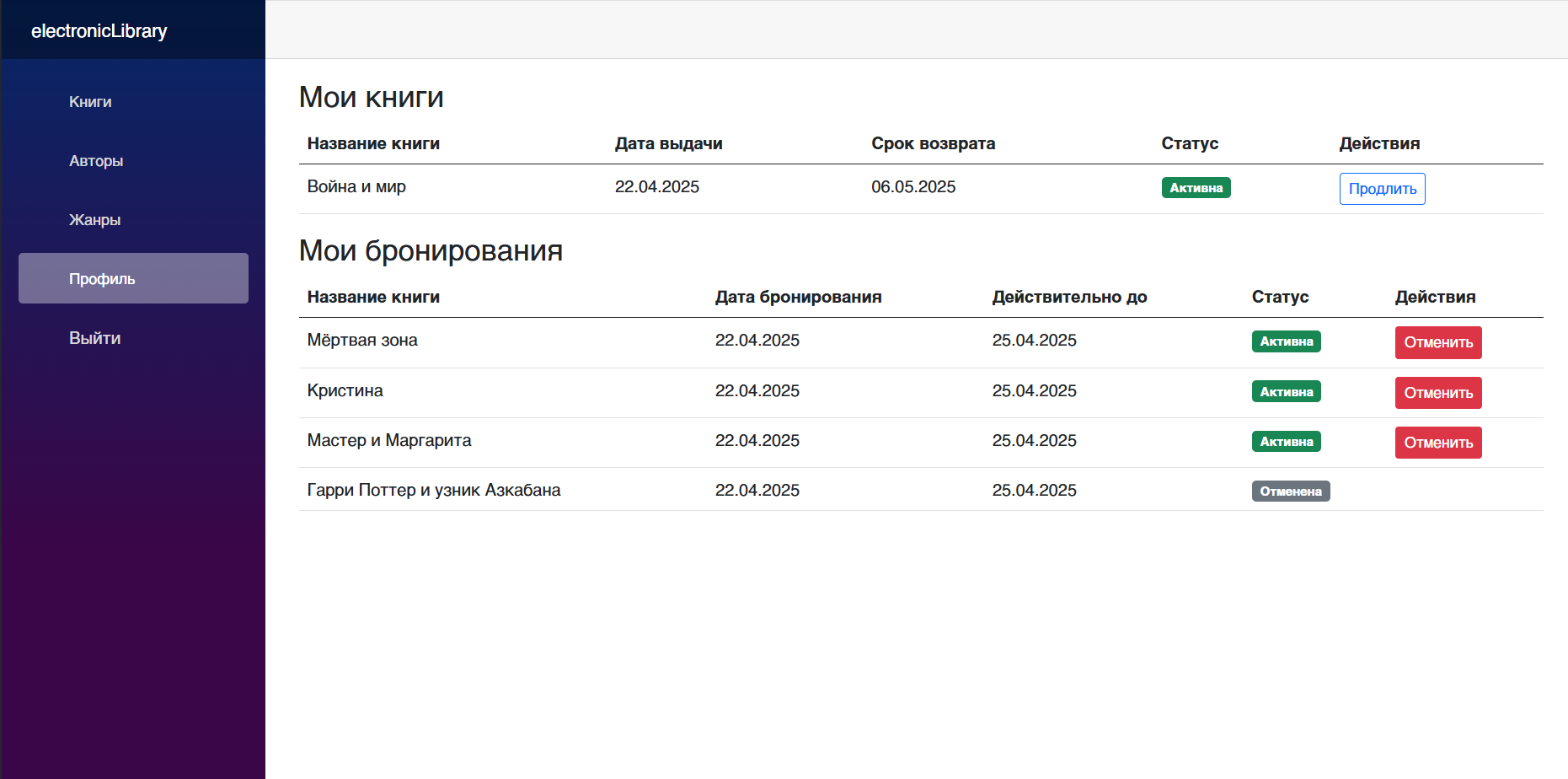


Рисунок 3.18 – Профиль читателя

**ЗАКЛЮЧЕНИЕ**

В ходе выполнения курсового проекта была разработана автоматизированная система «Электронная библиотека», реализующая эффективное управление традиционными библиотечными ресурсами.

Проведённый анализ существующих АБИС позволил определить функциональные и технические требования к разрабатываемой системе, на основе которых была спроектирована архитектура и база данных.

Реализованные с помощью платформы Blazor модули учета книг, управления пользователями, поиска и выдачи материалов обеспечивают удобный интерфейс и автоматизацию ключевых библиотечных процессов. Проведенное тестирование подтвердило работоспособность системы и её соответствие заявленным требованиям.

Разработанный прототип системы демонстрирует высокую практическую значимость, позволяя оптимизировать работу библиотек, повышать доступность информации и сокращать временные затраты на обслуживание пользователей, что подтверждает актуальность данной работы.

Результаты проекта могут быть использованы для дальнейшего развития и внедрения полноценной автоматизированной библиотечной информационной системы, отвечающей современным требованиям к эффективности и удобству работы с библиотечными фондами.

Таким образом, курсовая работа достигла поставленных целей и выполнила сформулированные задачи, продемонстрировав перспективность предложенного решения и открыв возможности для дальнейшего научного и прикладного исследования в области автоматизации библиотечных услуг.

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. Колкова, Н. И., Скипор, И. Л. Проектирование автоматизированных библиотечно-информационных систем : учебник для студентов направления подготовки «Библиотечно-информационная деятельность», профиль подготовки «Технология автоматизированных библиотечно-информационных систем», квалификация (степень) «бакалавр» / Н. И. Колкова, И. Л. Скипор – Кемерово: гос. ин-т культуры, 2020. - 382 с. - Текст : непосредственный. (дата обращения: 05.04.2025)
2. Рахматуллаев, М. А. Проектирование информационно-библиотечных систем : учебник / М.А. Рахматуллаев. - Москва : ИНФРА-М, 2024. - 287 с. - Текст : непосредственный. (дата обращения: 03.04.2025).
3. Сэйнти, К. Blazor в действии : руководство / К. Сэйнти ; перевод с английского Д. А. Беликовой. - Москва : ДМК Пресс, 2023. - 380 с. - Текст : электронный. (дата обращения: 10.04.2025).
4. Умрихин, Е.Д. Разработка веб-приложений с помощью Blazor : учебное пособие / Е.Д. Умрихин. - Санкт-Петербург : БХВ, 2025. - 400 с. - (Профессиональное программирование). - Текст : непосредственный. (дата обращения: 08.04.2025).
5. Blazor | Руководство : [сайт]. – URL: https://metanit.com/sharp/blazor/ (дата обращения: 06.04.2025) - Текст : электронный.

**ПРИЛОЖЕНИЕ А**

**Реализация модели данных**

**Файл Author.cs**

using System.ComponentModel.DataAnnotations;

namespace electronicLibrary.Data.Models

{

public class Author

{

public int Id { get; set; }

public string? FullName { get; set; }

public string? Biography { get; set; }

public List<BookAuthor> BookAuthors { get; set; } = new();

}

}

**Файл Book.cs**

using System.ComponentModel.DataAnnotations;

namespace electronicLibrary.Data.Models

{

public class Book

{

public int Id { get; set; }

public string Title { get; set; }

public string ISBN { get; set; }

public string? Publisher { get; set; }

public int Year { get; set; }

public int Pages { get; set; }

public string? Language { get; set; }

public string? Description { get; set; }

public int TotalCopies { get; set; }

public int AvailableCopies { get; set; }

public List<BookAuthor> BookAuthors { get; set; } = new();

public List<BookGenre> BookGenres { get; set; } = new();

public List<BookLoan> BookLoans { get; set; } = new();

public List<BookReservation> BookReservations { get; set; } = new();

}

}

**Файл Genre.cs**

using System.ComponentModel.DataAnnotations;

namespace electronicLibrary.Data.Models

{

public class Genre

{

public int Id { get; set; }

public string? Name { get; set; }

public string? Description { get; set; }

public List<BookGenre> BookGenres { get; set; } = new();

}

}

**Файл BookAuthor.cs**

namespace electronicLibrary.Data.Models

{

public class BookAuthor

{

public int BookId { get; set; }

public Book? Book { get; set; }

public int AuthorId { get; set; }

public Author? Author { get; set; }

}

}

**Файл BookGenre.cs**

namespace electronicLibrary.Data.Models

{

public class BookGenre

{

public int BookId { get; set; }

public Book? Book { get; set; }

public int GenreId { get; set; }

public Genre? Genre { get; set; }

}

}

**Файл ApplicationUser.cs**

using electronicLibrary.Data.Models;

using Microsoft.AspNetCore.Identity;

namespace electronicLibrary.Data

{

public class ApplicationUser : IdentityUser

{

public string? FullName { get; set; }

public DateTime RegistrationDate { get; set; } = DateTime.UtcNow;

public List<BookLoan> BookLoans { get; set; } = [];

public List<BookReservation> BookReservations { get; set; } = [];

}

}

**Файл BookLoan.cs**

using System.ComponentModel.DataAnnotations.Schema;

namespace electronicLibrary.Data.Models

{

public class BookLoan

{

public int Id { get; set; }

public int BookId { get; set; }

public Book? Book { get; set; }

public string? UserId { get; set; }

public ApplicationUser? User { get; set; }

private DateTime \_loanDate;

public DateTime LoanDate

{

get => \_loanDate;

set => \_loanDate = value <= DateTime.Now ? value : throw new ArgumentException("Loan date cannot be in the future");

}

private DateTime? \_returnDate;

public DateTime? ReturnDate

{

get => \_returnDate;

set => \_returnDate = value > LoanDate ? value : throw new ArgumentException("Return date cannot be before loan date");

}

public DateTime DueDate { get; set; }

[NotMapped]

public bool IsOverdue => DateTime.Now > DueDate && !ReturnDate.HasValue;

}

}

**Файл BookReservation.cs**

namespace electronicLibrary.Data.Models

{

public class BookReservation

{

public int Id { get; set; }

public int BookId { get; set; }

public Book? Book { get; set; }

public string? UserId { get; set; }

public ApplicationUser? User { get; set; }

public DateTime ReservationDate { get; set; } = DateTime.UtcNow;

public DateTime ExpiryDate { get; set; }

public bool IsActive { get; set; } = true;

}

}

**Файл ApplicationDbContext.cs**

using electronicLibrary.Data.Models;

using Microsoft.AspNetCore.Identity;

using Microsoft.AspNetCore.Identity.EntityFrameworkCore;

using Microsoft.EntityFrameworkCore;

using System.Reflection.Emit;

namespace electronicLibrary.Data

{

public class ApplicationDbContext(DbContextOptions<ApplicationDbContext> options) : IdentityDbContext<ApplicationUser, IdentityRole, string>(options)

{

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

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

public new DbSet<ApplicationUser> Users { get; set; }

public DbSet<Genre> Genres { get; set; }

public DbSet<BookLoan> BookLoans { get; set; }

public DbSet<BookAuthor> BookAuthors { get; set; }

public DbSet<BookGenre> BookGenres { get; set; }

public DbSet<BookReservation> BookReservations { get; set; }

protected override void OnModelCreating(ModelBuilder builder)

{

base.OnModelCreating(builder);

builder.Entity<BookLoan>(entity =>

{

entity.HasOne(bl => bl.User)

.WithMany(u => u.BookLoans)

.HasForeignKey(bl => bl.UserId)

.OnDelete(DeleteBehavior.Cascade);

entity.HasOne(bl => bl.Book)

.WithMany(b => b.BookLoans)

.HasForeignKey(bl => bl.BookId)

.OnDelete(DeleteBehavior.Cascade);

});

builder.Entity<BookReservation>(entity =>

{

entity.HasOne(br => br.User)

.WithMany(u => u.BookReservations)

.HasForeignKey(br => br.UserId)

.OnDelete(DeleteBehavior.Cascade);

entity.HasOne(br => br.Book)

.WithMany(b => b.BookReservations)

.HasForeignKey(br => br.BookId)

.OnDelete(DeleteBehavior.Cascade);

});

// Настройка связей многие-ко-многим (Book - Author)

builder.Entity<BookAuthor>()

.HasKey(ba => new { ba.BookId, ba.AuthorId });

builder.Entity<BookAuthor>()

.HasOne(ba => ba.Book)

.WithMany(b => b.BookAuthors)

.HasForeignKey(ba => ba.BookId);

builder.Entity<BookAuthor>()

.HasOne(ba => ba.Author)

.WithMany(a => a.BookAuthors)

.HasForeignKey(ba => ba.AuthorId);

// Настройка связей многие-ко-многим (Book - Genre)

builder.Entity<BookGenre>()

.HasKey(bg => new { bg.BookId, bg.GenreId });

builder.Entity<BookGenre>()

.HasOne(bg => bg.Book)

.WithMany(b => b.BookGenres)

.HasForeignKey(bg => bg.BookId);

builder.Entity<BookGenre>()

.HasOne(bg => bg.Genre)

.WithMany(g => g.BookGenres)

.HasForeignKey(bg => bg.GenreId);

// Уникальные индексы

builder.Entity<Book>()

.HasIndex(b => b.ISBN)

.IsUnique();

}

}

}

**ПРИЛОЖЕНИЕ Б**

**Реализация сервисов**

**Файл BookService.cs**

using electronicLibrary.Data;

using electronicLibrary.Data.interfaces;

using electronicLibrary.Data.Models;

using Microsoft.EntityFrameworkCore;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

namespace electronicLibrary.Data.Services

{

public class BookService : IBookService

{

private readonly ApplicationDbContext \_context;

public BookService(ApplicationDbContext context)

{

\_context = context;

}

public async Task<List<Book>> GetBooksAsync()

{

return await \_context.Books

.Include(b => b.BookAuthors)

.ThenInclude(ba => ba.Author)

.Include(b => b.BookGenres)

.ThenInclude(bg => bg.Genre)

.AsNoTracking()

.ToListAsync();

}

public async Task<Book?> GetBookByIdAsync(int id)

{

return await \_context.Books

.Include(b => b.BookAuthors)

.ThenInclude(ba => ba.Author)

.Include(b => b.BookGenres)

.ThenInclude(bg => bg.Genre)

.AsNoTracking()

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

}

public async Task<List<Book>> SearchBooksAsync(string searchTerm)

{

if (string.IsNullOrWhiteSpace(searchTerm))

return await GetBooksAsync();

searchTerm = searchTerm.ToLower();

return await \_context.Books

.Include(b => b.BookAuthors)

.ThenInclude(ba => ba.Author)

.Include(b => b.BookGenres)

.ThenInclude(bg => bg.Genre)

.Where(b => b.Title.ToLower().Contains(searchTerm) ||

b.ISBN.ToLower().Contains(searchTerm) ||

b.Publisher.ToLower().Contains(searchTerm) ||

b.BookAuthors.Any(ba => ba.Author.FullName.ToLower().Contains(searchTerm)))

.AsNoTracking()

.ToListAsync();

}

public async Task AddBookAsync(Book book, List<int> authorIds, List<int> genreIds)

{

if (book == null) throw new ArgumentNullException(nameof(book));

if (await \_context.Books.AnyAsync(b => b.ISBN == book.ISBN))

throw new InvalidOperationException("Книга с таким ISBN уже существует");

await \_context.Books.AddAsync(book);

await \_context.SaveChangesAsync();

foreach (var authorId in authorIds)

{

if (!await \_context.Authors.AnyAsync(a => a.Id == authorId))

continue;

\_context.BookAuthors.Add(new BookAuthor

{

BookId = book.Id,

AuthorId = authorId

});

}

foreach (var genreId in genreIds)

{

if (!await \_context.Genres.AnyAsync(g => g.Id == genreId))

continue;

\_context.BookGenres.Add(new BookGenre

{

BookId = book.Id,

GenreId = genreId

});

}

await \_context.SaveChangesAsync();

}

public async Task UpdateBookAsync(Book book, List<int> authorIds, List<int> genreIds)

{

if (book == null) throw new ArgumentNullException(nameof(book));

var existingBook = await \_context.Books

.Include(b => b.BookAuthors)

.Include(b => b.BookGenres)

.FirstOrDefaultAsync(b => b.Id == book.Id);

if (existingBook == null)

throw new KeyNotFoundException("Книга не найдена");

if (await \_context.Books.AnyAsync(b => b.ISBN == book.ISBN && b.Id != book.Id))

throw new InvalidOperationException("Книга с таким ISBN уже существует");

\_context.Entry(existingBook).CurrentValues.SetValues(book);

UpdateBookAuthors(existingBook, authorIds);

UpdateBookGenres(existingBook, genreIds);

await \_context.SaveChangesAsync();

}

private void UpdateBookAuthors(Book book, List<int> authorIds)

{

var authorsToRemove = book.BookAuthors

.Where(ba => !authorIds.Contains(ba.AuthorId))

.ToList();

foreach (var authorToRemove in authorsToRemove)

{

\_context.BookAuthors.Remove(authorToRemove);

}

var existingAuthorIds = book.BookAuthors.Select(ba => ba.AuthorId).ToList();

var authorsToAdd = authorIds

.Where(id => !existingAuthorIds.Contains(id))

.Select(id => new BookAuthor { BookId = book.Id, AuthorId = id });

foreach (var authorToAdd in authorsToAdd)

{

\_context.BookAuthors.Add(authorToAdd);

}

}

private void UpdateBookGenres(Book book, List<int> genreIds)

{

var genresToRemove = book.BookGenres

.Where(bg => !genreIds.Contains(bg.GenreId))

.ToList();

foreach (var genreToRemove in genresToRemove)

{

\_context.BookGenres.Remove(genreToRemove);

}

var existingGenreIds = book.BookGenres.Select(bg => bg.GenreId).ToList();

var genresToAdd = genreIds

.Where(id => !existingGenreIds.Contains(id))

.Select(id => new BookGenre { BookId = book.Id, GenreId = id });

foreach (var genreToAdd in genresToAdd)

{

\_context.BookGenres.Add(genreToAdd);

}

}

public async Task DeleteBookAsync(int id)

{

var book = await \_context.Books

.Include(b => b.BookAuthors)

.Include(b => b.BookGenres)

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

if (book == null)

throw new KeyNotFoundException("Книга не найдена");

if (book.TotalCopies != book.AvailableCopies)

throw new InvalidOperationException("Нельзя удалить книгу, так как есть выданные экземпляры");

\_context.Books.Remove(book);

await \_context.SaveChangesAsync();

}

}

}

**Файл UserService.cs**

using electronicLibrary.Data.interfaces;

using Microsoft.AspNetCore.Identity;

using Microsoft.EntityFrameworkCore;

namespace electronicLibrary.Data.Services

{

public class UserService : IUserService

{

private readonly ApplicationDbContext \_context;

private readonly UserManager<ApplicationUser> \_userManager;

private readonly RoleManager<IdentityRole> \_roleManager;

public UserService(

ApplicationDbContext context,

UserManager<ApplicationUser> userManager,

RoleManager<IdentityRole> roleManager)

{

\_context = context;

\_userManager = userManager;

\_roleManager = roleManager;

}

public async Task<List<ApplicationUser>> GetUsersAsync()

{

return await \_userManager.Users

.Include(u => u.BookLoans)

.ThenInclude(bl => bl.Book)

.OrderBy(u => u.FullName)

.AsNoTracking()

.ToListAsync();

}

public async Task<IdentityResult> CreateUserAsync(ApplicationUser user, string password)

{

if (user == null)

throw new ArgumentNullException(nameof(user));

if (string.IsNullOrWhiteSpace(password))

throw new ArgumentException("Password cannot be empty", nameof(password));

var existingUser = await \_userManager.FindByEmailAsync(user.Email);

if (existingUser != null)

throw new InvalidOperationException("User with this email already exists");

var result = await \_userManager.CreateAsync(user, password);

if (result.Succeeded)

{

await \_userManager.AddToRoleAsync(user, "User");

}

return result;

}

public async Task<IList<string>> GetUserRolesAsync(string userId)

{

var user = await \_userManager.FindByIdAsync(userId);

if (user == null)

throw new KeyNotFoundException("Пользователь не найден");

return await \_userManager.GetRolesAsync(user);

}

public async Task<IdentityResult> UpdateUserRolesAsync(string userId, List<string> rolesToAdd, List<string> rolesToRemove)

{

var user = await \_userManager.FindByIdAsync(userId);

if (user == null)

return IdentityResult.Failed(new IdentityError { Description = "Пользователь не найден" });

await EnsureRolesExistAsync();

foreach (var role in rolesToRemove)

{

if (await \_userManager.IsInRoleAsync(user, role))

{

var result = await \_userManager.RemoveFromRoleAsync(user, role);

if (!result.Succeeded)

return result;

}

}

foreach (var role in rolesToAdd)

{

if (!await \_userManager.IsInRoleAsync(user, role))

{

var result = await \_userManager.AddToRoleAsync(user, role);

if (!result.Succeeded)

return result;

}

}

return IdentityResult.Success;

}

public async Task EnsureRolesExistAsync()

{

var requiredRoles = new[] { "User", "Librarian", "Admin" };

foreach (var roleName in requiredRoles)

{

if (!await \_roleManager.RoleExistsAsync(roleName))

{

await \_roleManager.CreateAsync(new IdentityRole(roleName));

}

}

}

public async Task<bool> IsUserInRoleAsync(string userId, string roleName)

{

if (!await \_roleManager.RoleExistsAsync(roleName))

return false;

var user = await \_userManager.FindByIdAsync(userId);

if (user == null)

return false;

return await \_userManager.IsInRoleAsync(user, roleName);

}

public async Task<List<ApplicationUser>> GetRegularUsersAsync()

{

return (await \_userManager.GetUsersInRoleAsync("User"))

.OrderBy(u => u.FullName)

.ToList();

}

public async Task<List<IdentityRole>> GetAllRolesAsync()

{

return await \_roleManager.Roles

.OrderBy(r => r.Name)

.AsNoTracking()

.ToListAsync();

}

}

}

**Файл BookLoanService.cs**

using electronicLibrary.Data.interfaces;

using electronicLibrary.Data.Models;

using Microsoft.EntityFrameworkCore;

namespace electronicLibrary.Data.Services

{

public class BookLoanService: IBookLoanService

{

private readonly ApplicationDbContext \_context;

private readonly IBookService \_bookService;

public BookLoanService(ApplicationDbContext context, IBookService bookService)

{

\_context = context;

\_bookService = bookService;

}

public async Task<List<BookLoan>> GetFilteredLoansAsync(string? userId = null, bool overdueOnly = false)

{

var now = DateTime.UtcNow;

var query = \_context.BookLoans

.Include(bl => bl.Book)

.Include(bl => bl.User)

.Where(bl => !bl.ReturnDate.HasValue)

.AsQueryable();

if (!string.IsNullOrEmpty(userId))

{

query = query.Where(bl => bl.UserId == userId);

}

if (overdueOnly)

{

query = query.Where(bl => bl.DueDate < now);

}

return await query

.OrderByDescending(bl => bl.DueDate < now)

.ThenBy(bl => bl.DueDate)

.AsNoTracking()

.ToListAsync();

}

public async Task<BookLoan> LoanBookAsync(int bookId, string userId, int loanDays, bool skipReservationCheck = false)

{

if (loanDays <= 0)

throw new ArgumentException("Loan period must be positive", nameof(loanDays));

var book = await \_bookService.GetBookByIdAsync(bookId);

if (book == null)

throw new KeyNotFoundException("Book not found");

if (book.AvailableCopies <= 0)

throw new InvalidOperationException("No available copies of this book");

if (!skipReservationCheck && await HasUserBorrowedBookAsync(userId, bookId))

throw new InvalidOperationException("User already has this book");

if (!skipReservationCheck)

{

var activeReservations = await \_context.BookReservations

.AnyAsync(r => r.BookId == bookId && r.IsActive && r.UserId != userId);

if (activeReservations)

throw new InvalidOperationException("This book is reserved by another user");

}

var loan = new BookLoan

{

BookId = bookId,

UserId = userId,

LoanDate = DateTime.UtcNow,

DueDate = DateTime.UtcNow.AddDays(loanDays)

};

await \_context.BookLoans.AddAsync(loan);

book.AvailableCopies--;

\_context.Books.Update(book);

await \_context.SaveChangesAsync();

return loan;

}

public async Task<List<BookLoan>> GetUserCurrentLoansAsync(string userId)

{

return await \_context.BookLoans

.Where(bl => bl.UserId == userId && bl.ReturnDate == null)

.Include(bl => bl.Book)

.OrderBy(bl => bl.DueDate)

.ToListAsync();

}

public async Task<BookLoan> ReturnBookAsync(int loanId, string? requestingUserId = null)

{

var loan = await \_context.BookLoans

.Include(bl => bl.Book)

.FirstOrDefaultAsync(bl => bl.Id == loanId);

if (loan == null)

throw new KeyNotFoundException("Loan not found");

if (requestingUserId != null && loan.UserId != requestingUserId)

throw new UnauthorizedAccessException("You can only return your own books");

if (loan.ReturnDate.HasValue)

throw new InvalidOperationException("Book already returned");

loan.ReturnDate = DateTime.UtcNow;

loan.Book.AvailableCopies++;

await \_context.SaveChangesAsync();

return loan;

}

public async Task<BookLoan> RenewLoanAsync(int loanId, int additionalDays)

{

if (additionalDays <= 0)

throw new ArgumentException("Additional days must be positive", nameof(additionalDays));

var loan = await \_context.BookLoans.FindAsync(loanId);

if (loan == null)

throw new KeyNotFoundException("Loan not found");

if (loan.ReturnDate.HasValue)

throw new InvalidOperationException("Cannot renew returned book");

loan.DueDate = loan.DueDate.AddDays(additionalDays);

\_context.BookLoans.Update(loan);

await \_context.SaveChangesAsync();

return loan;

}

public async Task<bool> HasUserBorrowedBookAsync(string userId, int bookId)

{

return await \_context.BookLoans

.Where(bl => bl.UserId == userId &&

bl.BookId == bookId &&

bl.ReturnDate == null)

.AnyAsync();

}

public int GetDaysUntilDue(BookLoan loan)

{

return (int)(loan.DueDate - DateTime.UtcNow).TotalDays;

}

public int GetOverdueDays(BookLoan loan)

{

return (int)(DateTime.UtcNow - loan.DueDate).TotalDays;

}

}

}

**ПРИЛОЖЕНИЕ В**

**Реализация интерфейса (логики)**

**Файл BooksList.razor**

@page "/books"

@rendermode InteractiveServer

@inject IBookService BookService

@inject IBookReservationService ReservationService

@inject NavigationManager Navigation

@inject AuthenticationStateProvider AuthProvider

<PageTitle>Список книг</PageTitle>

<h3>Список книг</h3>

<div class="mb-3">

<div class="input-group">

<input @bind="searchTerm" class="form-control" placeholder="Поиск по названию, автору, ISBN..." />

<button @onclick="SearchBooks" class="btn btn-primary">Поиск</button>

</div>

</div>

@if (isLibrarian)

{

<div class="mt-3">

<button @onclick="AddNewBook" class="btn btn-primary">Добавить новую книгу</button>

</div>

}

@if (books == null)

{

<p>Загрузка...</p>

}

else if (!books.Any())

{

<p>Книги не найдены</p>

}

else

{

<table class="table">

<thead>

<tr>

<th>Название</th>

<th>Авторы</th>

<th>ISBN</th>

<th>Год</th>

<th>Доступно</th>

<th>Действия</th>

</tr>

</thead>

<tbody>

@foreach (var book in books)

{

<tr>

<td>@book.Title</td>

<td>

@if (book.BookAuthors?.Any() == true)

{

@string.Join(", ", book.BookAuthors.Select(ba => ba.Author.FullName))

}

</td>

<td>@book.ISBN</td>

<td>@book.Year</td>

<td>@book.AvailableCopies из @book.TotalCopies</td>

<td>

<button @onclick="() => ViewBookDetails(book.Id)" class="btn btn-sm btn-info">Подробнее</button>

@if (isLibrarian)

{

<button @onclick="() => EditBook(book.Id)" class="btn btn-sm btn-warning ms-1">Редактировать</button>

<button @onclick="() => DeleteBook(book.Id)" class="btn btn-sm btn-danger ms-1">Удалить </button>

<button @onclick="() => LoanBook(book.Id)" class="btn btn-sm btn-success ms-1">Выдать</button>

}

@if (isUser)

{

@if (book.AvailableCopies > 0)

{

<button @onclick="() => ReserveBook(book.Id)" class="btn btn-sm btn-primary ms-1">Забронировать</button>

}

else if (userReservations.Any(r => r.BookId == book.Id && r.IsActive))

{

<button @onclick="() => CancelReservation(book.Id)" class="btn btn-sm btn-warning ms-1">Отменить бронь</button>

}

else

{

<button disabled class="btn btn-sm btn-secondary ms-1">Нет в наличии</button>

}

}

</td>

</tr>

}

</tbody>

</table>

}

@code {

private List<Book> books = new();

private List<BookReservation> userReservations = new();

private string searchTerm = string.Empty;

private bool isLibrarian;

private bool isUser;

private bool isAuthenticated;

private string? userId;

protected override async Task OnInitializedAsync()

{

await LoadBooks();

await CheckUserRole();

if (isUser)

{

var authState = await AuthProvider.GetAuthenticationStateAsync();

userId = authState.User.FindFirst(ClaimTypes.NameIdentifier)?.Value;

if (!string.IsNullOrEmpty(userId))

{

userReservations = await ReservationService.GetUserReservationsAsync(userId);

}

}

}

private async Task LoadBooks()

{

books = await BookService.GetBooksAsync();

}

private async Task SearchBooks()

{

if (string.IsNullOrWhiteSpace(searchTerm))

{

await LoadBooks();

}

else

{

books = await BookService.SearchBooksAsync(searchTerm);

}

}

private async Task DeleteBook(int id)

{

try

{

await BookService.DeleteBookAsync(id);

await LoadBooks();

}

catch (Exception ex)

{

Console.WriteLine($"Ошибка при удалении: {ex.Message}");

}

}

private void ViewBookDetails(int bookId)

{

Navigation.NavigateTo($"/books/{bookId}");

}

private void EditBook(int bookId)

{

Navigation.NavigateTo($"/books/edit/{bookId}");

}

private void AddNewBook()

{

Navigation.NavigateTo("/books/add");

}

private void LoanBook(int bookId)

{

Navigation.NavigateTo($"/books/loan/{bookId}");

}

private async Task CheckUserRole()

{

var authState = await AuthProvider.GetAuthenticationStateAsync();

isAuthenticated = authState.User.Identity?.IsAuthenticated ?? false;

isLibrarian = authState.User.IsInRole("Librarian");

isUser = authState.User.IsInRole("User");

}

private void ReserveBook(int bookId)

{

Navigation.NavigateTo($"/books/reserve/{bookId}");

}

private async Task CancelReservation(int bookId)

{

var reservation = userReservations.FirstOrDefault(r => r.BookId == bookId && r.IsActive);

if (reservation != null)

{

await ReservationService.CancelReservationAsync(reservation.Id);

userReservations = await ReservationService.GetUserReservationsAsync(userId);

StateHasChanged();

}

}

}

**Файл BookLoan.razor**

@page "/books/loan/{bookId:int}"

@attribute [Authorize]

@rendermode InteractiveServer

@inject IBookLoanService LoanService

@inject IBookService BookService

@inject IUserService UserService

@inject NavigationManager Navigation

<PageTitle>Выдача книги</PageTitle>

<h3>Выдача книги</h3>

@if (book == null)

{

<p>Загрузка...</p>

}

else

{

<div class="card">

<div class="card-header">

<h4>@book.Title</h4>

</div>

<div class="card-body">

<p>Доступно экземпляров: @book.AvailableCopies</p>

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

<label for="userSelect">Читатель:</label>

<select @bind="selectedUserId" class="form-select" id="userSelect">

<option value="">Выберите читателя</option>

@foreach (var user in users)

{

<option value="@user.Id">@user.FullName (@user.Email)</option>

}

</select>

</div>

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

<label for="days">На сколько дней:</label>

<input type="number" @bind="loanDays" id="days" class="form-control" min="1" max="30" />

</div>

</div>

<div class="card-footer">

<button @onclick="ProcessLoan" class="btn btn-primary">Оформить</button>

<button @onclick="Cancel" class="btn btn-secondary ms-2">Отмена</button>

</div>

</div>

}

@if (!string.IsNullOrEmpty(errorMessage))

{

<div class="alert alert-danger mt-3">@errorMessage</div>

}

@code {

[Parameter]

public int BookId { get; set; }

private Book? book;

private List<ApplicationUser> users = new();

private string? selectedUserId;

private int loanDays = 14;

private string? errorMessage;

protected override async Task OnInitializedAsync()

{

book = await BookService.GetBookByIdAsync(BookId);

if (book == null)

{

errorMessage = "Книга не найдена";

}

users = await UserService.GetRegularUsersAsync();

}

private async Task ProcessLoan()

{

try

{

if (string.IsNullOrEmpty(selectedUserId))

{

errorMessage = "Выберите читателя";

return;

}

if (loanDays <= 0)

{

errorMessage = "Укажите корректный срок выдачи";

return;

}

await LoanService.LoanBookAsync(BookId, selectedUserId, loanDays, true);

Navigation.NavigateTo($"/books/{BookId}");

}

catch (Exception ex)

{

errorMessage = ex.Message;

}

}

private void Cancel()

{

Navigation.NavigateTo($"/books/{BookId}", forceLoad: true);

}

}