



## **КУРСОВОЙ ПРОЕКТ**

**Тема: Разработка программного модуля системы «Интернет магазин книг».**

**Специальность 09.02.07 Информационные системы и программирование**

**Выполнил студент(ка) группы** \_\_\_\_\_ **Р.В. Капитонов**

**Руководитель** \_\_\_\_\_ **В.Ю. Назаров**

**Москва 2023**



**УТВЕРЖДАЮ**  
**Зам. директора КМПО**  
**С.Ф. Гасанов**  
« \_\_\_\_\_ » \_\_\_\_\_ 2023 г.

## **ЗАДАНИЕ НА КУРСОВУЮ ПРОЕКТ**

**по дисциплине: МДК.01.01 Разработка программных модулей**  
**Специальность 09.02.07 Информационные системы и**  
**программирование**  
**Студент(ка) группы 31ИС-21 Капитонов Роман**  
**ТЕМА: Разработка программного модуля системы**  
**«Интернет магазин книг».**

Дата выдачи задания « \_\_\_\_\_ » \_\_\_\_\_ 2023 г.

Срок сдачи работы « \_\_\_\_\_ » \_\_\_\_\_ 2023 г.

**Москва 2023**

Федеральное государственное бюджетное образовательное учреждение  
высшего образования  
«РОССИЙСКАЯ АКАДЕМИЯ НАРОДНОГО ХОЗЯЙСТВА И ГОСУДАРСТВЕННОЙ  
СЛУЖБЫ ПРИ ПРЕЗИДЕНТЕ РОССИЙСКОЙ ФЕДЕРАЦИИ»  
КОЛЛЕДЖ МНОГОУРОВНЕВОГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ

**Задание**  
**на курсовой проект**

Дисциплина: МДК.01.01 Разработка программных модулей

Тема: Разработка программного модуля системы

«Интернет магазин книг»

Специальность: 09.02.07 Информационные системы и программирование

Группа: 31ИС-21

ФИО студента Капитонов Р.В.

ФИО руководителя Назаров В.Ю.

1. Проанализировать предметную область
2. Проанализировать готовые решения
3. Подготовить техническое задание
4. Подготовить план тестирования
5. Обосновать выбор инструментов и средств разработки
6. Описать реализацию технического задания
7. Выполнить тестирование

Задание выдано « \_\_\_\_ » \_\_\_\_\_ 2023 г.

Срок выполнения « \_\_\_\_ » \_\_\_\_\_ 2023 г.

Сроки защиты \_\_\_\_\_

Преподаватель: \_\_\_\_\_

Задание получил: \_\_\_\_\_

# СОДЕРЖАНИЕ

ВВЕДЕНИЕ .....	5
1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ.....	7
1.1 Особенности интернет-магазинов .....	7
1.2 Анализ существующих решений.....	8
1.3 Постановка задач.....	9
2 ПРОЕКТИРОВАНИЕ МОДУЛЯ СИСТЕМЫ .....	12
2.1 Выбор и описание программных инструментов.....	12
2.2 Требования к функциям, выполняемым системой .....	13
2.3 Модуль Classes .....	14
2.4 Модуль BookDesktop .....	15
2.5 Структура приложения.....	16
2.6 Структура интерфейса приложения.....	17
2.7 План тестирования .....	19
3 РЕАЛИЗАЦИЯ ПРОЕКТА МОДУЛЯ СИСТЕМЫ.....	20
3.1 Описание кодом функциональных узлов модуля BookDesktop.....	20
3.2 Описание кодом функциональных узлов модуля Classes .....	26
3.3 Тестирование приложения .....	28
ЗАКЛЮЧЕНИЕ .....	32
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ .....	34
Приложение 1 .....	36
Приложение 2 .....	37
Приложение 3 .....	38

## ВВЕДЕНИЕ

Актуальность темы заключается в том, что интернет-магазины книг занимают свою нишу среди любителей литературы. Это связано с тем, что они предлагают широкий ассортимент книг, низкие цены и удобство покупки. Однако, для обеспечения эффективной работы интернет-магазина книг необходимо наличие развитой информационной системы, в том числе для сотрудников организации.

В такой системе должны выполняться такие функции как:

- регистрация и авторизация пользователей;
- поиск книг;
- добавление книг в корзину;
- оформление заказа;
- оплата заказа;
- доставка заказа;
- реализация подписок на категории или авторов;
- обработка и поддержание информации о книгах в базе данных;
- процедуры обработки заказов (отмена, перенос).

Разработка программного модуля системы «Интернет магазин книг» является актуальной задачей, поскольку позволит улучшить работу интернет-магазина и повысить его конкурентоспособность.

Разрабатываемый программный модуль системы «Интернет магазин книг» может быть использован в различных интернет-магазинах книг. Он позволит автоматизировать основные бизнес-процессы, связанные с работой интернет-магазина, и повысить эффективность его работы.

Целью курсовой работы является разработка программного модуля системы «Интернет магазин книг».

Для достижения поставленной цели необходимо решить следующие задачи:

- провести анализ предметной области;

- провести анализ готовых решений;
- разработать логическую модель данных и проект модуля;
- реализовать программный модуль системы;
- провести тестирование программного модуля.

Объектом работы является процесс продажи книг.

Предметом работы является продажа книг через интернет-магазин.

В рамках курсовой работы будут рассмотрены авторизация пользователей, включая авторизацию пользователей с разным уровнем доступа, обработка заказов, поступающих в базу данных из внешнего сервиса, управление книгами в системе, включая их редактирование и добавление, а также просмотр действий других пользователей в системе при соответствующем уровне доступа.

# 1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

## 1.1 Особенности интернет-магазинов

Традиционные магазины книг представляют собой физическое строение, обладающее своей уникальной атмосферой и характеристиками. Их расположение обычно выбирается в центре города или в крупных торговых центрах, обеспечивая удобный доступ для посетителей. Дизайн магазина направлен на создание приятного визуального восприятия и привлечение внимания.

Одним из плюсов традиционных книжных магазинов является возможность физически присутствовать и выбирать объект для приобретения. Посетитель может физически прикоснуться к книге до покупки и прочитать то, что ему интересно. Но за это владельцам такого магазина приходится брать дополнительные расходы такие как зарплата кассирам, аренда помещения, что является нежелательными тратами.

Также физические магазины книг могут продавать не только книги, но и канцелярские предметы и другого рода вещи, необходимые в школе или творчестве.

В свою очередь интернет-магазины обеспечивают удобство онлайн-покупок с широким ассортиментом, доставкой на дом и возможностью ознакомления с отзывами других покупателей. Однако, в интернете, покупатель теряет возможность личного взаимодействия с продавцом и физического ощущения книг перед покупкой.

Основное различие заключается в способе взаимодействия с продуктом и окружающей обстановкой. Традиционные магазины книг стремятся создать уникальный опыт, в то время как интернет-магазины сосредотачиваются на удобстве и скорости онлайн-покупок.

Важными аспектами в сфере онлайн-торговли книгами являются несколько ключевых факторов. Прежде всего, это обширный каталог товаров,

организованный в четкие категории, чтобы обеспечить удобство выбора и навигации для покупателей.

Особое внимание также уделяется созданию интуитивных интерфейсов, использованию продвинутых систем фильтрации и рекомендаций для обеспечения удобства процесса покупки и навигации по виртуальному магазину.

Интернет-магазины книг обязательно предлагают интеграцию методов доставки и обработки заказов для обоих форматов – электронных и физических книг. Это включает в себя системы, способные обеспечить безопасность онлайн-платежей, а также эффективную мобильную совместимость и аналитические инструменты для оценки и улучшения производительности интернет-магазина.

Это основополагающий взгляд на технические и функциональные аспекты, которые будут учтены при разработке программного модуля для системы "Интернет магазин книг".

## 1.2 Анализ существующих решений

Существует множество интернет-магазинов книг, каждый из которых предлагает уникальные решения и подходы.

Amazon Books — это крупнейший онлайн-ритейлер, предоставляющий огромный каталог книг в физическом и электронном форматах. Специализируется не только на продаже книг, но и на других товарах. Система рекомендаций Amazon позволяет предлагать пользователям товары, основанные на их предыдущих покупках и интересах.

Book Depository — онлайн-магазин, известный бесплатной международной доставкой. Специализируется на продаже книг, включая широкий выбор заграничных изданий. Привлекает внимание покупателей, ищущих разнообразие и доступность мировой литературы.

AbeBooks выделяется как платформа для поиска и покупки редких, антикварных и коллекционных книг. Этот магазин предоставляет возможность



находить уникальные издания, которые могут быть трудно найти в других местах.

Google Play Books является частью экосистемы Google и предоставляет широкий выбор электронных книг и аудиокниг. Интегрируется с другими сервисами Google, что обеспечивает удобство чтения на различных устройствах, включая смартфоны и планшеты.

ЛитРес — российский интернет-магазин, специализирующийся на продаже электронных книг. Предоставляет возможность скачивания и чтения книг в электронном формате. Также предлагает подписку для доступа к библиотеке книг за фиксированную плату. Фокусируется на российской аудитории, предоставляя широкий выбор российских и зарубежных произведений.

Каждый из рассмотренных интернет-магазинов книг имеет свои преимущества и недостатки. Amazon Books и Book Depository предлагают широкий ассортимент книг по низким ценам, но доставка в некоторые страны может быть платной. AbeBooks предлагает широкий ассортимент книг, включая редкие и коллекционные экземпляры, но время доставки может быть длительным. Google Play Books предлагает широкий ассортимент электронных книг и возможность слушать аудиокниги, но не всегда доступны книги в печатном формате. ЛитРес предлагает широкий ассортимент электронных книг и возможность слушать аудиокниги, а также доступ к книгам на иностранных языках, но стоимость подписки может быть высокой. (см. Приложение 1)

### 1.3 Постановка задач

Перед постановкой задач и определением требований необходимо определить рамки модуля и функционал системы в целом.

Полная система должна включать минимум 3 модуля, в которые входит клиентский, редакторский и рабочий интерфейс.

Клиентский интерфейс принято делать удобным, понятным и быстрым. Для таких приложений есть свои критерии разработки и такое делается в виде веб-приложения, которое работает аналогично обычному просмотру веб-страницы, что удобно для обычного пользователя и для разработчика.

Редакторский интерфейс должен брать на себя часть верификации автора и издательства, то есть учесть все нормативно-правовые вопросы до того, как сотрудник сможет добавить книгу определённого автора или издательства.

Рабочий интерфейс предназначен для сотрудников, которые должны заниматься исключительно работой с системой в виде добавления и редактирования книг при необходимости или запросе, а также работу с заказами, которые необходимо контролировать и отменять в определённых случаях.

Это 3 основных модуля, которые можно брать за основу для разработки. К ним можно добавить такие модули как службу поддержки, которая сможет создавать запросы по просьбе клиентов или интерфейс мониторинга состояния системы, что является исключительно технической частью.

Так как целью курсового проекта является разработка настольного приложения, то наиболее уместным будет выбрать «рабочий» модуль, так как такое приложение должно устанавливаться в операционной системе и его файлы должны находиться на компьютере, что не приемлемо в случае разработки веб-приложения.

Такой модуль должен включать:

#### 1 Функции управления каталогом

Управление каталогом должно включать в себя возможность добавления новых книг в систему. Это предполагает введение основной информации о книге, такой как название, автор, жанр, цена и наличие на складе. Дополнительно, должен предоставляться функционал редактирования информации о существующих книгах, а также фильтрации списка книг для удобства пользователей, позволяя им выбирать книги в соответствии с интересующими жанрами.

## 2 Функции обработки заказов

Обработка заказов должна включать в себя сохранение информации о заказах и управление ими. Это подразумевает хранение деталей заказов, таких как перечень книг, количество экземпляров, стоимость и статус. Администратору предоставляется функционал изменения статуса заказа, что упрощает отслеживание и обработку заказов в системе.

## 3 Функции просмотра действий пользователей

Просмотр действий пользователей должен предоставляться администратору. А также отслеживание действий, таких как добавление, редактирование книг или изменение статуса заказов. Эта функциональность обеспечивает контроль за происходящим в системе и может быть полезной для обнаружения проблем или необычных событий.

## 4 Функции безопасности

Функции безопасности включают в себя механизмы аутентификации и авторизации. Они обеспечивают безопасный доступ администраторов к системе, предотвращая несанкционированный доступ. Это также включает в себя защиту конфиденциальных данных, таких как информация о пользователях, заказах и других важных аспектах системы. Аутентификация подтверждает легитимность пользователя, а авторизация определяет его права доступа в системе. Эти меры гарантируют, что только уполномоченные пользователи могут взаимодействовать с критическими частями приложения.

Модуль не включает регистрацию новых авторов и жанров, так как эти вопросы должны решаться другим модулем системы – редакторским.

## 2 ПРОЕКТИРОВАНИЕ МОДУЛЯ СИСТЕМЫ

### 2.1 Выбор и описание программных инструментов

Для написания десктопных приложений существует множество популярных языков программирования, каждый из которых имеет свои преимущества, особенности и недостатки, которые не очевидны при беглом просмотре.

Для написания приложения подходит C# (C Sharp). Этот язык, разработанный Microsoft, часто используется для создания десктопных приложений с использованием технологии .NET. Он предоставляет удобный синтаксис, богатые библиотеки и интеграцию с различными инструментами разработки. Лёгок в освоении, а также имеет большое сообщество программистов, которое привлекает в том числе обилие фреймворков. Ко всему этому следует добавить, что платформа .NET поддерживает кроссплатформенность приложений, а также включает ASP.NET, позволяющий работать с веб-приложениями.

Из множества фреймворков подходит .NET MAUI с Blazor. Он предоставляет инструментарий для создания кроссплатформенных мобильных, настольных и веб-приложений с использованием языка программирования C# и платформы .NET. Сочетание .NET MAUI с Blazor позволяет использовать привычные технологии для создания веб-приложений и переносить их на мобильные и настольные платформы.

Синтаксис представляет собой HTML разметку с использованием «@», когда необходимо обратиться к переменным кода или использовать условные и циклические структуры.

Также поддерживает «модели», что позволяет удобным образом создавать формы заполнения, устанавливая условия для полей и сообщения об ошибках, которые увидит пользователь при несоблюдении условий.

Для работы с базой данных используется EntityFramework. Он используется для работы с данными в приложениях на платформе .NET и предоставляет набор инструментов для работы с базами данных, абстрагируя слой доступа к данным и позволяя разработчикам взаимодействовать с базой данных с использованием объектно-ориентированной модели данных.

В качестве сред разработки были выбраны Visual Studio 2022 и JetBrains Rider. Каждая из сред имеет свои преимущества и недостатки. Visual Studio помогает писать простые и однотипные участки кода при помощи IntelliSense. Но несмотря на то, что Rider не имеет подобного, он намного удобнее при написании кода, а также отладке и изучении.

## 2.2 Требования к функциям, выполняемым системой

Функции управления каталогом включают в себя возможность добавления новых книг с указанием основной информации, такой как название, автор, жанр, цена и количество экземпляров. Также предусмотрена опция редактирования информации о уже имеющихся в каталоге книгах. Для облегчения поиска и выбора пользователю предоставляется функция фильтрации списка книг по жанрам.

Функции обработки заказов включают в себя хранение информации о заказах и возможность управления ими, включая изменение статуса заказов. Это обеспечивает систематизацию и эффективное ведение учета заказов в системе.

Функции просмотра действий пользователей предоставляют администратору системы возможность отслеживать действия других пользователей в системе. Эта функция обеспечивает контроль и мониторинг активности, что может быть полезным для анализа и обеспечения безопасности системы.

Функции безопасности включают в себя механизмы аутентификации и авторизации, обеспечивая безопасный доступ администраторов к системе. Эти меры гарантируют, что только уполномоченные пользователи могут вносить

изменения в каталог, обрабатывать заказы и просматривать действия других пользователей, что важно для поддержания целостности и конфиденциальности информации.

В приложении должны быть 2 строго разделённых модуля:

- Classes. Содержит классы, с которыми ведётся информационная работа.
- BookDesktop. Приложение с классами, которые его описывают.

## 2.3 Модуль Classes

Рассмотрим диаграмму классов (см. рис.1), которые помогают вести информационную работу.

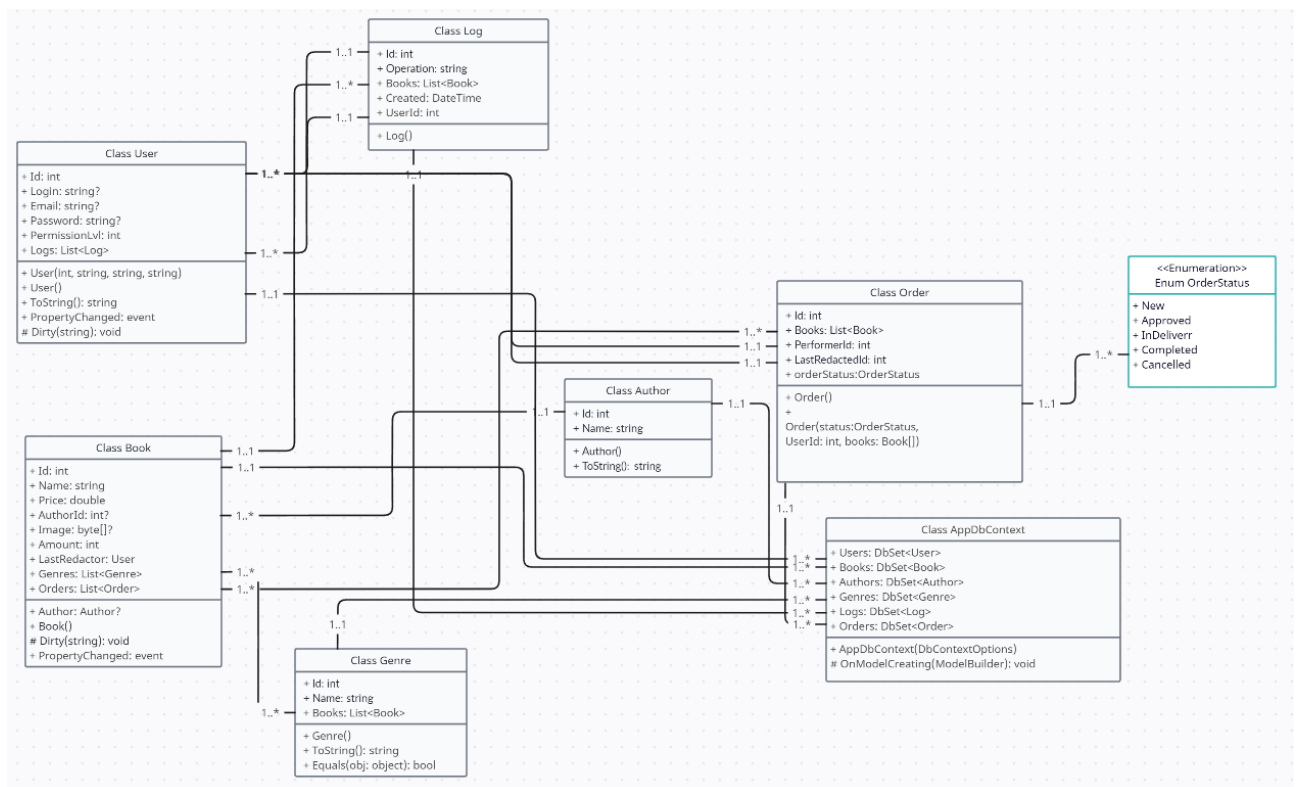


Рисунок1 - Диаграмма классов

На ней представлены основные классы, которые будут использоваться в программе. Также на их основе будет сделана база данных (см. Рисунок 2), обращение к которой будет проходить через объект класса ApplicationDbContext.

Этот класс, реализует работу с базой данных. Он хранится в этом же модуле для удобства работы с ним. Имеет ссылки на все остальные классы через

свойство DbSet. Через EntityFramework создаёт базу данных с таблицами и связями, которые соответствуют таковым в коде модуля.

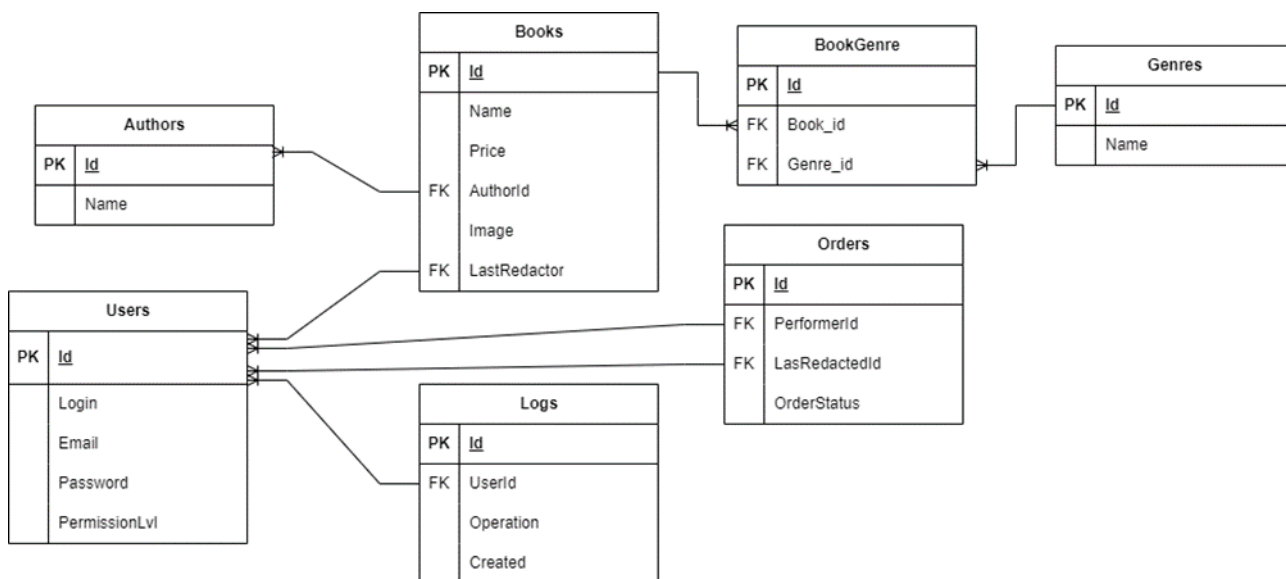


Рисунок 2 – ER-диаграмма базы данных

Почти все классы в модуле являются описанием таблиц, но для некоторых прописаны и переопределены методы для удобства работы с ними в приложении.

## 2.4 Модуль BookDesktop

Этот модуль написан на основе фреймворка .NET MAUI. При запуске открывается веб-приложение, которое является отдельным экземпляром браузера, который установлен в системе по умолчанию.

- Благодаря этому приложение само по себе является кроссплатформенным с проблемой исключительно в вёрстке.
- Также это позволит легко перейти на браузерную работу приложения при необходимости.

Связь с модулем Classes реализована через ссылку на сборку в модуле и директивы using там, где необходима прямая работа с классами первого модуля.

Класс AppDbContext здесь устанавливается как сервис. Для удобства проектирования приложения использовалась база данных SQLite, которая является файловой БД, т. е. хранится в одном файле вместе с описанием и самими

данными, что удобно с небольшими базами или при отладке. Благодаря использованию EntityFramework, приложение можно быстро перенести на PostgreSQL.

## 2.5 Структура приложения

Функционал, который предоставляет интерфейс, ограничивается в зависимости от уровня прав пользователя (см. Рисунок 3).

Предусмотрено 3 уровня доступа:

- 0 Root - полный доступ. Предназначен для администратора системы.
- 1 Администратор. Предназначен для администратора занимающимся исключительно аналитикой. Не обладает правами на редактирование.
- 2 Сотрудник. Предназначен для сотрудника занимающимся каталогизацией книг и обработкой заказов.

Все действия логируются и доступны пользователям с доступом к аналитике.

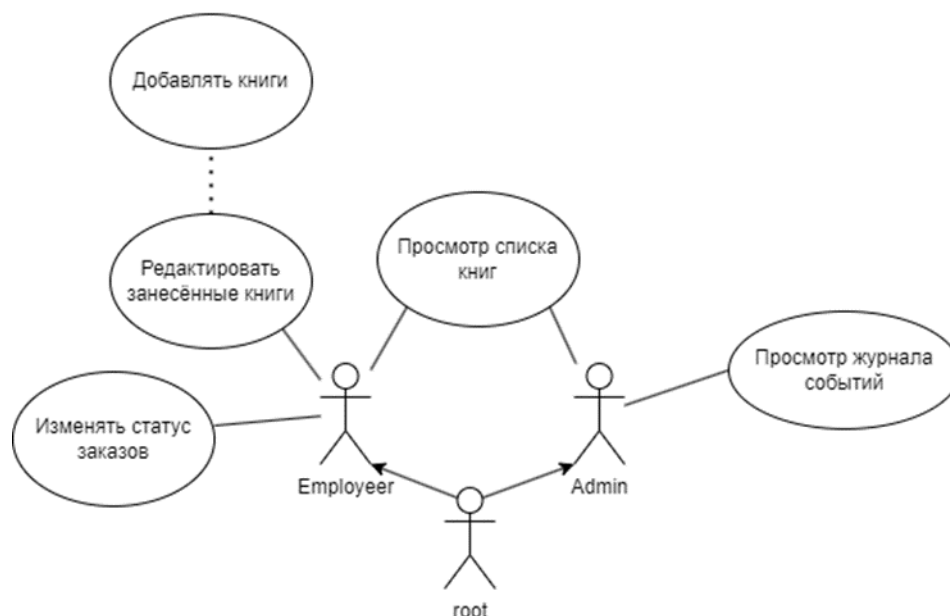


Рисунок 3 - Диаграмма прецедентов



## 2.6 Структура интерфейса приложения

При запуске приложения пользователь должен попадать на страницу авторизации. После ввода и подтверждения программа запоминает авторизованного пользователя, как объект класса User из модуля Classes.

Пользователю с соответствующими правами будут доступны:

- Главная страница
- Каталог
- Логи
- Заказы

### 2.6.1 Главная страница

На главной странице можно посмотреть основную информацию о пользователе, от имени которого производятся действия в программе (см. Рисунок 4).

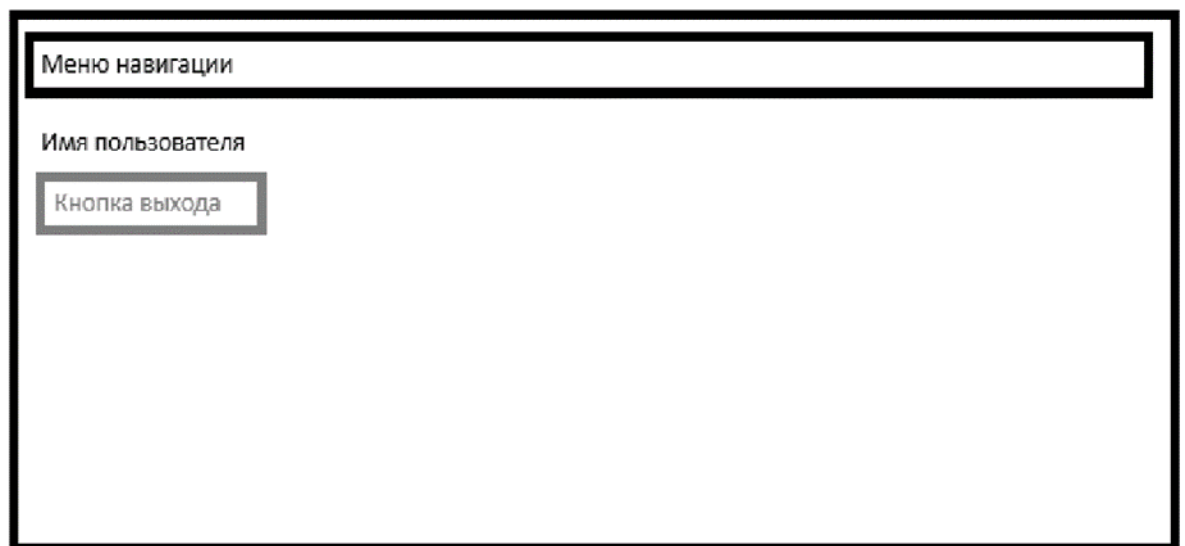


Рисунок 4 – Интерфейс главной страницы

Также на странице должна быть кнопка выйти для отладки приложения и удобства использования.

### 2.6.2 Каталог

Страница каталога должна отображать все книги, которые есть в базе данных и основную информацию о них в виде списка (см. Рисунок 5).

Должна иметь кнопки для изменения выбранной книги, добавления новой или фильтрации по жанрам.



Рисунок 5 –Интерфейс окна каталога

Функции редактирования и добавления не доступны для администратора(аналитика).

### 2.6.3 Логи

Страница аналитики. Отображает все действия, которые записаны системой в виде списка (см. Рисунок 6). Записываются только изменения, которые повлияли на данные в базе данных.

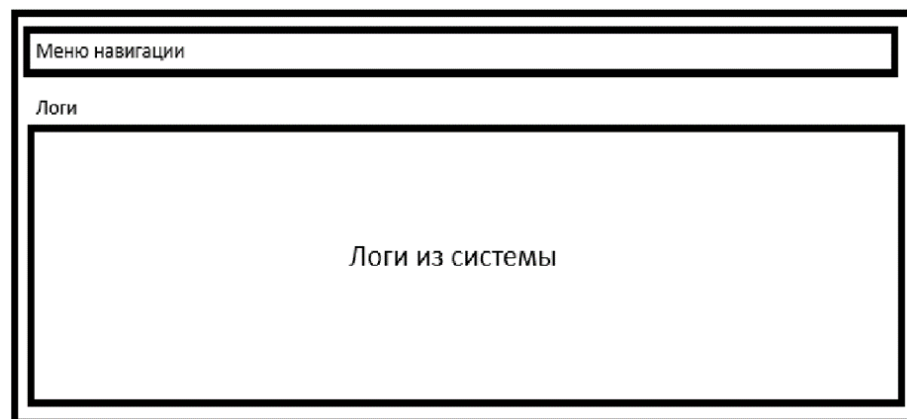


Рисунок 6 – Интерфейс окна логов

Доступ к этой странице должен быть только у администратора и системного администратора во избежание утечки данных и для безопасности.

#### 2.6.4 Заказы

Представляет все заказы, которые есть в БД в виде таблицы из карт. Позволяет изменять статус заказа.

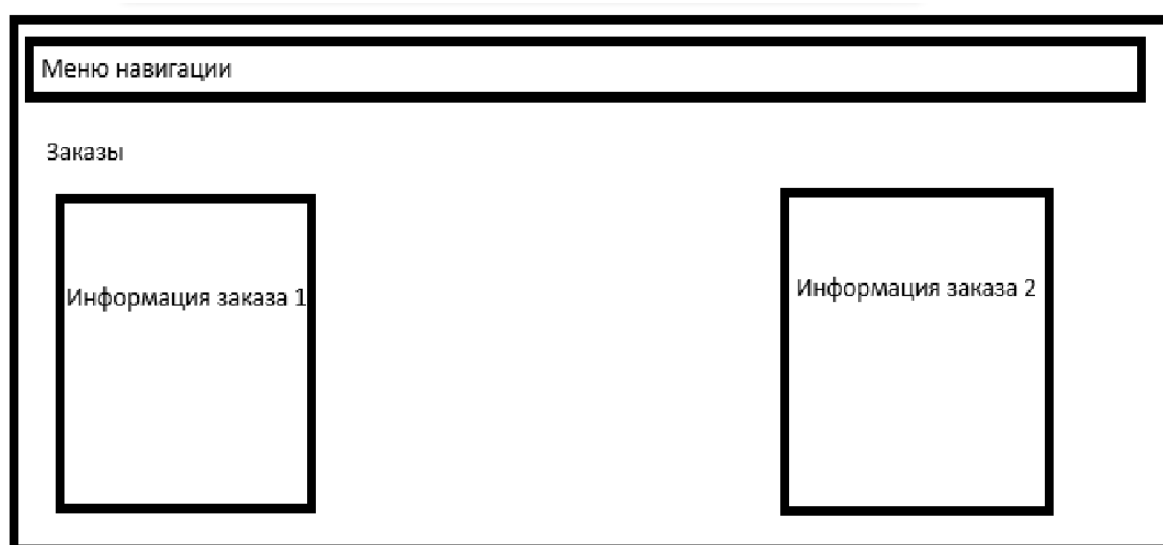


Рисунок 7 – Интерфейс окна заказов

#### 2.7 План тестирования

Разработан план тестирования, покрывающий некоторые сценарии программы (см. Приложение 2).

## 3 РЕАЛИЗАЦИЯ ПРОЕКТА МОДУЛЯ СИСТЕМЫ

### 3.1 Описание кодом функциональных узлов модуля BookDesktop

#### 3.1.1 Старт программы и подготовка

Программа начинает своё выполнение с класса MauiProgram, где приложения собирается. Производится добавление сервисов и их подготовка (см. Рисунок 8).

```
builder.Services.AddSingleton<AuthService>();
builder.Services.AddSingleton<BookChangeService>();
builder.Services.AddDbContext<AppDbContext>(optionsAction: (options) =>
{
    options.UseSqlite(connectionString: $"Data Source={Environment.CurrentDirectory + @"\MyDatabase.db"}");
});
builder.Services.AddSingleton<BinService>();
var app = builder.Build();
var dbContext = app.Services.GetRequiredService<AppDbContext>();
dbContext.Database.EnsureCreated();
```

Рисунок 8 – MauiProgram, инсталляция сервисов

В представленном участке кода явно видно, что используется база данных SQLite, но при необходимости можно подключаться к PostgreSQL, которая является более профессиональным и производительным решением.

Также в этом файле база данных заполняется тестовыми данными при запуске приложения в отладочном режиме и отсутствии базы.

После завершения подготовительных работ инициализируется интерфейс и появляется окно программы со страницей авторизации.

#### 3.1.2 Окно авторизации

Представляет собой форму с двумя полями. Проверка введённых пользователем значений производится благодаря использованию модели Person, в которой через атрибуты указаны условия и выводимые ошибки.

При корректности данных программа обращается к сервису для авторизации. Сервис возвращает булево значение в зависимости от того, авторизован пользователь или нет. Если нет, то поля ввода очищаются (см. Рисунок 10).

```
private void Submit()
{
    if (Person.Login != "" && Person.Password != "")
    {
        if (!authService.Login(navigation, Person.Login!, Person.Password!))
        {
            Person.Login = String.Empty;
            Person.Password = String.Empty;
            IsValid = true;
            InvokeAsync(StateHasChanged);
        }
    }
}
```

Рисунок 10 – Код формы для авторизации

Также при неправильных данных авторизации пользователь получает уведомление о неправильном вводе данных (см. Рисунок 12), что соответствует реакции программы при неправильных данных по плану тестирования (см. Приложение 2).

## Авторизация

Логин

Пароль


 Авторизоваться

Рисунок 11 – Окно авторизации

Интерфейс окна состоит из 2 полей и кнопки для авторизации (см. Рисунок 11).

```

@if (IsValid)
{
    <div class="d-flex border border-warning">
        <div class="p-2 w-100">
            <p>Некорректные данные авторизации</p>
        </div>
        <div class="p-2 flex-shrink-1">
            <button type="button" class="btn btn-warning" onClick="@(() => { IsValid = false; InvokeAsync(StateHasChanged); })">
                Закреть
            </button>
        </div>
    </div>
}

```

Рисунок 12 – Уведомление о некорректном вводе данных авторизации

### 3.1.3 Каталог

Страница с перечислением всех книг в базе данных. Рядом на строке с каждой книгой есть кнопка для изменения. Ниже часть кода, отвечающая за обработку действий пользователя.

Также, начиная с этой страницы, используется компонент для навигации, чтобы автоматически отслеживать активную страницу и корректно отображать доступные страницы.

Для реализации фильтра используется функция, которая запрашивает данные из базы, используя выбранные пользователем жанры.

```

async Task<List<Book>> GetBooksWithFilter()
{
    if (selectedGenres.Count == 0)
        return books;

    var filteredBooks :List<Book> = await dbcontext.Books // DbSet<Book>
        .Include( navigationPropertyPath: x :Book => x.Author) // IQueryable<Book, Author?>
        .Where(book => book.Genres.Any(bg :Genre => selectedGenres.Contains(bg))) // IQueryable<Book>
        .ToListAsync(); // Task<List<...>>

    return filteredBooks;
}

```

Рисунок 13 – Функция выборки книг по выбранным жанрам

Для изменения книги или добавления используются разные страницы, но отличаются они не сильно по своему функционалу. Их задача отобразить форму (см. Рисунок 14) для пользователя и проконтролировать введенные значения, что выполняется при попытке пользователем сохранить книгу.

## Добавление книги

Название

Автор

Стоимость

Количество

Рисунок 14 – Окно изменения/добавления книги

Рассмотрим подробнее работу этой функции. Сначала функция убирает лишние пробелы при помощи регулярного выражения, а после проверяет введённые значения на наличие пустой строки или отрицательных значений (см. Рисунок 15). Если были ошибки при написании, то выполнение идёт по ветке true и пользователь получает уведомление о некорректном вводе согласно плану тестирования (см. Приложение 2).

```
bookLocal.Name = Regex.Replace(input: bookLocal.Name, pattern: @"\s+", replacement: " ");
if (bookLocal.Name == "" || bookLocal.Price < 0 || bookLocal.Amount < 0)
{
    MessageError = "Ошибка при сохранении";
    HasError = true;
    CanEscape = false;
    InvokeAsync(StateHasChanged);
}
else
```

Рисунок 15 - Функция сохранения изменённой книги ветка true

Если ошибок не было, то программа идёт по ветке false (см. Рисунок 16) и обрабатывает добавление новой книги, также заносит запись в базу об изменении/добавлении книги.

```

else
{
    HasError = false;
    CanEscape = true;

    bookLocal.Genres.Clear();
    foreach (var id:int in checkedGenres.Where(x:KeyValuePair<int,bool> => x.Value).Select(x:KeyValuePair<int,bool> => x.Key))
    {
        bookLocal.Genres.Add(item:genres.First(x:Genre => x.Id == id));
    }

    bookLocal.LastRedactorId = authService.CurrentUser!.Id;

    dbcontext.Update(bookLocal);
    dbcontext.Logs.Add(entity:new Log
    {
        Operation = $"Внесены изменения в книгу с Id = {bookLocal.Id}",
        UserId = authService.CurrentUser!.Id,
    });
    await dbcontext.SaveChangesAsync();
    navigation.NavigateTo(uri:"books");
}

```

Рисунок 16 – Функция сохранения изменённой книги ветка else

Также в ходе тестирования добавления и удаления пользователей была обнаружена ошибка, из-за которой программа аварийно завершала работу. Это происходило из-за попытки EntityFramework обработать автора книги как новый объект, а не существующий.

Если нужно будет занести книгу, автора которого нет в базе, то придётся делать запрос напрямую в базу данных. Аналогичная ситуация с жанрами. Сделано это так, потому что такие процессы как добавление автора, жанра или издательства задача другого рода и не подходит под определённые при проектировании задачи, а также выходит за рамки разработки модуля.

### 3.1.4 Страница просмотра логов

На этой странице создаётся список из всех записей таблицы Logs, которые есть в базе данных.

При изменении вносятся записи следующих форматов:

- Заказы: (пользователь) изменил статус заказа (id заказа) на (статус заказа)
- Книги(изменение): Внесены изменения в книгу с id = (id книги)



- Книги(добавление): Добавлена книга с именем = (название книги)

<a href="#">Главная страница</a> <a href="#">Каталог</a> <a href="#">Логи</a> <a href="#">Заказы</a>		
Логи		
Исполнитель	Действие	Время
root	Внесены изменения в книгу с Id = 2	29.11.2023 23:11:44
root	Внесены изменения в книгу с Id = 5	29.11.2023 23:11:53
root	root изменил статус заказа(11) на Cancelled	29.11.2023 23:13:15
root	root изменил статус заказа(12) на Cancelled	29.11.2023 23:13:16
root	root изменил статус заказа(14) на Cancelled	29.11.2023 23:13:18
root	root изменил статус заказа(13) на Cancelled	29.11.2023 23:13:20
Chesunov	employee1 изменил статус заказа(11) на Approved	29.11.2023 23:15:56
Chesunov	employee1 изменил статус заказа(12) на Approved	29.11.2023 23:15:56
Chesunov	employee1 изменил статус заказа(13) на Approved	29.11.2023 23:15:58
Chesunov	employee1 изменил статус заказа(14) на Approved	29.11.2023 23:15:59
Chesunov	employee1 изменил статус заказа(16) на Approved	29.11.2023 23:16:01
root	root изменил статус заказа(11) на Cancelled	01.12.2023 11:16:22
Chesunov	employee1 изменил статус заказа(12) на Completed	01.12.2023 11:18:12
root	root изменил статус заказа(11) на Completed	03.12.2023 21:11:19
Chesunov	Внесены изменения в книгу с Id = 3	10.12.2023 17:40:05
Chesunov	Внесены изменения в книгу с Id = 1	10.12.2023 17:40:35
Chesunov	employee1 изменил статус заказа(11) на Approved	10.12.2023 17:40:47
Chesunov	Внесены изменения в книгу с Id = 1	10.12.2023 18:34:27
Chesunov	employee1 изменил статус заказа(11) на Cancelled	10.12.2023 18:36:04
root	Внесены изменения в книгу с Id = 5	13.12.2023 0:30:40

Рисунок 17 – Страница логов

Записи в эту таблицу вносятся при любом значимом изменении (см. Рисунок 17).

### 3.1.5 Страница просмотра заказов

На этой странице отображаются все заказы, которые занесены в систему (см. Рисунок 18) в виде таблицы с элементами card.

<a href="#">Главная страница</a> <a href="#">Каталог</a> <a href="#">Логи</a> <a href="#">Заказы</a>	
Заказы	
<div> <div>Номер: 11</div> <div>Статус: Completed</div> <div>Входящие книги</div> <div> <div>Как на самом деле работают компьютеры.</div> <div>Практическое руководство по внутреннему устройству машины</div> <div>Автор: Адам Фримен</div> <div>Количество: 130</div> <div>Цена: 0</div> </div> <div> <div>Изучаем C#</div> <div>Автор: Дженифер Грин</div> <div>Количество: 60</div> <div>Цена: 600</div> </div> <div>Изменить статус +</div> </div>	<div> <div>Номер: 12</div> <div>Статус: Completed</div> <div>Входящие книги</div> <div> <div>Параллельное программирование на современном C++</div> <div>Мэттью Джастис</div> <div>Количество: 20</div> <div>Цена: 200</div> </div> <div> <div>Эффективный C. Профессиональное программирование</div> <div>Райнер Гримм</div> <div>Количество: 30</div> <div>Цена: 3000</div> </div> <div>Изменить статус +</div> </div>
<div> <div>Номер: 13</div> <div>Статус: New</div> <div>Входящие книги</div> </div>	<div> <div>Номер: 14</div> <div>Статус: Completed</div> <div>Входящие книги</div> </div>

Рисунок 18 – Окно просмотра заказов

Создать заказ нельзя, так как подразумевается, что это происходит во внешнем сервисе или приложении. На этой стороне происходит лишь обработка статуса. Также заказ не имеет возможности «транзакций». То есть при статусе «Выполнен» в системе не будет изменяться количество оставшихся экземпляров книг.

```
public void ChangeOrderStatus(Order order, OrderStatus status)
{
    order.orderStatus = status;
    dbcontext.Logs.Add(entity: new Log
    {
        Operation = $"{AuthService.CurrentUser.Login} изменил статус заказа({order.Id}) на {status}",
        UserId = authService.CurrentUser.Id,
    });
    dbcontext.SaveChanges();
    InvokeAsync(StateHasChanged);
}
```

Рисунок 19 – Обработка изменения статуса заказа

При полноценной структуре системы у сотрудников не должна быть возможность напрямую изменить статус заказа на «Выполнен» или «Отменить». Это должно проводиться через запрос клиента в специальной форме. Здесь есть такая функция, чтобы показать рабочий функционал, а также факт учёта такого статуса в системе (см. Рисунок 19).

### 3.2 Описание кодом функциональных узлов модуля Classes

Модуль состоит из 7 классов: 6 функциональных и 1 служебного. Является библиотекой и связь с модулем BookDesktop осуществляется через ссылку на проект, а точнее путём компиляции dll файла в директорию основного модуля.

#### 3.2.1 Работа с БД в контексте класса AppDbContext

Этот класс служебный и выполняет роль посредника между кодом и СУБД, т. е. является интерфейсом. Через свойства DbSet указаны классы и названия

таблиц, которые должен создать EntityFramework, если их не будет в базе данных.

В ходе разработки потребовалось создание таблиц многие ко многим, поэтому через отношения это было прописано отдельно (см. Рисунок 20).

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    modelBuilder.Entity<Book>() // EntityTypeBuilder<Book>
        .HasMany( navigationExpression: b :Book => b.Genres) // CollectionNavigationBuilder<Book,Genre>
        .WithMany( navigationExpression: bg :Genre => bg.Books);

    modelBuilder.Entity<Order>() // EntityTypeBuilder<Order>
        .HasMany( navigationExpression: o :Order => o.Books) // CollectionNavigationBuilder<Order,Book>
        .WithMany( navigationExpression: b :Book => b.Orders);

    modelBuilder.Entity<User>() // EntityTypeBuilder<User>
        .HasMany( navigationExpression: x :User => x.Logs) // CollectionNavigationBuilder<User,Log>
        .WithOne( navigationExpression: b :Log => b.User);
}
```

Рисунок 20 – Отношения многие-ко-многим

EntityFramework сам регулирует их и прямых операций со сводными таблицами не требуется, поэтому они не указаны как одно из свойств типа DbSet.

### 3.2.2 Функциональные классы

Класс Author. Простой, но нужный класс для соответствия базы данных третьей нормальной форме, а также упрощает операции добавления и изменения книг.

Класс Book. Хранит в себе основную информацию о книгах такую как название, цену, автора, изображение в виде бинарного массива, количество экземпляров, список жанров, который контролируется через EntityFramework, как список заказов, а также пользователя, который последним редактировал конкретную книгу.

Класс Genre. Вспомогательный для Book, так как представляет собой жанры. Также хранит список книг, которые можно получить при необходимости

через EntityFramework. Это позволяет узнать сколько книг есть у каждого жанра простым запросом.

Класс Log. Представляет записи о действиях пользователей. Хранит текстовое описание операции, дату создания записи и пользователя.

Класс Order. Для этого класса было создано перечисление, чтобы удобно работать со статусами заказов. Хранит список книг, пользователя, которому назначен заказ (если такое необходимо), пользователя, который последний раз редактировал этот заказ, а также конкретный элемент из перечисления.

Класс User. Самый важный класс, который заставляет работать половину приложения (не считая служебный AppDbContext). Содержит логин, почту, пароль и уровень доступа пользователя в качестве целочисленного значения. Также была переопределена стандартная функция ToString для удобства работы с доступами.

### 3.3 Тестирование приложения

Протестируем программу с учётом плана тестирования плану тестирования (см. Приложение 2). А именно авторизацию, работу фильтра в каталоге, изменение статуса заказов и посмотрим записывает ли система внесённые изменения на странице просмотра логов.

#### 3.3.1 Тестирование окна авторизации

При вводе неверных данных пользователь должен увидеть уведомление о некорректном вводе (см. Рисунок 21).

## Авторизация

Некорректные данные авторизации Заккрыть

Логин

Пароль

Войти

Рисунок 21 – Тестирование. Авторизация: некорректный ввод  
Результат: отображение сообщения и очистка полей авторизации.

При вводе верных данных авторизации (существующего пользователя) пользователя должно перенаправить на главную страницу (см. Рисунок 22).

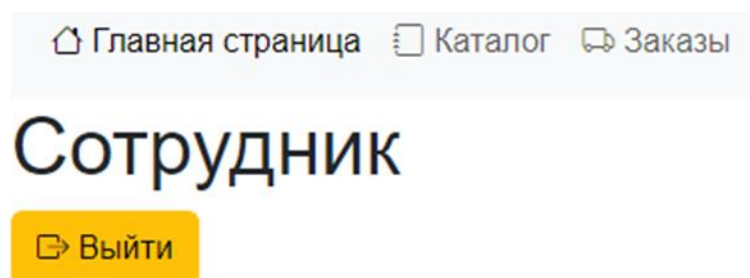


Рисунок 22 – Успешная авторизация от пользователя Kalidsky  
На главной странице должна быть написана группа пользователя.

### 3.3.2 Тестирование окна каталога

Рассмотрим 2 варианта. Выбор нескольких жанров (см. Рисунок 23) и пустой список (см. Рисунок 24).

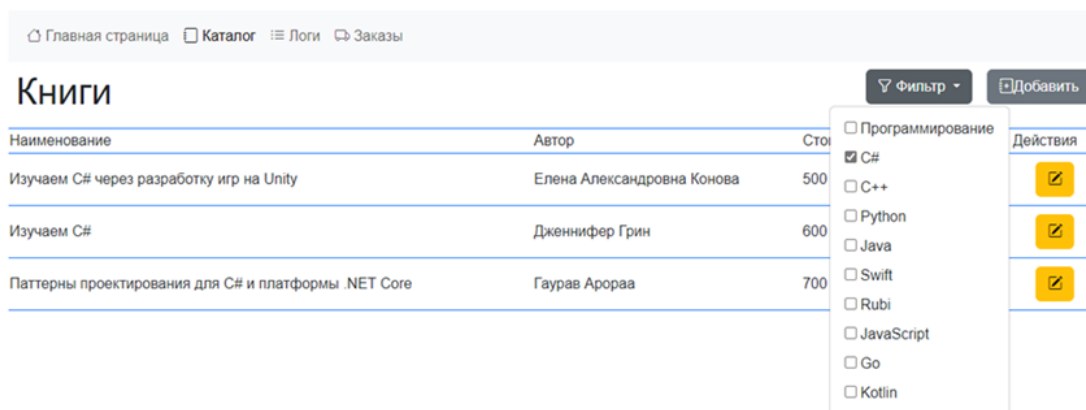


Рисунок 23 – Тестирование. Каталог: без жанров

<a href="#">Главная страница</a> <a href="#">Каталог</a> <a href="#">Логи</a> <a href="#">Заказы</a>				
Книги				
Наименование	Автор	<input type="checkbox"/> Программирование <input type="checkbox"/> C# <input type="checkbox"/> C++ <input type="checkbox"/> Python <input type="checkbox"/> Java <input type="checkbox"/> Swift <input type="checkbox"/> Rubi <input type="checkbox"/> JavaScript <input type="checkbox"/> Go <input type="checkbox"/> Kotlin		Действия
Как на самом деле работают компьютеры. Практическое руководство по внутреннему устройству машины	Адам Фримен			<input checked="" type="checkbox"/>
Параллельное программирование на современном C++	Мэттью Джастис			<input checked="" type="checkbox"/>
Эффективный C. Профессиональное программирование	Райнер Гримм			<input checked="" type="checkbox"/>
Алгоритмы и программы. Язык C++ : учебное пособие для СПО	Роберт С. Сихорд			<input checked="" type="checkbox"/>
Изучаем C# через разработку игр на Unity	Елена Александровна Конова			<input checked="" type="checkbox"/>
Изучаем C#	Дженнифер Грин	600	60	<input checked="" type="checkbox"/>
Паттерны проектирования для C# и платформы .NET Core	Гаурав Арора	700	70	<input checked="" type="checkbox"/>
Экстремальный Си. Параллелизм, ООП и продвинутые возможности	Камран Амини	800	80	<input checked="" type="checkbox"/>
Обратные вызовы в C++	Виталий Евгеньевич Ткаченко	900	90	<input checked="" type="checkbox"/>

Рисунок 24 – Тестирование. Каталог: несколько жанров

Стоит отметить, что выбираются книги по принципу ANY, то есть наличие хотя бы одного из выбранных жанров.

### 3.3.3 Тестирование изменения статуса заказов

Изменим статус заказа под номером 12 (см. Рисунок 25) на «Выполнен» (см. Рисунок 26).

Номер: 12
Статус: InDeliver

Рисунок 25 – Заказ 12 до выполнения изменения статуса

<a href="#">Главная страница</a> <a href="#">Каталог</a> <a href="#">Логи</a> <a href="#">Заказы</a>	
Заказы	
<div> <div>Номер: 11</div> <div>Статус: Completed</div> <div>Входящие книги</div> <div> <div>Название</div> <div>Как на самом деле работают компьютеры. Практическое руководство по внутреннему устройству машины</div> </div> <div> <div>Автор</div> <div>Адам Фримен</div> </div> <div> <div>Количество</div> <div>130</div> </div> <div> <div>Цена</div> <div>0</div> </div> <div> <div>Название</div> <div>Изучаем C#</div> </div> <div> <div>Автор</div> <div>Дженнифер Грин</div> </div> <div> <div>Количество</div> <div>60</div> </div> <div> <div>Цена</div> <div>600</div> </div> <div>Изменить статус</div> </div>	<div> <div>Номер: 12</div> <div>Статус: Completed</div> <div>Входящие книги</div> <div> <div>Название</div> <div>Параллельное программирование на современном C++</div> </div> <div> <div>Автор</div> <div>Мэттью Джастис</div> </div> <div> <div>Количество</div> <div>20</div> </div> <div> <div>Цена</div> <div>200</div> </div> <div> <div>Название</div> <div>Эффективный C. Профессиональное программирование</div> </div> <div> <div>Автор</div> <div>Райнер Гримм</div> </div> <div> <div>Количество</div> <div>30</div> </div> <div> <div>Цена</div> <div>3000</div> </div> <div>Изменить статус</div> </div>

Рисунок 26 – Изменение статуса заказа

Каждый заказ представлен в виде элементы таблицы и объекта card, благодаря чему можно удобным способом отображать книги, которые входят в заказ, но из-за этого заказы могут отображаться некорректно с визуальной точки зрения.

### 3.3.4 Просмотр окна логов и тестирование логирования системы

При тестировании заказов система записала действие изменения статуса заказа (см. Рисунок 27).

root	Внесены изменения в книгу с Id = 3	16.12.2023 19:08:32
root	Внесены изменения в книгу с Id = 4	16.12.2023 19:08:41
root	Внесены изменения в книгу с Id = 5	16.12.2023 19:08:48
root	Внесены изменения в книгу с Id = 6	16.12.2023 19:08:54
root	Внесены изменения в книгу с Id = 7	16.12.2023 19:09:03
root	Внесены изменения в книгу с Id = 8	16.12.2023 19:09:11
root	Внесены изменения в книгу с Id = 9	16.12.2023 19:09:19
root	root изменил статус заказа(12) на Completed	16.12.2023 19:13:57

Рисунок 27 – Просмотр логов приложения

На рисунке можно увидеть, что в ходе тестирования под учётной записью с логином root были изменены книги и внесено изменение в заказ.

## ЗАКЛЮЧЕНИЕ

По итогу курсового проекта был разработан модуль системы «Интернет магазин книг». Который представляет работу с каталогом книг, просмотр и обработку заказов, а также мониторинг действий сотрудников в системе.

Разработанный модуль является решением для автоматизации внутренних бизнес-процессов интернет-магазина книг. Он позволяет обеспечить удобную и эффективную работу интернет-магазина, повысить его конкурентоспособность и привлечь новых клиентов.

Реализация этого модуля позволяет интернет-магазину книг повысить эффективность работы сотрудников и обработки заказов. Это поможет сократить расходы и время сотрудников в таких процессах как каталогизация книг, их изменение и добавление, обработка заказов, что экономит время и ресурсы организации/площадки. За действиями сотрудников смогут наблюдать менеджеры и администраторы магазинов, что поможет при анализе работы сотрудников и потребностях клиентов.

Разработка такого программного модуля системы «Интернет магазин книг» является актуальной задачей ввиду множества площадок и магазинов, а также перспективной ввиду их расширения.

Также модуль был протестирован и в ходе тестирования были исправлены критические ошибки, которые могли нарушить работу модуля.

На основе полученных результатов работы сделаны следующие выводы:

Есть необходимость в разработке и интеграции дополнительных функций для программного модуля. В частности возможность регистрации новых сотрудников, так как сейчас это возможно только посредством прямого запроса в базу данных.

Улучшение интерфейса работы с заказами, а именно создать и организовать процедуры работы с заказами, а также добавить возможность поиска и фильтрации заказов.



Упрощение работы с книгами: добавить поиск по названию и автору книг в каталоге, в том числе возможность фильтровать по количеству и цене.

Добавить страницу для просмотра и обработки запросов от других модулей и сотрудников, которые могут пользоваться системой.

Следует также рассмотреть возможность расширения модуля и системы, а именно создание и организация остальных модулей системы таких как:

- клиентский интерфейс с оформлением заказов и соответствующими процедурами;
- редакторский интерфейс с возможностью добавления издательств и авторов;
- техническая поддержка с возможностью создавать запросы для остальных модулей.

Для исправления невыявленных ошибок и оптимизации приложения необходимо провести анализ эффективности работы разработанного модуля в реальных условиях.

Реализация этих вопросов позволит улучшить функциональность программного модуля и повысить его эффективность.

## СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

### Нормативно-правовые источники

1. ГОСТ 7.32–2017 Система стандартов по информации, библиотечному и издательскому делу. Отчет о научно-исследовательской работе. Структура и правила оформления
2. ГОСТ 7.1–2003. Межгосударственный стандарт. Система стандартов по информации, библиотечному и издательскому делу. Библиографическая запись. Библиографическое описание. Общие требования и правила составления
3. ГОСТ 7.9—95 Система стандартов по информации, библиотечному и издательскому делу. Реферат и аннотация. Общие требования
4. ГОСТ 7.11—2004 (ИСО 832:1994) Система стандартов по информации, библиотечному и издательскому делу. Библиографическая запись. Сокращение слов и словосочетаний на иностранных европейских языках
5. ГОСТ 7.12—93 Система стандартов по информации, библиотечному и издательскому делу. Библиографическая запись. Сокращение слов на русском языке. Общие требования и правила
6. ГОСТ 7.80—2000 Система стандартов по информации, библиотечному и издательскому делу. Библиографическая запись. Заголовок. Общие требования и правила составления
7. ГОСТ 7.82—2001 Система стандартов по информации, библиотечному и издательскому делу. Библиографическая запись. Библиографическое описание электронных ресурсов. Общие требования и правила составления
8. ГОСТ Р 7.0.97–2016 Национальный стандарт Российской Федерации. Система стандартов по информации, библиотечному и издательскому делу. Организационно-распорядительная документация. Требования к оформлению документов
9. ГОСТ Р 7.0.100–2018 Библиографическая запись. Библиографическое описание. Общие требования и правила составления
10. ГОСТ 19.101–77 Виды программ и программных документов
11. ГОСТ 19.102–77 Стадии разработки

### Учебники, учебные пособия, статьи

12. Эндрю Троелсен, Филипп Джепикс Язык программирования C# 7 и платформы .NET и .NET Core. 1330 страниц.
13. Стивен Клири Конкурентность в C#. Асинхронное, параллельное и многопоточное программирование. 304 страницы.
14. Эндрю Стилмен, Дженнифер Грин Изучаем C#. 768 страниц
15. Сусликов А.А. Программирование на C#. 192 страницы
16. Рикардо Террелл Конкурентность и параллелизм на платформе .NET.  
URL: <https://sd.blackball.lv/en/books/18890>

### Интернет-источники

17. Отношения между типами в операциях запросов LINQ (C#).  
URL: <https://learn.microsoft.com/ru-ru/dotnet/csharp/linq/get-started/type-relationships-in-linq-query-operations> (17.12.2023)
18. Blazor для разработчиков ASP.NET Web Forms. URL: <https://learn.microsoft.com/ru-ru/dotnet/architecture/blazor-for-web-forms-developers/> (17.12.2023)
19. Started with EF Core. URL: <https://learn.microsoft.com/en-us/ef/core/get-started/overview/first-app?tabs=netcore-cli> (17.12.2023)
20. Документация по использованию компонентов Bootstrap. URL: <https://getbootstrap.com/docs/5.3/getting-tarted/introduction/> (17.12.2023)
21. UserControl Класс. URL: <https://learn.microsoft.com/ru-ru/dotnet/api/system.windows.controls.usercontrol?view=windowsdesktop-8.0>
22. Использование внедрения зависимостей в ASP.NET Core. URL: <https://learn.microsoft.com/ru-ru/aspnet/core/fundamentals/dependency-injection?view=aspnetcore-8.0>
23. .NET MAUI. URL: <https://learn.microsoft.com/ru-ru/dotnet/maui/what-is-maui?view=net-maui-8.0#single-project>

Таблица 1 Анализ существующих решений

Характеристика	Amazon Books	Book Depository	AbeBooks	Google Play Books	ЛитРес
Ассортимент книг	Широкий	Широкий	Широкий	Широкий	Широкий
Цены	Низкие	Низкие	Средние	Средние	Средние
Доставка	Быстрая	Бесплатная по всему миру	Длительная	Быстрая	Быстрая
Электронные книги	Да	Да	Да	Да	Да
Аудиокниги	Да	Нет	Нет	Да	Нет

## Приложение 2

### План тестирования

No	Наименование функциональности	Наименования поля	Тестовый набор	Результат (должно получиться)	Результат тестирования
1	Редактирование, добавление книги	Название	Введены текстовые данные	Согласен с разработчиком	Сообщение: «Ошибка сохранения книги»
		Цена	Введены цифры меньше 0	Согласен с разработчиком	Сообщение: «Ошибка сохранения книги»
		Количество	Введены цифры меньше 0	Согласен с разработчиком	Сообщение: «Ошибка сохранения книги»
		Список жанров	Ничего не выбрано	Согласен с разработчиком	Создание книги без жанров
		Название	Все пробелы	Согласен с разработчиком	Сообщение: «Ошибка сохранения книги»
		Название	Введены текстовые данные с пробелами в начале и середине слова	Согласен с разработчиком	Книга добавлена, пробелы нормализованы
		-	Добавление книг с одинаковыми данными	Ошибка: «Книга уже добавлена»	Добавлена новая книга с существующим и значениями.
2	Авторизация	Логин и пароль	Ввод любых не подходящих значений	Согласен с разработчиком	Ошибка: «Неверные данные авторизации»
		Логин и пароль	Правильные данные авторизации	Согласен с разработчиком	Пользователь авторизован
3	Фильтр книг	Список жанров	Пустой список. Ничего не выбрано	Согласен с разработчиком	Отображается весь список.
		Список жанров	Выбрано несколько жанров	Согласен с разработчиком	Отобраны книги, в которых есть выбранные жанры

## Листинг 1. Функциональная часть страница авторизации

```
@code
{
    Person Person = default!;
    bool IsInvalid = false;

    private void Submit()
    {
        if (Person.Login != "" && Person.Password != "")
        {
            if (!authService.Login(navigation, Person.Login!, Person.Password!))
            {
                Person.Login = String.Empty;
                Person.Password = String.Empty;
                IsInvalid = true;
                InvokeAsync(StateHasChanged);
            }
        }
    }

    protected override void OnInitialized()
    {
        base.OnInitialized();
        Person = new();
    }
}
```

## Листинг 2. Функциональная часть главной страницы

```
@code {
    public void Logout()
    {
        authService.CurrentUser = null;
        navigation.NavigateTo("/");
    }
}
```

## Листинг 3. Функциональная часть страницы каталога

```
@code {
    bool IsLoading = true;
    List<Book> books = new();
    List<Genre> genres = new();
    List<Genre> selectedGenres = new();

    async Task<List<Book>> GetBooksWithFilter()
    {
        if (selectedGenres.Count == 0)
            return books;

        var filteredBooks = await dbcontext.Books
```

```

        .Include(x => x.Author)
        .Where(book => book.Genres.Any(bg => selectedGenres.Contains(bg)))
        .ToListAsync();

    return filteredBooks;
}

async Task Load()
{
    books = await dbcontext.Books
        .Include(x => x.Author)
        .ToListAsync();
    genres = await dbcontext.Genres.ToListAsync();

    IsLoading = false;
    await InvokeAsync(StateHasChanged);
}

protected override async Task OnInitializedAsync()
{
    await base.OnInitializedAsync();
    await InvokeAsync(Load);
}

private void ChangeBook(Book book)
{
    navigation.NavigateTo($"book_redact/{book.Id}");
}

private void ToggleGenreFilter(Genre genre)
{
    if (selectedGenres.Contains(genre))
    {
        selectedGenres.Remove(genre);
    }
    else
    {
        selectedGenres.Add(genre);
    }

    InvokeAsync(StateHasChanged);
}
}

```

#### Листинг 4. Функциональная часть страницы добавления книги

```

@code {
    Book bookNew = new();
    bool IsLoading = true;
    private bool ShowGenres = false;

    string MessageError = "";
    bool HasError = false;
    bool CanEscape = true;

    private Dictionary<int, bool> checkedGenres = new();

    List<Author> authors = new();
    List<Genre> genres = new();

    protected override async Task OnInitializedAsync()

```

```

{
    await base.OnInitializedAsync();

    authors = await dbcontext.Authors.ToListAsync();
    genres = await dbcontext.Genres.ToListAsync();
    foreach (var genre in genres)
    {
        checkedGenres.Add(genre.Id, false);
    }

    IsLoading = false;
    await InvokeAsync(StateHasChanged);
}

private async void BookCommit_change()
{
    bookNew.Name = Regex.Replace(bookNew.Name, @"\s+", " ");
    if (bookNew.Name == "" || bookNew.Price < 0 || bookNew.Amount < 0)
    {
        MessageError = "Ошибка при сохранении";
        HasError = true;
        CanEscape = false;
        InvokeAsync(StateHasChanged);
    }
    else
    {
        HasError = false;
        CanEscape = true;

        bookNew.Name = String.Join(' ', bookNew.Name.Split());

        foreach (var id in checkedGenres.Where(x => x.Value).Select(x =>
x.Key))
        {
            bookNew.Genres.Add(genres.First(x => x.Id == id));
        }

        bookNew.LastRedactorId = authService.CurrentUser!.Id;

        bookNew.AuthorId ??= authors.First().Id;

        try
        {
            dbcontext.Update(bookNew);

            dbcontext.Logs.Add(new Log
            {
                Operation = $"Добавлена книга с именем = {bookNew.Name}",
                UserId = authService.CurrentUser!.Id,
            });
        }
        catch (Exception ex)
        {
            Console.WriteLine(ex.Message);
        }

        await dbcontext.SaveChangesAsync();

        navigation.NavigateTo("books");
    }
}
}

```



## Листинг 5. Функциональная часть страницы редактирования книги

@code {

```
[Parameter] public int bookId { get; set; }

public bool ShowGenres = false;

public Dictionary<int, bool> checkedGenres = new();

string MessageError = "";
bool HasError = false;
bool CanEscape = true;

Book bookLocal = default!;
bool IsLoading = true;
List<Author> authors = new();
List<Genre> genres = new();

protected async override Task OnInitializedAsync()
{
    await base.OnInitializedAsync();

    bookLocal = await dbcontext.Books.Include(x => x.Genres).FirstAsync(b =>
b.Id == bookId);
    authors = await dbcontext.Authors.ToListAsync();
    genres = await dbcontext.Genres.ToListAsync();
    foreach (var genre in genres)
    {
        checkedGenres.Add(genre.Id, bookLocal.Genres.Any(x => x.Id ==
genre.Id));
    }

    IsLoading = false;
    await InvokeAsync(StateHasChanged);
}

public async void BookCommit_change()
{
    bookLocal.Name = Regex.Replace(bookLocal.Name, @"\s+", " ");
    if (bookLocal.Name == "" || bookLocal.Price < 0 || bookLocal.Amount < 0)
    {
        MessageError = "Ошибка при сохранении";
        HasError = true;
        CanEscape = false;
        InvokeAsync(StateHasChanged);
    }
    else
    {
        HasError = false;
        CanEscape = true;

        bookLocal.Genres.Clear();
        foreach (var id in checkedGenres.Where(x => x.Value).Select(x =>
x.Key))
        {
            bookLocal.Genres.Add(genres.First(x => x.Id == id));
        }

        bookLocal.LastRedactorId = authService.CurrentUser!.Id;

        dbcontext.Update(bookLocal);
    }
}
```

```

        dbcontext.Logs.Add(new Log
        {
            Operation = $"Внесены изменения в книгу с Id = {bookLocal.Id}",
            UserId = authService.CurrentUser!.Id,
        });
        await dbcontext.SaveChangesAsync();
        navigation.NavigateTo("books");
    }
}

```

## Листинг 6. Функциональная часть страницы просмотра логов

```

@code {
    bool IsLoading = true;
    public List<Log> Logs { get; set; } = new();

    protected override async Task OnInitializedAsync()
    {
        await base.OnInitializedAsync();

        Logs = dbcontext.Logs.Include(x => x.User).ToList();

        IsLoading = false;
        await InvokeAsync(StateHasChanged);
    }
}

```

## Листинг 7. Функциональная часть страницы заказов

```

@code {
    bool IsLoading = true;
    public List<Order> OrderList { get; set; } = new();

    protected override async Task OnInitializedAsync()
    {
        await base.OnInitializedAsync();

        OrderList = dbcontext.Orders
            .Include(x => x.Books)
            .ThenInclude(b => b.Author)
            .Include(x => x.Performer)
            .ToList();

        IsLoading = false;
        await InvokeAsync(StateHasChanged);
    }

    public void ChangeOrderStatus(Order order, OrderStatus status)
    {
        order.orderStatus = status;
        dbcontext.Logs.Add(new Log
        {
            Operation = $"{authService.CurrentUser.Login} изменил статус
заказа({order.Id}) на {status}",
            UserId = authService.CurrentUser.Id,
        });
        dbcontext.SaveChangesAsync();
    }
}

```

```

        InvokeAsync(StateHasChanged);
    }
}

```

### Листинг 8. Модель пользователя на странице авторизации

```

public class Person
{
    [StringLength(15, ErrorMessage = "Login is too long."), Required]
    public string? Login { get; set; }

    [StringLength(20, ErrorMessage = "Password is too long."), Required]
    public string? Password { get; set; }
}

```

### Листинг 9. Сервис авторизации

```

public class AuthService
{
    private AppDbContext dbContext { get; set; }
    public User? CurrentUser { get; set; }

    public AuthService(AppDbContext DbContext)
    {
        dbContext = DbContext;
    }

    /// <summary>
    ///
    /// </summary>
    /// <param name="login"></param>
    /// <param name="password"></param>
    public bool Login(NavigationManager navigation, string login, string
password)
    {
        var user = dbContext.Users.FirstOrDefault(x => x.Login == login &&
x.Password == password);
        if (user == null)
            return false;
        CurrentUser = user;
        navigation.NavigateTo("personal_page");
        return true;
    }
}

```

### Листинг 10. Файл инициализации программы. Класс MauiProgram

```

public static class MauiProgram
{
    public static MauiApp CreateMauiApp()
    {
        if (AppDomain.CurrentDomain.BaseDirectory.Contains(@"\bin\"))
        {
            Environment.CurrentDirectory =
                AppDomain.CurrentDomain.BaseDirectory.Substring(0,
AppDomain.CurrentDomain.BaseDirectory.IndexOf("BookDesktop"));
        }
    }
}

```

```

        var builder = MauiApp.CreateBuilder();
        builder
            .UseMauiApp<App>()
            .ConfigureFonts(fonts => { fonts.AddFont("OpenSans-Regular.ttf",
"OpenSansRegular"); });

        builder.Services.AddMauiBlazorWebView();

#if DEBUG
        builder.Services.AddBlazorWebViewDeveloperTools();
        builder.Logging.AddDebug();
#endif

        builder.Services.AddSingleton<AuthService>();
        builder.Services.AddSingleton<BookChangeService>();
        builder.Services.AddDbContext<AppDbContext>(options) =>
        {
            options.UseSqlite($"Data Source={Environment.CurrentDirectory +
@"\MyDatabase.db"}");
        });
        builder.Services.AddSingleton<BinService>();
        var app = builder.Build();
        var dbContext = app.Services.GetRequiredService<AppDbContext>();
        dbContext.Database.EnsureCreated();

#if DEBUG
        DbDataFill(dbContext);
#endif
        return app;
    }

    private async static void DbDataFill(AppDbContext dbContext)
    {
        if (dbContext.Users.ToList().Count < 4) //root
        {
            dbContext.Users.Add(new User
            {
                Id = 1,
                Email = "RootEmail@gmail.com",
                Login = "root",
                Password = "root",
                PermissionLvl = 0
            });
            dbContext.Users.Add(new User //admin
            {
                Id = 2,
                Email = "AdminEmail@gmail.com",
                Login = "admin",
                Password = "admin",
                PermissionLvl = 1
            });
            dbContext.Users.Add(new User //employeeer1
            {
                Id = 3,
                Email = "EmplEmail@gmail.com",
                Login = "employeeer1",
                Password = "employeeer1",
                PermissionLvl = 2
            });
            dbContext.Users.Add(new User //employeeer2
            {
                Id = 4,
                Email = "Emp2Email@gmail.com",

```

```

        Login = "employeeer2",
        Password = "employeeer2",
        PermissionLvl = 2
    });
    dbContext.SaveChanges();
}

if (dbContext.Authors.Where(author => author.Name ==
"AuthorTestName").ToList().Count == 0) //Author
{
    dbContext.Authors.Add(new Author { Id = 1, Name = "AuthorTestName"
});
    dbContext.SaveChanges();
}

if (dbContext.Books.Where(book =>
book.Name.Contains("testbookName")).ToList().Count == 0) //Book
{
    for (int i = 1; i <= 10; i++)
    {
        var lastGenre = new Genre();
        if (dbContext.Genres.Where(x => x.Id == i).FirstOrDefault() ==
null)
        {
            lastGenre = new Genre { Id = i, Name = $"{i}TestGenreName"
};
            dbContext.Genres.Add(lastGenre);
        }

        var book = new Book(
            i, $"testbookName{i}", dbContext.Authors.First(), i * 100, i
* 10);

        book.Genres.Add(lastGenre);
        dbContext.Books.Add(book);
    }
    dbContext.SaveChanges();
}

if (await dbContext.Orders.CountAsync() == 0)
{
    foreach (var book in await dbContext.Books.ToListAsync())
    {
        dbContext.Orders.Add(new
Order(Enum.GetValues<OrderStatus>().RandomElement(),
dbContext.Users.RandomElement().Id, book, dbContext.Books.RandomElement()));
    }
    dbContext.SaveChanges();
}

}

public static T RandomElement<T>(this IEnumerable<T> enumerable)
{
    return enumerable.RandomElementUsing<T>(new Random());
}

public static T RandomElementUsing<T>(this IEnumerable<T> enumerable, Random
rand)
{
    int index = rand.Next(0, enumerable.Count());
    return enumerable.ElementAt(index);
}
}

```

## Листинг 11. Класс AppDbContext

```
public class AppDbContext : DbContext
{
    public AppDbContext(DbContextOptions options) : base(options)
    {
    }

    protected AppDbContext()
    {
    }

    public DbSet<User> Users { get; set; }
    public DbSet<Book> Books { get; set; }
    public DbSet<Author> Authors { get; set; }
    public DbSet<Genre> Genres { get; set; }
    public DbSet<Order> Orders { get; set; }
    public DbSet<Log> Logs { get; set; }

    //public string databasepath = $"Data
Source={@"C:\Projects\CourseWork\MyDatabase.db"}";

    //"Host=127.0.0.1;Port=5432;Database=KRVBooks;Username=TestGroupLocalhost;Passwo
rd=postgres;";

    protected override void OnConfiguring(DbContextOptionsBuilder
optionsBuilder)
    {
        // Путь к базе данных SQLite. Файл будет создан в текущей директории
        приложения. (bin/Debug/)
        //var databasePath =
        System.IO.Path.Combine(System.Environment.CurrentDirectory, "MyDatabase.db");

        //optionsBuilder.UseSqlite($"Data Source={databasePath}");
        //optionsBuilder.UseNpgsql(databasepath);
        //optionsBuilder.UseSqlite(databasepath);
    }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<Book>()
            .HasMany(b => b.Genres)
            .WithMany(bg => bg.Books);

        modelBuilder.Entity<Order>()
            .HasMany(o => o.Books)
            .WithMany(b => b.Orders);

        modelBuilder.Entity<User>()
            .HasMany(x => x.Logs)
            .WithOne(b => b.User);
    }
}
```

## Листинг 12. Класс Author

```
public class Author
{
    [Key, DatabaseGenerated(DatabaseGeneratedOption.Identity)]
    public int Id { get; set; }
    public string Name { get; set; } = "AuthorName";
}
```

```

public Author()
{

}

public override string ToString()
{
    return Name;
}
}

```

### Листинг 13. Класс Book

```

public class Book : INotifyPropertyChanged
{
    [Key, DatabaseGenerated(DatabaseGeneratedOption.Identity)]
    public int Id { get; set; }
    public string Name { get; set; } = "BookName";
    public double Price { get; set; }
    public int? AuthorId { get; set; }
    [ForeignKey(nameof(AuthorId))]
    public Author? Author { get; set; } = null;
    public byte[]? Image { get; set; }
    public int Amount { get; set; } = 0;
    public virtual List<Genre> Genres { get; set; } = new();
    public virtual List<Order> Orders { get; set; } = new();
    public int LastRedactorId { get; set; }

    public Book(int id, string name, Author author, double price, int amount) :
    this() =>
        (this.Id, this.Name, this.Author, this.Price, this.Amount) = (id, name,
        author, price, amount);

    public Book()
    {

    }

    public Book(Book book)
    {
        Name = book.Name;
        AuthorId = book.AuthorId;
        LastRedactorId = book.LastRedactorId;
        Genres.AddRange(book.Genres);
        Price = book.Price;
        Amount = book.Amount;
    }

    public event PropertyChangedEventHandler? PropertyChanged;

    protected virtual void Dirty(string propertyName)
    {
        PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(nameof(Id)));
        PropertyChanged?.Invoke(this, new
        PropertyChangedEventArgs(nameof(Name)));
        PropertyChanged?.Invoke(this, new
        PropertyChangedEventArgs(nameof(Author)));
        PropertyChanged?.Invoke(this, new
        PropertyChangedEventArgs(nameof(Price)));
        PropertyChanged?.Invoke(this, new
        PropertyChangedEventArgs(nameof(Genres)));
    }
}

```

```
}  
}
```

#### Листинг 14. Класс Genre

```
public class Genre  
{  
    [Key, DatabaseGenerated(DatabaseGeneratedOption.Identity)]  
    public int Id { get; set; }  
    public string Name { get; set; }  
  
    public Genre()  
    {  
  
    }  
  
    public override string ToString()  
        => Name;  
  
    public virtual List<Book> Books { get; set; } = new();  
  
    public override bool Equals(object? obj)  
    {  
        if (obj is Genre genre)  
        {  
            return this.Id == genre.Id;  
        }  
        return false;  
    }  
}
```

#### Листинг 15. Класс Log

```
public class Log  
{  
    [Key, DatabaseGenerated(DatabaseGeneratedOption.Identity)]  
    public int Id { get; set; }  
    public string Operation { get; set; }  
    public DateTime Created { get; set; } = DateTime.Now;  
    public int UserId { get; set; }  
    [ForeignKey(nameof(UserId))]  
    public User? User { get; set; }  
  
    public Log() { }  
}
```

#### Листинг 16. Класс Order

```
public enum OrderStatus  
{  
    New,  
    Approved,  
    InDeliver,  
    Completed,  
    Cancelled  
}  
  
public class Order  
{
```



```

[Key, DatabaseGenerated(DatabaseGeneratedOption.Identity)]
public int Id { get; set; }
public virtual List<Book> Books { get; set; } = new();
public int PerformerId { get; set; }
[ForeignKey(nameof(PerformerId))]
public User? Performer { get; set; }
public int LastRedactedId { get; set; }
[ForeignKey(nameof(LastRedactedId))]
public User? LastRedacted { get; set; }
public OrderStatus orderStatus { get; set; }
public Order()
{
}

public Order(OrderStatus status, int UserId=1, params Book[] books) : this()
{
    this.Books = books.ToList();
    this.orderStatus = status;
    this.LastRedactedId = UserId;
    this.PerformerId = UserId;
}
}

```

## Листинг 17. Класс User

```

public class User : INotifyPropertyChanged
{
    [Key, DatabaseGenerated(DatabaseGeneratedOption.Identity)]
    public int Id { get; set; }
    public string? Login { get; set; }
    public string? Email { get; set; }
    public string? Password { get; set; }
    public int PermissionLvl { get; set; }
    public User(int id, string name, string email = "", string password = "") :
this() =>
        (this.Id, this.Login, this.Email, this.Password) = (id, name, email,
password);

    public virtual List<Log> Logs { get; set; } = new();

    public User()
    {
    }

    public override string ToString()
    {
        return PermissionLvl switch
        {
            0 => "root",
            1 => "Администратор",
            2 => "Сотрудник",
            _ => "???"
        };
    }

    public event PropertyChangedEventHandler? PropertyChanged;

    protected virtual void Dirty(string propertyName)
    {
        PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(nameof(Id)));
    }
}

```

```
        PropertyChanged?.Invoke(this, new
PropertyChangedEventArgs(nameof(Login)));
        PropertyChanged?.Invoke(this, new
PropertyChangedEventArgs(nameof(Email)));
        PropertyChanged?.Invoke(this, new
PropertyChangedEventArgs(nameof>Password));
        PropertyChanged?.Invoke(this, new
PropertyChangedEventArgs(nameof(PermissionLvl)));
    }
}
```