**ВВЕДЕНИЕ**

В последние годы электронная коммерция стала неотъемлемой частью повседневной жизни. Интернет-магазины позволяют людям приобретать товары и услуги без необходимости покидать дом, экономя время и усилия. Особенно стремительно развивается рынок цифровой продажи книг, где пользователи могут удобно находить, приобретать и скачивать литературу в электронном виде или заказывать печатные издания с доставкой [1]. Это делает систему управления интернет-магазином книг актуальной и востребованной.

Современные пользователи ожидают от онлайн-магазина не только широкий ассортимент книг, но и интуитивно понятный интерфейс, персонализированные рекомендации, систему отзывов, а также простой и безопасный процесс оформления заказов [2]. Поэтому создание специализированного веб-приложения для продажи книг требует комплексного подхода, сочетающего удобство, функциональность и безопасность.

Целью данной выпускной квалификационной работы является разработка полнофункционального веб-приложения интернет-магазина книг с возможностью управления заказами, пользователями, книжными наборами (комбо-наборами) и отзывами. В приложении реализована авторизация с использованием JWT, разграничение прав доступа по ролям (администратор, сотрудник, покупатель) [3], система управления каталогом книг и комбо-наборов, оформление и отслеживание заказов, а также модуль рекомендаций книг.

Для реализации проекта были использованы современные технологии:

- язык программирования **Go** и фреймворк **Gin** для серверной части;

- библиотека **GORM** для работы с базой данных **PostgreSQL**;

- механизм **JWT** для аутентификации и авторизации;

- **Docker** для контейнеризации приложения;

- а также **Vue 3, TypeScript, Pinia, i18n, Tailwind CSS** для клиентской части;

- в отдельных модулях — поддержка загрузки и хранения изображений, а также PDF-файлов.

Разработка веб-приложения осуществлялась по модели клиент-сервер с REST-архитектурой. Особое внимание было уделено модульной структуре, безопасности и расширяемости решения. Приложение позволяет администраторам и сотрудникам управлять ассортиментом, получать статистику и настраивать параметры магазина, в то время как покупатели могут регистрироваться, просматривать книги, оформлять заказы и оставлять отзывы.

Работа включает в себя анализ существующих решений, формализацию требований, проектирование архитектуры, реализацию ключевых функций, тестирование, а также описание процесса развертывания системы.

Таким образом, результатом данной работы является веб-приложение интернет-магазина книг, готовое к использованию и дальнейшему расширению

**1. Анализ аналогов**

Перед началом проектирования и разработки собственного веб-приложения для интернет-магазина книг важно провести анализ существующих аналогов. Это позволит выявить сильные и слабые стороны конкурентов, а также определить основные функции, которые ожидают пользователи от современного книжного онлайн-магазина [4].

**1.1 Ozon (ozon.ru) [5]**

**Tiêu đề**

|  |  |
| --- | --- |
| **Достоинства** | **Недостатки** |
| Огромный выбор книг в различных форматах (печатные, аудиокниги, электронные) | Сложность интерфейса для новых пользователей |
| Удобная система поиска и фильтрации по жанру, автору, рейтингу и цене | Слишком общий каталог, где книги – лишь часть большого ассортимента товаров |
| Интеграция с бонусной программой и системой персональных рекомендаций | Отсутствие гибкого выбора комбо-наборов книг по интересам |
| Возможность оставлять отзывы и оценки |  |
| Быстрая доставка и отслеживание заказов |  |

**1.2 ЛитРес (litres.ru) [6]**

**Tiêu đề**

|  |  |
| --- | --- |
| **Достоинства** | **Недостатки** |
| Специализация именно на электронных и аудиокнигах | Ограниченная функциональность по части пользовательских настроек и кастомных подборок |
| Удобное приложение и синхронизация между устройствами | Отсутствие системы доставки печатных книг |
| Возможность чтения онлайн без загрузки | Нет возможности приобретать книги в составе тематических наборов |
| Интеграция с авторскими профилями и сериями книг |  |

**1.3 Read.ru [7]**

**Tiêu đề**

|  |  |
| --- | --- |
| **Достоинства** | **Недостатки** |
| Упор на печатную продукцию (учебники, художественная литература) | Устаревший пользовательский интерфейс |
| Возможность заказа книг по учебным программам | Слабая адаптация под мобильные устройства |
| Простая система оформления заказа | Ограниченный функционал личного кабинета |

**1.4 Выводы по анализу аналогов**

На основе анализа можно выделить ключевые элементы, которые необходимо включить в разрабатываемое веб-приложение:

* Четкое разделение ролей пользователей: администратор, сотрудник, покупатель.
* Поддержка как цифрового (PDF), так и печатного формата книг.
* Удобная система комбо-наборов книг (по жанру, теме, возрасту и т.п.).
* Простой, интуитивно понятный интерфейс [8][9].
* Интерактивный каталог с отзывами, рейтингами и фильтрами.
* Поддержка хранения и отображения обложек книг и сопроводительных материалов (например, аннотации в формате PDF).

Таким образом, предлагаемое решение будет сочетать лучшие черты существующих платформ и расширять возможности пользователя за счёт гибкости управления, кастомизации, и интеграции с современными средствами аутентификации и хранения данных.

**2. Анализ требований**

**2.1. Функциональные и нефункциональные требования**

Для определения требований к веб-приложению интернет-магазина были проанализированы следующие источники:

- **Законодательство РФ**, регулирующее электронную торговлю книгами — в частности, **Федеральный закон «О защите прав потребителей» № 2300-1**, статья 8, 10 и 12. Закон предписывает предоставление достоверной информации о товаре, прозрачность условий покупки и соблюдение прав пользователя при дистанционной торговле [1].

- Функциональность аналогов, рассмотренная в главе 1 (Ozon, ЛитРес, Read.ru).

- Также были использованы открытые исследования пользовательского поведения в интернет-магазинах: Отчёт **Data Insight**: "Книжный рынок в онлайне" [2]; **Социологическое исследование поведения покупателей** в интернет-магазине — CyberLeninka [3].

На основании этих источников сформулированы бизнес-требования, из которых вытекают требования к программному обеспечению.

**а. Функциональные требования**

Функциональные требования описывают действия, которые система должна уметь выполнять. Они делятся на основные, вспомогательные и дополнительные в зависимости от их влияния на ключевую бизнес-цель — продажу книг и управление заказами.

Основные функциональные требования (ключевые бизнес-процессы):

- Пользователь может просматривать каталог книг по категориям и авторам.

- Пользователь может добавлять книги в корзину и оформлять заказ.

- Администратор может добавлять, редактировать и удалять книги.

- Система должна поддерживать регистрацию, вход и разграничение ролей (Admin, Staff, Customer).

- Сотрудник может просматривать список заказов и изменять их статус (в обработке, отправлен, доставлен).

- Система должна сохранять историю заказов пользователей.

- Система должна предоставлять API-интерфейс для всех функций.

Вспомогательные функциональные требования (сопутствующие функции):

- Возможность создавать и управлять комбо-наборами книг.

- Возможность загружать обложку книги и файл PDF.

- Возможность фильтровать книги по скидкам или популярности.

- Администратор может управлять системными настройками магазина (логотип, сообщения, языки).

Дополнительные функциональные требования (дополняют удобство):

- Возможность мультиязычного интерфейса (русский / аглиский).

- Возможность интеграции хелпер-чата для помощи в выборе книг.

- Возможность просмотра статистики продаж (для администратора).

- Возможность поиска книг по ключевым словам.

- Покупатель может оставить отзыв о книге после покупки.

**в. Нефункциональные требования**

Нефункциональные требования описывают, как система должна работать:

- Приложение должно быть доступно 24/7, за исключением периодов обновлений.

- Аутентификация должна быть реализована через JWT с истекающим временем токена.

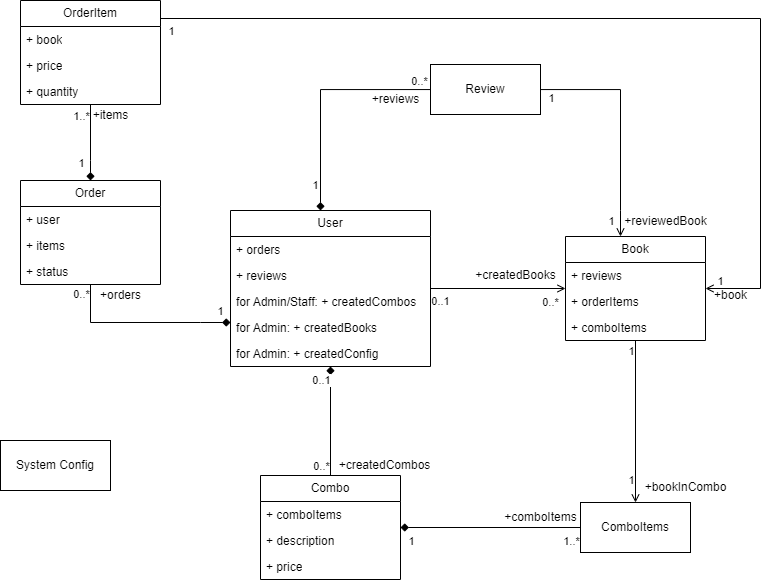
- Все данные должны храниться в реляционной базе PostgreSQL.

- Приложение должно отвечать на REST-запросы не дольше 1 секунды.

- Поддержка контейнеризации с помощью Docker.

**2.2. Модель предметной области**

Представленная модель предметной области была разработана на основе анализа пользовательских сценариев и требований к функциональности. Она обеспечивает целостное представление об основных объектах системы и логике их взаимодействия, что важно на этапе проектирования базы данных и программной архитектуры. На рисунке представлена модель предметной области веб-приложения интернет-магазина книг. Сущность User является центральной и взаимодействует с другими элементами системы, такими как Order, Book, Review и Combo. Модель отражает логические связи между объектами, возникающие в процессе использования приложения: оформление заказов, публикация отзывов, управление ассортиментом и настройками системы.



Tên hình

Модель предметной области построена на основе бизнес-требований и отражает ключевые процессы взаимодействия пользователей с системой. Пользователь (User) может выполнять действия в зависимости от своей роли:

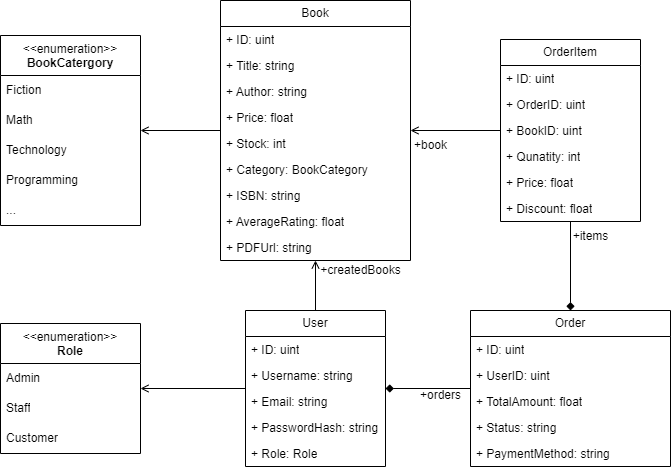
- Покупатель оформляет заказы (Order), оставляет отзывы (Review) и просматривает книги (Book).

- Сотрудник или администратор может добавлять книги и формировать комбо-наборы (Combo).

- Каждая книга может входить одновременно в несколько заказов, отзывов и комбо-наборов — что реализовано через сущности OrderItem, Review и ComboItem.

- Сущность SystemConfig содержит глобальные настройки приложения и доступна только администраторам.

**2.3. Реализация классов предметной области**



Tên hình

На рисунке представлена диаграмма классов, отражающая ключевые сущности веб-приложения и их взаимосвязи. Диаграмма построена на основе моделей GORM, используемых в проекте. Основные элементы диаграммы:

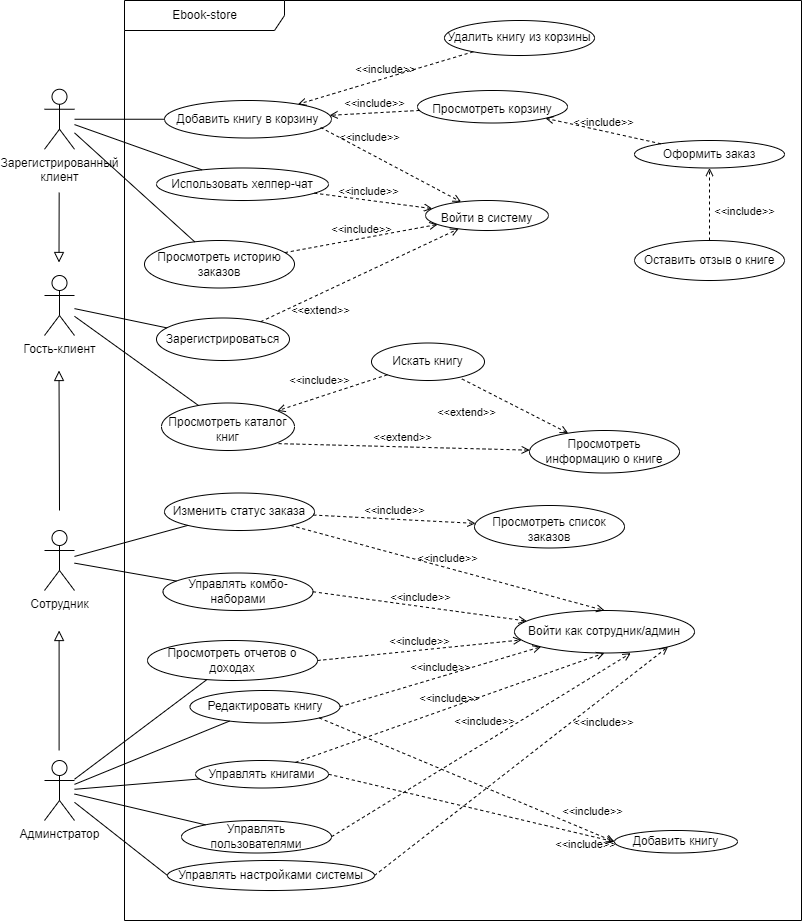
- Пользователь (User) связан с несколькими заказами (Order) и отзывами (Review);

- Книга (Book) может участвовать во многих заказах и иметь множество отзывов;

- Заказ (Order) содержит одну или несколько книг через промежуточную таблицу OrderItem;

**2.4. Диаграмма вариантов использования**

Варианты использования, представленные на рисунке, позволяют сформировать представление о логике доступа и разделении ответственности между ролями.



Tên hình

Модель ролевого доступа (RBAC — Role-Based Access Control) реализована на уровне бизнес-логики и поддерживается в интерфейсе пользователя. Каждая роль в системе имеет строго определённый набор допустимых действий: Покупатель — только просмотр и покупка книг, а также отзывы; Сотрудник — доступ к управлению заказами, но не к книгам и пользователям; Администратор — полный контроль над сущностями: книги, комбо-наборы, заказы, пользователи, конфигурации. Такой подход полностью соответствует рекомендациям OWASP по управлению правами в веб-приложениях [1].

Структура вариантов использования построена модульно, и позволяет легко масштабировать проект за счёт добавления новых функций: Например, в будущем можно добавить сценарии: «Рекомендация книг на основе истории заказов», «Онлайн-чат с консультантом», «Просмотр статистики продаж» — и это не потребует пересмотра существующих ролей. Варианты использования уже хорошо отделены друг от друга — это упрощает рефакторинг и внедрение микросервисов при необходимости.

Важно отметить, что use-case диаграмма не только определяет действия пользователя, но и отражает основные бизнес-процессы, такие как: Принятие и обработка заказа; Управление ассортиментом; Обслуживание покупателя (через отзывы и личный кабинет). Это делает диаграмму не просто техническим артефактом, а важным инструментом формализации бизнес-логики проекта.

**2.5. Диаграмма деятельности — сценарии взаимодействия пользователей**

В рамках данной работы была составлена обобщённая диаграмма деятельности (xem hình), моделирующая типичный сценарий: от входа в систему до оформления заказа и последующего отзыва. Данный процесс включает действия всех трёх участников системы: пользователя, интерфейса приложения и серверной части.

Типовой пользовательский путь включает следующие основные этапы:

- запуск приложения и авторизация;

- просмотр и фильтрация каталога книг;

- добавление книги в корзину;

- оформление и подтверждение заказа;

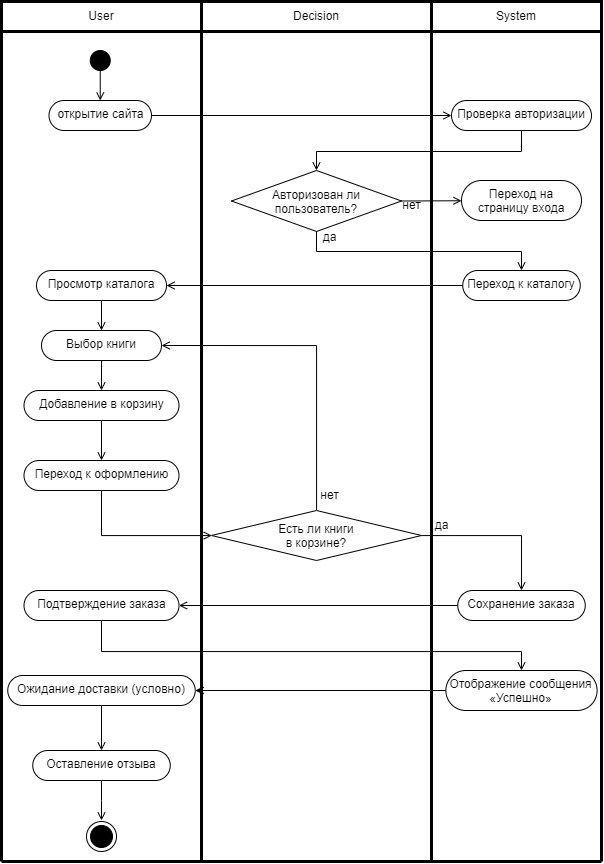
- получение подтверждения от системы;

- после доставки — написание отзыва.

Диаграмма включает разветвления (decision nodes), например:

- проверка авторизации: если пользователь не авторизован — его перенаправляют на страницу входа;

- проверка наличия книг в корзине: если корзина пуста — оформление заказа невозможно.



Activity Diagram – Обобщённое взаимодействие

**3. Реализация серверной части (backend)**

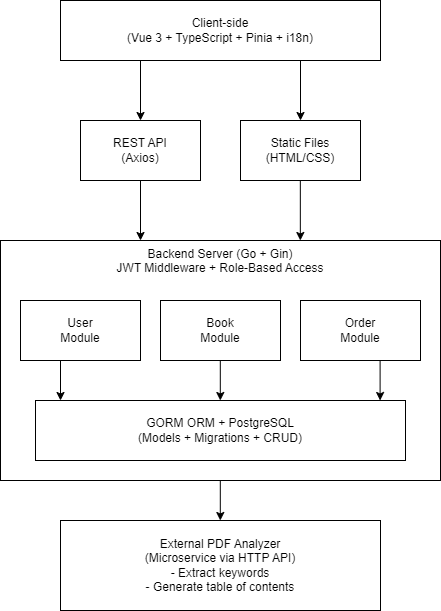
На основании сформулированных требований, рассмотренных в предыдущей главе, была спроектирована и реализована веб-система, обеспечивающая полный цикл функционирования интернет-магазина: от регистрации пользователя до оформления заказов, управления ассортиментом и получения отзывов. При разработке были учтены как функциональные, так и нефункциональные требования, в частности — безопасность, масштабируемость, поддержка нескольких ролей и гибкость расширения.

Проект реализован в виде клиент-серверного веб-приложения, где серверная часть отвечает за бизнес-логику и взаимодействие с базой данных, а клиентская — за отображение информации и интерфейсную логику. Архитектура строится на основе принципов REST и разделения ответственности между слоями.

Следующие разделы главы посвящены подробному разбору архитектурных решений, технологического стека, логики API, структуры данных и взаимодействия компонентов системы.

**3.1. Архитектура приложения**

В процессе реализации приложения особое внимание было уделено выбору архитектурного подхода, обеспечивающего гибкость, масштабируемость и техническую устойчивость проекта. Система построена по принципу клиент-серверной модели с чётким разделением ответственности между компонентами, каждый из которых решает строго определённую задачу. Это решение позволило добиться независимости пользовательского интерфейса от внутренней бизнес-логики и упростило развертывание как в среде разработки, так и в продуктивной инфраструктуре.



Tên hình Архитектура приложения

Серверная часть разработана на языке Go с использованием фреймворка Gin. Она представляет собой API-ориентированное приложение, предоставляющее доступ к ключевым функциям системы — от регистрации пользователей до оформления заказов, публикации отзывов и управления каталогом книг. Логика доступа реализована через middleware-механизмы с использованием JWT, что позволило разграничить роли и минимизировать уязвимости на уровне авторизации. В качестве средства работы с базой данных используется ORM-библиотека GORM, что обеспечило согласованность моделей с предметной областью и ускорило разработку за счёт автоматизации миграций и управления связями между сущностями.

Особенностью архитектуры является наличие отдельного сервиса, реализующего анализ содержимого книг. Этот микросервис подключается к основной системе через HTTP-интерфейс и обрабатывает загруженные PDF-файлы с целью извлечения ключевых слов и построения структуры оглавления. Данный подход позволил вынести ресурсоёмкую задачу за пределы основного backend-приложения, сохранив производительность и улучшив масштабируемость. Такая изоляция логики анализа также упростила возможное обновление алгоритмов или замену технологического стека без вмешательства в основную кодовую базу.

Клиентская часть реализована с использованием Vue 3 и TypeScript, что позволило создать компонентную архитектуру с поддержкой повторного использования и расширяемости. Взаимодействие с сервером осуществляется через REST API посредством библиотеки Axios, а управление состоянием — с помощью хранилища Pinia. Механизм интернационализации обеспечивается за счёт i18n, что особенно важно в контексте целевой аудитории приложения, включающей пользователей, использующих русский и английский языки.

Хотя система в текущем виде содержит один монолитный backend-сервер, наличие внешнего сервиса анализа PDF-документов свидетельствует о постепенном переходе к микросервисной архитектуре. Такая модель открывает возможности для дальнейшей декомпозиции системы на независимые модули — например, вынос логики рекомендаций книг, генерации статистики или обработки изображений в отдельные сервисы. Таким образом, архитектура приложения закладывает фундамент для эволюции системы в сторону гибкой распределённой среды, способной адаптироваться под растущие требования бизнеса.

**3.2. API и маршруты**

В рамках реализации серверной части приложения был разработан полноценный REST API, предоставляющий доступ ко всем основным функциям, определённым в разделе требований. Концепция проектирования маршрутов была основана на принципах согласованности, предсказуемости и разделения ответственности. Каждый маршрут чётко соответствует одной сущности предметной области и реализует типичные CRUD-операции, а также специализированные действия, характерные для конкретной бизнес-логики (например, оформление заказа или публикация отзыва).

Организация маршрутов построена по модели группировки по функциональным модулям: книги, пользователи, заказы, отзывы, комбо-наборы и системные настройки. Для каждого из этих блоков предусмотрены как публичные, так и защищённые маршруты. Публичная часть охватывает действия, доступные всем пользователям — такие как просмотр каталога книг или регистрация. Защищённая часть требует наличия валидного JWT-токена и проходит через цепочку middleware, включая проверку авторизации, логирования и контроля доступа по ролям.

func RoleCheck(requiredRole string) gin.HandlerFunc {

    return func(c \*gin.Context) {

        userRole := c.GetString("userRole")

        if userRole != requiredRole {

            c.AbortWithStatusJSON(

                http.StatusForbidden,

                gin.H{"error": "Access denied"})

            return

        }

        c.Next()

    }

}

Tên listing

В целях безопасности все чувствительные маршруты, связанные с изменением данных, требуют соответствующего уровня прав: сотрудники, например, могут обновлять статусы заказов, но не имеют доступа к управлению книгами, тогда как администраторы обладают расширенными возможностями. Эти ограничения реализуются на уровне middleware **role\_check.go**, который проверяет привязку пользователя к определённой роли и принимает решение о возможности выполнения запроса.

Особое внимание при проектировании API было уделено предсказуемости структуры маршрутов, что выражается в соблюдении соглашений REST: использование HTTP-методов (GET, POST, PUT, DELETE) в соответствии с действием, логичное наименование конечных точек (/books, /orders, /auth/login), и передача параметров через URL или тело запроса в зависимости от контекста.

func SetupRouter(

    db \*gorm.DB,

    cfg \*config.Config) \*gin.Engine {

    api := router.Group("/api")

    api.POST("/auth/register", authController.Register)

    api.POST("/auth/login", authController.Login)

    books := api.Group("/books")

    books.GET("", bookController.GetAll)

    books.GET("/:id", bookController.GetByID)

    admin := api.Group("/admin").Use(

        AuthMiddleware(),

        RoleCheck("admin"))

    admin.POST("/books", bookController.Create)

    admin.PUT("/books/:id", bookController.Update)

    admin.DELETE("/books/:id", bookController.Delete)

}

Tên listing

Взаимодействие с внешним микросервисом анализа PDF-файлов также происходит через отдельный API. Этот сервис не имеет своего пользовательского интерфейса и используется исключительно backend-приложением. При добавлении или редактировании книги, содержащей PDF-документ, основной сервер отправляет запрос к вспомогательному сервису, получает от него структуру содержимого и ключевые слова, и сохраняет их в связанную с книгой запись в базе данных. Такая организация взаимодействия способствует расширяемости архитектуры и позволяет при необходимости заменить анализатор или изменить формат взаимодействия без вмешательства в остальной код.

Таким образом, API-прослойка приложения выполняет роль основного связующего элемента между интерфейсом, бизнес-логикой и вспомогательными сервисами. Чёткая структура маршрутов, разграничение прав и соответствие принципам REST обеспечивают надёжность, расширяемость и прозрачность всей системы.