Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ   
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина: Модели данных и системы управления базами данных

*К защите допустить*:

И.О. Заведующего кафедрой

\_\_\_\_\_\_\_\_\_\_\_\_С.И. Сиротко

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовой работе

на тему

**Программное средство для интернет-магазина книг**

БГУИР КП 1-40 04 01 01 016 ПЗ

Студент Н.В. Захаренко

Руководитель А.В.Давыдчик

Нормоконтролер А.В.Давыдчик

Минск 2024

СОДЕРЖАНИЕ

[Введение 6](#_Toc185820514)

[1 Архитектура программного обеспечения 8](#_Toc185820515)

[1.1 Общая структура системы 8](#_Toc185820516)

[1.2 Особенности клиент-серверной архитектуры 8](#_Toc185820517)

[1.3 Обоснование выбора клиент-серверного подхода 9](#_Toc185820518)

[1.4 Преимущества и ограничения выбранной архитектуры 9](#_Toc185820519)

[1.5 Масштабируемость и адаптивность системы 10](#_Toc185820520)

[1.6 Обеспечение безопасности данных 10](#_Toc185820521)

[1.7 Поддержка интеграции с внешними системами 11](#_Toc185820522)

[2 Платформа программного обеспечения 12](#_Toc185820523)

[2.1 Выбор серверной части: Node.js 12](#_Toc185820524)

[2.2 Выбор клиентской части: React.js 12](#_Toc185820525)

[2.3 Используемая СУБД: PostgreSQL 13](#_Toc185820526)

[2.4 Средства разработки и интеграции 14](#_Toc185820527)

[2.5 Обоснование выбора платформы 14](#_Toc185820528)

[3 Теоретическое обоснование разработки программного продукта 15](#_Toc185820529)

[3.1 Необходимость разработки системы 15](#_Toc185820530)

[3.2 Применяемые технологии программирования 15](#_Toc185820531)

[3.3 Связь архитектуры с функциональностью программы 17](#_Toc185820532)

[3.4 Преимущества использования современных технологий 17](#_Toc185820533)

[3.5 Разработка системы рекомендаций 18](#_Toc185820534)

[3.6 Интеграция с внешними сервисами 19](#_Toc185820535)

[3.7 Тестирование и оптимизация системы 20](#_Toc185820536)

[4 Проектирование функциональных возможностей программного обеспечения 21](#_Toc185820537)

[4.1 Каталог книг 21](#_Toc185820538)

[4.2 Корзина покупок 21](#_Toc185820539)

[4.3 Оформление заказа 22](#_Toc185820540)

[4.4 Административная панель 23](#_Toc185820541)

[5 Проектирование базы данных программного обеспечения 24](#_Toc185820542)

[5.1 Инфологическая структура базы данных 24](#_Toc185820543)

[5.2 Физическая реализация базы данных 27](#_Toc185820544)

[5.3 Оптимизация и поддержка базы данных 28](#_Toc185820545)

[5.4 Разработка функциональных компонентов базы данных 29](#_Toc185820546)

[Заключение 32](#_Toc185820547)

[Список литературных источников 34](#_Toc185820548)

[Приложение А (обязательное) Листинг программного кода 36](#_Toc185820549)

[Приложение Б (обязательное) Конечная схема базы данных 46](#_Toc185820550)

[Приложение В (обязательное) Ведомость документов 47](#_Toc185820551)

# **ВВЕДЕНИЕ**

Интернет-магазины стали неотъемлемой частью современной коммерции, предоставляя пользователям удобный способ приобретения товаров без необходимости посещения физических магазинов. Книжные магазины, как важная часть рынка, активно переходят в онлайн-среду, что связано с ростом цифровых технологий и изменением предпочтений покупателей. Однако многие существующие решения не учитывают особенностей работы с книгами, таких как необходимость расширенных фильтров по жанрам, авторам, издательствам и персонализированных рекомендаций. Некоторые платформы ограничены функционально или требуют значительных затрат на настройку и обслуживание.

Среди ключевых преимуществ разработки собственного интернет-магазина книг можно выделить возможность интеграции персонализированных рекомендаций для пользователей. На основе данных о предыдущих покупках, предпочтениях и поисковых запросах можно создавать систему рекомендаций, которая предложит книги, соответствующие интересам клиента. Это не только увеличит продажи, но и улучшит пользовательский опыт, помогая клиентам быстрее находить интересующую литературу. Такая функциональность особенно важна в условиях высокой конкуренции на рынке электронных книг, где удобство и персонализация играют решающую роль.

Создание специализированного программного средства для интернет-магазина книг позволяет учитывать уникальные потребности этого сегмента, предоставляя пользователям простой доступ к широкому ассортименту товаров, а владельцам бизнеса – гибкие инструменты управления.

Целью данного проекта является разработка программного средства для интернет-магазина книг, которое позволит автоматизировать процесс продаж, упростить управление товарами и заказами, а также обеспечить удобный интерфейс для пользователей. Для достижения поставленной цели необходимо выполнить несколько ключевых задач. Прежде всего, спроектировать базу данных, поддерживающую хранение информации о книгах, пользователях и заказах. Также требуется разработать серверную часть, которая будет обрабатывать пользовательские запросы и обеспечивать безопасное взаимодействие с данными. Кроме того, важным этапом является создание клиентского интерфейса, который обеспечит удобное взаимодействие пользователей с магазином.

Другая важная задача заключается в реализации административного функционала, включающего добавление, редактирование и удаление товаров, а также управление заказами и учетными записями пользователей. После завершения разработки необходимо провести тестирование системы для проверки её надежности, производительности и удобства.

На сегодняшний день существует множество платформ для создания интернет-магазинов, таких как Shopify, WooCommerce и OpenCart. Эти системы предлагают базовый функционал для электронной коммерции, однако они не всегда подходят для книжных магазинов из-за ряда ограничений. Прежде всего, им недостает узкоспециализированных функций, таких как рекомендации на основе жанров и авторов. Возможности настройки фильтров и категорий также зачастую ограничены. Более того, использование таких платформ может быть связано с высокими затратами на адаптацию и поддержку.

Создание собственного программного продукта позволяет преодолеть эти недостатки. Это решение полностью адаптировано под потребности книжного магазина, обеспечивает гибкость в управлении товарами и заказами, а также предоставляет современный пользовательский интерфейс, упрощающий взаимодействие. Разработка интернет-магазина книг с нуля дает высокую степень контроля над функционалом и архитектурой системы, что особенно важно для удовлетворения специфических требований пользователей и администраторов.

Дополнительно, создание уникального программного обеспечения для интернет-магазина позволяет учитывать региональные особенности и предпочтения. Например, можно реализовать поддержку локальных платежных систем, интеграцию с региональными службами доставки и настройку интерфейса под языковые предпочтения пользователей. Это сделает платформу более привлекательной для широкой аудитории, обеспечивая доступность товаров для клиентов из разных регионов. Учитывая растущую глобализацию, адаптация под локальные рынки становится важным аспектом успешного функционирования интернет-магазина.

Таким образом, актуальность разработки программного средства для интернет-магазина книг обусловлена потребностью в специализированных решениях, которые не только упрощают процесс покупки для клиентов, но и предоставляют владельцам бизнеса эффективные инструменты управления.

# **1 АРХИТЕКТУРА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ**

* 1. **Общая структура системы**

Архитектура интернет-магазина книг строится на клиент-серверной модели, что позволяет разделить задачи между компонентами системы, обеспечивая их независимость и упрощая разработку, масштабирование и поддержку. Система включает три основных компонента: клиентская часть, серверная часть и база данных.[1]

Клиентская часть реализована на основе библиотеки React.js и отвечает за визуализацию данных, предоставление интуитивно понятного интерфейса и взаимодействие пользователя с приложением. Это решение позволяет динамически обновлять данные на странице без необходимости её перезагрузки, что делает работу приложения более плавной и удобной.

Серверная часть построена на платформе Node.js. Она выполняет роль посредника между клиентом и базой данных, обрабатывая запросы, выполняя бизнес-логику и возвращая результаты в виде JSON-объектов. Такой подход обеспечивает гибкость и высокую производительность системы, а использование REST API делает её легко интегрируемой с другими платформами.

База данных построена на основе PostgreSQL, которая является документно-ориентированной системой управления базами данных. Её выбор обусловлен способностью работать с большими объемами данных, гибкой структурой, возможностью быстрого масштабирования и удобной интеграцией с Node.js.

## **1.2 Особенности клиент-серверной архитектуры**

Клиент-серверная архитектура имеет ряд особенностей, которые делают её наиболее подходящим решением для разработки интернет-магазина книг. Во-первых, такая архитектура разделяет систему на независимые слои, где каждый слой выполняет строго определенные функции. Это позволяет разрабатывать, тестировать и обновлять каждый компонент независимо от других.[2]

Во-вторых, клиент-серверная модель упрощает масштабирование системы. В случае увеличения нагрузки можно выделить дополнительные серверы для обработки запросов или распределить данные между несколькими базами данных. Кроме того, клиент-серверная архитектура обеспечивает гибкость при подключении новых клиентов, таких как мобильные приложения или сторонние сервисы.

Наконец, важной характеристикой архитектуры является безопасность. Сервер отвечает за обработку конфиденциальных данных, таких как информация о клиентах и заказах, что позволяет минимизировать риски, связанные с их утечкой.

## **1.3 Обоснование выбора клиент-серверного подхода**

Выбор клиент-серверной архитектуры обусловлен её соответствием требованиям к функциональности и надежности системы. Такая архитектура позволяет чётко разделить логику на уровни представления, обработки и хранения данных, что облегчает разработку и поддержку приложения.

Клиент-серверная модель предоставляет возможность быстрого взаимодействия с сервером через REST API, что обеспечивает универсальность и лёгкость интеграции с другими приложениями. Кроме того, разделение системы на независимые компоненты делает её более устойчивой к изменениям: добавление нового функционала на клиентской или серверной части не требует кардинальной перестройки системы.

Среди других преимуществ клиент-серверной архитектуры стоит отметить возможность работы с различными устройствами и платформами, что особенно актуально для интернет-магазина, где пользователи могут работать как с настольных компьютеров, так и с мобильных устройств.

## **1.4 Преимущества и ограничения выбранной архитектуры**

Клиент-серверная архитектура обладает значительными преимуществами. Прежде всего, это высокая гибкость и модульность, которые упрощают процесс разработки, тестирования и обновления компонентов системы. Кроме того, использование REST API позволяет интегрировать систему с внешними сервисами, такими как платёжные шлюзы, системы доставки или аналитические платформы.

Ещё одно преимущество — лёгкость масштабирования. По мере роста числа пользователей или объёмов данных систему можно адаптировать, добавляя серверные мощности или распределяя нагрузку между базами данных.

Однако архитектура имеет и ограничения. Во-первых, производительность системы может зависеть от качества сетевого соединения, так как все данные передаются между клиентом и сервером. Во-вторых, для обеспечения безопасности передачи данных требуется дополнительная защита, такая как использование SSL-сертификатов или шифрование.

Тем не менее, преимущества выбранной архитектуры значительно превышают её недостатки, что делает клиент-серверную модель оптимальным решением для разработки интернет-магазина книг.

## **1.5 Масштабируемость и адаптивность системы**

Одним из ключевых требований к архитектуре программного обеспечения для интернет-магазина книг является её способность к масштабированию. Масштабируемость достигается за счёт разделения системы на независимые компоненты: клиентская часть, сервер и база данных могут быть адаптированы или заменены без необходимости полной перестройки архитектуры.

Для горизонтального масштабирования возможно добавление серверов в кластер, что увеличивает производительность при росте числа пользователей. Вертикальное масштабирование реализуется путём увеличения вычислительных ресурсов серверов или базы данных. Например, PostgreSQL позволяет распределять данные по нескольким узлам, обеспечивая доступность и быстродействие при увеличении объёмов данных.

Адаптивность системы обеспечивает её работу на различных устройствах. Клиентская часть, разработанная на React.js, поддерживает адаптивный дизайн, что позволяет пользователям комфортно взаимодействовать с приложением как на компьютерах, так и на мобильных устройствах. Это важно для интернет-магазина, где значительная доля покупок совершается с мобильных платформ.

## **1.6 Обеспечение безопасности данных**

Важной задачей при проектировании архитектуры является защита пользовательских данных. Интернет-магазины часто работают с конфиденциальной информацией, такой как личные данные клиентов, реквизиты для доставки и информация о платежах.[3]

Для обеспечения безопасности система использует следующие подходы:

Шифрование данных. Передача данных между клиентом и сервером осуществляется с использованием HTTPS, что предотвращает их перехват злоумышленниками.

Аутентификация и авторизация. Для защиты учётных записей пользователей реализуется проверка личности с использованием токенов доступа (JWT) или сессий. Это предотвращает несанкционированный доступ к личным кабинетам.

Регулярное обновление компонентов. Используемые библиотеки и платформы регулярно обновляются для защиты от известных уязвимостей.

Резервное копирование. Регулярное создание резервных копий базы данных позволяет минимизировать потери информации в случае сбоев или кибератак.

Эти меры обеспечивают надёжную защиту системы от потенциальных угроз и повышают доверие пользователей к интернет-магазину.[4]

## **1.7 Поддержка интеграции с внешними системами**

Современные интернет-магазины редко функционируют изолированно, часто интегрируясь с внешними сервисами, такими как платёжные шлюзы, службы доставки и CRM-системы. Для этого архитектура интернет-магазина книг предусматривает использование REST API, который обеспечивает стандартизированный способ взаимодействия с внешними приложениями.

Интеграция с платёжными системами позволяет пользователям оплачивать покупки различными способами, такими как банковские карты, электронные кошельки или мобильные платежи. Подключение служб доставки упрощает процесс логистики, автоматически рассчитывая стоимость доставки и отслеживая статусы заказов.

Взаимодействие с CRM-системами помогает владельцам магазина анализировать поведение клиентов, настраивать маркетинговые кампании и управлять программами лояльности. Таким образом, поддержка интеграции расширяет функциональность интернет-магазина и делает его более удобным для пользователей и администраторов.

1. **ПЛАТФОРМА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ**
   1. **Выбор серверной части: Node.js**

Серверная часть системы построена с использованием платформы Node.js. Этот выбор обусловлен её особенностями, которые идеально подходят для реализации интернет-магазина. Node.js представляет собой среду выполнения JavaScript, которая работает на стороне сервера. Она основана на движке V8, который обеспечивает высокую производительность.

Одним из ключевых преимуществ Node.js является её асинхронная и событийно-ориентированная природа. Это позволяет серверу обрабатывать множество запросов одновременно, не блокируя выполнение операций. Такая модель особенно эффективна для интернет-магазинов, где система должна обрабатывать запросы на поиск товаров, управление корзиной, оформление заказов и другие действия, выполняемые большим числом пользователей.

Дополнительным преимуществом Node.js является огромное количество готовых модулей и библиотек, доступных через менеджер пакетов npm. Это ускоряет процесс разработки и позволяет использовать проверенные решения для аутентификации, работы с базами данных, интеграции с платёжными системами и других задач.[5]

Node.js также легко интегрируется с PostgreSQL, что упрощает создание полноценной связки серверной части и базы данных. Эта платформа поддерживает REST API, обеспечивая надёжное взаимодействие между клиентской и серверной частями.

## **2.2 Выбор клиентской части: React.js**

Клиентская часть разработана с использованием React.js — популярной библиотеки для создания пользовательских интерфейсов. React.js позволяет создавать динамические, адаптивные и отзывчивые интерфейсы, что особенно важно для интернет-магазина, где пользовательский опыт напрямую влияет на конверсию.

React.js использует компонентный подход, позволяя разделить интерфейс на независимые элементы. Это упрощает разработку, тестирование и поддержку системы. Каждый компонент может быть повторно использован в различных частях приложения, что сокращает время на создание новых функций.

Особенностью React.js является использование виртуального DOM, который ускоряет обновление интерфейса. Вместо прямого изменения структуры страницы React.js сравнивает старую и новую версии виртуального DOM и обновляет только изменённые элементы. Это значительно повышает производительность, особенно на устройствах с низкой мощностью.[6]

Библиотека React.js также обладает большим сообществом разработчиков, что позволяет находить готовые решения для реализации сложных интерфейсных задач. Например, существуют плагины и библиотеки для реализации маршрутизации, управления состоянием и интеграции с REST API.

## **2.3 Используемая СУБД: PostgreSQL**

Для хранения данных в интернет-магазине книг используется PostgreSQL, одна из самых мощных и популярных реляционных систем управления базами данных. PostgreSQL идеально подходит для приложений, где требуется работа с жёстко структурированными данными и сложными запросами.

PostgreSQL обеспечивает высокий уровень соответствия стандарту SQL, что делает её подходящей для сложных аналитических задач. Она поддерживает транзакции с соблюдением ACID, что гарантирует целостность данных даже в случае сбоев.

Важным преимуществом PostgreSQL является её расширяемость. Система поддерживает пользовательские типы данных, индексы и функции, что позволяет адаптировать базу данных под специфические требования интернет-магазина.

Дополнительные возможности PostgreSQL включают:

Индексация. Поддержка B-деревьев, хэш-индексов и других структур для ускорения поиска.

Масштабируемость. Возможность работы в режиме репликации для повышения производительности и отказоустойчивости.

Работа с JSON. Поддержка вложенных структур данных, что полезно для хранения метаинформации о товарах.

PostgreSQL идеально интегрируется с Node.js с использованием библиотек, таких как pg, которые обеспечивают удобный интерфейс для взаимодействия с базой данных.[7]

## **2.4 Средства разработки и интеграции**

Разработка интернет-магазина велась с использованием современных инструментов, которые обеспечили удобство, гибкость и надёжность процесса:

Visual Studio Code. Эта интегрированная среда разработки предоставляет удобный интерфейс, поддержку большого количества языков и инструментов отладки. Её расширяемость позволяет добавлять плагины для работы с React.js, Node.js и PostgreSQL.

Postman. Использовался для тестирования REST API. Этот инструмент позволяет создавать запросы к серверу, тестировать их корректность и анализировать ответы.

Git. Система контроля версий обеспечила удобное управление исходным кодом и совместную работу над проектом. Репозиторий позволял отслеживать изменения, возвращаться к предыдущим версиям и быстро исправлять ошибки.

Библиотеки и фреймворки. В проекте использовались такие популярные библиотеки, как Axios для взаимодействия с REST API, Redux для управления состоянием и pg для работы с PostgreSQL.

## **2.5 Обоснование выбора платформы**

Выбор перечисленных платформ обусловлен их производительностью, удобством интеграции и активным сообществом разработчиков. Node.js обеспечивает высокую скорость работы сервера и возможность обработки большого количества запросов. React.js предоставляет мощные инструменты для создания адаптивных интерфейсов, которые легко расширяются и модернизируются. PostgreSQL, в свою очередь, обеспечивает надёжное хранение данных, поддержку сложных аналитических запросов и лёгкость масштабирования.

Совокупность этих технологий позволяет создать интернет-магазин, который отвечает современным требованиям производительности, удобства и безопасности. Платформы хорошо взаимодействуют между собой, обеспечивая надёжную работу системы и её адаптивность к будущим изменениям.[8]

1. **ТЕОРЕТИЧЕСКОЕ ОБОСНОВАНИЕ РАЗРАБОТКИ ПРОГРАММНОГО ПРОДУКТА**
   1. **Необходимость разработки системы**

Современные интернет-магазины становятся неотъемлемой частью розничной торговли, а потребность в их автоматизации и оптимизации возрастает с каждым годом. Особенность интернет-магазина книг заключается в том, что он должен обеспечивать широкий ассортимент товаров, удобный поиск и фильтрацию, а также бесперебойное управление заказами и запасами.

Существующие решения для создания интернет-магазинов, такие как Shopify, WooCommerce и другие платформы, предлагают базовый функционал для электронной коммерции. Однако большинство из них не предназначены для специфики книжной торговли. В частности, ограниченные возможности для работы с большими каталогами книг, а также необходимость интеграции с различными внешними сервисами, такими как платёжные системы, службы доставки и CRM-системы, делают стандартные платформы неудобными для этого сегмента.

Разработка собственного программного продукта позволяет создать решение, которое будет идеально подходить для нужд конкретного интернет-магазина книг. Это обеспечит гибкость в работе с ассортиментом, индивидуальный подход к каждому пользователю, а также упрощённое управление всеми процессами, от добавления товара до доставки заказа.

Кроме того, создание собственного решения предоставляет возможность интеграции с внешними сервисами и расширения функционала с учётом специфических потребностей бизнеса, таких как персонализированные рекомендации, создание лояльности через бонусные программы и другие инструменты для увеличения продаж.

## **3.2 Применяемые технологии программирования**

Для разработки интернет-магазина книг использовались современные технологии, которые обеспечивают высокую производительность, гибкость и удобство работы. В данном проекте были использованы следующие ключевые технологии:

JavaScript — это основной язык программирования как для клиентской, так и для серверной части системы. Он позволяет создавать динамичные веб-приложения, в которых данные обновляются в реальном времени без необходимости перезагрузки страницы. Для серверной части был выбран Node.js, который обеспечивает асинхронную обработку запросов и высокую производительность. Это решение идеально подходит для приложений с большим количеством пользователей, так как оно эффективно обрабатывает большое количество одновременных запросов, не блокируя выполнение операций.

React.js — это библиотека JavaScript для создания пользовательских интерфейсов, которая позволяет легко обновлять и рендерить компоненты при изменении данных. React.js использует виртуальный DOM, что значительно ускоряет работу интерфейса и минимизирует нагрузку на сервер. Использование React.js позволило создать адаптивный и интуитивно понятный интерфейс, который удобно использовать как на десктопных, так и на мобильных устройствах.

В качестве системы управления базой данных был выбран PostgreSQL — реляционная СУБД, которая обеспечивает высокую производительность, поддержку сложных запросов и масштабируемость. PostgreSQL идеально подходит для интернет-магазинов, так как она может эффективно обрабатывать большое количество данных, таких как информация о книгах, заказах и пользователях. Она также поддерживает транзакции с соблюдением ACID, что гарантирует надёжность и целостность данных.

Вся коммуникация между клиентом и сервером осуществляется через REST API, что позволяет упростить интеграцию с другими системами и платформами, а также обеспечить гибкость при обмене данными между клиентом и сервером. REST API является универсальным подходом, который упрощает работу с различными клиентами и позволяет интегрировать систему с внешними сервисами, такими как платёжные шлюзы и службы доставки.

Для управления состоянием приложения использовался Redux — библиотека для JavaScript, которая позволяет централизованно хранить состояние приложения и передавать его между компонентами. Это особенно полезно для веб-приложений, таких как интернет-магазины, где состояние приложения постоянно меняется в зависимости от действий пользователя (например, добавление товара в корзину, изменение количества товара и т. д.).

* 1. **Связь архитектуры с функциональностью программы**

Выбор архитектуры, основанной на клиент-серверной модели с использованием React.js для клиентской части и Node.js для серверной, позволяет легко разделять логику приложения и управлять данными. Это разделение даёт возможность разрабатывать, тестировать и поддерживать систему по частям, что упрощает процесс разработки и масштабирования.

Архитектура обеспечивает поддержку всех ключевых функций интернет-магазина:

Поиск и фильтрация товаров. Реализация этой функции достигается через серверную часть, которая обрабатывает запросы на поиск и фильтрацию, и клиентскую часть, которая отображает результаты поиска в реальном времени.

Корзина покупок. Все изменения в корзине, такие как добавление или удаление товара, автоматически обновляют состояние клиента через Redux, а серверная часть обрабатывает эти запросы и сохраняет их в базе данных.

Оформление заказа. При оформлении заказа информация о пользователе и списке товаров передаётся на сервер, который затем обрабатывает заказ и сохраняет его в базе данных.

Административный интерфейс. Простой и удобный интерфейс для администрирования товаров и заказов обеспечивает быструю и удобную работу с магазином.

Кроме того, архитектура позволяет легко интегрировать дополнительные функции, такие как персонализированные рекомендации, создание лояльности через бонусные программы и возможность интеграции с платёжными системами и службами доставки.

## **3.4 Преимущества использования современных технологий**

Использование современных технологий, таких как Node.js, React.js и PostgreSQL, даёт несколько ключевых преимуществ:

Высокая производительность. Node.js позволяет обрабатывать большое количество одновременных запросов, что критически важно для интернет-магазинов, где количество пользователей и операций может значительно возрасти.

Гибкость и масштабируемость. Архитектура, построенная на REST API, позволяет легко расширять функционал системы и интегрировать её с внешними сервисами. PostgreSQL обеспечивает удобное масштабирование базы данных по мере роста объёмов данных.

Удобство разработки. Использование React.js и Redux позволяет ускорить процесс разработки и сделать интерфейс более отзывчивым и интерактивным. Эти технологии дают возможность легко управлять состоянием приложения и обновлять компоненты интерфейса без перезагрузки страницы.

Безопасность и надёжность. PostgreSQL предоставляет высокий уровень безопасности, поддерживая транзакции и индексацию, что критически важно для работы с конфиденциальными данными, такими как информация о пользователях и их заказах.

Совокупность этих технологий позволяет создать интернет-магазин, который не только отвечает современным требованиям производительности и удобства, но и легко масштабируется и адаптируется к изменениям в будущем.

## **3.5 Разработка системы рекомендаций**

Одним из ключевых элементов современных интернет-магазинов является система рекомендаций, которая помогает пользователям находить интересующие товары, основываясь на их предпочтениях и предыдущем поведении. В случае с интернет-магазином книг, такой функционал становится особенно важным, поскольку позволяет предложить пользователям книги, которые соответствуют их интересам и потребностям.

В рамках разработки данной системы рекомендаций было решено использовать несколько методов, основанных на алгоритмах машинного обучения и анализа данных. Одним из основных подходов является коллаборативная фильтрация, которая позволяет делать рекомендации на основе предпочтений других пользователей с похожими интересами. Другим методом является контентная фильтрация, которая использует информацию о содержимом книг (жанры, авторы, ключевые слова и т.д.) для предложений на основе предпочтений пользователя.

В зависимости от поведения пользователя в магазине (например, книги, которые он просматривает или добавляет в корзину), система будет генерировать рекомендации. Алгоритм будет учитывать популярность книг, отзывы, схожесть с другими произведениями и рейтинг пользователей. Это значительно улучшит пользовательский опыт, обеспечивая персонализированные предложения.

Кроме того, система рекомендаций будет гибко настраиваться, позволяя добавлять новые методы анализа данных, а также проводить экспериментальные тестирования для оптимизации качества рекомендаций. Система будет интегрирована с основным каталогом товаров и алгоритмами поиска, чтобы эффективно взаимодействовать с другими функциональными компонентами магазина.

## **3.6 Интеграция с внешними сервисами**

Для эффективного функционирования интернет-магазина, особенно в сфере электронной коммерции, необходима интеграция с различными внешними сервисами и платформами. В данном проекте особое внимание было уделено интеграции с платёжными системами и службами доставки, которые играют ключевую роль в процессах оформления заказа и доставки товаров.

Интеграция с платёжными системами:

Для обеспечения безопасности и удобства оплаты был выбран интеграционный модуль с платёжными шлюзами, такими как Stripe и PayPal. Эти платёжные системы поддерживают различные способы оплаты, включая кредитные и дебетовые карты, электронные кошельки и даже криптовалюту.

Кроме того, поддержка многовалютности и многоязычности позволяет расширить аудиторию интернет-магазина, предлагая пользователям возможность выбирать наиболее удобную для них валюту и язык интерфейса. Такой подход значительно повышает привлекательность магазина для международной аудитории.

Интеграция с курьерскими службами:

Для автоматизации логистики и упрощения процесса доставки был добавлен модуль для интеграции с курьерскими службами, такими как DHL, FedEx и СДЭК. Интеграция с этими службами позволяет автоматизировать расчёт стоимости доставки и отслеживание посылок в реальном времени.

Кроме того, система автоматически рассчитывает предполагаемое время доставки в зависимости от региона, веса заказа и выбранной службы доставки. Это позволяет клиентам интернет-магазина получать точную информацию о статусе их заказа и времени его получения.

Синхронизация с CRM-системами и аналитикой:

Интеграция с CRM-системами и аналитическими платформами, такими как HubSpot и Google Analytics, позволяет собрать полную картину о поведении клиентов на сайте, их предпочтениях и покупательской активности.

С помощью таких инструментов возможно реализовать сегментацию пользователей, настройку персонализированных маркетинговых акций и предложения, а также оптимизацию рекламных кампаний, что способствует увеличению конверсии и улучшению взаимодействия с клиентами.[9]

## **3.7 Тестирование и оптимизация системы**

После разработки всех компонентов системы важно провести тестирование, чтобы убедиться в её надёжности, функциональности и производительности. Тестирование включает несколько этапов, каждый из которых имеет свои особенности и цели.

Основная цель функционального тестирования — проверить корректность работы всех функциональных компонентов системы, таких как поиск и фильтрация товаров, оформление заказа, управление корзиной, персонализированные рекомендации и другие функции. Для этого используется набор тестов, которые моделируют действия пользователей, проверяя правильность обработки запросов сервером и корректность отображения данных на клиентской части.

Нагрузочное тестирование важно для определения того, как система будет работать при увеличении числа пользователей. Оно позволяет выявить узкие места в производительности системы и оптимизировать её для работы с большим количеством одновременных пользователей. В ходе тестирования проверяются такие аспекты, как скорость отклика сервера, время загрузки страниц и производительность базы данных.

Тестирование безопасности направлено на выявление уязвимостей в системе, которые могут быть использованы злоумышленниками для получения несанкционированного доступа к данным. Важными аспектами являются проверка аутентификации и авторизации пользователей, защита данных при передаче через сеть, а также обеспечение безопасности базы данных. В процессе тестирования также проверяется устойчивость системы к различным типам атак, таким как SQL-инъекции, XSS и CSRF.[10]

# **4 ПРОЕКТИРОВАНИЕ ФУНКЦИОНАЛЬНЫХ ВОЗМОЖНОСТЕЙ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ**

## **4.1 Каталог книг**

Каталог книг является центральным элементом любого интернет-магазина книг. Это раздел, в котором пользователи могут просматривать, искать и фильтровать товары. Он должен быть интуитивно понятным, с удобными функциями для поиска по ключевым словам, авторам, жанрам и рейтингу.

Основные задачи при проектировании каталога включают:

Отображение товаров. Каждая книга в каталоге должна быть представлена с полным описанием, изображением обложки, автором, жанром, ценой и наличием на складе. Для удобства пользователей важен правильный подбор информации и её форматирование.

Поиск и фильтрация. Пользователь должен иметь возможность искать книги по различным критериям, таким как автор, жанр, цена, рейтинг и другие параметры. Для этого будет реализована система фильтров, которая будет динамически обновлять результаты поиска в зависимости от выбранных критериев.

Сортировка. Возможность сортировки по цене, популярности, новизне и другим параметрам. Это улучшит пользовательский опыт и ускорит поиск нужной книги.[11]

Персонализированные рекомендации. Включение системы рекомендаций, которая будет предлагать пользователю книги на основе его истории покупок или предпочтений. Эти рекомендации будут выводиться в виде отдельного блока на странице каталога.

Функционал каталога обеспечит удобство выбора книг, а также будет способствовать улучшению взаимодействия с пользователем. Важно, чтобы интерфейс был понятным и не перегружал пользователя лишней информацией.

## **4.2 Корзина покупок**

Корзина покупок является важной частью функционала интернет-магазина, так как она позволяет пользователю управлять выбранными товарами до оформления заказа. Проектирование корзины должно обеспечить её простоту и удобство, при этом поддерживая все необходимые функциональные возможности.

Основные задачи при проектировании корзины:

Добавление товаров. При добавлении товара в корзину пользователь должен видеть актуальное количество выбранных единиц и стоимость. Товар будет добавляться по кнопке "Добавить в корзину", и при этом будет отображаться уведомление об успешном добавлении.

Изменение количества. В корзине пользователь должен иметь возможность изменять количество товаров, а также удалять их из корзины. При изменении количества будет автоматически пересчитываться итоговая сумма заказа.

Просмотр итоговой суммы. Внизу страницы корзины должна отображаться итоговая стоимость товаров, включая возможные скидки и стоимость доставки. Это позволит пользователю заранее оценить, сколько он потратит на покупку.

Сохранение данных. Корзина должна сохранять свои данные между сессиями пользователя, чтобы он мог вернуться к ним позже, не теряя информации о товарах.

Промо-коды и скидки. В корзине также будет реализована возможность ввода промо-кодов, которые позволяют получить скидку на товары. Это также влияет на итоговую сумму заказа.

Для реализации корзины будет использован Redux для управления состоянием, что позволит централизованно хранить данные о корзине и эффективно обновлять её в зависимости от действий пользователя.[12]

## **4.3 Оформление заказа**

Оформление заказа — это ключевая часть функционала интернет-магазина, которая включает в себя ввод данных о пользователе, выборе способа доставки и оплаты. Процесс оформления должен быть простым, понятным и безопасным для пользователей.

Основные задачи при проектировании оформления заказа:

Форма для ввода данных. Пользователь должен предоставить свою контактную информацию, включая имя, адрес доставки, номер телефона и e-mail. Также можно предложить регистрацию или авторизацию на сайте для более быстрой обработки заказов в будущем.

Выбор способа доставки. На этом этапе пользователь может выбрать удобный способ доставки. В зависимости от региона и товаров, система будет автоматически предлагать доступные варианты с указанием стоимости и времени доставки.

Оплата. Оформление заказа также включает выбор способа оплаты. В интернет-магазине будут поддерживаться различные способы оплаты, такие как кредитные карты, электронные кошельки и платёжные системы, например, PayPal или Stripe.[13]

Подтверждение заказа. После ввода всех данных, система покажет итоговую информацию о заказе: товары, способ доставки и сумма к оплате. Также пользователь получит уведомление о подтверждении заказа и ожидаемом времени доставки.

Данный этап важен для повышения конверсии, и поэтому проектирование должно включать несколько этапов проверки введённых данных, чтобы избежать ошибок и повысить удовлетворённость клиентов.

## **4.4 Административная панель**

Административная панель предназначена для управления товарами, заказами, клиентами и всеми другими аспектами функционирования магазина. Её дизайн и функциональность должны быть интуитивно понятными, чтобы администраторы могли эффективно выполнять свою работу.

Основные задачи административной панели:

Управление товарами. Администраторы должны иметь возможность добавлять новые книги, редактировать их описание, менять цену, а также удалять товары из каталога. Для удобства будет предусмотрена загрузка изображений и указание меток для фильтрации (например, жанры, популярность и т.д.).[14]

Управление заказами. Панель должна позволять отслеживать все заказы, изменять их статус (например, «обрабатывается», «отправлен» или «доставлен»), а также осуществлять возвраты или отмену заказов.

Отчёты и аналитика. Для администраторов будет предоставлена возможность просматривать отчёты о продажах, анализировать поведение пользователей, оценивать эффективность маркетинговых кампаний и оптимизировать ассортимент товара.

Управление пользователями. Администраторы смогут управлять профилями пользователей, а также отслеживать их активность, например, частоту покупок или рейтинг оставленных отзывов.

Интерфейс административной панели будет разработан таким образом, чтобы обеспечить быстрое выполнение всех операций, не перегружая систему лишними данными.

# **5 ПРОЕКТИРОВАНИЕ БАЗЫ ДАННЫХ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ**

База данных для интернет-магазина книг играет ключевую роль в поддержании функциональности системы. Она организует хранение данных о книгах, пользователях, заказах, а также управляет связями между различными сущностями. В проектировании базы данных необходимо учесть все аспекты, от хранения информации о товарах до управления пользовательскими аккаунтами.[15]

## **5.1 Инфологическая структура базы данных**

При разработке приложения крайне важно создать четкую и логически последовательную инфологическую структуру базы данных. Эффективное управление данными напрямую влияет на производительность системы и качество предоставляемых услуг. Чтобы достичь этой цели, необходимо учитывать различные уровни моделирования данных. [16]

Цель инфологического моделирования – обеспечить наиболее естественные для человека способы сбора и представления информации, которую предполагается хранить в создаваемой базе данных. Поэтому инфологическая модель данных строится по аналогии с естественным языком.

Описание концептуальной, логической и физической моделей данных помогает детализировать и систематизировать информацию о сущностях, их атрибутах и взаимосвязях.

Концептуальная модель данных представляет собой абстрактное представление системы, которое фокусируется на ключевых сущностях и их взаимосвязях без детального описания атрибутов. Она помогает определить основные объекты, важные для бизнес-логики приложения. Эта модель служит основой для дальнейшего проектирования логической и физической моделей данных, обеспечивая четкое понимание структуры системы. Концептуальная модель часто используется на начальных этапах разработки, позволяя участникам проекта согласовать требования и ожидания. На рисунке 5.1 представлена концептуальная модель данных, иллюстрирующая взаимосвязи между основными сущностями приложения.[17]

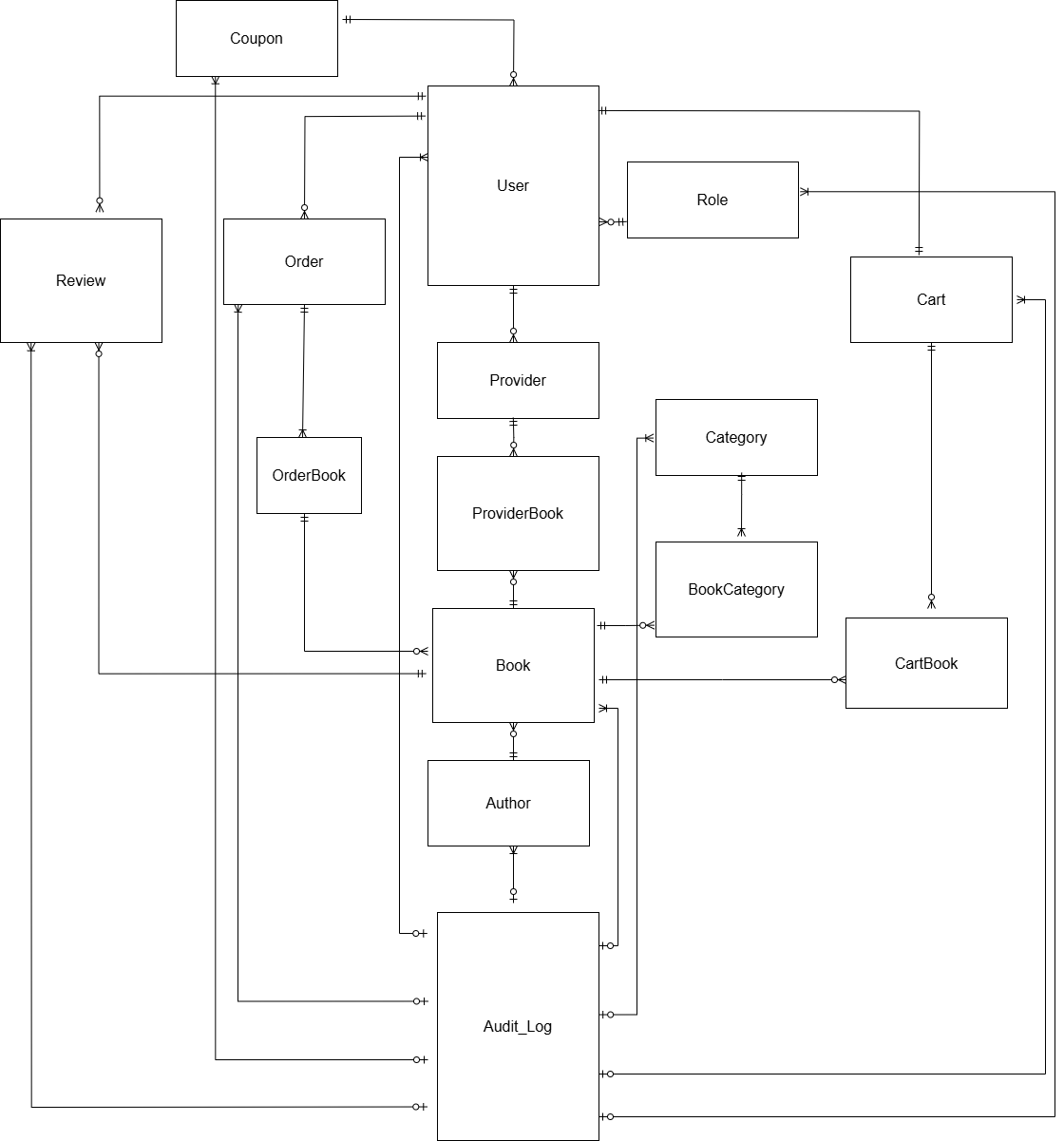


Рисунок 5.1 – Концептуальная модель данных

Логическая модель данных является более детализированным представлением по сравнению с концептуальной моделью, включающим все необходимые сущности и их атрибуты. Она описывает структуру данных и взаимосвязи между сущностями, определяя уникальные и внешние ключи, а также ограничения, такие как уникальность и диапазоны значений. Логическая модель служит основой для создания физической модели, обеспечивая понимание бизнес-правил и требований к данным. На рисунке 5.2 представлена логическая модель данных:

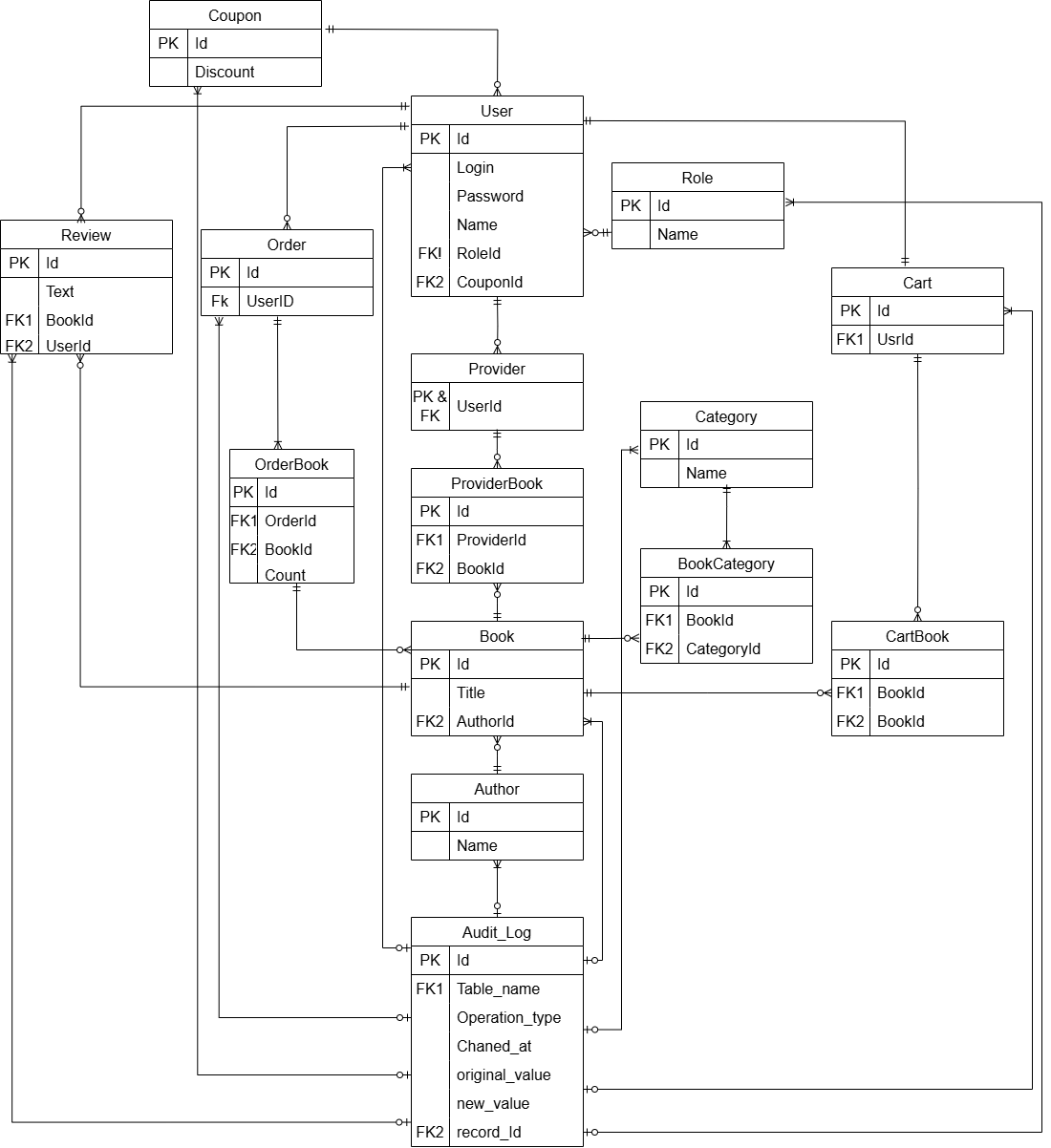


Рисунок 5.2 – Логическая модель данных

Физическая модель данных представляет собой конкретизацию логической модели, описывающую структуру таблиц, столбцов и их типы данных для реализации в базе данных. Она также учитывает оптимизацию производительности, включая стратегии индексации, чтобы ускорить доступ к данным и повысить эффективность запросов. Кроме того, физическая модель описывает способы хранения данных, такие как формат хранения и способы резервного копирования.[18]

## **5.2 Физическая реализация базы данных**

Для реализации физической модели будет использована реляционная СУБД PostgreSQL. Важной частью будет создание индексов на наиболее часто используемые поля для ускорения поиска, таких как название книги, автор и категория. На рисунке 3.4 представлена физическая модель данных:

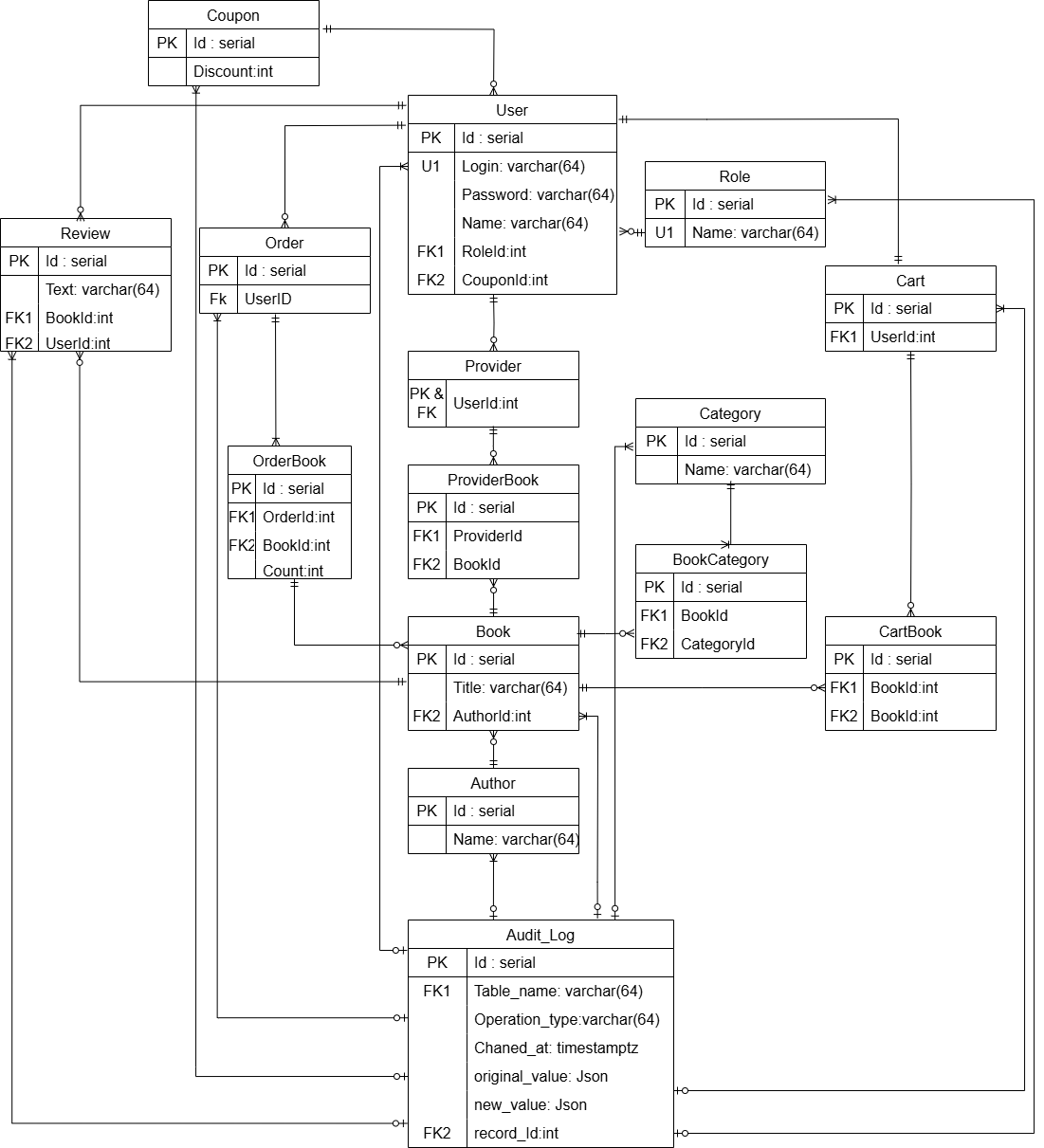


Рисунок 5.3 – Физическая модель данных

Таким образом в приложении используются следующие сущности:

1. User. Содержит информацию о пользователях системы, включая логин, пароль, имя и связь с ролями (например, клиент или администратор). Также может быть привязан купон для скидок.
2. Role. Определяет роли пользователей, такие как клиент, администратор или поставщик.
3. Coupon. Содержит данные о купонах, включая размер скидки.
4. Book. Хранит информацию о книгах, включая название и связь с автором.
5. Author. Описывает авторов книг, включая их имя.
6. Category. Описывает категории книг, такие как жанры или типы литературы.
7. Cart. Представляет корзину пользователя, содержащую добавленные книги.
8. Order. Содержит информацию о заказах, включая пользователя, дату заказа и список книг.
9. Review. Содержит отзывы пользователей на книги, включая текст отзыва и связь с пользователем и книгой.
10. Provider. Определяет поставщиков, которые работают с системой, и связывает их с пользователем.
11. Audit\_Log. Содержит информацию о действиях пользователей.

Внешние ключи обеспечат правильность связей между таблицами, обеспечивая целостность данных в системе.

## **5.3 Оптимизация и поддержка базы данных**

Оптимизация базы данных будет включать:

Кэширование. Использование кэширования для часто запрашиваемых данных, таких как популярные книги или категории, что позволит снизить нагрузку на базу данных.

Резервное копирование. Регулярное создание резервных копий базы данных обеспечит её восстановление в случае сбоя.

Мониторинг. Будет использоваться мониторинг производительности, чтобы оперативно выявлять узкие места и выполнять оптимизацию запросов.

Таким образом, проектирование базы данных для интернет-магазина книг обеспечит её стабильную работу, эффективное управление данными и высокую производительность, что критически важно для функционирования системы с большим количеством пользователей и товаров.[19]

## **Разработка функциональных компонентов базы данных**

**Индексы:**

Индексы представляют собой специальные структуры данных, которые улучшают скорость выполнения операций поиска в базе данных. Их использование играет ключевую роль в повышении производительности приложения, особенно когда речь идет о больших объемах данных. Индексы позволяют значительно сократить время, необходимое для доступа к данным, организуя их в упорядоченные структуры, что делает процесс поиска более эффективным.

Индексы можно рассматривать как набор ссылок на данные, упорядоченных по определенному столбцу в таблице. Столбцы, для которых созданы индексы, называются индексированными. Эти структуры хранятся в таблице и служат полезным внутренним механизмом системы управления базами данных, позволяя оптимизировать доступ к данным.

При определении ограничений столбцов, таких как первичный ключ или уникальные значения, автоматически создаются уникальные индексы.

При установлении ограничения первичного ключа на один или несколько столбцов в PostgreSQL автоматически создается уникальный индекс. PostgreSQL по умолчанию создает уникальный некластеризованный индекс для первичного ключа. Однако при создании первичного ключа также существует возможность задать параметры на создание кластеризованного индекса, что позволяет физически упорядочить строки в таблице в соответствии с порядком значений этого индекса. Если в таблице уже существует кластеризованный индекс, то будет создан уникальный некластеризованный индекс для первичного ключа.

Для ограничения уникальности значений PostgreSQL также создает уникальный некластеризованный индекс по умолчанию. Тем не менее, разработчик может явно указать создание уникального кластеризованного индекса, если на таблице еще не был создан кластеризованный индекс. Это позволяет оптимизировать доступ к данным, обеспечивая их уникальность и ускоряя операции поиска.

В дополнение к стандартным индексам было принято решение добавить дополнительные индексы для повышения производительности системы и оптимизации запросов. Они включают следующие индексы:

**Триггеры:**

Триггер – это специальный тип хранимой процедуры, которая автоматически срабатывает при выполнении определённых операций изменения данных в заданной таблице, таких как добавление, удаление или редактирование строк. Триггеры не вызываются пользователем напрямую; их выполнение инициируется автоматически в ответ на изменения данных. Они могут использоваться как до выполнения операций (например, для валидации данных), так и после. Кроме того, триггеры могут выполнять функции, заменяющие стандартные операции над данными.

В рамках данного проекта были реализованы триггеры для автоматического создания и обновления профилей пользователей. При добавлении или обновлении записи пользователя в таблице User триггер проверяет роль пользователя и создает соответствующий профиль в таблице Profile. Это позволяет обеспечить актуальность и целостность данных о пользователях в базе данных.

Пример триггера для обработки операций вставки и обновления пользователей:

CREATE OR REPLACE TRIGGER check\_admin\_will\_remain\_after\_delete

BEFORE DELETE ON users

FOR EACH ROW

WHEN (OLD.role\_id = 2)

EXECUTE FUNCTION check\_admin\_will\_remain\_after\_delete();

**Хранимые процедуры:**

Хранимая процедура – это предопределенный набор SQL-инструкций, который сохраняется в базе данных как именованный объект и выполняется в виде единого блока кода. Эти процедуры могут принимать входные параметры и возвращать результаты, что делает их универсальным инструментом для выполнения сложных операций в базе данных. Когда хранимая процедура создается, сервер компилирует ее и сохраняет в кэше, позволяя нескольким пользователям использовать скомпилированный код, что значительно повышает производительность.

Хранимые процедуры похожи на функции в языках программирования высокого уровня. Они могут иметь входные и выходные параметры, а также локальные переменные. Внутри процедур возможно выполнение арифметических операций, работа со строками и управление потоками данных, что позволяет реализовать сложную логику обработки данных.

В рамках данного проекта была создана процедура create\_or\_update\_user\_profile, предназначенная для автоматического создания или обновления профиля пользователя в таблице Profile. Эта процедура выполняется в ответ на операции вставки или обновления записи в таблице User, что обеспечивает автоматизацию управления профилями пользователей и гарантирует актуальность информации.

Код используемой хранимой процедуры:

CREATE OR REPLACE FUNCTION select\_books\_by\_user\_cart(\_user\_id int)

RETURNS TABLE(id INTEGER, image VARCHAR(64),price REAL,title VARCHAR(64),count INTEGER) AS $$

BEGIN

RETURN QUERY SELECT

b.id,

b.image,

b.price,

b.title,

cb.count

FROM carts\_books cb

JOIN carts c ON cb.cart\_id = c.id

JOIN users u ON c.user\_id = u.id AND u.id = \_user\_id

JOIN books b ON cb.book\_id = b.id;

END;

$$ LANGUAGE plpgsql;

CREATE OR REPLACE FUNCTION check\_admin\_will\_remain(u RECORD)

RETURNS RECORD AS $$

BEGIN

IF ((SELECT COUNT(\*) FROM users WHERE role\_id = 2) > 1) THEN

RETURN u;

ELSE

RAISE EXCEPTION 'Cant execute operation because there must be at least one admin';

END IF;

END

$$ LANGUAGE plpgsql;

CREATE OR REPLACE FUNCTION check\_admin\_will\_remain\_after\_delete()

RETURNS TRIGGER AS $$

BEGIN RETURN check\_admin\_will\_remain(OLD); END

$$ LANGUAGE plpgsql;

# **ЗАКЛЮЧЕНИЕ**

В рамках выполнения курсового проекта была разработана информационная система для интернет-магазина книг, ориентированная на удобство пользователей и эффективность управления процессами. В процессе работы проведен полный цикл проектирования, включающий разработку архитектуры системы, выбор современных технологий, проектирование функциональных возможностей и базы данных.

Интернет-магазин книг представляет собой клиент-серверное приложение, разработанное с использованием передовых технологий, таких как Node.js для серверной части, React.js для клиентской и PostgreSQL для управления данными. Эти инструменты обеспечили высокую производительность системы, её масштабируемость и гибкость.

В ходе проектирования были реализованы следующие ключевые функции:

1. Каталог товаров с возможностью поиска и фильтрации книг по различным параметрам.
2. Удобная корзина для управления покупками и оформления заказов.
3. Административная панель, позволяющая эффективно управлять товарами, заказами и пользователями.

База данных проекта была детально спроектирована с учётом всех функциональных требований. Нормализация структуры данных, настройка индексов и поддержка целостности позволили создать надёжную и производительную систему, способную работать с большими объёмами информации.

Реализация интернет-магазина книг демонстрирует, как современные технологии и подходы к проектированию программного обеспечения могут быть использованы для создания высококачественного продукта. Система не только отвечает требованиям современного рынка, но и имеет значительный потенциал для развития.

Перспективы развития системы включают:

1. Реализацию рекомендаций на основе машинного обучения.
2. Автоматизацию маркетинговых кампаний с использованием анализа пользовательского поведения.
3. Расширение функционала для поддержки международных пользователей, включая локализацию интерфейса и поддержку различных валют.

В результате выполнения проекта была создана система, которая полностью соответствует поставленным задачам и требованиям. Она может быть использована в реальной практике для автоматизации работы книжного интернет-магазина, повышения его конкурентоспособности и улучшения опыта взаимодействия пользователей с платформой.

Разработанный проект интернет-магазина книг подчёркивает важность интеграции современных технологий для достижения высокого уровня удобства и функциональности. Система была спроектирована таким образом, чтобы её основные компоненты могли легко масштабироваться и адаптироваться к изменениям требований рынка.

Использование PostgreSQL в качестве системы управления базами данных стало важным выбором, позволяющим обеспечивать высокую надёжность и производительность системы. Реляционная структура базы данных с тщательно продуманными связями между сущностями позволяет минимизировать избыточность данных и эффективно управлять информацией о товарах, пользователях и заказах. Особое внимание было уделено оптимизации запросов, что позволяет системе работать быстро даже при значительном увеличении объёмов данных.[20]

Функциональные возможности интернет-магазина тщательно продуманы с точки зрения потребностей конечных пользователей. Реализованный каталог товаров обеспечивает удобный поиск книг по различным критериям, такие как жанр, автор или ценовой диапазон. Внедрение персонализированных рекомендаций позволяет пользователям находить книги, которые соответствуют их интересам, что способствует увеличению продаж.

Особое внимание в проекте было уделено безопасности данных. Использование современных методов аутентификации и авторизации обеспечивает защиту информации о пользователях, заказах и платежах. Система была спроектирована так, чтобы исключить риски, связанные с несанкционированным доступом или уязвимостями в структуре данных.

В процессе выполнения проекта были продемонстрированы основные этапы жизненного цикла разработки программного обеспечения: от постановки задач и анализа требований до проектирования, реализации и тестирования. Это позволило не только создать работоспособный продукт, но и углубить понимание принципов построения современных информационных систем.

**СПИСОК ЛИТЕРАТУРНЫХ ИСТОЧНИКОВ**

[1] "Node.js Design Patterns" - Packt Publishing (2016). [Электронный ресурс]. – Режим доступа: https://www.packtpub.com/product/node-js-design-patterns-second-edition/9781785885587 – Дата доступа: 07.12.2023.

[2] "Mastering PostgreSQL 13" - Packt Publishing (2020). [Электронный ресурс]. – Режим доступа: https://www.packtpub.com/product/mastering-postgresql-13/9781800567495 – Дата доступа: 07.12.2023.

[3] "React - The Complete Guide (incl Hooks, React Router, Redux)" - Udemy (2023). [Электронный ресурс]. – Режим доступа: https://www.udemy.com/course/react-the-complete-guide-incl-redux/ – Дата доступа: 06.12.2023.

[4] "Modern Full-Stack Development" - O'Reilly Media (2020). [Электронный ресурс]. – Режим доступа: https://www.oreilly.com/library/view/modern-full-stack/9781492056466/ – Дата доступа: 06.12.2023.

[5] "Learning React: Functional Web Development with React and Redux" - O'Reilly Media (2020). [Электронный ресурс]. – Режим доступа: https://www.oreilly.com/library/view/learning-react/9781492051720/ – Дата доступа: 06.12.2023.

[6] "PostgreSQL Administration Cookbook" - Packt Publishing (2017). [Электронный ресурс]. – Режим доступа: https://www.packtpub.com/product/postgresql-administration-cookbook/9781787125421 – Дата доступа: 07.12.2023.

[7] "REST API Design Rulebook" - O'Reilly Media (2011). [Электронный ресурс]. – Режим доступа: https://www.oreilly.com/library/view/rest-api-design/9781449317904/ – Дата доступа: 05.12.2023.

[8] "JavaScript: The Good Parts" - O'Reilly Media (2008). [Электронный ресурс]. – Режим доступа: https://www.oreilly.com/library/view/javascript-the-good/9780596517748/ – Дата доступа: 05.12.2023.

[9] "React and React Native" - Packt Publishing (2020). [Электронный ресурс]. – Режим доступа: https://www.packtpub.com/product/react-and-react-native-third-edition/9781839211140 – Дата доступа: 06.12.2023.

[10] "Pro Git" - Apress (2014). [Электронный ресурс]. – Режим доступа: https://git-scm.com/book/en/v2 – Дата доступа: 05.12.2023.

[11] "Database Design for Mere Mortals" - Addison-Wesley Professional (2013). [Электронный ресурс]. – Режим доступа: https://www.pearson.com/store/p/database-design-for-mere-mortals-a-hands-on-guide-to-relational-database-design/P100000152264 – Дата доступа: 05.12.2023.

[12] "Learning Node.js Development" - Packt Publishing (2018). [Электронный ресурс]. – Режим доступа: https://www.packtpub.com/product/learning-node-js-development/9781785283307 – Дата доступа: 07.12.2023.

[13] "PostgreSQL: Up and Running" - O'Reilly Media (2017). [Электронный ресурс]. – Режим доступа: https://www.oreilly.com/library/view/postgresql-up-and/9781491963284/ – Дата доступа: 07.12.2023.

[14] "Building RESTful Web APIs with Node.js" - Packt Publishing (2017). [Электронный ресурс]. – Режим доступа: https://www.packtpub.com/product/building-restful-web-apis-with-node-js/9781788396332 – Дата доступа: 06.12.2023.

[15] "Frontend Development with React" - O'Reilly Media (2022). [Электронный ресурс]. – Режим доступа: https://www.oreilly.com/library/view/frontend-development-with/9781492066858/ – Дата доступа: 06.12.2023.

[16] "High Performance PostgreSQL" - Packt Publishing (2021). [Электронный ресурс]. – Режим доступа: https://www.packtpub.com/product/high-performance-postgresql/9781788395359 – Дата доступа: 07.12.2023.

[17] "JavaScript Patterns" - O'Reilly Media (2010). [Электронный ресурс]. – Режим доступа: https://www.oreilly.com/library/view/javascript-patterns/9781449399115/ – Дата доступа: 05.12.2023.

[18] "Eloquent JavaScript" - No Starch Press (2018). [Электронный ресурс]. – Режим доступа: https://eloquentjavascript.net/ – Дата доступа: 05.12.2023.

[19] "The Road to React" - Leanpub (2023). [Электронный ресурс]. – Режим доступа: https://www.roadtoreact.com/ – Дата доступа: 06.12.2023.

[20] "Learning PostgreSQL" - Packt Publishing (2015). [Электронный ресурс]. – Режим доступа: https://www.packtpub.com/product/learning-postgresql/9781783988983 – Дата доступа: 07.12.2023.

**ПРИЛОЖЕНИЕ А**

**(обязательное)**

**Листинг программного кода**

const ApiError = require('../errors/apiError');

const AuthorRepository = require('../repositories/AuthorRepository');

class AuthorController {

async create(req, res, next) {

const {name} = req.body;

const authorWithSameName = await AuthorRepository.getByName(name);

if (authorWithSameName) {

return next(ApiError.badRequest("Author with same name already exists."));

}

const author = await AuthorRepository.create({name});

return res.json(author);

}

async getAll(req, res) {

const authors = await AuthorRepository.getAll();

return res.json(authors);

}

async update(req, res, next) {

try {

const {id} = req.params;

const {name} = req.body;

const author = await AuthorRepository.update(id, name);

return res.json(author);

} catch (e) {

return next(ApiError.badRequest(e.message));

}

}

async delete(req, res, next) {

try {

const {id} = req.params;

const result = await AuthorRepository.delete(id);

res.status(204).json();

} catch (e) {

return next(ApiError.badRequest(e.message));

}

}

}

module.exports = new AuthorController();

const uuid = require('uuid');

const path = require('path');

const ApiError = require('../errors/apiError');

const pool = require('../db');

const BookRepository = require('../repositories/BookRepository');

class BookController {

async create(req, res, next) {

try {

const {title, price, authorId} = req.body;

let fname;

try {

if (req.files === null) {

fname = "image\_default.png";

} else {

const {image} = req.files;

const filename = uuid.v4() + '.jpg';

image.mv(path.resolve(\_\_dirname, '..', 'static', filename));

fname = filename;

}

} catch (error) {

fname = "image\_default.png";

}

const bookWithSameTitle = await BookRepository.getByTitle(title);

if (bookWithSameTitle) {

return next(ApiError.badRequest(JSON.stringify({field: 'title', text: "Book with same title already exists."})));

}

const book = await BookRepository.create({title, price, authorId, image: fname});

return res.json(book);

} catch (e) {

return next(ApiError.badRequest(e.message));

}

}

async getAll(req, res) {

let {authorId} = req.query;

let books = await BookRepository.getAll(authorId);

books = {rows: books};

return res.json(books);

}

async getById(req, res) {

const {id} = req.params;

const books = await BookRepository.getById(id);

return res.json(books)

}

async update(req, res, next) {

try {

const {id} = req.params;

const newBook = {...req.body};

try {

if (req.files === null) {

} else {

const {image} = req.files;

const filename = uuid.v4() + '.jpg';

image.mv(path.resolve(\_\_dirname, '..', 'static', filename));

newBook.image = filename;

}

} catch (error) {

}

const book = await BookRepository.update(id, newBook);

return res.json(book);

} catch (e) {

return next(ApiError.badRequest(e.message));

}

};

async updateCategories(req, res, next) {

try {

const {id} = req.params;

const {categoriesId} = req.body;

const newCategories = await BookRepository.updateCategories(id, categoriesId);

return res.json(newCategories);

} catch (e) {

return next(ApiError.badRequest(e.message));

}

}

async updateProviders(req, res, next) {

try {

const {id} = req.params;

const {providersId} = req.body;

const newProviders = await BookRepository.updateProviders(id, providersId);

return res.json(newProviders);

} catch (e) {

return next(ApiError.badRequest(e.message));

}

}

async getCategories(req, res, next) {

try {

const {id} = req.params;

const categories = await BookRepository.getCategories(id);

return res.json(categories);

} catch (e) {

return next(ApiError.badRequest(e.message));

}

}

async getProviders(req, res, next) {

try {

const {id} = req.params;

const providers = await BookRepository.getProviders(id);

return res.json(providers);

} catch (e) {

return next(ApiError.badRequest(e.message));

}

}

async delete(req, res, next) {

try {

const {id} = req.params;

const result = await BookRepository.delete(id);

res.status(204).json();

} catch (e) {

return next(ApiError.badRequest(e.message));

}

};

}

module.exports = new BookController();

const ApiError = require('../errors/apiError');

const CategoryRepository = require('../repositories/CategoryRepository');

class CategoryController {

async create(req, res, next) {

const {name} = req.body;

const categotyWithSameName = await CategoryRepository.getByName(name);

if (categotyWithSameName) {

return next(ApiError.badRequest("Author with same name already exists."));

}

const category = await CategoryRepository.create({name});

return res.json(category);

}

async getAll(req, res) {

const categories = await CategoryRepository.getAll();

return res.json(categories);

}

async getBooks(req, res, next) {

try {

const {id} = req.params;

const books = await CategoryRepository.getBooks(id);

return res.json(books);

} catch (e) {

return next(ApiError.badRequest(e.message));

}

}

async update(req, res, next) {

try {

const {id} = req.params;

const {name} = req.body;

const category = await CategoryRepository.update(id, name);

return res.json(category);

} catch (e) {

return next(ApiError.badRequest(e.message));

}

}

async delete(req, res, next) {

try {

const {id} = req.params;

const result = await CategoryRepository.delete(id);

res.status(204).json();

} catch (e) {

return next(ApiError.badRequest(e.message));

}

}

}

module.exports = new CategoryController();

const ApiError = require('../errors/apiError');

const CouponRepository = require('../repositories/CouponRepository');

class CouponController {

async create(req, res, next) {

const {discount} = req.body;

const couponWithSameDiscount = await CouponRepository.getByDiscount(discount)

if (couponWithSameDiscount) {

return next(ApiError.badRequest("Coupon with same discount already exists."));

}

const coupon = await CouponRepository.create({discount});

return res.json(coupon);

}

async getAll(req, res) {

const coupons = await CouponRepository.getAll();

return res.json(coupons);

}

async getByUserId(req, res, next) {

const {id} = req.params;

const books = await CouponRepository.getByUserId(id);

return res.json(books)

}

async update(req, res, next) {

try {

const {id} = req.params;

const {discount} = req.body;

const coupon = await CouponRepository.update(id, discount);

return res.json(coupon);

} catch (e) {

return next(ApiError.badRequest(e.message));

}

}

async delete(req, res, next) {

try {

const {id} = req.params;

const result = await CouponRepository.delete(id);

res.status(204).json();

} catch (e) {

return next(ApiError.badRequest(e.message));

}

}

}

module.exports = new CouponController();

**ПРИЛОЖЕНИЕ Б**

**(обязательное)**

**Конечная схема базы данных**

# **ПРИЛОЖЕНИЕ В**

# **(обязательное)**

# **Ведомость документов**