МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ

Федеральное государственное бюджетное образовательное учреждение высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет	*Кафедра*
вычислительной техники	МОиПЭВМ

Направление подготовки 09.03.02 «Информационные системы и технологии»

Направленность Информационные системы и технологии

БАКАЛАВРСКАЯ РАБОТА

на тему

Веб–приложение для автоматизации процесса книгообмена между читателями

Студент _____ Кадацкая Алена Денисовна
(подпись, дата) (ФИО полностью)

Руководитель _____ Такташкин Д.В.
(подпись, дата) (фамилия, инициалы)

Нормоконтролер _____ Такташкин Д.В.
(подпись, дата) (фамилия, инициалы)

Работа допущена к защите (протокол заседания кафедры от 03.06.2026г. № 12)

Заведующий кафедрой _____ Козлов А.Ю.
(подпись) (фамилия, инициалы)

Работа защищена с отметкой _____ (протокол заседания ГЭК от _____№____)

Секретарь ГЭК _____ Попова Н.А.
(подпись) (фамилия, инициалы)

Пенза, 2026

Федеральное государственное бюджетное образовательное учреждение высшего образования
ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет	*Кафедра*
Вычислительной техники	МО и ПЭВМ

«Утверждаю»
Заведующий кафедрой
_____А.Ю. Козлов
«26» декабря 2025 г

ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ
БАКАЛАВРА

1. Студент Кадацкая Алена Денисовна гр. 22ВИ1 факультета ВТ
<Фамилия, имя, отчество полностью>
2.Тема работы Веб–приложение для автоматизации процесса книгообмена между читателями

Тема утверждена приказом ПГУ №1583/о от "30"декабря 2025 г.

3. Руководитель работы доцент кафедры «МОиПЭВМ», доцент, к.т.н., Такташкин Д.В.

4. Задание на работу (назначение разработки, исходные данные и т.п.)

1) Требуется разработать веб–приложение для автоматизации процесса книгообмена между читателями.
2) Среда выполнения: операционная система Windows 11, Web–браузер
3) Языковые средства разработки: Java, JavaScript, UML
4) Инструментальные средства: Project Libre, Visual Paradigm 17.2, Figma, NetBeans
5) Основные функции:
а) регистрация и аутентификация пользователей;
б) добавление книги в каталог;
в) осуществление обмена;
г) ведение переписки между участниками обмена;
д) просмотр новостной ленты;
е) просмотр каталога книг;
ж) админ–панель и панель модератора для управления приложением;

5. Перечень подлежащих разработке вопросов

1) Анализ предметной области и постановка задачи
2) Проектирование
3) Реализация
4) Контроль качества программного обеспечения
5) План разработки и оценка бюджета
6) Оформление пояснительной записки

5. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Анализ предметной области и требований к системе	10%	07.04.2026 – 13.04.2026	
2	Проектирование программных средств	20%	14.04.2026 – 25.04.2026	
3	Реализация программных средств	30%	27.04.2026 – 11.05.2026	
4	Тестирование программных средств	20%	12.05.2026 – 21.05.2026	
5	Оформление пояснительной записки	20%	22.05.2026 – 02.06.2026	

Дата выдачи задания «26» декабря 2025г.

Руководитель бакалаврской работы _____ 26.12.2025 г.
(подпись, дата)

Задание к исполнению принял студент _____ 26.12.2025 г.
(подпись, дата)

Работу к защите допустить
Декан ФВТ _____ Л.Р. Фионова
(подпись, дата)

Реферат

Пояснительная записка содержит 103 лист, 24 рисунка, 15 таблиц, 20 использованных источников и 3 приложения.

ВЕБ–ПРИЛОЖЕНИЕ, КНИГООБМЕН, NETBEANS, JAVA, UML, JAVASCRIPT, MVC

Объектом исследования являются процесс книгообмена между читателями и веб–технологии для его автоматизации.

Предметом исследования являются программные средства и методы разработки веб–приложений, обеспечивающего регистрацию пользователей, публикацию информации о книгах, организацию обмена литературой и коммуникацию между участниками.

Разработка приложения проводилась на языке программирования JavaScript и Java. В качестве среды разработки использовалась Project Libre, Visual Paradigm 17.2, Figma, NetBeans.

Среда исполнения – Microsoft Windows 10 и выше, web-браузер.

Результаты работы – в ходе выполнения выпускной квалификационной работы были созданы клиентская и серверная части веб–приложения для автоматизации процесса книгообмена между читателями.

Содержание

Введение 7
1 Постановка задачи и анализа требований к web–приложению для автоматизации процесса книгообмена между читателями 9
1.1 Анализ предметной области и постановка задачи 9
1.2 Обзор аналогов 10
1.2.1 LiveLib 10
1.2.2 BookCrossing.ru 11
1.2.3 Группы в социальных сетях 12
1.2.4 Сравнение продуктов–аналогов 13
1.3 Анализ функциональных требований 14
1.4 Анализ нефункциональных требований 17
2 Проектирование веб–инструмента по созданию и редактированию интерактивных диалогов в игровых приложениях 21
2.1 Архитектура приложения 21
2.2 Проектирование структуры хранения 24
2.3 Проектирование программного средства 28
2.4 Проектирование пользовательского интерфейса 33
2.5 Планирование разработки и оценка бюджета 35
3 Реализация программных средств веб–приложения для автоматизации процесса книгообмена 38
3.1 Реализация слоя данных 38
3.2 Реализация слоя бизнес–логики 44
3.3 Разработка слоя представления 48
4 Тестирование программного модуля «Добавление книги» 55
4.1 Тестирование методом черного ящика 55
4.1.1 Классы эквивалентности 55
4.1.2 Функциональная диаграмма и таблица решений 56

4.2 Модульное тестирование добавление книг 59
4.3 Анализ сложности кода модуля 61
Заключение 66
Список использованных источников 68
Приложение А (справочное) Глоссарий 71
Приложение Б (обязательное) Листинг программы 75
Приложение В (рекомендуемое) Листинг автоматизированных тестов 98

Введение

В современном мире развитие информационных технологий охватывает все сферы человеческой деятельности, и книжная культура не является исключением. Традиционные способы распространения книг через библиотеки и книжные магазины дополняются новыми форматами взаимодействия между читателями. Одним из таких форматов является буккроссинг – общественное движение, основанное на принципе свободного обмена книгами между людьми.

Несмотря на растущую популярность этого движения отсутствие централизованной платформы для каталогизации книг, отслеживания статуса обмена и обеспечения коммуникации между участниками создает существенные трудности для масштабирования буккроссинг-сообществ. Современные веб-технологии позволяют создавать специализированное программное решение, способное автоматизировать все этапы книгообмена и объединить читателей в рамках единой экосистемы.

Объект исследования – процесс книгообмена между читателями и веб–технологии для его автоматизации.

Предмет исследования – программные средства и методы разработки веб-приложений, обеспечивающего регистрацию пользователей, публикацию информации о книгах, организацию обмена литературой и коммуникацию между участниками.

Целью выпускной квалификационной работы является разработка веб-приложения для автоматизации процесса книгообмена между читателями, предоставляющего удобный инструментарий для каталогизации книг, поиска литературы, организации безопасного обмена и взаимодействия внутри сообщества.

Для достижения поставленной цели необходимо выполнить следующие задачи:

– провести сравнительный анализ существующих платформ для книгообмена, выявить их достоинства и функциональные ограничения;

– определить и проанализировать функциональные и нефункциональные требования к разрабатываемому веб–приложению;

– спроектировать архитектуру программной системы, включая структуру базы данных, серверную и клиентскую части;

– осуществить разработку программного продукта с использованием современных технологий и фреймворков;

– провести тестирование разработанного приложения и оценить его соответствие заявленным требования;

– оформить пояснительную записку в соответствии с установленными стандартами.

Веб-приложение для автоматизации процесса книгообмена между читателями позволит упростить публикацию и поиск книг, обеспечить прозрачность обменных операций и создать удобную среду для взаимодействия участников буккроссинг–движения. Практическая реализация будет способствовать популяризации чтения, снижению затрат на потребление литературы и развитию культуры обмена книгами среди широкой аудитории пользователей.

1 Постановка задачи и анализа требований к web-приложению для автоматизации процесса книгообмена между читателями

1.1 Анализ предметной области и постановка задачи

Современное общество постепенно проявляет интерес к более осознанному потреблению ресурсов. Домашние библиотеки копят большое количество прочитанных книг, которые зачастую пылятся на полках, в то время как другие читатели нуждаются именно в этих изданиях.

Старые способы обмена, такие как буккроссинг-зоны в общественных местах, социальные сети или тематические форумы, имеют ряд существенных ограничений. Они не имеют структурированного каталога доступных книг, отслеживания перемещения изданий и полноценной коммуникации между читателями.

Проектирование веб-приложения для автоматизации книгообмена является актуальной задачей, решение которой позволит перевести процесс книгообмена на новый уровень. Такая система должна представлять собой централизованную платформу, основными сущностями которой являются:

– читатели, которые выступают в роли владельцев книг и желающих их получить;

– книги, представленные конкретными экземплярами с детальным описанием;

– сессии обмена, которые фиксируют договоренности между пользователями и статусы книг.

Программное средство должно иметь понятный пользовательский интерфейс, корректно отображающий как каталог доступных книг, так и личный кабинет пользователя с его активными и завершенными обменами.

Цель данного инструмента – создание эффективной и безопасной цифровой среды, позволяющей снизить временные и трудозатраты на поиск литературы и организацию обмена.

1.2 Обзор аналогов

Для определения текущего состояния рынка и выявления сильных и слабых сторон существующих решений был проведен анализ нескольких сервисов, предоставляющих возможность книгообмена или функционально близких к разрабатываемому веб-приложению.

1.2.1 LiveLib

LiveLib – это крупнейшая русскоязычная социальная сеть читателей, которая помимо ведения читательских дневников, рецензий и подборок предоставляет пользователям возможность обмениваться книгами. На сайте существует специальный раздел, где читатели могут разместить объявление о передаче книги, продаже или о поиске нужного издания [1]. Интерфейс раздела книгообмена на LiveLib приведен на рисунке 1.

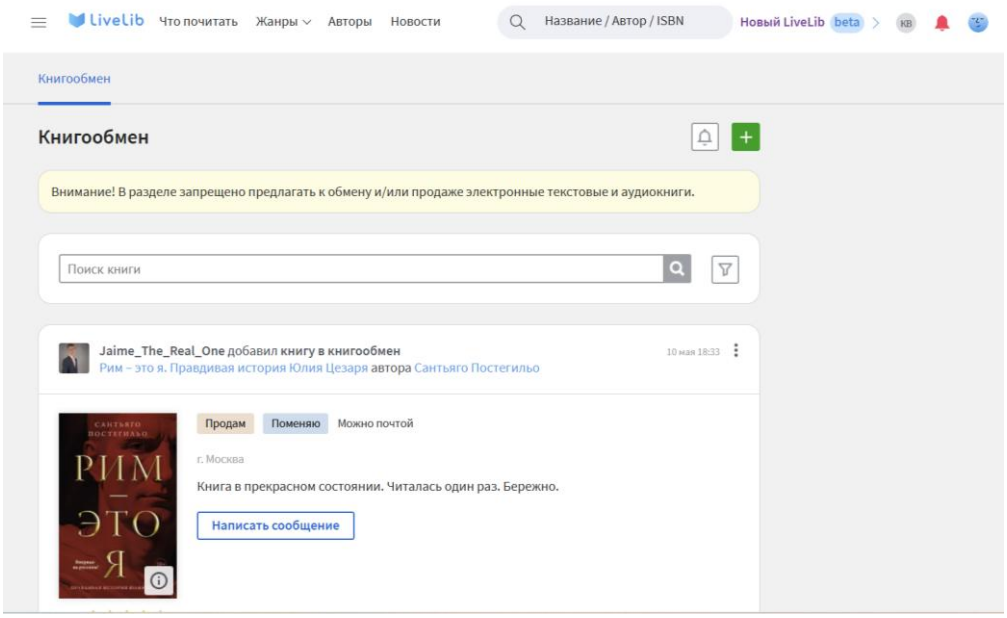

Рисунок 1 – Интерфейс раздела книгообмена «LiveLib»

К достоинствам сервиса можно отнести большую базу пользователей по всей стране, что увеличивает шансы найти нужную книгу. Кроме того, сайт предоставляет удобные инструменты для каталогизации домашней библиотеки с привязкой к конкретным изданиям, а также позволяет отмечать желаемые книги.

К недостаткам стоит отнести то, что книгообмен здесь является лишь

второстепенной функцией, а не основным направлением. Вся коммуникация сводится к личным сообщениям, а сам обмен происходит вне системы и никак в ней не фиксируется. Нельзя отследить, какие книги пользователь отдал, а какие получил. Таким образом, отсутствие полноценного процесса обмена является ключевым ограничением данного сервиса.

1.2.2 BookCrossing.ru

BookCrossing.ru – один из старейших российских сайтов, посвященных непосредственно буккроссингу. Он позволяет регистрировать книги, присваивать им уникальные идентификаторы и отслеживать их путешествие от одного читателя к другому [2]. Интерфейс сайта представлен на рисунке 2.

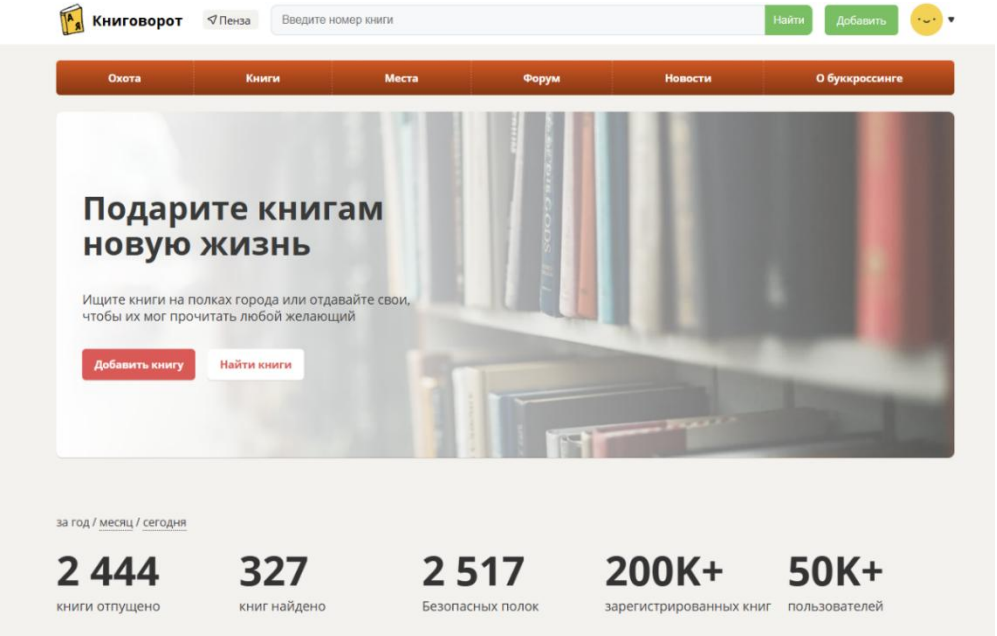

Рисунок 2 – Интерфейс сайта «BookCrossing.ru»

Главным достоинством является идея отслеживания пути книги. У каждого зарегистрированного экземпляра есть своя страница с историей перемещений.

Однако сайт имеет и серьезные недостатки. Интерфейс морально устарел и неудобен. Самое главное – на сайте отсутствует механизм прямого обмена между читателями: нельзя отправить запрос на книгу, нельзя принять или отклонить его. Пользователь может лишь зарегистрировать книгу и отпустить ее

в свободное путешествие, но сам процесс передачи никак не управляется внутри системы. Кроме того, нет возможности вести диалог между читателями. Следовательно, отсутствие инструментов для организации самого обмена и устаревший интерфейс являются основными ограничениями данного ресурса.

1.2.3 Группы в социальных сетях

На сегодняшний день самым популярным способом книгообмена остаются тематические группы в социальных сетях, таких как ВКонтакте и Telegram. Пользователи публикуют посты с описанием книг, которые они готовы отдать или хотят получить, после чего договариваются о встрече или пересылке в личных сообщениях [3]. Пример такой группы приведен на рисунке 3.

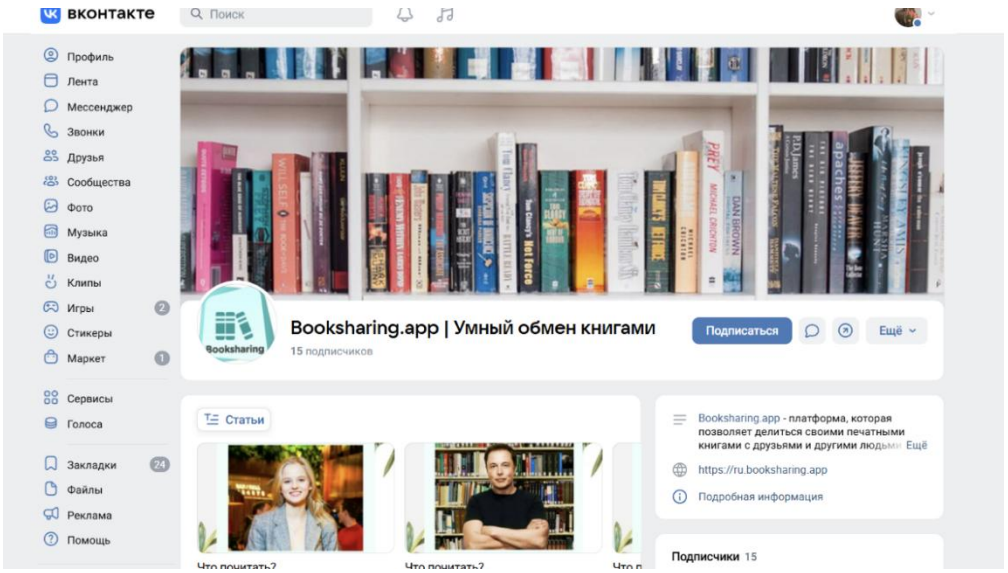

Рисунок 3 – Пример группы книгообмена в социальной сети

К достоинствам такого подхода можно отнести простоту и доступность, так как не нужно регистрироваться на отдельном сайте, а публикация объявления занимает считанные минуты. Кроме того, в популярных городских группах состоит большое количество участников, что обеспечивает широкий охват.

Однако этот способ обладает критическими недостатками для полноценного книгообмена. Во-первых, все объявления публикуются вперемешку, а поиск конкретной книги превращается в ручной просмотр десятков постов из-за отсутствия структурированного каталога. Во-вторых,

невозможно отследить историю своих обменов – какие книги были отданы, а какие получены. В-третьих, все общение происходит исключительно в личных сообщениях, без какой-либо связи с карточкой книги. В результате поиск книги занимает много времени, а сам процесс обмена остается полностью неструктурированным.

1.2.4 Сравнение продуктов–аналогов

Проведем сравнительный анализ рассмотренных выше сервисов книгообмена.

Для оценки будет использоваться трехбалльная система, в которой 0 означает «Неудовлетворительно», 1 – «Удовлетворено частично», а 2 – «Удовлетворительно» рассматриваемой функции. Результаты сравнения приведены в таблице 1.

Таблица 1 – Сравнение программных средств аналогичного назначения

Критерий оценивания	LiveLib	BookCrossing.ru	Группы в социальных сетях
Удобство каталогизации книг	2	1	0
Отправка запроса на обмен и ответ на него	0	0	0
Переписка между читателями	1	0	1
Сохранение желаемых книг	1	2	0
История отпущенных и полученных книг	0	2	0
Интуитивно понятный интерфейс	2	1	1
ИТОГО	6	6	2

Таким образом, можно сделать вывод, что ни одно из рассмотренных решений не предоставляет полноценного механизма для организации книгообмена. LiveLib хорошо справляется с каталогизацией книг, но полностью лишен инструментов для самого обмена. BookCrossing.ru позволяет отслеживать перемещение книг и сохранении желаемых, но не дает возможности договориться об обмене внутри платформы. Группы в социальных сетях наиболее популярны, но в них отсутствует какая-либо структура и автоматизация процесса.

В результате анализа принято решение о целесообразности разработки

собственного веб-приложения. Разрабатываемое приложение будет стремиться объединить удобный каталог книг и список желаемого с полноценным процессом обмена в интуитивно понятном интерфейсе.

1.3 Анализ функциональных требований

На основе анализа предметной области и поставленных задач были сформулированы следующие функциональные требования к разрабатываемому веб-приложению для автоматизации процесса книгообмена между читателями:

Для незарегистрированного пользователя (гостя):

- просматривать каталог доступных для обмена книг;
- искать книги по названию, автору или жанру;
- пройти регистрацию в системе;
- изучить карточку книги.

Для зарегистрированного читателя:

- авторизоваться в системе;
- добавить книгу в свой список с указанием детальной информации;
- редактировать информацию о своей книге;
- удалить книгу из своего списка;
- просматривать каталог книг, доступных для обмена;
- провести поиск книги по названию, автору или жанру;
- просмотреть карточку книги;
- нажать «Хочу эту книгу», после чего книга сохраняется в разделе «Желаемые» личного кабинета;
- отправить запрос на обмен владельцу интересующей книги;
- принять или отклонить запрос на обмен от другого читателя;
- вести переписку с другим читателем внутри приложения;
- просматривать историю своих обменов «Отпущенные» и «Пойманные»;
- просмотреть новостную ленту;

– добавить аватарку в личный кабинет;

– написать в техподдержку.

Для описания функциональных требований к системе была составлена диаграмма вариантов использования.

Диаграмма вариантов использования – диаграмма, описывающая, какой функционал разрабатываемой программной системы доступен каждой группе пользователей [4].

Она отражает отношения между актерами и вариантами использования. Актер обозначает набор ролей пользователя, взаимодействующего с некоторой сущностью, например, системой, подсистемой или классом. В качестве актера может выступать человек, внешняя сущность, класс или другая система. Вариант использования представляет собой выполняемые системой действия, приводящие к наблюдаемым актерами результатам. Имя варианта использования связано со сценарием, который является конкретной последовательностью действий, иллюстрирующей поведение.

Диаграмма вариантов использования представлена на рисунке 4.

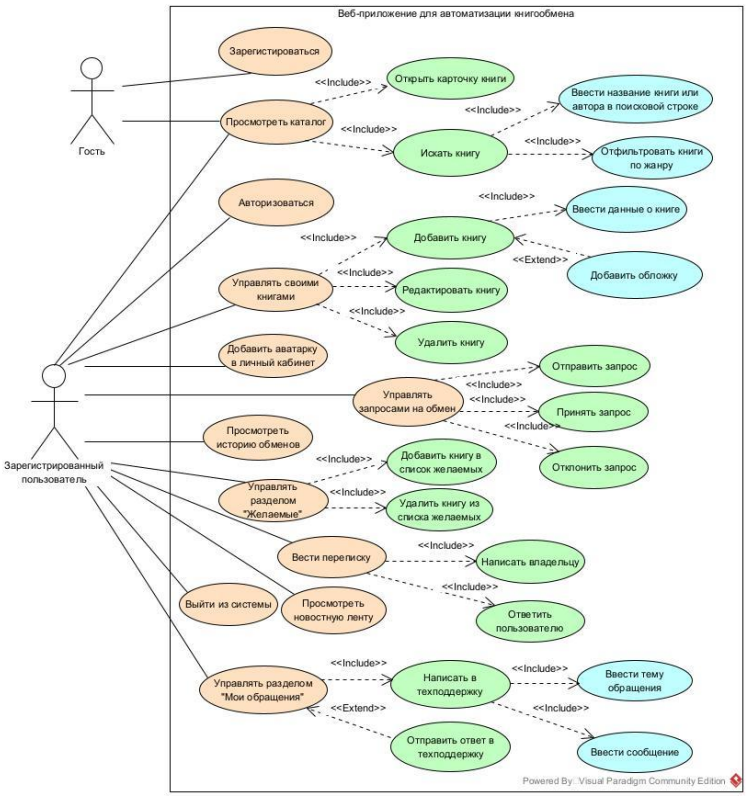

Рисунок 4 – Диаграмма вариантов использования веб-приложения для книгообмена

Спецификация прецедента «Отправить запрос на обмен» представлена в таблице 2.

Таблица 2 – Спецификация прецедента «Отправить запрос на обмен»

Прецедент: Отправить запрос на обмен
ID: 1
Краткое описание: Пользователь отправляет владельцу запрос на обмен для получения книги
Главные актеры: Зарегистрированный читатель
Предусловие: Пользователь авторизован, открыта карточка книги
Основной поток: 1. Пользователь нажимает кнопку «Отправить запрос на обмен»; 2. Система создает запрос и связывает его с книгой и читателями; 3. Система отправляет запрос владельцу книги; 4. Запрос появляется в разделе «Запросы на обмен». Альтернативные потоки: 1.а. Пользователь несколько раз отправляет запрос на эту книгу. Система не создает дубликат заявки.
Постусловия: Владелец книги получил запрос на обмен

Спецификация прецедента «Принять запрос на обмен» представлена в таблице 3.

Таблица 3 – Спецификация прецедента «Принять запрос на обмен»

Прецедент: Принять запрос на обмен
ID: 2
Краткое описание: Владелец подтверждает передачу книги запрашивающему читателю
Главные актеры: Зарегистрированный читатель
Предусловие: Владелец авторизован, запрос на обмен получен, переписка проведена
Основной поток: 1. Владелец открывает «Запросы на обмен»; 2. При необходимости обсуждает детали в переписке; 3. Нажимает кнопку «Принять запрос»; 4. Система переносит книгу из «Мои книги» в «Отпущенные» у владельца; 5. У запрашивающего читателя книга появляется в разделе «Мои книги» и «Пойманные». Альтернативные потоки: 3.а. Владелец нажимает «Отклонить запрос», а система отображает ответ в «Запросы на обмен».
Постусловия: Обмен зафиксирован, книга сменила владельца

Спецификация прецедента «Добавить книгу» представлена в таблице 4.

Таблица 4 – Спецификация прецедента «Добавить книгу»

Прецедент: Добавить книгу
ID: 3
Краткое описание: Пользователь размещает книгу для обмена
Главные актеры: Зарегистрированный читатель
Предусловие: Пользователь авторизован
Основной поток: 1. Пользователь нажимает «Добавить книгу»; 2. Заполняет форму: автор, название, описание, состояние, год выпуска, язык, загружает фото; 3. Нажимает «Сохранить»; 4. Система добавляет книгу в каталог и в раздел «Мои книги». Альтернативные потоки: 2.а. Заполнены не все обязательные поля. Система выводит сообщение об ошибке.
Постусловия: Книга успешно добавлена и доступна другим читателям в общем каталоге

В результате анализа функциональных требований были выделены варианты использования программы. Для основных вариантов использования были представлены их спецификации, которые позволяют отобразить последовательность действий пользователя и системы при выполнении определенных функций.

1.4 Анализ нефункциональных требований

В дополнение к функциональным требованиям были сформулированы нефункциональные требования, предъявляемые к разрабатываемому веб-приложению. Нефункциональные требования (non-functional requirements) описывают, как программа выполняет свои функции, какие у нее характеристики и ограничения. Они задают критерии качества, которым должен соответствовать продукт [5].

Данные требования разделены на две группы: требования к клиентской части (интерфейс пользователя) и требования к серверной части (бэкенд, база данных, инфраструктура).

Требования к клиентской части:

а) производительность:

1) время загрузки любой страницы при скорости соединения не

менее 10 Мбит/с не должно превышать 2 секунд;

2) отклик интерфейса на действия пользователя должен составлять не более 0,5 секунды без учета времени обработки запроса на сервере;

3) интерфейс должен сохранять приемлемую отзывчивость при отображении каталога, содержащего до 10 000 книг.

b) удобство использования:

1) интерфейс должен быть интуитивно понятным: новый пользователь должен освоить основные операции за не более 5 минут без обращения к документации;

2) навигация по разделам личного кабинета должна быть доступна не более чем за 2 клика с главной страницы;

3) форма добавления книги должна содержать обязательные поля («Название», «Автор», «Жанр», «Состояние книги») и опциональные поля («Год издания», «Язык», «ISBN», «Описание», «Обложка»), а также подсказки по заполнению.

c) адаптивность и кроссплатформенность:

1) веб-приложение должно корректно отображаться и функционировать в последних стабильных версиях следующих веб-браузеров: Google Chrome, Mozilla Firefox, Safari, Microsoft Edge, Yandex Browser;

2) интерфейс должен обеспечивать корректное отображение на устройствах с различным разрешением экрана – от 1366×768 пикселей до 3840×2160 пикселей (4K).

d) доступность:

1) все интерактивные элементы (кнопки, ссылки, поля ввода) должны иметь понятные подписи и визуальные индикаторы состояния (активное, наведение, фокус);

2) сообщения об ошибках валидации формы (например, незаполненные обязательные поля) должны выводиться в понятном виде рядом с соответствующим полем.

Требования к серверной части:

a) производительность и нагрузка:

1) серверная часть должна обеспечивать обработку не менее 100 запросов в секунду при штатной нагрузке;

2) время обработки запроса на сервере для операций чтения не должно превышать 200 мс при нагрузке до 5000 одновременных пользователей;

3) время обработки запросов записи не должно превышать 500 мс при нагрузке до 5000 одновременных пользователей.

b) масштабируемость

1) система управления базами данных PostgreSQL должна поддерживать увеличение объема хранимых данных до 100 ГБ без деградации производительности запросов.

c) безопасность

1) пароли пользователей должны храниться в базе данных в хешированном виде;

2) доступ к персональным данным пользователей и переписке должен быть строго ограничен и предоставляться только авторизованным участникам обмена, а также модератору и администратору в рамках их полномочий;

3) административная панель и функции модератора должны быть доступны только пользователям с соответствующей ролью и иметь дополнительную защиту от несанкционированного доступа;

4) функция блокировки пользователей модератором должна автоматически запрещать доступ к системе заблокированному пользователю на указанный срок.

d) надежность и отказоустойчивость

1) система должна обеспечивать сохранность данных пользователя. Информация должна храниться без потерь и корректно восстанавливаться при любых сбоях;

2) при возникновении ошибок на сервере пользователю должно выводиться сообщение с кодом ошибки и предложением повторить действие

позднее; системные ошибки не должны раскрывать внутреннюю структуру приложения (стек вызовов, пути к файлам).

e) совместимость

1) серверная часть должна функционировать под управлением веб-сервера Apache Tomcat версии 9.0 или выше;

2) приложение должно быть совместимо с СУБД PostgreSQL версии 13 или выше;

3) взаимодействие с базой данных осуществляется через JDBC-драйвер версии 42.5.0 или выше;

4) серверная логика реализована на языке Java версии 11 или 17 (Java SE 11 LTS или Java SE 17 LTS);

5) для сборки и развертывания используется среда разработки Apache NetBeans версии 17 или выше;

6) клиентская часть использует JSP (Jakarta Server Pages) версии 3.0, HTML5, CSS3 и JavaScript ES6.

f) объем памяти

1) прогнозируемый объем базы данных PostgreSQL на начальном этапе эксплуатации не должен превышать 500 МБ;

2) максимальный расчетный объем базы данных на пять лет эксплуатации не должен превышать 10 ГБ;

3) размер одной записи о книге (без учета изображения обложки) в базе данных не должен превышать 5 КБ;

4) допустимый размер одного изображения обложки – не более 5 МБ, аватара – не более 2 МБ;

5) общий объем хранилища для загружаемых изображений не должен превышать 50 ГБ на начальном этапе с возможностью расширения до 500 ГБ.

Все эти аспекты будут учтены при разработке и реализации проекта для обеспечения его успешной работы и удовлетворения потребностей пользователей.

2 Проектирование веб-инструмента по созданию и редактированию интерактивных диалогов в игровых приложениях

2.1 Архитектура приложения

Ключевым архитектурным решением при проектировании веб-инструмента для автоматизации книгообмена является его реализация с использованием паттерна проектирования Model View Controller (MVC). Данный подход был выбран как классическое и хорошо зарекомендовавшее себя решение для серверных веб-приложений на платформе Java. MVC разделяет приложение на три взаимосвязанных компонента, обеспечивая четкое разделение ответственности [6].

Для проектирования приложения выбрана многослойная архитектура, которая позволяет четко разграничить задачи каждого компонента системы. Рассмотрим основные слои системы.

1. Слой представления (Presentation Layer) – отвечает за отображение данных и взаимодействие с пользователем [7]. В разрабатываемом приложении этот слой реализован в виде набора динамических веб-страниц на базе JSP (JavaServer Pages). Страницы формируются на стороне сервера и передаются в браузер пользователя в виде готового HTML-документа. Этот слой включает страницы для просмотра каталога книг, карточки книги, личного кабинета, форм для добавления книг, а также страницы для ведения переписки и просмотра новостной ленты. Основной задачей слоя представления является предоставление интуитивно понятного и отзывчивого пользовательского интерфейса, который отвечает поставленным функциональным и нефункциональным требованиям.

2. Слой бизнес–логики (Business Layer) – содержит набор компонентов, которые отвечают за обработку полученных от уровня представлений данных, реализует всю необходимую логику приложения, все вычисления, взаимодействует с базой данных и передает уровню представления результат обработки [8]. В приложении этот слой представлен контроллерами

(сервлетами). Каждый контроллер обрабатывает запросы от пользователя, поступающие через слой представления, выполняет необходимые проверки и преобразования данных, а затем обращается к слою доступа к данным для получения или сохранения информации. Именно здесь реализована логика отправки запроса на обмен, принятия или отклонения запроса, подтверждения передачи книги и ведения переписки между читателями. Контроллеры также отвечают за маршрутизацию запросов и выбор соответствующей страницы для отображения результата. Слой включает в себя интерфейсы для доступа к бизнес–логике, что обеспечивает возможность модульного тестирования и упрощает дальнейшее развитие системы.

3. Слой доступа к данным (Data Access Layer) – это компонент программной архитектуры, отвечающий за хранение и извлечение данных в приложении [9]. Этот слой представлен классами–моделями, которые отражают сущности предметной области, и компонентами для выполнения операций с базой данных. Для работы с PostgreSQL используется JDBC-драйвер, а все SQL-запросы выполняются с использованием параметризованных запросов (PreparedStatement), что обеспечивает защиту от SQL-инъекций. Слой данных предоставляет интерфейс для выполнения операций создания, чтения, обновления и удаления записей.

4. База данных (Database) – отвечает за хранение данных. В качестве системы управления базами данных используется PostgreSQL. База данных хранит всю информацию о читателях, книгах, запросах на обмен, переписке и истории обменов, обеспечивая целостность и согласованность данных.

Клиентская часть приложения разработана с использованием JSP и HTML, что обеспечивает доступ ко всем функциям системы через браузер без необходимости установки дополнительного программного обеспечения. Серверная часть реализована на языке Java с использованием сервлетов, которые работают под управлением веб-сервера Apache Tomcat. Для разработки приложения используется среда NetBeans.

Для визуализации компонентов, входящих в состав приложения, была

разработана диаграмма компонентов, построенная с помощью UML-средств. Диаграмма компонентов позволяет получить высокоуровневое представление о компонентах системы [10].

Пунктирные стрелки показывают отношения взаимозависимости между компонентами. На диаграмме отображены основные компоненты каждого из трех слоев: JSP-страницы слоя представления, сервлеты-контроллеры слоя бизнес–логики и классы–модели с утилитами доступа к базе данных слоя данных.

Диаграмма компонентов обеспечивает согласованный переход от логического представления к конкретной реализации проекта в форме программного кода. На рисунке 5 изображена диаграмма компонентов приложения.

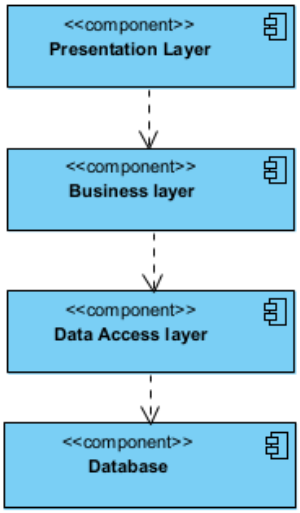

Рисунок 5 – Диаграмма компонентов веб–приложения для автоматизации книгообмена между читателями

Технологии и инструменты, используемые при проектировании:

– Visual Paradigm для создания диаграмм;

– NetBeans для написания клиентской и серверной части приложения;

– Java для реализации серверной логики;

– JSP и HTML для создания динамических веб–страниц своя представления;

– Apache Tomcat для развертывания и запуска приложения;

– PostgreSQL для хранения данных;

– JDBC для взаимодействия Java-приложения с базой данных PostgreSQL.

Итогом этапа проектирования стала выработка многослойной архитектурной модели на основе паттерна MVC. Такая архитектурная организация обладает рядом преимуществ:

– относительная простота в понимании и реализации;

– высокий уровень абстракции, позволяющий изменять реализацию одного слоя без существенного влияния на другие;

– повышенная безопасность за счет изоляции прямого доступа к базе данных от пользовательского интерфейса.

2.2 Проектирование структуры хранения

Одной из ключевых этапов проектирования веб-приложения для автоматизации книгообмена является разработка надежной и эффективной структуры базы данных. Поскольку система работает с большим количеством взаимосвязанных сущностей, было принято решение использовать реляционную систему управления базами данных PostgreSQL.

PostgreSQL – это свободно распространяемая объектно-реляционная система управления базами данных (СУБД) с открытым исходным кодом, написанном на языке C [11].

Такой выбор обеспечивает целостность данных, поддержку транзакций и возможность выполнения сложных запросов.

Центральными сущностями предметной области являются «Пользователь» (users), «Книга» (books), «Обмен» (exchanges) и «Сообщение» (messages). В рамках анализа были выделены их ключевые атрибуты и связи с вспомогательными сущностями, на основе чего спроектирована логическая модель базы данных. ER-диаграмма разработанной базы данных представлена на рисунке 6.

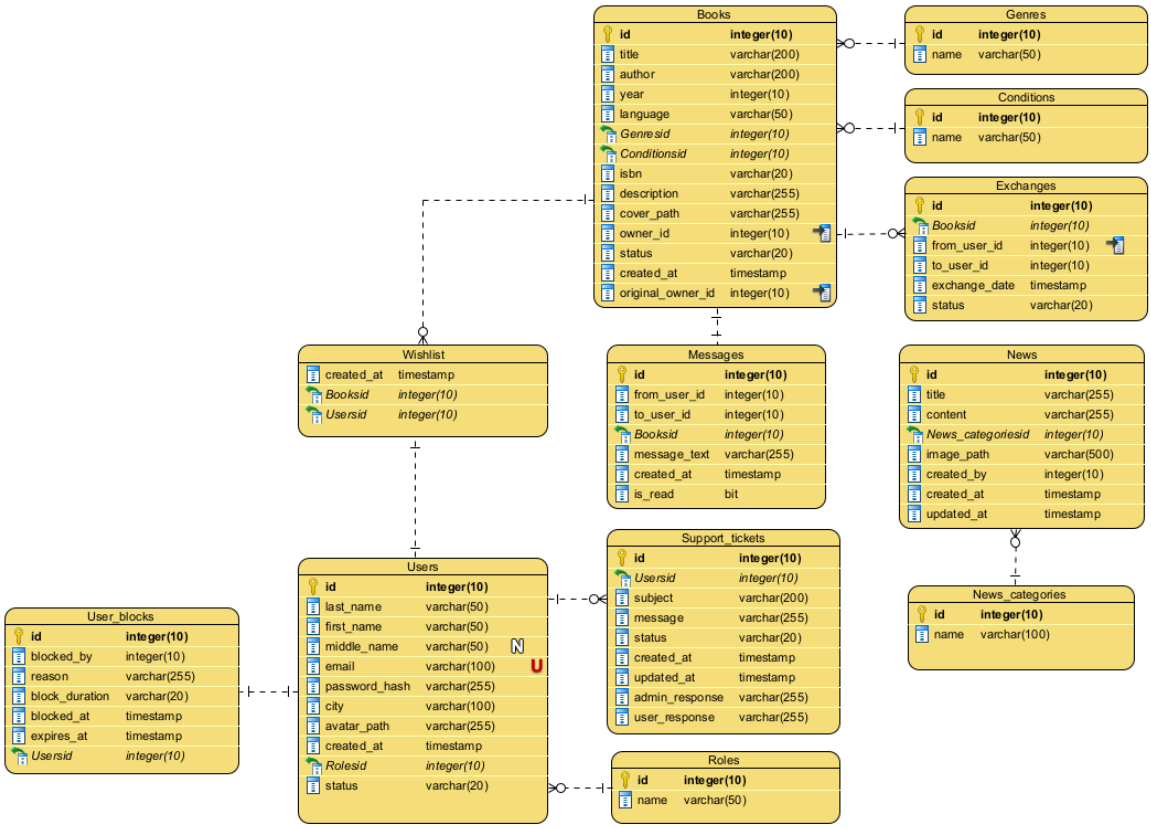

Рисунок 6 – ER-диаграмма БД для веб–приложения книгообмена

Таблица users является центральной в системе и содержит информацию о зарегистрированных пользователях. Подробное описание полей приведено в таблице 5.

Таблица 5 – Сущность «Users»

Поле	Тип данных	Описание
id	Integer	Уникальный идентификатор пользователя. Является первичным ключом таблицы.
last_name	Character varying	Фамилия пользователя.
middle_name	Character varying	Имя пользователя.
email	Character varying	Адрес электронной почты пользователя, используется для входа в систему
password_hash	Character varying	Хеш пароля пользователя для обеспечения безопасности. Хранится в зашифрованном виде.
city	Character varying	Город проживания пользователя
avatar_path	Character varying	Путь к файлу изображения аватара пользователя на сервере.
created_at	Timestamp	Дата и время регистрации пользователя в системе.
role_id	Integer	Определяет уровень доступа пользователя в системе
status	Character varying	Текущий статус учетной записи пользователя

Таблица books представляет собой каталог книг, предлагаемых для обмена.

Она хранит ключевую информацию о каждом издании и его текущем владельце. Описание ее полей представлено в таблице 6.

Таблица 6 – Сущность «Books»

Поле	Тип данных	Описание
id	Integer	Уникальный идентификатор пользователя. Является первичным ключом таблицы.
title	Character varying	Полное название книги.
author	Character varying	Автор(ы) произведения
year	Integer	Год издания данного экземпляра книги
language	Character varying	Язык, на котором написана книга
genre_id	Integer	Определяет жанровую принадлежность книги
condition_id	Integer	Определяет физическое состояние экземпляра книги
isbn	Character varying	Международный стандартный книжный номер
description	Character varying	Краткое описание или аннотация книги, предоставляемая владельцем.
cover_path	Character varying	Путь к файлу изображения обложки книги на сервере.
owner_id	Integer	Идентифицирует текущего владельца книги.
status	Character varying	Текущий статус книги в системе
created_at	Timestamp	Дата и время публикации объявления о готовности книги к обмену.
original_owner_id	Integer	Идентифицирует первоначального владельца книги.

Таблица exchanges является центральной для автоматизации процесса обмена. Она фиксирует факт заключения и выполнения сделки между двумя пользователями в отношении конкретной книги. Детальная структура таблицы представлена в таблице 7.

Таблица 7 – Сущность «Exchanges»

Поле	Тип данных	Описание
id	Integer	Уникальный идентификатор пользователя. Является первичным ключом таблицы.
book_id	Integer	Указывает на книгу, которая является предметом данного обмена.
from_user_id	Integer	Идентифицирует текущего владельца книги, который передает ее в рамках обмена.
to_user_id	Integer	Идентифицирует пользователя, который получает книгу в результате обмена
exchange_date	Timestamp	Дата и время, когда обмен был фактически совершен или согласован сторонами
status	Character varying	Текущий статус сделки обмена

Таблица messages обеспечивает коммуникацию между пользователями по поводу конкретной книги в рамках потенциального или текущего обмена.

Описание ее полей представлено в таблице 8.

Таблица 8 – Сущность «Messages»

Поле	Тип данных	Описание
id	Integer	Уникальный идентификатор пользователя. Является первичным ключом таблицы.
book_id	Integer	Связывает сообщение с конкретной книгой, по поводу которой ведется обсуждение.
from_user_id	Integer	Идентифицирует отправителя сообщения.
to_user_id	Integer	Идентифицирует получателя сообщения
message_text	Character varying	Текстовое содержимое сообщения, которым обмениваются пользователи для уточнения деталей обмена.
created_at	Timestamp	Дата и время отправки сообщения.
is_read	Boolean	Флаг, указывающий, было ли сообщение прочитано получателем.

Остальные таблицы выполняют справочные и вспомогательные функции, обеспечивая нормализацию данных и расширенную функциональность приложения.

Таблица genres хранит перечень литературных жанров, а таблица conditions содержит возможные варианты физического состояния книги.

Таблица roles определяет роли пользователей в системе (читатель, модератор, администратор), разграничивая уровни доступа.

Таблица wishlist связывает пользователя с книгой, формируя список желаемых к получению изданий.

Таблица news и news_categories обеспечивают функционирование новостной ленты приложения.

Таблица support_tickets хранит обращения пользователей в службу технической поддержки, связывая пользователя с текстом обращения и статусом его обработки.

Таблица user_blocks предназначена для администрирования системы и фиксирует факты блокировки пользователей.

На основе полученной структуры базы данных будет реализован слой данных приложения. Четко определенные сущности и их взаимосвязи, позволят реализовать весь необходимый функционал для автоматизации процесса

книгообмена, гарантируя логическую целостность и эффективное управление информацией о пользователях, книгах и транзакциях.

2.3 Проектирование программного средства

На основе всестороннего анализа предметной области, тщательно сформулированных функциональных и нефункциональных требований, а также детально проработанной структуры базы данных, было осуществлено комплексное проектирование программных средств разрабатываемого веб–приложения. Основополагающим этапом данного процесса являлось явилось определение основных классов системы, их атрибутов, функциональных методов и сложных взаимосвязей, формирующих архитектурное ядро приложения.

Для обеспечения гибкости и тестируемости кода, серверная часть приложения спроектирована с использованием многослойной архитектуры, реализованной на языке Java в среде разработки NetBeans. Данный подход обеспечивает четкое разделение ответственности между компонентами системы. Основными слоями серверной части являются: Busines layer (бизнес–логика), Data Access Layer (Доступ к данным) и Databases (База данных).

Контроллеры служат точкой входа для обработки HTTP-запросов от клиентской части. Они принимают запросы, вызывают соответствующие методы сервисов для выполнения бизнес-логики и возвращают ответы, формируя основу REST API приложения. В рамках системы спроектированы следующие контроллеры:

- AuthController – обрабатывает запросы на аутентификацию и регистрацию пользователей;
- BookController – управляет каталогом книг, обеспечивая добавление, редактирование, удаление и поиск книг с фильтрацией по жанрам и другим параметрам;
- ExchangeController – обеспечивает создание запроса, отправку и получение ответа на него, а также успешное завершение обмена;

- UserController – предоставляет функциональность для управления профилями пользователей;
- MessageController – обеспечивает обмен сообщениями между пользователями в рамках конкретной книги;
- AdminController и ModeratorController – предоставляют расширенные функции для управления системой, пользователями и контентом для администратора и модератора;
- NewsController – отвечает за управление новостной лентой приложения;
- TicketController – обрабатывает запросы пользователей в службу технической поддержки;
- HomeController – отвечает за открытие главной страницы при запуске приложения.

Сервисы инкапсулируют в себе ключевую бизнес-логику приложения и выступают в роли связующего звена между контроллерами и слоем доступа к данным (DAO). В проекте определены следующие сервисы:

- UserService – отвечает за регистрацию и аутентификацию пользователей с хешированием паролей, загрузку аватаров, проверку ролей и статуса блокировки, получение книг пользователя, блокировку и разблокировку пользователей, а также получение статистики и CRUD-операции с пользователями;
- BookService – обеспечивает получение каталога книг с фильтрацией по жанру и поиску, CRUD книг, загрузку и удаление обложек, обновление статуса книги и передачу права собственности;
- ExchangeService – отвечает за создание запроса на обмен с проверкой валидности, а также принятие, отклонение и завершение обмена со сменой владельца книги;
- MessageService – отвечает за получение диалогов пользователя, переписки по конкретной книге, отправку сообщений, отметку о прочтении и подсчет непрочитанных сообщений;

- AdminService – управляет пользователями, книгами, обменами, а также за просмотром новостей и обращений в поддержку;
- ModeratorService – отвечает за блокировку и разблокировку пользователей, просмотр пользователей и книг с фильтрацией, удаление обложек книг и просмотр новостей;
- NewsService – отвечает за CRUD-операции над новостной лентой, получение категорий новостей, загрузку и удаление изображений;
- TicketService – обеспечивает создание обращений в поддержку, получение обращений пользователя, добавление ответа пользователя, а для администратора – получение всех обращений с фильтром по статусу и обновление статуса с ответом;

Слой доступа к данным (DAO) отвечает за взаимодействие с базой данных PostgreSQL. Каждый DAO-класс реализует типовые операции создания, чтения, обновления и удаления (CRUD) для своей сущности. Использование паттерна DAO позволяет в будущем легко заменить источник данных без внесения изменений в бизнес–логику. Спроектированы следующие DAO-классы: UserDAO, BookDAO, ExchangeDAO, MessageDAO, GenreDAO, ConditionDAO, NewsDAO, NewsCategoryDAO, TicketDAO, BlockDAO.

Модели представляют собой объекты передачи данных, которые используются для обмена информацией между слоями приложения. Ключевыми моделями являются: User, Book, Exchange, Message, Genre, Condition, Role, News, NewsCategory, Ticket, Owner, Dialog, BlockInfo.

Для обеспечения наглядного и структурированного представления этих фундаментальных элементов и их комплексной организации в рамках проектируемой системы традиционно прибегают к использованию диаграммы классов. Диаграмма классов предназначена для описания структуры системы на высоком уровне абстракции, демонстрируя не только сами классы, но и их внутреннее устройство через атрибуты и операции, а также взаимосвязи между ними [12]. Диаграмма классов ключевых компонентов приложения представлена на рисунке 7.

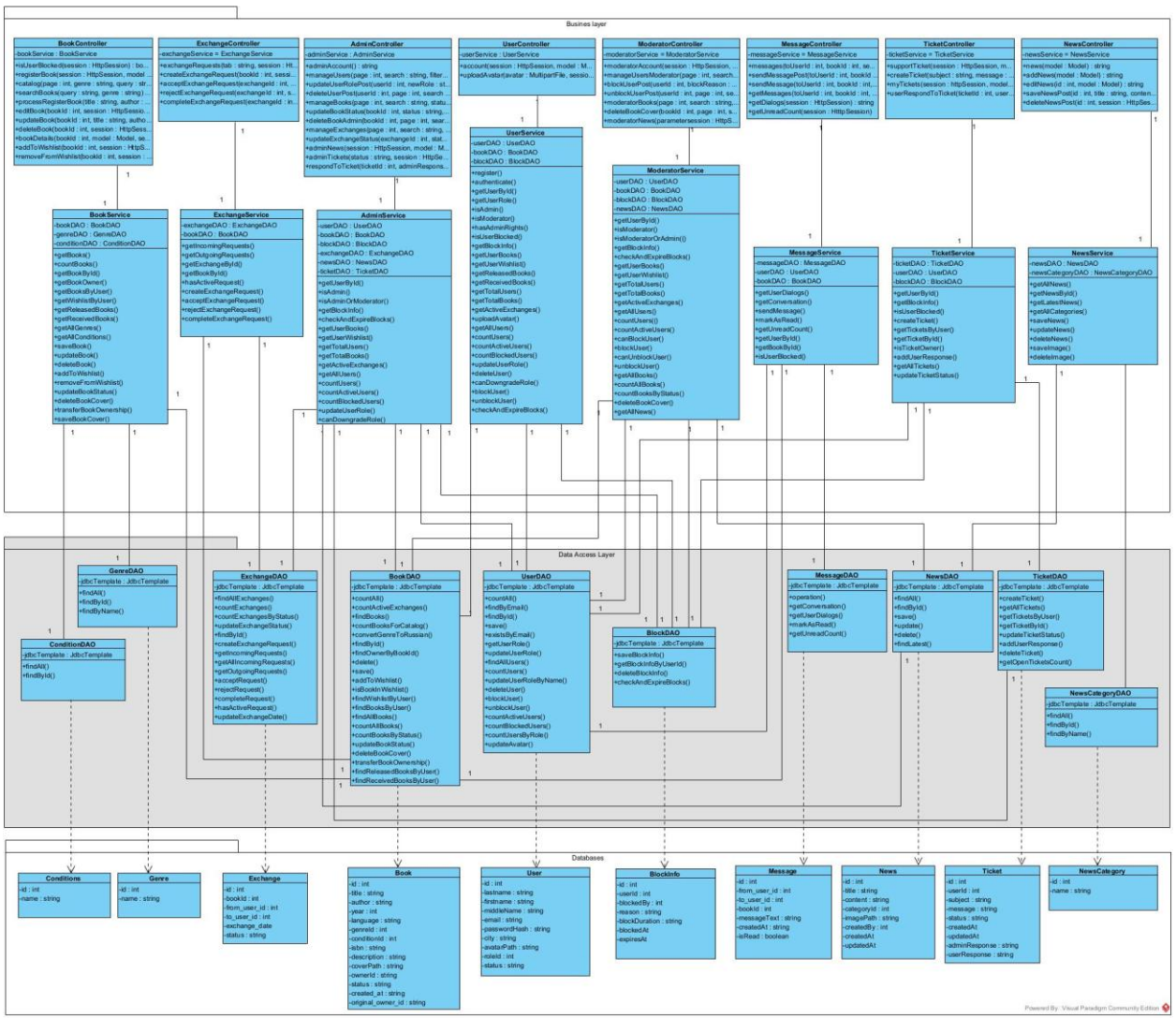

Рисунок 7 – Диаграмма классов ключевых компонентов приложения

На диаграмме отображены три логических слоя системы. Слой бизнес-логики содержит контроллеры и сервисы, связанные отношением ассоциации, поскольку сервисы внедряются в контроллеры через конструктора и используются на протяжении всего времени жизни контроллера. Слой доступа к данным представлен DAO-классами, с которыми сервисы также связаны отношением ассоциации. Слой базы данных содержит модели данных, с которыми DAO-классы связаны отношением зависимости, так как DAO создает и возвращает объекты моделей, но не хранит их как поля класса.

Для сценария, описывающего основной функционал приложения, а именно добавление новой книги в каталог, была разработана диаграмма деятельности [13], представленная на рисунке 8.

На данной диаграмме выделены две основные дорожки: «Пользователь», представляющая действия читателя, взаимодействующего с интерфейсом, и «Система», отражающая внутренние процессы и реакции веб–приложения на действия пользователя.

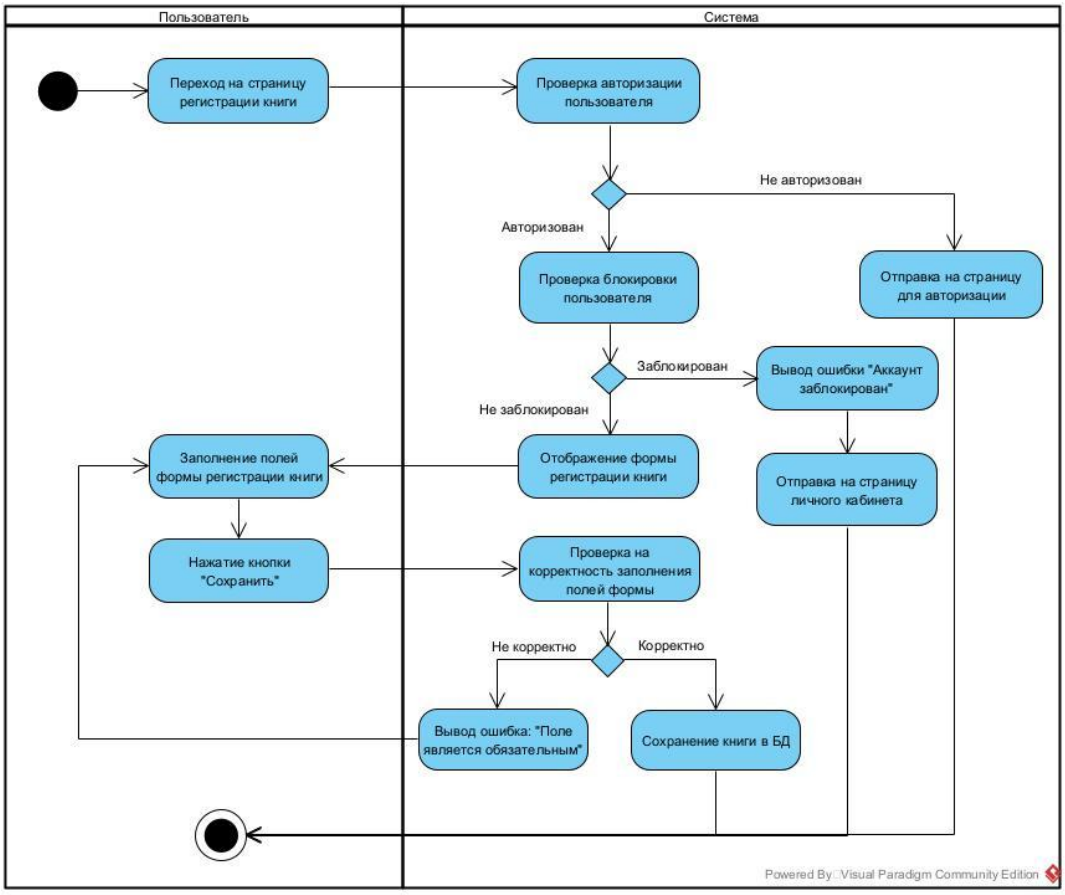

Рисунок 8 – Диаграмма деятельности добавления книги

Процесс начинается с того, что пользователь переходит на страницу регистрации книги. Система проверяет, авторизован ли пользователь. Если авторизован, то система проверяет заблокирован ли пользователь, и если нет, то система отображает форму регистрации книги. Пользователь заполняет обязательные поля, указывает дополнительные сведения и загружает обложку. Для сохранения книги должна быть нажата кнопка «Сохранить» и, если все обязательные поля заполнены, книга успешно добавляется в БД.

В альтернативном случае, если пользователь не авторизован или заблокирован, система выведет соответствующее сообщение. Если были

заполнены не все обязательные поля, система выведет на экран предупреждение и не даст сохранить книгу, пока нарушение не будет исправлено.

Таким образом, в рамках данного раздела было осуществлено проектирование программных средств, что выразилось в определении структуры классов и детализации ключевых сценариев взаимодействия пользователя с системой посредством диаграмм деятельности и последовательности, формируя основу для последующей разработки.

2.4 Проектирование пользовательского интерфейса

Пользовательский интерфейс веб-приложения для книгообмена должен быть интуитивно понятен и удобен в освоении пользователем. Поскольку приложение разрабатывается с использованием Java и технологий Jakarta Server Pages (JSP), клиентская часть реализована в виде серверного рендеринга страниц, где каждый экран формируется на стороне сервера и передается браузеру в готовом виде.

Для визуального проектирования эскизов интерфейса использовался инструмент Figma. Это онлайн графический редактор, который используется для создания дизайна интерфейсов, прототипирования и работы с векторными изображениями. Одним из главных преимуществ Figma является возможность совместной работы над проектами в реальном времени, что делает процесс дизайна более эффективным и гибким [14].

При первом запуске приложения пользователь попадает на главный экран, который представлен на рисунке 9. Он содержит навигационную панель с кнопками «Войти», «Зарегистрироваться», «Добавить книгу» и «Найти книгу», а также приветственный баннер с названием платформы и кратким описанием концепции буккроссинга. В центральной части страницы расположены информационные карточки, поясняющие преимущества сервиса: возможность бесплатного обмена книгами, удобный поиск и безопасные сделки.

Если пользователь не авторизован, при нажатии кнопки «Добавить книгу», его перенаправит на страницу авторизации.

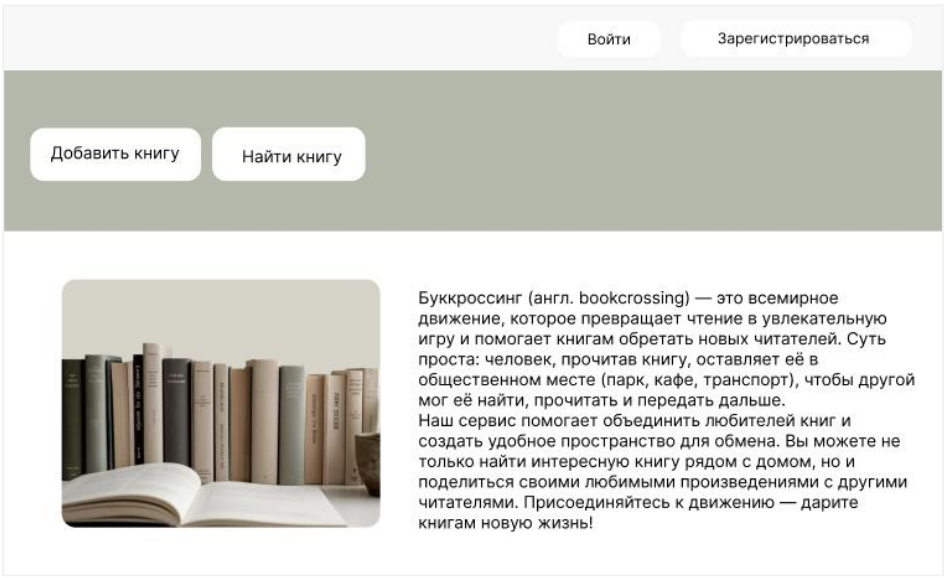

Рисунок 9 – Прототип главного экрана

Основным рабочим пространством для добавления книг является страница регистрации книги (рисунок 10). Она содержит форму с обязательными полями: «Название», «Автор», «Жанр», «Состояние книги», а также опциональными полями: «Год издания», «Язык», «ISBN», «Описание» и «Обложка». Для загрузки обложки реализован механизм предварительного просмотра изображения, что позволяет пользователю убедиться в корректности выбранного файла до сохранения. При редактировании существующей книги используется та же форма, но с заполненными данными.

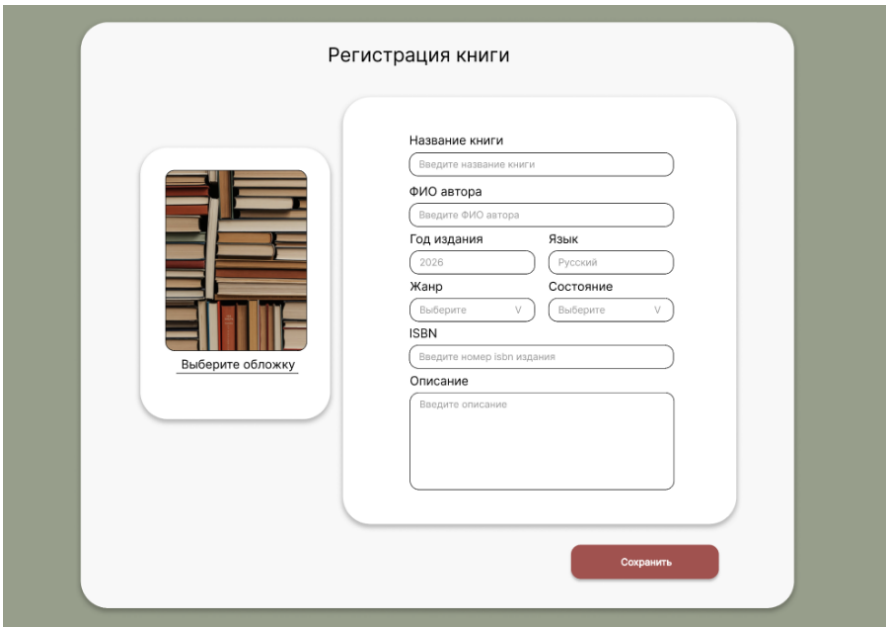

Рисунок 10 – Прототип окна регистрирования книги

Для просмотра списка доступных для обмена книг, пользователь должен перейти на страницу «Каталог» (рисунок 11). На странице реализована фильтрация по жанрам, поиск по названию книги или автору, а также пагинация для навигации по большому количеству записей. Каталог книг организован в виде сетки карточек, что обеспечивает удобный просмотр большого количества изданий. Каждая карточка книги содержит изображение обложки, название, автора, жанр и состояние экземпляра, а для ее открытия необходимо нажать кнопку «Подробнее».

Рисунок 11 – Прототип окна каталога книг

Таким образом, проектирование пользовательского интерфейса было направлено на создание интуитивно понятной и удобной рабочей среды для читателей, что было реализовано через разработку прототипов ключевых экранов, охватывающих все основные аспекты взаимодействия пользователя с веб-инструментом.

2.5 Планирование разработки и оценка бюджета

Планирование разработки и оценка бюджета проекта была выполнена при помощи Project Libre. Дата начала проекта – 8 апреля 2026 года. В результате планирования было выделено 36 задач. Задачи охватывают все этапы жизненного цикла разработки: от анализа предметной области до оформления

пояснительной записки.

Задачи проекта, а также их длительность и данные о предшествующих задачах представлены на рисунке 12.

	Название	Продолжительность	Начало	Окончание	Предшествующие	Название ресурса
1	**Начало проекта**	38 дней	**08.04.26 8:00**	**03.06.26 17:00**		
2	**Постановка задачи и анализ требований**	7 дней	**08.04.26 8:00**	**16.04.26 17:00**		
3	Анализ предметной области	2 дней	08.04.26 8:00	09.04.26 17:00		Аналитик;Компьютер
4	Обзор аналогов	2 дней	11.04.26 8:00	14.04.26 17:00	3	Аналитик;Компьютер
5	Анализ функциональных требований	2 дней	15.04.26 8:00	16.04.26 17:00	4	Аналитик;Компьютер
6	Анализ нефункциональных требований	2 дней	15.04.26 8:00	16.04.26 17:00	4	Аналитик;Компьютер
7	Завершение постановки задачи и анализа требований	0 дней	16.04.26 17:00	16.04.26 17:00	6	
8	**Проектирование программных средств**	12 дней	**17.04.26 8:00**	**06.05.26 17:00**		
9	Проектирование архитектуры приложения	3 дней	17.04.26 8:00	21.04.26 17:00		Аналитик;Компьютер;Программист;Проектировщик
10	Проектирование структуры хранения	2 дней	22.04.26 8:00	23.04.26 17:00	9	Компьютер;Проектировщик
11	Проектирование приложения	4 дней	24.04.26 8:00	29.04.26 17:00	10	Компьютер;Программист;Проектировщик
12	Проектирование пользовательского интерфейса	3 дней	30.04.26 8:00	06.05.26 17:00	11	Дизайнер;Компьютер;Программист;Проектировщик
13	Завершение проектирования программных средств	0 дней	06.05.26 17:00	06.05.26 17:00	12	
14	**Реализация программных средств**	12 дней	**05.05.26 8:00**	**21.05.26 17:00**		
15	Реализация слоя данных приложения	3 дней	05.05.26 8:00	07.05.26 17:00		Программист;Компьютер
16	Реализация слоя бизнес-логики приложения	5 дней	08.05.26 8:00	15.05.26 17:00	15	Программист;Компьютер
17	Реализация слоя представления приложения	4 дней	18.05.26 8:00	21.05.26 17:00	16	Дизайнер;Компьютер;Программист;Проектировщик
18	Завершение реализации программных средств	0 дней	21.05.26 17:00	21.05.26 17:00	17	
19	**Тестирование**	8 дней	**21.05.26 8:00**	**01.06.26 17:00**		
20	Тестирование методом черного ящика	3 дней	21.05.26 8:00	25.05.26 17:00		Компьютер;Тестировщик
21	Модульное тестирование добавления книги	3 дней	26.05.26 8:00	28.05.26 17:00	20	Тестировщик;Компьютер
22	Анализ сложности кода модуля	2 дней	29.05.26 8:00	01.06.26 17:00	21	Программист;Тестировщик;Компьютер
23	Подготовка документации	0 дней	01.06.26 17:00	01.06.26 17:00	22	Бумага;Принтер;Компьютер;Аналитик
24	Оформление и печать документации	2 дней	02.06.26 8:00	03.06.26 17:00	23	Бумага;Принтер;Компьютер;Аналитик
25	Завершение проекта	0 дней	03.06.26 17:00	03.06.26 17:00	24	

Рисунок 12 – Список задач проекта

Для расчета бюджета проекта были определены трудовые ресурсы, участвующие в разработке. Для каждого ресурса установлена почасовая ставка, что позволило рассчитать итоговую стоимость проекта. Список ресурсов, используемых при выполнении задач представлен на рисунке 13.

№	Название ресурса	Тип	Стандартная ставка, руб/ч	Начисление	Базовый календарь	Ставка сверхурочный, руб/ч	Макс. Единиц	Затраты на использование
1	Аналитик	Трудовой	252,00	Пропорционально	Стандартный	348,00	100%	0,00 ₽
2	Программист	Трудовой	300,00	Пропорционально	Стандартный	450,00	100%	0,00 ₽
3	Дизайнер	Трудовой	265,00	Пропорционально	Стандартный	415,00	100%	0,00 ₽
4	Проектировщик	Трудовой	290,00	Пропорционально	Стандартный	435,00	100%	0,00 ₽
5	Компьютер	Материальный	60 000,00 ₽	В начале				0,00 ₽
6	Бумага	Материальный	500,00 ₽	В начале				0,00 ₽
7	Принтер	Материальный	7 000,00 ₽	В начале				0,00 ₽

Рисунок 13 – Список ресурсов

На основе выделенных задач и закреплённых за ними ресурсов автоматически была построена диаграмма Ганта (рисунок 14). Она показывает не только сроки, но взаимосвязь рабочих этапов. Помогает узнать, когда и кому нужно заняться очередной задачей, что упрощает координацию действий персонала. [15].

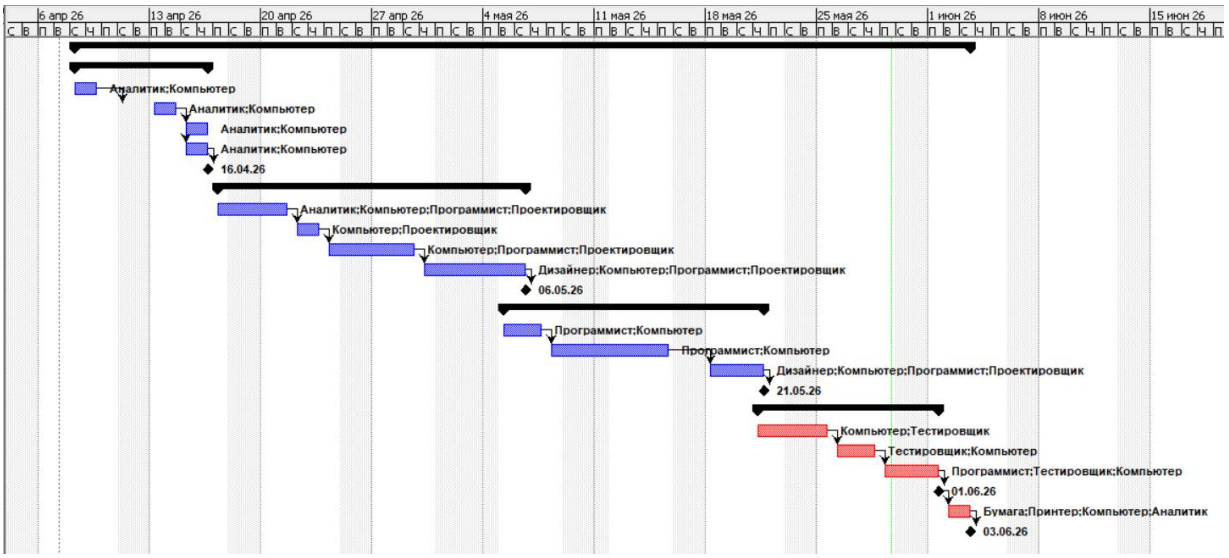

Рисунок 14 – Диаграмма Ганта для задач проекта

Таким образом, разработка проекта составит 38 полных рабочих дней с 08.04.26 по 03.06.26. Оценка бюджета проекта составляет 187 709,00 рублей.

3 Реализация программных средств веб–приложения для автоматизации процесса книгообмена

Слой данных в веб-приложении отвечает за структурированное представление, хранение и управление информацией о пользователях, книгах, обменах, сообщениях и сопутствующих метаданных. Реализация слоя данных сосредоточена на серверной стороне и включает два ключевых компонента: модели данных и DAO-классы для взаимодействия с базой данных PostgreSQL.

3.1 Реализация слоя данных

Модели данных представляют собой простые Java-классы, атрибуты которых соответствуют полям таблиц базы данных. Каждая модель содержит приватные поля, публичные геттеры и сеттеры для доступа к ним. Основными моделями приложения являются Book, User, Exchange, Message, а также вспомогательные модели Genre, Condition, Role, News, Ticket, BlockInfo.

В качестве примера взят фрагмент модели Book, который представлен в листинге 1.

Листинг 1 – Фрагмент модели Book

```
public class Book {
    private int id;
    private String title;
    private String author;
    private Integer year;
    private String language;
    private int genreId;
    private String genreName;
    private int conditionId;
    private String conditionName;
    private String isbn;
    private String description;
    private String coverPath;
    private int ownerId;
    private String ownerName;
    private String ownerEmail;
    private String status;

    public int getId() { return id; }
    public void setId(int id) { this.id = id; }
    public String getTitle() { return title; }
    public void setTitle(String title) {
```

```
        this.title = title;
        if (title != null && !title.isEmpty()) {
            this.initials = title.substring(0, 1).toUpperCase();
        }
    }
```

Некоторые поля, такие как genreName, conditionName, ownerName и ownerEmail не хранятся непосредственно в таблице books, а заполняются через JOIN-запросы при получении данных из базы. Такой подход позволяет избежать дублирования информации и сохранить нормализованную структуру базы данных. Аналогичный принцип применяется и в других моделях, например, модель Exchange содержит поля bookTitle и fromUserName, заполняемые из связанных таблиц при чтении данных.

DAO-классы инкапсулируют всю логику взаимодействия с базой данных, предоставляя типовые операции создания, чтения, обновления и удаления для каждой сущности. Каждый DAO-класс использует аннотацию @Repository и JdbcTemplate, внедряемый через @Autowired.

В проекте реализован набор DAO-классов, охватывающих все сущности приложения. Каждый из них содержит методы, специфичные для своей предметной области и обеспечивающие выполнение всех необходимых операций с соответствующей таблицей базы данных. В качестве подробного примера рассмотрен класс BookDAO, отвечающий за взаимодействие с таблицей books и реализующий полный набор операций для управления книгами в системе. Листинг 2 демонстрирует фрагмент класса BookDAO, реализующего метод сохранения новой книги в базе данных.

Листинг 2 – Фрагмент BookDAO с методом сохранения книги

```
@Repository
public class BookDAO {
    @Autowired
    private JdbcTemplate jdbcTemplate;

public int save(Book book) {
        String sql = "INSERT INTO Books (title, author, year,
language, genre_id, condition_id, " +
                     "isbn, description, cover_path, owner_id,
status) " +
                     "VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)";
```

```
        KeyHolder keyHolder = new GeneratedKeyHolder();
        jdbcTemplate.update(connection -> {
            PreparedStatement ps =
connection.prepareStatement(sql, Statement.RETURN_GENERATED_KEYS);
            ps.setString(1, book.getTitle());
            ps.setString(2, book.getAuthor());
            ps.setObject(3, book.getYear());
            ps.setString(4, book.getLanguage());
            ps.setInt(5, book.getGenreId());
            ps.setInt(6, book.getConditionId());
            ps.setString(7, book.getIsbn());
            ps.setString(8, book.getDescription());
            ps.setString(9, book.getCoverPath());
            ps.setInt(10, book.getOwnerId());
            ps.setString(11, "available");
            return ps;
        }, keyHolder);
        Map<String, Object> keys = keyHolder.getKeys();
        if (keys != null && keys.containsKey("id")) {
            return ((Number) keys.get("id")).intValue();
        }
        return 0;
    }
```

Метод save() выполняет SQL-запрос INSERT, передавая все атрибуты книги через параметризованный запрос. Каждый вопросительный знак в строке запроса является заполнителем, на место которого подставляется конкретное значение через соответствующий метод объекта PreparedStatement. Для строковых полей используется метод setString(), для целочисленных – setInt(), а для полей которые могут принимать значение NULL, применяется метод setObject(). Для получения идентификатора новой записи используется GeneratedKeyHolder, который позволяет извлечь значение первичного ключа после выполнения вставки. Статус книги по умолчанию устанавливается как «available», т.е. доступна для обмена.

Для получения списка книг с фильтрацией и пагинацией в классе BookDAO реализован метод findBooks(). Он формирует динамический SQL-запрос с JOIN-объединениями таблиц Genres, Conditions и Users, что позволяет получить полную информацию о книге, включая название жанра, состояние и имя владельца, в одном запросе. Фильтрация по жанру и поиск по названию книги или автору добавляются к запросу в зависимости от переданных параметров. Фрагмент этого метода представлен в листинге 3.

Листинг 3 – Фрагмент метода получения книг с фильтрацией

```
public List<Book> findBooks(String genre, String query, int page,
int pageSize) {
        int offset = (page - 1) * pageSize;
        StringBuilder sql = new StringBuilder(
            "SELECT b.*, g.name as genre_name, c.name as
condition_name, " +
            "u.last_name || ' ' || u.first_name as owner_name,
u.email as owner_email " +
            "FROM Books b " +
            "JOIN Genres g ON b.genre_id = g.id " +
            "JOIN Conditions c ON b.condition_id = c.id " +
            "JOIN Users u ON b.owner_id = u.id " +
            "WHERE 1=1"
        );

        List<Object> params = new ArrayList<>();

        // Фильтрация по жанру
        if (!genre.equals("all")) {
            String genreRu = convertGenreToRussian(genre);
            sql.append(" AND g.name = ?");
            params.add(genreRu);
        }

        // Поиск по названию или автору
        if (query != null && !query.trim().isEmpty()) {
            sql.append(" AND (LOWER(b.title) LIKE ? OR
LOWER(b.author) LIKE ?)");
            String searchPattern = "%" +
query.toLowerCase().trim() + "%";
            params.add(searchPattern);
            params.add(searchPattern);
        }
        sql.append(" ORDER BY b.id LIMIT ? OFFSET ?");
        params.add(pageSize);
        params.add(offset);

        return jdbcTemplate.query(sql.toString(),
params.toArray(), bookRowMapper);
    }
```

Обновление книги в базе данных выполняется методом update(), который принимает объект модели с измененными атрибутами и выполняет SQL-запрос UPDATE к таблице Books. Листинг 4 демонстрирует фрагмент этого метода.

Листинг 4 – Метод обновления книги

```
public void update(Book book) {
    String sql = "UPDATE Books SET title = ?, author = ?, year =
?, language = ?, " +
```

```
                "genre_id = ?, condition_id = ?, isbn = ?,
description = ?, " +
                "cover_path = ? WHERE id = ?";
    jdbcTemplate.update(sql,
        book.getTitle(),
        book.getAuthor(),
        book.getYear(),
        book.getLanguage(),
        book.getGenreId(),
        book.getConditionId(),
        book.getIsbn(),
        book.getDescription(),
        book.getCoverPath(),
        book.getId());
    System.out.println("Book updated: id=" + book.getId() + ",
title=" + book.getTitle());
}
```

Метод обновляет все редактируемые поля книги: название, автор, год издания, язык, жанр, состояние, ISBN, описание и путь к обложке. Идентификатор книги используется в условии WHERE для определения обновляемой записи. Для обновления только статуса книги используется отдельный метод updateBookStatus(), который выполняет запрос, изменяющий только поле status. Такое разделение позволяет оптимизировать выполнение частых операций изменения статуса, которые не требуют передачи полного набора данных о книге.

Удаление книги из базы данных реализовано методом delete(), который выполняет каскадное удаление связанных данных. Перед удалением книги из таблицы Books метод последовательно удаляет все связанные записи из таблицы Wishlist (список желаний), затем из таблицы Messages (сообщения, связанные с книгой). Такой порядок обеспечивает соблюдение ссылочной целостности данных и предотвращает ошибки внешних ключей, которые могли бы возникнуть при попытке удаления записи, на которую ссылаются другие таблицы. Блок try-catch обеспечивает обработку возможных исключений при выполнении операций с базой данных. Фрагмент данного метода представлен в листинге 5.

Листинг 5 – Метод удаления книги

```
    public int delete(int bookId) {
        try {
```

```
            String deleteWishlistSql = "DELETE FROM Wishlist WHERE
book_id = ?";
            jdbcTemplate.update(deleteWishlistSql, bookId);

            String deleteMessagesSql = "DELETE FROM Messages WHERE
book_id = ?";
            jdbcTemplate.update(deleteMessagesSql, bookId);

            String sql = "DELETE FROM Books WHERE id = ?";
            int result = jdbcTemplate.update(sql, bookId);
            System.out.println("delete: bookId=" + bookId + ",
result=" + result);
            return result;
        } catch (Exception e) {
            e.printStackTrace();
            return 0;
        }
    }
```

Помимо базовых CRUD-операций, BookDAO содержит ряд специализированных методов. Метод findBooksByUser() получает список книг, принадлежащих конкретному пользователю, фильтруя записи по полю owner_id. Метод findWishlistByUser() получает список желаемых книг пользователя, используя JOIN с таблицей Wishlist. Методы findReleasedBooksByUser() и findReceivedBooksByUser() формируют историю обменов, определяя, какие книги пользователь отдал (является оригинальным владельцем, но текущий владелец изменился) и какие получил (является текущим владельцем, но не является оригинальным).

Метод transferBookOwnership() реализует передачу права собственности на книгу при завершении обмена. Он получает текущего владельца книги, при необходимости фиксирует оригинального владельца (если это первый обмен) через функцию COALESCE(), после чего изменяет владельца и возвращает статус книги в «available».

Реализация слоя данных обеспечивает как управление информацией через полный набор CRUD-операций. Данные хранятся и передаются между компонентами в четко определенных структурах моделей, с использованием DAO-классов, инкапсулирующих SQL-логику и применяющих параметризованные запросы для защиты от SQL-инъекций, что обеспечивает безопасность и возможность обмена данными между всеми слоями приложения.

3.2 Реализация слоя бизнес-логики

Слой бизнес-логики в разрабатываемом веб-приложении отвечает за обработку пользовательских действий, управление бизнес-правилами, валидацию данных и координацию взаимодействия между слоем представления и слоем данных. Основная бизнес–логика сосредоточена в сервисных классах, которые выступают в роли связующего звена между контроллерами и DAO-классами, а также в самих контроллерах, обрабатывающих HTTP-запросы от клиентской части.

Каждый контроллер реализован как класс с аннотацией @Controller и содержит методы с @GetMapping или @PostMapping, которые связывают URL-адреса с конкретными обработчиками. Контроллеры получают параметры запроса, вызывают соответствующие методы сервисов для выполнения бизнес-логики и возвращают имя JSP-страницы для отображения результата. Листинг 6 демонстрирует фрагмент BookController, отвечающего за обработку запросов, связанных с книгами.

Листинг 6 – Фрагмент BookController с методом добавления книги

```
@Controller
public class BookController {
    private final BookService bookService;

@PostMapping("/book/register.htm")
    public String processRegisterBook(
            @RequestParam("title") String title,
            @RequestParam("author") String author,
            @RequestParam("genreId") int genreId,
            @RequestParam(value = "year", required = false)
Integer year,
            @RequestParam(value = "language", required = false)
String language,
            @RequestParam("conditionId") int conditionId,
            @RequestParam(value = "isbn", required = false) String
isbn,
            @RequestParam(value = "description", required = false)
String description,
            @RequestParam(value = "cover", required = false)
MultipartFile cover,
            HttpSession session) {

        Integer userId = (Integer) session.getAttribute("userId");
```

```
        if (userId == null) return "redirect:/login.htm";

        Book book = new Book();
        book.setTitle(title);
        book.setAuthor(author);
        book.setYear(year);
        book.setLanguage(language);
        book.setGenreId(genreId);
        book.setConditionId(conditionId);
        book.setIsbn(isbn);
        book.setDescription(description);
        book.setOwnerId(userId);
        book.setStatus("available");

        if (cover != null && !cover.isEmpty()) {
            try {
                String appPath =
session.getServletContext().getRealPath("/");
                String coverPath =
bookService.saveBookCover(cover, userId, appPath);
                book.setCoverPath(coverPath);
            } catch (Exception e) {
                book.setCoverPath("/resources/images/default-
book.jpg");
            }
        } else {
            book.setCoverPath("/resources/images/default-
book.jpg");
        }

        bookService.saveBook(book);
        return "redirect:/catalog.htm";
    }
```

При обработке GET-запроса на отображение формы регистрации книги контроллер сначала проверяет авторизацию пользователя через объект HttpSession. Если идентификатор пользователя отсутствует в сессии, выполняется перенаправление на страницу входа. Затем вызывается метод isUserBlocked(), который проверяет статус учетной записи пользователя. Если аккаунт заблокирован, в модель добавляется сообщение об ошибке, и пользователь перенаправляется в личный кабинет. В случае успешной проверки через сервис BookService загружаются списки жанров и состояний книг, которые передаются в модель для отображения в выпадающих списках формы.

При обработке POST-запроса на сохранение книги контроллер создает новый объект модели Book и заполняет его атрибуты данными из параметров

запроса. Для загружаемого файла обложки выполняется проверка на наличие. Если файл предоставлен, он сохраняется в файловую систему сервера через метод saveBookCover() сервиса BookService, а путь к изображению устанавливается в объекте книги. В противном случае используется изображение по умолчанию.

Сервисы являются центральным звеном бизнес-логики, инкапсулируя правила предметной области и координируя взаимодействие между контроллерами и DAO-классами. Каждый сервис аннотирован @Service и получает необходимые DAO-объекты через внедрение зависимостей в конструкторе. Наиболее показательным примером реализации бизнес–правил является логика создания запроса на обмен книгой, реализованная в ExchangeService. Фрагмент данного сервиса представлен в листинге 7.

Листинг 7 – Фрагмент ExchangeService с методом создания запроса на обмен

```
@Service
public class ExchangeService {

    private final ExchangeDAO exchangeDAO;
    private final BookDAO bookDAO;

    public ExchangeService(ExchangeDAO exchangeDAO, BookDAO
bookDAO) {
        this.exchangeDAO = exchangeDAO;
        this.bookDAO = bookDAO;
    }
public int createExchangeRequest(int bookId, int fromUserId) {
        Book book = bookDAO.findById(bookId);
        if (book == null) {
            throw new IllegalArgumentException("Книга не
найдена");
        }
        if (book.getOwnerId() == fromUserId) {
            throw new IllegalStateException("Нельзя запросить
обмен на свою книгу");
        }
        if (!"available".equals(book.getStatus())) {
            throw new IllegalStateException("Книга уже в обмене
или забронирована");
        }
        if (exchangeDAO.hasActiveRequest(bookId, fromUserId)) {
            throw new IllegalStateException("Вы уже отправили
запрос на эту книгу");
        }
```

```
        Date exchangeDate = new Date();
        return exchangeDAO.createExchangeRequest(bookId,
fromUserId, book.getOwnerId(), exchangeDate);
    }
```

Метод createExchangeRequest() выполняет последовательную проверку бизнес–правил перед созданием запроса на обмен. Сначала проверяется существование книги в базе данных. Если книга не найдена, выбрасывается исключение IllegalArgumentException. Затем проверяется, что пользователь не пытается запросить собственную книгу – сравнение идентификатора владельца книги с идентификатором запрашивающего пользователя. Далее выполняется проверка статуса книги, так как обмен возможен только для книг со статусом «available». Затем через метод hasActiveRequest() объекта ExchangeDAO проверяется отсутствие уже существующего активного запроса от данного пользователя на данную книгу. Если все проверки пройдены успешно, создается новый запрос с текущей датой и статусом «pending», после чего идентификатор созданной записи возвращается контроллеру.

Остальные сервисы приложения реализуют бизнес-логику для своих предметных областей по аналогичному принципу. К примеру, MessageService при отправке сообщения автоматически устанавливает флаг isRead в значение false, фиксирует временную метку создания и связывает сообщение с конкретной книгой и участниками обмена. Метод получения диалогов формирует список бесед, сгруппированных по книге и собеседнику, с отображением последнего сообщения и подсчетом количества непрочитанных. Сервис UserService при регистрации выполняет хеширование пароля с использованием алгоритма BCrypt, проверяет уникальность email в базе данных и назначает новому пользователю роль «читатель» по умолчанию. При аутентификации введенный пароль сравнивается с хешем, хранящимся в базе, и в случае успеха в сессию сохраняются идентификатор и роль пользователя. AdminService и ModeratorService предоставляют расширенные функции для управления новостной лентой, книгами, пользователями, обменами и обработки обращений в службу поддержки.

Функция удаления книги в контроллере BookController также демонстрирует взаимодействие слоев. Контроллер получает идентификатор удаляемой книги и текущего пользователя из сессии, после чего вызывает метод getBookById() сервиса BookService для получения объекта книги. Сервис проверяет, что текущий пользователь является владельцем книги, и только при выполнении этого условия вызывает метод delete() объекта BookDAO, который выполняет каскадное удаление книги и всех связанных с ней записей.

Реализованная архитектура слоя бизнес-логики обеспечивает четкое разделение ответственности между компонентами системы. Контроллеры принимают и маршрутизируют HTTP-запросы, делегируя выполнение бизнес-операций специализированным сервисам. Сервисы, в свою очередь, реализуют правила предметной области – от валидации входных данных и проверки прав доступа до выполнения сложных сценариев. Благодаря использованию внедрения зависимостей через конструкторы каждый компонент может быть легко заменен или протестирован изолированно от остальных. В приложении Б приведены листинги основных модулей разрабатываемого приложения, а полный исходный код проекта доступен в репозитории GitHab [16].

3.3 Разработка слоя представления

При разработке графического интерфейса веб-приложения использовались технологии Jakarta Server Pages (JSP) и HTML. Такой подход позволил создать серверно-рендеринговые страницы, формируемые на стороне сервера и передаваемые браузеру в виде готового HTML-документа.

Серверный рендеринг (Server-Side Rendering, или SSR) – метод отрисовки веб-страницы, при котором HTML-код генерируется на сервере до отправки в браузер [17].

Для стилизации интерфейса применялся CSS, а для реализации динамического поведения страниц – теги JSTL.

При первом входе в приложение пользователь попадает на главный экран, представленный на рисунке 15. Данный экран встречает посетителя названием

платформы и слоганом «Подарите книгам новую жизнь». В верхней части страницы расположена навигационная панель, содержимое которой динамически изменяется в зависимости от статуса авторизации пользователя. Для реализации этого поведения используется конструкция <c:choose> библиотеки JSTL, которая проверяет наличие идентификатора пользователя в сессии. Если атрибут userId отсутствует, отображаются кнопки «Войти» и «Зарегистрироваться», обеспечивающие быстрый доступ к формам аутентификации. Для авторизованных пользователей навигационная панель расширяется ссылками «Новости», «Личный кабинет», «Сообщения» и «Выйти». При этом для пользователей с ролями «ADMIN» и «MODERATOR» ссылка на сообщения скрывается, так как административный персонал не участвует в обмене книгами.

Центральным элементом главной страницы является приветственная секция с кнопками: «Добавить книгу» и «Найти книгу». Кнопка «Добавить книгу» ведет на страницу регистрации книги и для администраторов с модераторами отображается в неактивном состоянии, что сигнализирует о недоступности данной функции для этих ролей. Кнопка «Найти книгу» доступна всем категориям пользователей и ведет на страницу каталога. Под приветственной секцией расположены информационные блоки, знакомящие посетителей с концепцией буккроссинга и принципами работы платформы.

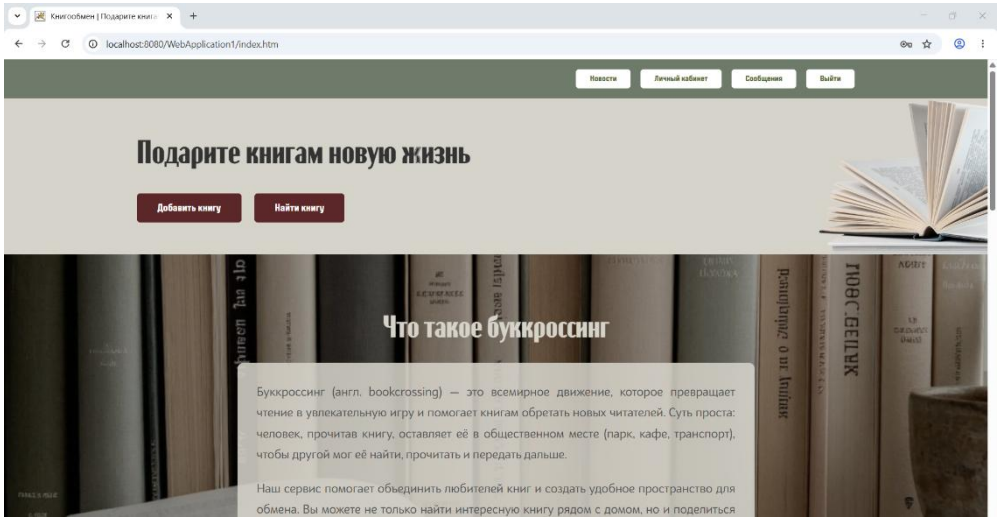

Рисунок 15 – Интерфейс главной страницы приложения

После авторизации пользователь получает доступ к странице каталога книг, представленной на рисунке 16. Каталог является основным инструментом поиска книг для обмена. На странице реализована фильтрация по жанрам с помощью выпадающего списка, загружаемого из базы данных через сервис BookService. Поиск по названию книги или автору выполняется через текстовое поле, результаты фильтруются на серверной стороне с помощью SQL-запроса с оператором LIKE. Для навигации по большому количеству записей реализована пагинация с расчетом общего количества страниц на основе числа найденных книг. Каждая карточка книги содержит изображение обложки, название, автора, жанр, состояние экземпляра и кнопку «Подробнее» для перехода к детальной информации.

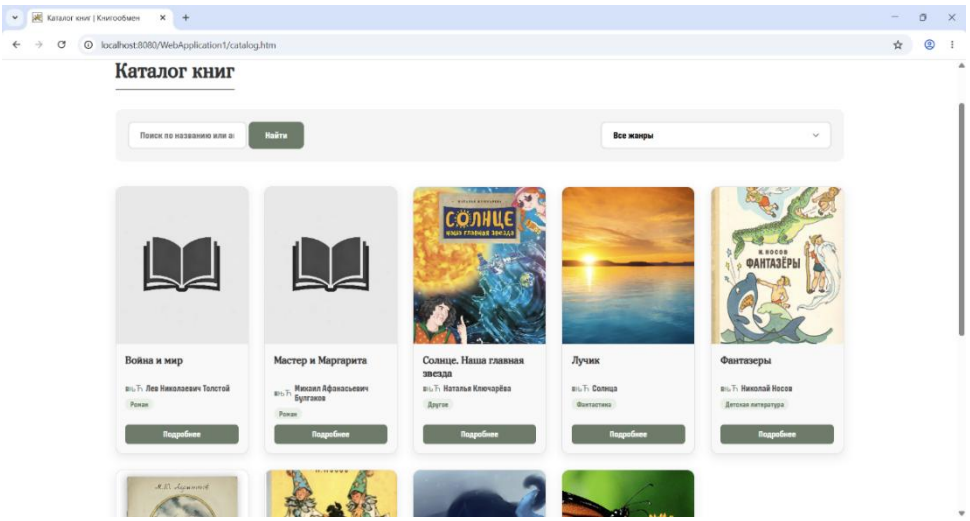

Рисунок 16 – Интерфейс страницы каталога книг

Переход по кнопке «Подробнее» открывает страницу деталей книги, представленную на рисунке 17. На ней отображается изображение обложки, полное название, автор, год издания, язык, ISBN, жанр, состояние, а также информация о владельце – имя, город и рейтинг. Для авторизованных пользователей, не являющихся владельцами данной книги, на странице доступны кнопки «Предложить обмен» и «Добавить в список желаний». Кнопка «Предложить обмен» инициирует создание запроса на обмен через ExchangeController, который вызывает соответствующий метод ExchangeService.

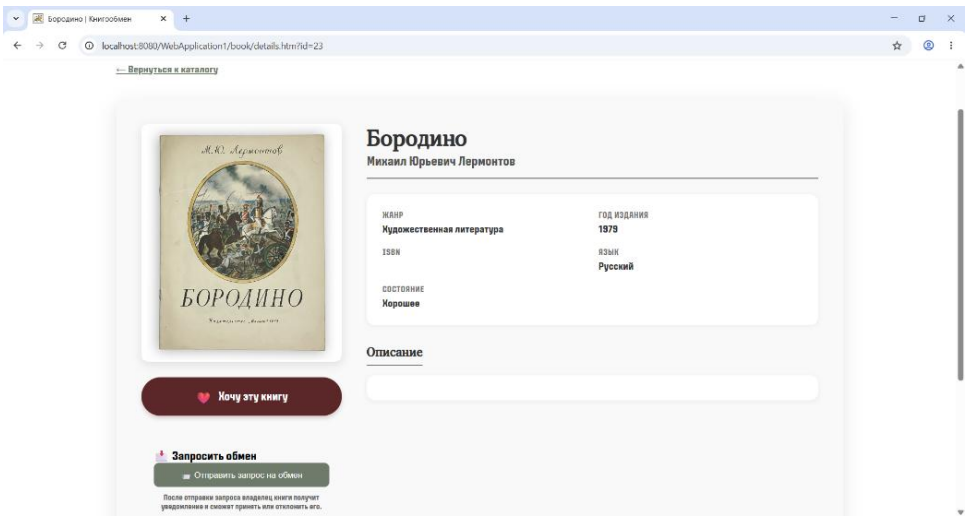

Рисунок 17 – Интерфейс страницы деталей книги

Личный кабинет пользователя, показанный на рисунке 18, является центральным разделом для управления своими книгами и отслеживания истории обменов. Страница содержит несколько вкладок, реализованных с использованием параметра tab в URL. Вкладка «Мои книги» отображает список изданий, добавленных пользователем для обмена, с возможностью редактирования и удаления каждой записи. Вкладка «Желаемые» показывает книги, отмеченные пользователем как желаемые к получению. Вкладка «Отпущенные» содержит книги, которые пользователь передал другим читателям, а «Полученные» – книги, полученные в результате обмена. Для каждой книги во всех вкладках отображается изображение обложки, название, автор, жанр и текущий статус.

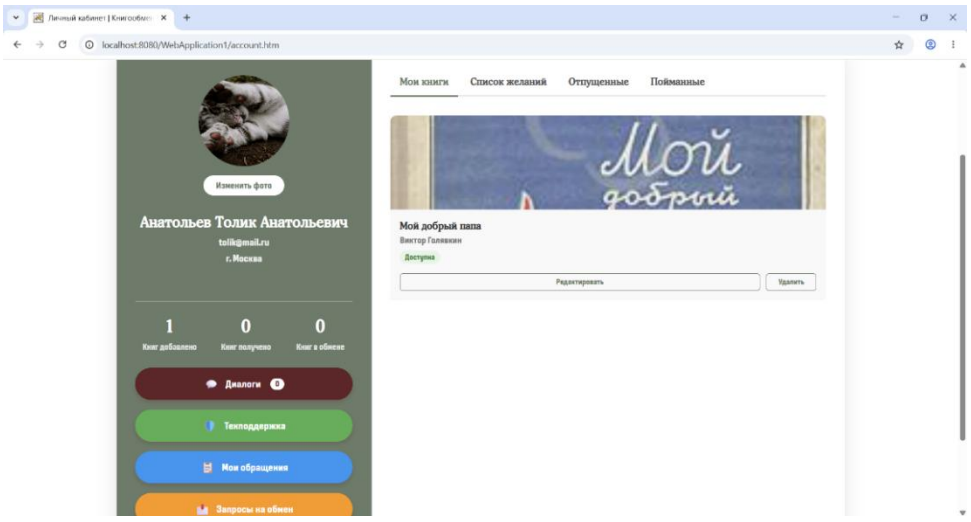

Рисунок 18 – Интерфейс личного кабинета пользователя

Страница запросов на обмен, обеспечивает управление процессом обмена между читателями. Она содержит две вкладки: «Входящие» – запросы от других пользователей на книги текущего владельца, и «Исходящие» – запросы, отправленные текущим пользователем. Для каждого запроса отображается информация о книге, пользователе-инициаторе, дате создания и текущем статусе. Статусы обмена визуально выделяются цветом: оранжевый для ожидающих запросов, зеленый для принятых, красный для отклоненных и синий для завершенных. Владелец книги может принять запрос, отклонить его или завершить обмен после физической передачи книги. При принятии запроса система автоматически изменяет статус книги на «exchanged» и удаляет ее из списков желаний других пользователей. При завершении обмена происходит передача права собственности на книгу новому владельцу. Страница запросов представлена на рисунке 19.

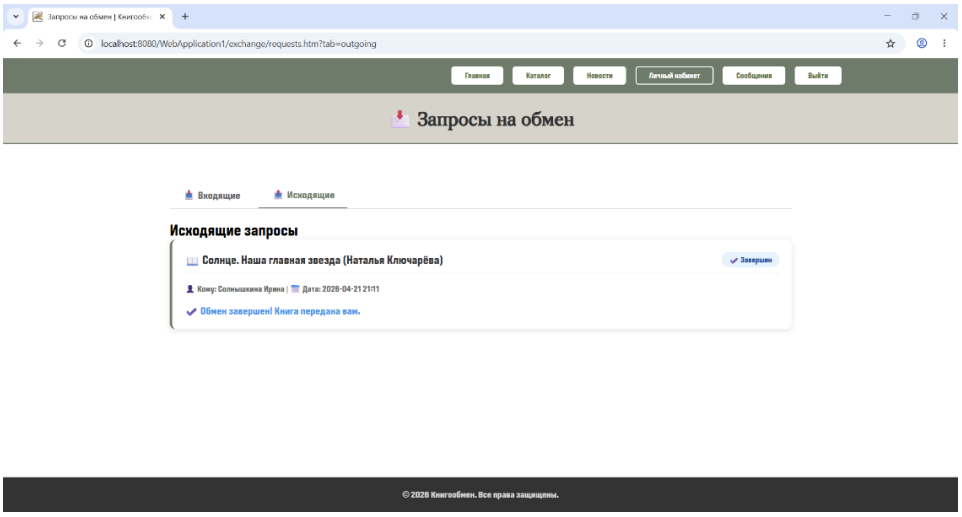

Рисунок 19 – Интерфейс страницы запросов на обмен

Для обеспечения коммуникации между участниками обмена разработана страница сообщений (рисунок 20), позволяющая вести переписку в рамках конкретной книги. Сообщения группируются по книгам и собеседникам, для каждого диалога отображается последнее сообщение и количество непрочитанных. При открытии переписки все сообщения отмечаются как прочитанные.

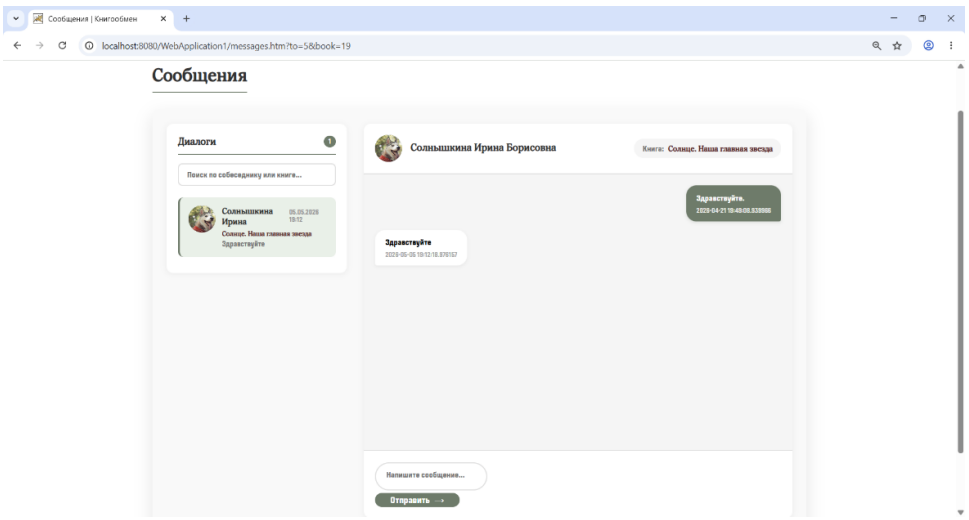

Рисунок 20 – Интерфейс страницы сообщений

Новостная лента приложения информирует пользователей о событиях платформы: новых функциях, правилах сообщества и другой важной информации. Новости сгруппированы по категориям и отсортированы по дате публикации. Для администраторов и модераторов страница новостей дополнительно содержит элементы управления для создания, редактирования и удаления новостных публикаций.

Для администраторов и модераторов разработаны панели управления с расширенными возможностями. Административная панель предоставляет доступ к управлению пользователями, книгами и обменами с фильтрацией по статусам, выполнять CRUD-операции над новостной лентой и отвечать на обращения в техническую поддержку.

Панель модератора позволяет блокировать/разблокировать пользователей на определенный срок, удалять обложки книг, если нарушают цензуру, выполнять CRUD-операции над новостной лентой и обрабатывать обращения в службу технической поддержки.

Таким образом, слой представления web-приложения предоставляет пользователю полный набор визуальных средств для эффективного поиска книг, организации обмена, управления личной библиотекой, коммуникации с другими читателями и отслеживания истории своих обменов. Использование JSP и JSTL позволило создать динамический интерфейс с серверным рендерингом,

обеспечивающий доступ ко всем функциям системы через браузер без необходимости установки дополнительного программного обеспечения.

Была успешно осуществлена практическая реализация веб-приложения для автоматизации процесса книгообмена. Реализована многослойная архитектура приложения: слой данных обеспечил управление информацией о пользователях, книгах, обменах и сообщениях через модели и DAO-классы с использованием параметризованных запросов для защиты от SQL-инъекций; слой бизнес-логики инкапсулировал ключевые бизнес-правила в контроллерах и сервисных классах, включая валидацию запросов на обмен, проверку прав доступа и передачу права собственности; слой представления предоставил интуитивно понятный пользовательский интерфейс с динамической навигацией и серверным рендерингом страниц. Взаимодействие компонентов построено на внедрении зависимостей через конструкторы, что обеспечивает слабую связанность и высокую связность системы. Приложение Б содержит исходный код программного средства. Итогом работы стал функциональный прототип веб–приложения, соответствующий поставленным требованиям и готовый к дальнейшему тестированию и развитию.

4 Тестирование программного модуля «Добавление книги»

4.1 Тестирование методом черного ящика

Тестирование методом черного ящика определяется как методика тестирования, при которой функциональность тестируемого приложения, тестируется без учета внутренней структуры кода, деталей реализации и знания внутренних путей программного обеспечения. Этот тип тестирования полностью основан на требованиях и спецификациях программного обеспечения. В тестирование методом черного ящика фокусируются на входах и выходах программной системы, не заботясь о внутренних знаниях программ. Вышеупомянутый черного ящик может быть любой программной системой, которую нужно протестировать. В тестировании методом черного ящика можно протестировать эти приложения, просто сосредоточившись на входах и выходах, не зная их внутренней реализации кода [18].

4.1.1 Классы эквивалентности

При добавлении книги, требуется указать полное ее название, содержащее 2 – 200 символов: русские буквы, латинские буквы, цифры, дефис, знаки препинания. Оно должно начинаться с буквы или цифры. В таблице 9 приведены классы эквивалентности для названия книги.

Таблица 9 – Классы эквивалентности для краткого имени

	Правильные КЭ	Неправильные КЭ
Длина краткого имени реакции	2 – 200 (1)	0 – 1 (2) 201 – ∞ (3)
Структура	Начинается с буквы или цифры (4)	Начинается не с буквы или цифры (5)
Содержание	Русские буквы (6) Латинские буквы (7) Цифры (8) Дефис (9) Знаки препинания (10)	Остальное (11)

Графическое представление классов эквивалентности изображено на рисунках 21 и 22. Тест–кейсы для поля «Название» представлено в таблице 10.

Рисунок 21 – Класс эквивалентности (длина названия книги)

Рисунок 22 – Класс эквивалентности (содержание)

Таблица 10 – Тест-кейсы для поля «Название»

Номер теста	Классы Эквивалентности	Входные данные	Выходные данные
1	1,4,6,7,8,9,10	Harry Potter and Sorcerer’s Stone	Успешно
2	2,4,6	H	Ошибка: Меньше 2–х символов
3	3,4,6,7,8,9,10	Shfkrjekpew…,kc…	Ошибка: Больше 200 символов
4	1,5,6	–Война и мир	Ошибка: Начинается не с буквы и не с цифры
5	1,4,6,14	Мастер и Маргарита	Ошибка: Недопустимые символы

Таким образом, были построены классы эквивалентности для тестирования краткого имени.

4.1.2 Функциональная диаграмма и таблица решений

Чтобы добавить книгу, пользователь должен иметь возможность ввода соответствующих характеристик, а именно название книги, имя автора и выбор жанра. Для этого данные должна иметь корректную длину и допустимые символы. Кроме того, обязательным условием для успешного добавления книги является выбор жанра. Таким образом была сформирована таблица 11.

Таблица 11 – Набор причин и следствий

Причина	Следствие
П1	Длина названия книги корректна (2 – 200)
П2	Название состоит из допустимых символов
П3	Длина названия автора корректна (3 – 50)
П4	Имя автора состоит из допустимых символов
П5	Жанр выбран
С1	Сообщение: «Длина названия книги некорректна»
С2	Сообщение: «Название состоит из недопустимых символов»
С3	Сообщение: «Длина названия автора некорректна»
С4	Сообщение: «Имя автора состоит из недопустимых символов»
С5	Сообщение: «Жанр не выбран»
С6	Сообщение: «Книга добавлена»

В качестве графического представление причин и следствий приведена функциональная диаграмма на рисунке 23.

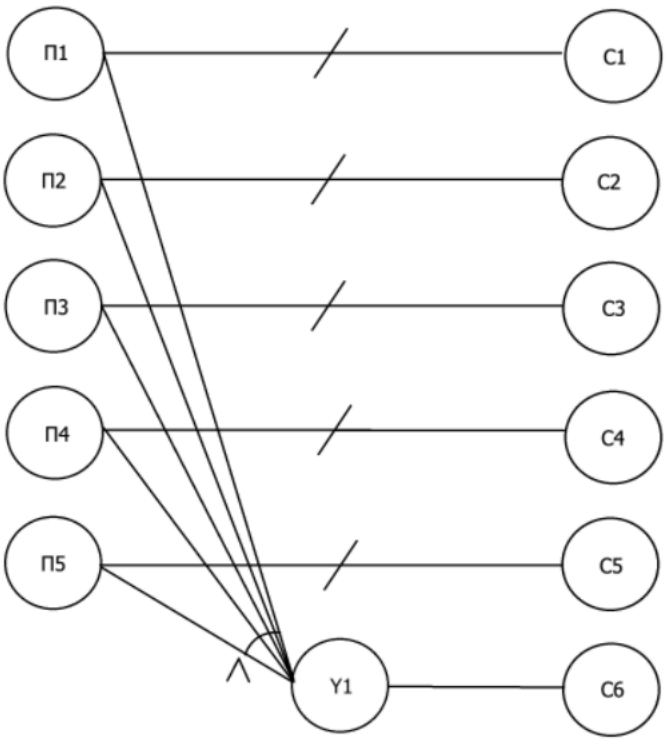

Рисунок 23 – Функциональная диаграмма

Таблицы с представлением допустимых ограничений в случае полного перебора всех комбинаций определяют 10368 тестовых сценария. С целью сокращения количества вхождений во множество допустимых тестов

используется комбинаторный метод тестирования программного обеспечения, называемый попарным тестированием, который для каждой пары входных параметров системы проверяет все возможные дискретные комбинации этих параметров. Данная техника позволяет уменьшить количество тестовых сценариев с минимальным вмешательством в качество проводимого тестирования.

Для проработки каждого тестируемого варианта, была применена техника попарного тестирования.

Попарное тестирование – это метод тестирования программного обеспечения, при котором используются перестановки и комбинации для проверки всех возможных пар входных параметров. Этот метод позволяет сократить количество тестовых примеров, при этом обеспечивая эффективное покрытие взаимодействий между входными данными [19].

Результат применения техники попарного тестирования оставил из 10368 тестовых сценариев лишь 29. Тестовые сценарии после применения техники попарного тестирования представлены на таблице 12.

Таблица 12 – Тестовые сценарии после применения техники попарного тестирования

	Название книги (длина)	**Название книги (содержание)**	**Автор (длина)**	**Автор (содержание)**	**Автор (структура)**	**Количество авторов**	**Жанр**
1	0 – 2	Цифры	> 50	Знаки препинания	Фамилия	2 – 5	true
2	0 – 2	Дефис	0 – 3	Дефис	Фамилия – Фамилия	> 5	false
3	2 – 200	Дефис	3 – 50	Цифры	Фамилия Имя Отчество	1	true
4	2 – 200	Знаки препинания	> 50	Кириллица	Фамилия И.О.	2 – 5	false
5	2 – 200	Остальное	0 – 3	Латинские	Фамилия	> 5	true
6	2 – 200	Кириллица	3 – 50	Знаки препинания	Фамилия – Фамилия	0	false
7	> 200	Цифры	3 – 50	Кириллица	Фамилия	0	true
8	> 200	Дефис	> 50	Латинские	Фамилия – Фамилия	1	false
9	> 200	Знаки препинания	0 – 3	Знаки препинания	Фамилия Имя Отчество	0	true
10	> 200	Остальное	3 – 50	Дефис	Фамилия И.О.	1	false

Продолжение таблицы 12

	Название книги (длина)	**Название книги (содержание)**	**Автор (длина)**	**Автор (содержание)**	**Автор (структура)**	**Количество авторов**	**Жанр**
11	> 200	Кириллица	> 50	Остальное	Фамилия Имя Отчество	2 – 5	true
12	> 200	Латинские	0 – 3	Цифры	Фамилия И.О.	> 5	false
13	0 – 2	Дефис	0 – 3	Дефис	Фамилия Имя Отчество	> 5	true
14	0 – 2	Знаки препинания	3 – 50	Остальное	Фамилия И.О.	0	false
15	0 – 2	Остальное	> 50	Цифры	Фамилия	1	true
16	0 – 2	Кириллица	0 – 3	Кириллица	Фамилия – Фамилия	0	false
17	0 – 2	Латинские	3 – 50	Латинские	Фамилия Имя Отчество	1	true
18	0 – 2	Цифры	> 50	Знаки препинания	Фамилия И.О.	2 – 5	false
19	2 – 200	Знаки препинания	> 50	Кириллица	Фамилия Имя Отчество	2 – 5	true
20	2 – 200	Остальное	0 – 3	Латинские	Фамилия И.О.	> 5	false
21	2 – 200	Латинские	> 50	Дефис	Фамилия И.О.	1	false
22	2 – 200	Цифры	0 – 3	Остальное	Фамилия	0	true
23	2 – 200	Дефис	3 – 50	Цифры	Фамилия – Фамилия	1	false
24	> 200	Остальное	3 – 50	Дефис	Фамилия	1	true
25	> 200	Кириллица	> 50	Остальное	Фамилия – Фамилия	2 – 5	false
26	> 200	Латинские	0 – 3	Цифры	Фамилия Имя Отчество	> 5	true
27	> 200	Цифры	3 – 50	Кириллица	Фамилия И.О.	0	false
28	> 200	Дефис	> 50	Латинские	Фамилия Имя Отчество	1	true
29	> 200	Знаки препинания	0 – 3	Знаки препинания	Фамилия И.О.	0	false

Результат выполнения каждого тестового сценария, определенного на таблице 13, привел к следствию с сообщением «Книга добавлена».

Таким образом, можно сделать вывод о том, что результат выполнения технического задания произведен успешно и разработанное программное обеспечение работает корректно.

4.2 Модульное тестирование добавление книг

Для проверки корректности работы разработанного веб-приложения были созданы Unit-тесты.

Unit-тестирование, или модульное – это разновидность тестирования в

программной разработке, которое заключается в проверке работоспособности отдельных функциональных модулей, процессов или частей кода приложения [20].

В качестве критерия проверки была использована нагрузка на сайт при выполнении операции добавления записей в таблицу Books базы данных PostgreSQL. Тестирование выполнялось для трех объемов данных: 100, 500 и 1000 книг. Замерялось общее время выполнения транзакции, включающей вставку всех записей для указанного объема. Полученные результаты сведены в таблицу 13.

Таблица 13 – Сравнительный анализ добавления тестовых данных

Кол–во данных	Время выполнения	Среднее время на 1 запись
100 книг	15 мс	0.15 мс
500 книг	30 мс	0.06 мс
1000 книг	47 мс	0.047 мс

Приложение В содержит пример исходного кода тестов, результаты проверки которых представлены на рисунке 24.

```
ЗАМЕЧАНИЕ:  Найдено пользователей: 10, от ID 1 до 10
ЗАМЕЧАНИЕ:  === Вставка 100 книг ===
ЗАМЕЧАНИЕ:  Время выполнения: 15 мс
ЗАМЕЧАНИЕ:  Количество записей: 100
ЗАМЕЧАНИЕ:  ----------------------------------------

Successfully run. Total query runtime: 56 msec.
10 rows affected.
ЗАМЕЧАНИЕ:  === Вставка 500 книг ===
ЗАМЕЧАНИЕ:  Время выполнения: 30 мс
ЗАМЕЧАНИЕ:  Количество записей: 500
ЗАМЕЧАНИЕ:  ----------------------------------------

Successfully run. Total query runtime: 74 msec.
10 rows affected.
ЗАМЕЧАНИЕ:  === Вставка 1000 книг ===
ЗАМЕЧАНИЕ:  Время выполнения: 47 мс
ЗАМЕЧАНИЕ:  Количество записей: 1000
ЗАМЕЧАНИЕ:  ----------------------------------------

Successfully run. Total query runtime: 126 msec.
10 rows affected.
```

Рисунок 24 – Результаты добавления 100, 500 и 1000 книг

Полученные данные показали ожидаемую тенденцию к увеличению времени выполнения при росте объема вставляемых данных. Среднее время на 1 запись показало, что при вставке 100 книг каждая запись требовала в среднем 0,15 мс, то при вставке 1000 книг этот показатель улучшился до 0,047 мс. Это означает, что ресурсы СУБД оптимально использованы.

4.3 Анализ сложности кода модуля

Для измерения характеристик и критериев качества используют метрики. Метрика качества программ – это система измерений качества программ. Эти измерения могут проводиться на уровне критериев качества программ или на уровне отдельных характеристик качества. В первом случае система измерений позволяет непосредственно сравнивать программы по качеству, при этом сами измерения не могут быть проведены без субъективных оценок свойств программ. Во втором случае измерения характеристик можно выполнить объективно и достоверно, но оценка качества ПО в целом будет связана с субъективной интерпретацией получаемых оценок. В листинге 8 приведен код метода контроллера, отвечающего за обработку POST-запроса для добавления новой книги в каталог, на основании которого будет производиться расчет метрик.

Листинг 8 – Метод добавления книги

```
@PostMapping("/book/register.htm")
public String processRegisterBook(
        @RequestParam("title") String title,
        @RequestParam("author") String author,
        @RequestParam("genreId") int genreId,
        @RequestParam(value = "year", required = false) Integer
year,
        @RequestParam(value = "language", required = false) String
language,
        @RequestParam("conditionId") int conditionId,
        @RequestParam(value = "isbn", required = false) String
isbn,
        @RequestParam(value = "description", required = false)
String description,
        @RequestParam(value = "cover", required = false)
MultipartFile cover,
        HttpSession session) {

    Integer userId = (Integer) session.getAttribute("userId");
```

```
    if (userId == null) return "redirect:/login.htm";

    Book book = new Book();
    book.setTitle(title);
    book.setAuthor(author);
    book.setYear(year);
    book.setLanguage(language);
    book.setGenreId(genreId);
    book.setConditionId(conditionId);
    book.setIsbn(isbn);
    book.setDescription(description);
    book.setOwnerId(userId);
    book.setStatus("available");

    if (cover != null && !cover.isEmpty()) {
        try {
            String appPath =
session.getServletContext().getRealPath("/");
            String coverPath = bookService.saveBookCover(cover,
userId, appPath);
            book.setCoverPath(coverPath);
        } catch (Exception e) {
            book.setCoverPath("/resources/images/default-
book.jpg");
        }
    } else {
        book.setCoverPath("/resources/images/default-book.jpg");
    }

    bookService.saveBook(book);
    return "redirect:/catalog.htm";
}
```

Расчет метрики SLOC на примере кода метода из листинга 8 представлен в таблице 14.

Таблица 14 – Расчет метрики SLOC для кода из листинга 8

Метрика	**Результат**
Физические строки кода	40
Логические строки кода	27
Исходные строки кода	40
Пустые строки	2

Результат расчета метрики показывает, что код функции, представленный на листинге 8, очень компактна и не имеет достаточной сложности, его просто анализировать и использовать.

Традиционной характеристикой размера программ является количество строк исходного текста. Под строкой понимается любой оператор программы, поскольку именно оператор, а не отдельно взятая строка является тем

интеллектуальным «квантом» программы, опираясь на который можно строить метрики сложности ее создания.

Оценка размера программы недостаточна для принятия решения о ее сложности, но вполне применима для классификации программ, существенно различающихся объемами. При уменьшении различий в объеме программ на первый план выдвигаются оценки других факторов, оказывающих влияние на сложность. Таким образом, оценка размера программы есть оценка по номинальной шкале, на основе которой определяются только категории программ без уточнения оценки для каждой категории. К группе оценок размера программ можно отнести метрику Холстеда.

Основу метрики Холстеда составляют четыре измеряемых характеристики программы:

n1 – число уникальных операторов программы, включая символы разделители, имена процедур и знаки операций (словарь операторов);

n2 – число уникальных операндов программы (словарь операндов);

N1 – общее число операторов в программе;

N2 – общее число операндов в программе.

В таблице 15 приведены результаты расчета количества операторов и операндов для тестируемого блока кода.

Таблица 15 – Операторы и операнды

№	Операнды		Операторы	
1	title	2	@PostMapping	1
2	author	2	@RequestParam	8
3	genreid	2	public	1
4	year	2	String	10
5	language	2	int	2
6	conditionId	2	Integer	3
7	isbn	2	MultipartFile	1
8	description	2	HttpSession	1
9	cover	4	class	1
10	session	3	=	13
11	userid	4	return	3
12	book	14	new	1
13	"redirect:/login.htm"	1	.	27
14	"available"	1	==	1

Продолжение таблицы 15

№	Операнды		Операторы	
15	"/resources/images/default–book.jpg"	2	&&	1
16	appPath	2	!	1
17	coverPath	2	null	2
18	"redirect:/catalog.htm"	1	.isEmpty()	1
19	e	1	try	1
20			catch	1
21			if	2
22			else	1
23			()	2
24			{}	4

Произведем расчет метрики Холстеда на примере кода из листинга 1.

Длина кода программы рассчитывается по формуле:

$$N = N1 + N2 = 89 + 51 = 140,$$

где N – длина программы;

N_1 – общее число операторов в программе;

N_2 – общее число операндов в программе.

Словарь программы рассчитывается по формуле:

$$\eta = \eta_1 + \eta_2 = 24 + 19 = 43.$$

Вычислительная сложность программы по метрике Холстеда вычисляется по формуле:

$$V = N \log_2 \eta = 88 \times 5{,}426 = 759.64,$$

где V – вычислительная сложность программы по метрике Холстеда;

N – длина программы;

η – словарь программы.

Сложность кода программы рассчитывается по формуле:

$$D = \frac{\eta_1}{2} \times \frac{N_2}{\eta_2} = 32.21,$$

где D – сложность кода программы;

N_2 – общее число операндов в программе;

η_1 – число уникальных операторов программы;

η_2 – число уникальных операндов программы.

Усилие для написания и понимания кода рассчитывается по формуле:

$$E = D \times V = 32.21 \times 759.64 = 24\ 468,$$

где E – Усилие для написания и понимания кода;

D – сложность кода программы;

V – вычислительная сложность программы по метрике Холстеда.

В результате расчета сложности кода по метрике Холстеда вышло, что программный код рассматриваемой функции удовлетворяет требуемым ограничениям на значения метрик качества. Удовлетворение требуемым ограничениям определяет код рассматриваемой программы как успешный вариант реализации.

Таким образом, в рамках данного раздела было проведено тестирование веб–приложения по добавлению книг. Анализ сложности кода показал, что код программы является качественным, не обладает избыточной сложностью и легко поддерживаемым. Тестирование методом белого ящика подтвердило корректность внутренней логики алгоритмов сохранения книги в каталог.

Заключение

В ходе выполнения выпускной квалификационной работы было разработано веб-приложение для автоматизации процесса книгообмена между читателями. Целью работы являлась разработка программного средства для создания удобной цифровой среды, позволяющей читателям публиковать информацию о книгах, искать литературу, обмениваться ею и вести коммуникацию внутри сообщества, что было успешно достигнуто.

Проведен анализ предметной области, изучены материалы по тематике буккроссинга и существующим способам книгообмена. Осуществлен сравнительный анализ существующих программных продуктов и платформ-аналогов (LiveLib, BookCrossing.ru, тематические группы в социальных сетях), выявлены их сильные и слабые стороны. На основании полученных результатов были сформулированы и проанализированы функциональные и нефункциональные требования к разрабатываемому веб–приложению.

Выполнено проектирование программных средств. Разработана многослойная архитектура приложения на основе паттерна Model-View-Controller (MVC). Спроектирована реляционная структура базы данных в PostgreSQL для хранения информации. Разработан пользовательский интерфейс в виде прототипов ключевых страниц (главная, регистрация книги, каталог).

Осуществлена реализация программного продукта. Разработка велась с использованием языка Java для серверной логики, Jakarta Server Pages (JSP) и HTML, CSS, JavaScript для клиентской части, а также СУБД PostgreSQL. Реализованы основные функции, включая регистрацию и аутентификацию пользователей, добавление и каталогизацию книг, поиск и фильтрацию, создание и обработку запросов на обмен, ведение переписки между участниками, формирование истории обменов («Отпущенные» и «Пойманные» книги), а также административную панель для управления системой и панель модератора.

Проведено тестирование разработанного программного обеспечения. Выполнено функциональное тестирование ключевого модуля «Добавление книги» методом черного ящика с использованием классов эквивалентности и

метода попарного тестирования, подтвердившее корректность работы. Проведены замеры производительности добавления книг, показавшие оптимальное использование ресурсов СУБД. Осуществлен анализ сложности кода по метрике Холстеда, который подтвердил его качество и поддерживаемость.

Выполнено планирование разработки и оценка бюджета проекта с использованием Project Libre. Установлено, что длительность проекта составила 31 рабочий день, а суммарные затраты – 187 709,00 рублей.

Таким образом, в результате выполнения выпускной квалификационной работы разработано веб-приложение для автоматизации процесса книгообмена между читателями. Результаты данной работы могут быть полезны широкому кругу читателей, желающим находить новые книги, делиться уже прочитанными и экономно пополнять свои домашние библиотеки, а также для развития и популяризации буккроссинг–движения.

Список использованных источников

1. LiveLib : страница сайта // LiveLib [Электронный ресурс]. URL: https://www.livelib.ru/bookswap (дата обращения: 10.04.2026).

2. Bookcrossing.ru : страница сайта // Bookcrossing.ru [Электронный ресурс]. URL: https://bookcrossing.ru/ (дата обращения: 10.04.2026).

3. Booksharing.app | Умный обмен книгами // Сообщество ВКонтакте [Электронный ресурс]. URL: https://vk.com/booksharingapp (дата обращения: 10.04.2026).

4. Использование диаграммы вариантов использования UML при проектировании программного обеспечения: // Хабр [Электронный ресурс] URL: https://habr.com/ru/articles/566218/ (Дата обращения: 14.04.2026).

5. Функциональные и нефункциональные требования к ПО: что нужно знать перед запуском проекта // SkillFactory [Электронный ресурс] URL: https://blog.skillfactory.ru/funktsionalnye–i–nefunktsionalnye–trebovaniya–k–po/ (дата обращения: 16.04.2026).

6. Шаблоны архитектуры программного обеспечения: руководство для разработчиков // KursHub [Электронный ресурс]. URL: https://kurshub.ru/journal/blog/shablony–arhitektury–programmnogo–obespecheniya–rukovodstvo–dlya–razrabotchikov/ (дата обращения: 19.04.2026).

7. Чистая архитектура: что это такое и как она устроена // Яндекс образование [Электронный ресурс]. URL: https://education.yandex.ru/handbook/flutter/article/clean–architecture (дата обращения: 21.04.2026).

8. Многоуровневая архитектура в ASP.NET MVC 5 // Metanit [Электронный ресурс]. URL: https://metanit.com/sharp/mvc5/23.5.php (дата обращения: 23.04.2026).

9. Data-Access Layer // GeeksforGeeks [Электронный ресурс]. URL: https://www.geeksforgeeks.org/dbms/data-access-layer/ (дата обращения: 25.04.2026).

10. Простое руководство по диаграмме компонентов // Creately

[Электронный ресурс]. URL: https://creately.com/blog/ru/uncategorized–ru/учебное–пособие–по–компонентной–диаг/ (дата обращения: 27.04.2026).

11. PostgreSQL: всё, что нужно знать для быстрого старта // Skillbox [Электронный ресурс]. URL: https://skillbox.ru/media/code/postgresql-vsye-chto-nuzhno-znat-dlya-bystrogo-starta/ (дата обращения: 30.04.2026).

12. UML: Class Diagram [Электронный ресурс]. URL: https://shesterov.by/tpost/pgu0yucas1–uml–class–diagram (дата обращения: 02.05.2026).

13. Проектирование диаграммы деятельности UML (Activity Diagram) // WorldSkills Russia National Team [Электронный ресурс]. URL: https://nationalteam.worldskills.ru/skills/proektirovanie–diagrammy–deyatelnosti–uml–activity–diagram/ (дата обращения: 05.05.2026).

14. Пошаговый гид по Figma: всё, что нужно знать о программе // Synergy [Электронный ресурс]. URL: https://synergy.ru/akademiya/design/kak_rabotaet_programma_figma#h_caption_0 (дата обращения: 08.05.2026).

15. Диаграмма Ганта: полное руководство по созданию и использованию // Мегаплан [Электронный ресурс]. URL: https://megaplan.ru/blog/crm/diagramma-ganta-polnoe-rukovodstvo/ (дата обращения: 15.05.2026).

16. book–exchange–app // GitHab [Электронный ресурс]. URL: https://github.com/kadazkaya02–lgtm/book–exchange–app.

17. Что такое SSR // Топвизор [Электронный ресурс]. URL: https://journal.topvisor.com/ru/dictionary/what–is–ssr/ (дата обращения: 21.05.2026).

18. Майерс, Г. Искусство тестирования программ / Г. Майерс, Т. Баджетт, К. Сандлер ; пер. с англ. – 3–е изд. – М. : ООО «И.Д. Вильямс», 2012. – 272 с. (дата обращения: 24.05.2026).

19. Pairwise Software Testing // GeeksforGeeks [Электронный ресурс]. URL: https://journal.topvisor.com/ru/dictionary/what–is–ssr/ (дата обращения: 25.05.2026).

20. Unit-тестирование // SkillFactory [Электронный ресурс]. URL: https://blog.skillfactory.ru/glossary/unit-testirovanie/ (дата обращения: 27.05.2026).

Приложение А
(справочное)
Глоссарий

Буккроссинг (Bookcrossing) – общественное движение, основанное на принципе свободного обмена книгами между людьми. В контексте работы – предметная область, для автоматизации которой разрабатывается веб-приложение.

DAO (Data Access Object) – паттерн проектирования, предоставляющий абстрактный интерфейс для доступа к данным в базе данных. В работе используется для инкапсуляции SQL-логики и отделения слоя данных от бизнес-логики.

ER–диаграмма (Entity-Relationship Diagram) – тип диаграммы, используемый для визуализации сущностей предметной области, их атрибутов и связей между ними. В работе применена для проектирования структуры базы данных PostgreSQL.

Java – объектно-ориентированный язык программирования, используемый для разработки серверной логики веб–приложения и реализации бизнес-правил.

JavaServer Pages (JSP) – технология, позволяющая создавать динамические веб-страницы на основе HTML и XML. В работе используется для реализации слоя представления.

JSTL (Jakarta Standard Tag Library) – библиотека стандартных тегов, расширяющая возможности JSP и упрощающая реализацию динамического поведения страниц.

JDBC (Java Database Connectivity) – стандартный API для взаимодействия Java-приложений с реляционными базами данных. В работе используется для выполнения SQL-запросов к PostgreSQL.

MVC (Model-View-Controller) – архитектурный паттерн, разделяющий приложение на три взаимосвязанных компонента: модель (данные и бизнес-логика), представление (пользовательский интерфейс) и контроллер (обработка запросов). Выбран в качестве основной архитектуры для разрабатываемого веб-приложения.

PostgreSQL – объектно-реляционная система управления базами данных (СУБД), используемая для хранения информации о пользователях, книгах, обменах и сообщениях.

REST API – архитектурный стиль взаимодействия компонентов распределенного приложения. В работе используется для организации обмена данными между клиентской и серверной частями.

SQL-инъекция – один из распространенных типов атак на веб-приложения, при котором злоумышленник внедряет произвольный SQL-код в параметры запроса. В работе защита от SQL-инъекций обеспечивается использованием параметризованных запросов.

Tomcat (Apache Tomcat) – веб-сервер и контейнер сервлетов, используемый для развертывания и запуска разработанного веб-приложения.

UML (Unified Modeling Language) – унифицированный язык графического моделирования для описания, проектирования и документирования программных систем. В работе применялся для создания диаграмм вариантов использования, классов, компонентов и деятельности.

Бизнес-логика – часть программного обеспечения, реализующая правила предметной области. В работе представлена контроллерами и сервисными классами.

Каскадное удаление – механизм поддержания ссылочной целостности в реляционных базах данных, при котором удаление записи в родительской таблице автоматически влечет удаление связанных записей в дочерних таблицах.

Контроллер – компонент архитектуры MVC, отвечающий за обработку HTTP-запросов от клиента, вызов соответствующей бизнес-логики и возврат ответа (обычно в виде JSP-страницы).

Многослойная архитектура – подход к проектированию программных систем, при котором приложение разделяется на логические слои (представления, бизнес–логики, доступа к данным), каждый из которых имеет свои зоны ответственности.

Модель (Model) – компонент архитектуры MVC, представляющий данные и бизнес-логику приложения. В работе реализована в виде Java-классов с приватными полями и публичными геттерами и сеттерами.

Нефункциональные требования – требования, описывающие, как программа выполняет свои функции, какие у нее характеристики и ограничения (производительность, безопасность, совместимость, объем памяти и т.д.).

Пагинация – механизм разбиения больших наборов данных на отдельные страницы для улучшения навигации и производительности. В работе используется на странице каталога книг.

Параметризованный запрос (PreparedStatement) – способ выполнения SQL-запросов, при котором значения параметров передаются отдельно от текста команды, что обеспечивает защиту от SQL-инъекций.

Попарное тестирование – комбинаторный метод тестирования программного обеспечения, который для каждой пары входных параметров системы проверяет все возможные дискретные комбинации этих параметров, позволяя сократить количество тестовых сценариев.

Серверный рендеринг (Server-Side Rendering, SSR) – метод отрисовки веб-страницы, при котором HTML-код генерируется на сервере до отправки в браузер. В работе используется для создания динамических страниц на JSP.

Сервлет – Java-класс, расширяющий возможности веб-сервера и обрабатывающий HTTP-запросы. В работе служит основой для реализации контроллеров.

Сервис (Service) – слой приложения, инкапсулирующий бизнес-логику и координирующий взаимодействие между контроллерами и DAO-классами.

Тестирование методом черного ящика – методика тестирования, при которой функциональность проверяется без учета внутренней структуры кода, исключительно на основе требований и спецификаций.

Функциональные требования – требования, описывающие, какие именно функции должна выполнять система (например, регистрация, добавление книги, отправка запроса на обмен).

Приложение Б

(обязательное)

Листинг программы

Листинг Б.1 – Файл BookDAO.java

```
package com.bookexchange.dao;

import com.bookexchange.model.Book;
import com.bookexchange.model.Owner;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.jdbc.core.RowMapper;
import org.springframework.jdbc.support.GeneratedKeyHolder;
import org.springframework.jdbc.support.KeyHolder;
import org.springframework.stereotype.Repository;

import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.ArrayList;
import java.util.List;
import java.util.Map;

@Repository
public class BookDAO {

    @Autowired
    private JdbcTemplate jdbcTemplate;

    private final RowMapper<Book> bookRowMapper = new
RowMapper<Book>() {
        @Override
        public Book mapRow(ResultSet rs, int rowNum) throws
SQLException {
            Book book = new Book();
            book.setId(rs.getInt("id"));
            book.setTitle(rs.getString("title"));
            book.setAuthor(rs.getString("author"));
            book.setYear(rs.getInt("year"));
            book.setLanguage(rs.getString("language"));
            book.setGenreId(rs.getInt("genre_id"));
            book.setGenreName(rs.getString("genre_name"));
            book.setConditionId(rs.getInt("condition_id"));
            book.setConditionName(rs.getString("condition_name"));
            book.setIsbn(rs.getString("isbn"));
            book.setDescription(rs.getString("description"));
            book.setCoverPath(rs.getString("cover_path"));
            book.setOwnerId(rs.getInt("owner_id"));
            book.setOwnerName(rs.getString("owner_name"));
            book.setOwnerEmail(rs.getString("owner_email"));
            book.setStatus(rs.getString("status"));
            return book;
        }
    };
```

```
    private final RowMapper<Owner> ownerRowMapper = new
RowMapper<Owner>() {
        @Override
        public Owner mapRow(ResultSet rs, int rowNum) throws
SQLException {
            Owner owner = new Owner();
            owner.setId(rs.getInt("id"));
            owner.setName(rs.getString("last_name") + " " +
rs.getString("first_name"));

owner.setInitials(rs.getString("first_name").substring(0, 1) +

rs.getString("last_name").substring(0, 1));
            owner.setCity(rs.getString("city"));
            owner.setEmail(rs.getString("email"));
            owner.setAvatarPath(rs.getString("avatar_path"));
            return owner;
        }
    };

    public int countAll() {
        String sql = "SELECT COUNT(*) FROM books WHERE status =
'available'";
        return jdbcTemplate.queryForObject(sql, Integer.class);
    }

    public int countActiveExchanges() {
        String sql = "SELECT COUNT(*) FROM exchanges WHERE status
IN ('pending', 'accepted', 'in_transit')";
        return jdbcTemplate.queryForObject(sql, Integer.class);
    }

    // Получить книги с фильтрацией и пагинацией (для каталога)
    public List<Book> findBooks(String genre, String query, int
page, int pageSize) {
        int offset = (page - 1) * pageSize;
        StringBuilder sql = new StringBuilder(
            "SELECT b.*, g.name as genre_name, c.name as
condition_name, " +
            "u.last_name || ' ' || u.first_name as owner_name,
u.email as owner_email " +
            "FROM Books b " +
            "JOIN Genres g ON b.genre_id = g.id " +
            "JOIN Conditions c ON b.condition_id = c.id " +
            "JOIN Users u ON b.owner_id = u.id " +
            "WHERE 1=1"
        );

        List<Object> params = new ArrayList<>();

        // Фильтрация по жанру
        if (!genre.equals("all")) {
            String genreRu = convertGenreToRussian(genre);
```

```
            sql.append(" AND g.name = ?");
            params.add(genreRu);
        }

        // Поиск по названию или автору
        if (query != null && !query.trim().isEmpty()) {
            sql.append(" AND (LOWER(b.title) LIKE ? OR
LOWER(b.author) LIKE ?)");
            String searchPattern = "%" +
query.toLowerCase().trim() + "%";
            params.add(searchPattern);
            params.add(searchPattern);
        }

        sql.append(" ORDER BY b.id LIMIT ? OFFSET ?");
        params.add(pageSize);
        params.add(offset);

        return jdbcTemplate.query(sql.toString(),
params.toArray(), bookRowMapper);
    }

    // Подсчет количества книг для каталога
    public int countBooksForCatalog(String genre, String query) {
        StringBuilder sql = new StringBuilder(
            "SELECT COUNT(*) FROM Books b " +
            "JOIN Genres g ON b.genre_id = g.id " +
            "WHERE 1=1"
        );

        List<Object> params = new ArrayList<>();

        if (!genre.equals("all")) {
            String genreRu = convertGenreToRussian(genre);
            sql.append(" AND g.name = ?");
            params.add(genreRu);
        }

        if (query != null && !query.trim().isEmpty()) {
            sql.append(" AND (LOWER(b.title) LIKE ? OR
LOWER(b.author) LIKE ?)");
            String searchPattern = "%" +
query.toLowerCase().trim() + "%";
            params.add(searchPattern);
            params.add(searchPattern);
        }

        return jdbcTemplate.queryForObject(sql.toString(),
params.toArray(), Integer.class);
    }

    // Вспомогательный метод для конвертации жанров
    private String convertGenreToRussian(String genre) {
```

```
        if (genre == null) return "all";
        switch(genre) {
            case "fiction": return "Художественная литература";
            case "nonfiction": return "Научная литература";
            case "detective": return "Детектив";
            case "fantasy": return "Фантастика";
            case "romance": return "Роман";
            case "history": return "История";
            case "biography": return "Биография";
            case "poetry": return "Поэзия";
            case "child": return "Детская литература";
            case "other": return "Другое";
            default: return genre;
        }
    }

    // Найти книгу по id
    public Book findById(int id) {
        String sql =
            "SELECT b.*, g.name as genre_name, c.name as
condition_name, " +
            "u.last_name || ' ' || u.first_name as owner_name,
u.email as owner_email " +
            "FROM Books b " +
            "JOIN Genres g ON b.genre_id = g.id " +
            "JOIN Conditions c ON b.condition_id = c.id " +
            "JOIN Users u ON b.owner_id = u.id " +
            "WHERE b.id = ?";
        List<Book> books = jdbcTemplate.query(sql, bookRowMapper,
id);
        return books.isEmpty() ? null : books.get(0);
    }

    // Найти владельца книги
    public Owner findOwnerByBookId(int bookId) {
    String sql = "SELECT u.id, u.first_name, u.last_name,
u.middle_name, u.city, u.email, u.avatar_path " +
                 "FROM Users u " +
                 "JOIN Books b ON u.id = b.owner_id " +
                 "WHERE b.id = ?";

    List<Owner> owners = jdbcTemplate.query(sql, new
Object[]{bookId}, (rs, rowNum) -> {
        Owner owner = new Owner();
        owner.setId(rs.getInt("id"));

        String firstName = rs.getString("first_name");
        String lastName = rs.getString("last_name");
        String middleName = rs.getString("middle_name");

        String fullName = lastName + " " + firstName;
        if (middleName != null && !middleName.isEmpty()) {
            fullName += " " + middleName;
```

```
        }
        owner.setName(fullName);
        owner.setCity(rs.getString("city") != null ?
rs.getString("city") : "Не указан");
        owner.setEmail(rs.getString("email"));

        // Инициалы
        String initials = "";
        if (firstName != null && !firstName.isEmpty()) {
            initials += firstName.substring(0, 1);
        }
        if (lastName != null && !lastName.isEmpty()) {
            initials += lastName.substring(0, 1);
        }
        owner.setInitials(initials.isEmpty() ? "??" :
initials.toUpperCase());

        // Аватар
        owner.setAvatarPath(rs.getString("avatar_path"));

        return owner;
    });

    return owners.isEmpty() ? null : owners.get(0);
    }

// Удалить книгу
public boolean delete(int bookId) {
    try {
        System.out.println("DAO: Starting delete for bookId=" +
bookId);

        // Удаляем из Wishlist
        String deleteWishlistSql = "DELETE FROM Wishlist WHERE
book_id = ?";
        int wishlistDeleted =
jdbcTemplate.update(deleteWishlistSql, bookId);
        System.out.println("DAO: Deleted from Wishlist: " +
wishlistDeleted);

        // Удаляем из Messages
        String deleteMessagesSql = "DELETE FROM Messages WHERE
book_id = ?";
        int messagesDeleted =
jdbcTemplate.update(deleteMessagesSql, bookId);
        System.out.println("DAO: Deleted from Messages: " +
messagesDeleted);

        // Удаляем из Exchanges если есть
        try {
            String deleteExchangesSql = "DELETE FROM Exchanges
WHERE book_id = ?";
            jdbcTemplate.update(deleteExchangesSql, bookId);
```

```
        } catch (Exception e) {
            System.out.println("DAO: Exchanges table may not
exist");
        }

        // Удаляем саму книгу
        String sql = "DELETE FROM Books WHERE id = ?";
        int result = jdbcTemplate.update(sql, bookId);
        System.out.println("DAO: Book deleted, result=" + result);

        if (result == 0) {
            System.err.println("DAO: Book not found in
database!");
            return false;
        }

        return true;
    } catch (Exception e) {
        System.err.println("DAO: Error deleting book: " +
e.getMessage());
        e.printStackTrace();
        throw new RuntimeException("Failed to delete book: " +
e.getMessage(), e);
    }
}

    // Сохранить новую книгу
    public int save(Book book) {
        String sql = "INSERT INTO Books (title, author, year,
language, genre_id, condition_id, " +
                     "isbn, description, cover_path, owner_id,
status) " +
                     "VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)";

        KeyHolder keyHolder = new GeneratedKeyHolder();

        jdbcTemplate.update(connection -> {
            PreparedStatement ps =
connection.prepareStatement(sql, Statement.RETURN_GENERATED_KEYS);
            ps.setString(1, book.getTitle());
            ps.setString(2, book.getAuthor());
            ps.setObject(3, book.getYear());
            ps.setString(4, book.getLanguage());
            ps.setInt(5, book.getGenreId());
            ps.setInt(6, book.getConditionId());
            ps.setString(7, book.getIsbn());
            ps.setString(8, book.getDescription());
            ps.setString(9, book.getCoverPath());
            ps.setInt(10, book.getOwnerId());
            ps.setString(11, "available");
            return ps;
        }, keyHolder);
```

```
        Map<String, Object> keys = keyHolder.getKeys();
        if (keys != null && keys.containsKey("id")) {
            return ((Number) keys.get("id")).intValue();
        }

        return 0;
    }

    // Добавить книгу в список желаний
    public void addToWishlist(int userId, int bookId) {
        String sql = "INSERT INTO Wishlist (user_id, book_id)
VALUES (?, ?)";
        jdbcTemplate.update(sql, userId, bookId);
    }

    // Проверить, есть ли книга в списке желаний
    public boolean isBookInWishlist(int userId, int bookId) {
        String sql = "SELECT COUNT(*) FROM Wishlist WHERE user_id
= ? AND book_id = ?";
        Integer count = jdbcTemplate.queryForObject(sql,
Integer.class, userId, bookId);
        return count != null && count > 0;
    }

    // Получить список желаний пользователя
    public List<Book> findWishlistByUser(int userId) {
        String sql =
            "SELECT b.*, g.name as genre_name, c.name as
condition_name, " +
            "u.last_name || ' ' || u.first_name as owner_name,
u.email as owner_email " +
            "FROM Books b " +
            "JOIN Genres g ON b.genre_id = g.id " +
            "JOIN Conditions c ON b.condition_id = c.id " +
            "JOIN Users u ON b.owner_id = u.id " +
            "JOIN Wishlist w ON b.id = w.book_id " +
            "WHERE w.user_id = ? " +
            "ORDER BY w.created_at DESC";
        return jdbcTemplate.query(sql, bookRowMapper, userId);
    }

    // Получить книги пользователя
    public List<Book> findBooksByUser(int userId) {
        String sql =
            "SELECT b.*, g.name as genre_name, c.name as
condition_name, " +
            "u.last_name || ' ' || u.first_name as owner_name,
u.email as owner_email " +
            "FROM Books b " +
            "JOIN Genres g ON b.genre_id = g.id " +
            "JOIN Conditions c ON b.condition_id = c.id " +
            "JOIN Users u ON b.owner_id = u.id " +
            "WHERE b.owner_id = ? " +
```

```
            "ORDER BY b.id";
        return jdbcTemplate.query(sql, bookRowMapper, userId);
    }

    // Получить все книги с фильтрацией и пагинацией (для админ-
панели)
    public List<Book> findAllBooks(String search, String
statusFilter, int page, int pageSize) {
        int offset = (page - 1) * pageSize;
        StringBuilder sql = new StringBuilder(
            "SELECT b.*, g.name as genre_name, c.name as
condition_name, " +
            "u.last_name || ' ' || u.first_name as owner_name,
u.email as owner_email " +
            "FROM Books b " +
            "JOIN Genres g ON b.genre_id = g.id " +
            "JOIN Conditions c ON b.condition_id = c.id " +
            "JOIN Users u ON b.owner_id = u.id " +
            "WHERE 1=1"
        );

        List<Object> params = new ArrayList<>();

        // Поиск по названию или автору
        if (search != null && !search.trim().isEmpty()) {
            sql.append(" AND (LOWER(b.title) LIKE ? OR
LOWER(b.author) LIKE ?)");
            String searchPattern = "%" +
search.toLowerCase().trim() + "%";
            params.add(searchPattern);
            params.add(searchPattern);
        }

        // Фильтр по статусу
        if (statusFilter != null && !statusFilter.equals("all")) {
            sql.append(" AND b.status = ?");
            params.add(statusFilter);
        }

        sql.append(" ORDER BY b.id LIMIT ? OFFSET ?");
        params.add(pageSize);
        params.add(offset);

        return jdbcTemplate.query(sql.toString(),
params.toArray(), bookRowMapper);
    }

    // Подсчет общего количества книг с фильтрацией (для админ-
панели)
    public int countAllBooks(String search, String statusFilter) {
        StringBuilder sql = new StringBuilder(
            "SELECT COUNT(*) FROM Books b WHERE 1=1"
        );
```

```
        List<Object> params = new ArrayList<>();

        if (search != null && !search.trim().isEmpty()) {
            sql.append(" AND (LOWER(b.title) LIKE ? OR
LOWER(b.author) LIKE ?)");
            String searchPattern = "%" +
search.toLowerCase().trim() + "%";
            params.add(searchPattern);
            params.add(searchPattern);
        }

        if (statusFilter != null && !statusFilter.equals("all")) {
            sql.append(" AND b.status = ?");
            params.add(statusFilter);
        }

        return jdbcTemplate.queryForObject(sql.toString(),
params.toArray(), Integer.class);
    }

    // Подсчет количества книг по статусу
    public int countBooksByStatus(String status) {
        String sql = "SELECT COUNT(*) FROM Books WHERE status =
?";
        return jdbcTemplate.queryForObject(sql, Integer.class,
status);
    }

    // Обновить статус книги
    public void updateBookStatus(int bookId, String status) {
        String sql = "UPDATE Books SET status = ? WHERE id = ?";
        int result = jdbcTemplate.update(sql, status, bookId);
        System.out.println("updateBookStatus: bookId=" + bookId +
", status=" + status + ", result=" + result);
    }

    // Удалить обложку книги и установить дефолтную
    public void deleteBookCover(int bookId) {
        String sql = "UPDATE Books SET cover_path =
'/resources/images/default-book.jpg' WHERE id = ?";
        int result = jdbcTemplate.update(sql, bookId);
        System.out.println("deleteBookCover: bookId=" + bookId +
", result=" + result);
    }

    // Передать книгу новому владельцу и сохранить историю
    public void transferBookOwnership(int bookId, int newOwnerId)
{
        // Сначала получаем текущего владельца
        Book book = findById(bookId);
        if (book != null) {
            int oldOwnerId = book.getOwnerId();
```

```
            // Если original_owner_id еще не установлен (первый
обмен), устанавливаем его
            String setOriginalOwnerSql = "UPDATE books SET
original_owner_id = COALESCE(original_owner_id, ?) WHERE id = ?";
            jdbcTemplate.update(setOriginalOwnerSql, oldOwnerId,
bookId);

            // Меняем владельца и статус
            String sql = "UPDATE books SET owner_id = ?, status =
'available' WHERE id = ?";
            jdbcTemplate.update(sql, newOwnerId, bookId);
        }
    }

    // Получить отпущенные книги пользователя (книги, которые
пользователь отдал)
    public List<Book> findReleasedBooksByUser(int userId) {
        String sql =
            "SELECT b.*, g.name as genre_name, c.name as
condition_name, " +
            "u.last_name || ' ' || u.first_name as owner_name,
u.email as owner_email " +
            "FROM Books b " +
            "JOIN Genres g ON b.genre_id = g.id " +
            "JOIN Conditions c ON b.condition_id = c.id " +
            "JOIN Users u ON b.owner_id = u.id " +
            "WHERE b.original_owner_id = ? AND b.owner_id != ? " +
            "ORDER BY b.id DESC";
        return jdbcTemplate.query(sql, bookRowMapper, userId,
userId);
    }

    // Получить полученные книги пользователя (книги, которые
пользователь получил в обмен)
    public List<Book> findReceivedBooksByUser(int userId) {
        String sql =
            "SELECT b.*, g.name as genre_name, c.name as
condition_name, " +
            "u.last_name || ' ' || u.first_name as owner_name,
u.email as owner_email " +
            "FROM Books b " +
            "JOIN Genres g ON b.genre_id = g.id " +
            "JOIN Conditions c ON b.condition_id = c.id " +
            "JOIN Users u ON b.owner_id = u.id " +
            "WHERE b.original_owner_id != ? AND b.owner_id = ? " +
            "ORDER BY b.id DESC";
        return jdbcTemplate.query(sql, bookRowMapper, userId,
userId);
    }

    // Удалить книгу из списка желаний
    public void removeFromWishlist(int userId, int bookId) {
```

```
        String sql = "DELETE FROM Wishlist WHERE user_id = ? AND
book_id = ?";
        jdbcTemplate.update(sql, userId, bookId);
    }

    public void update(Book book) {
    String sql = "UPDATE Books SET title = ?, author = ?, year =
?, language = ?, " +
                 "genre_id = ?, condition_id = ?, isbn = ?,
description = ?, " +
                 "cover_path = ? WHERE id = ?";
    jdbcTemplate.update(sql,
        book.getTitle(),
        book.getAuthor(),
        book.getYear(),
        book.getLanguage(),
        book.getGenreId(),
        book.getConditionId(),
        book.getIsbn(),
        book.getDescription(),
        book.getCoverPath(),
        book.getId());
    System.out.println("Book updated: id=" + book.getId() + ",
title=" + book.getTitle());
}
}
```

Листинг Б.2 – Файл ExchangeService.java

```
package com.bookexchange.service;

import com.bookexchange.dao.BookDAO;
import com.bookexchange.dao.ExchangeDAO;
import com.bookexchange.model.Book;
import com.bookexchange.model.Exchange;
import org.springframework.stereotype.Service;

import java.util.Date;
import java.util.List;

@Service
public class ExchangeService {

    private final ExchangeDAO exchangeDAO;
    private final BookDAO bookDAO;

    public ExchangeService(ExchangeDAO exchangeDAO, BookDAO
bookDAO) {
        this.exchangeDAO = exchangeDAO;
        this.bookDAO = bookDAO;
    }

    // Получить входящие запросы на обмен для пользователя
```

```
    public List<Exchange> getIncomingRequests(int userId) {
        return exchangeDAO.getAllIncomingRequests(userId);
    }

    // Получить исходящие запросы на обмен от пользователя
    public List<Exchange> getOutgoingRequests(int userId) {
        return exchangeDAO.getOutgoingRequests(userId);
    }

    // Найти обмен по ID
    public Exchange getExchangeById(int exchangeId) {
        return exchangeDAO.findById(exchangeId);
    }

    // Получить книгу по ID
    public Book getBookById(int bookId) {
        return bookDAO.findById(bookId);
    }

    // Проверить, существует ли уже активный запрос на обмен этой
книги от пользователя
    public boolean hasActiveRequest(int bookId, int fromUserId) {
        return exchangeDAO.hasActiveRequest(bookId, fromUserId);
    }

    // Создать новый запрос на обмен
    public int createExchangeRequest(int bookId, int fromUserId) {
        Book book = bookDAO.findById(bookId);
        if (book == null) {
            throw new IllegalArgumentException("Книга не
найдена");
        }
        if (book.getOwnerId() == fromUserId) {
            throw new IllegalStateException("Нельзя запросить
обмен на свою книгу");
        }
        if (!"available".equals(book.getStatus())) {
            throw new IllegalStateException("Книга уже в обмене
или забронирована");
        }
        if (exchangeDAO.hasActiveRequest(bookId, fromUserId)) {
            throw new IllegalStateException("Вы уже отправили
запрос на эту книгу");
        }

        Date exchangeDate = new Date();
        return exchangeDAO.createExchangeRequest(bookId,
fromUserId, book.getOwnerId(), exchangeDate);
    }

    // Принять запрос на обмен
    public void acceptExchangeRequest(int exchangeId, int userId)
{
```

```
        Exchange exchange = exchangeDAO.findById(exchangeId);
        if (exchange == null) {
            throw new IllegalArgumentException("Запрос не
найден");
        }
        if (exchange.getToUserId() != userId) {
            throw new IllegalStateException("Вы не можете принять
этот запрос");
        }

        exchangeDAO.acceptRequest(exchangeId);
        bookDAO.updateBookStatus(exchange.getBookId(),
"exchanged");
        bookDAO.removeFromWishlist(exchange.getFromUserId(),
exchange.getBookId());
    }

    // Отклонить запрос на обмен
    public void rejectExchangeRequest(int exchangeId, int userId)
{
        Exchange exchange = exchangeDAO.findById(exchangeId);
        if (exchange == null) {
            throw new IllegalArgumentException("Запрос не
найден");
        }
        if (exchange.getToUserId() != userId) {
            throw new IllegalStateException("Вы не можете
отклонить этот запрос");
        }

        exchangeDAO.rejectRequest(exchangeId);
    }

    // Завершить обмен и передать книгу новому владельцу
    public void completeExchangeRequest(int exchangeId, int
userId) {
        Exchange exchange = exchangeDAO.findById(exchangeId);
        if (exchange == null) {
            throw new IllegalArgumentException("Запрос не
найден");
        }
        if (exchange.getToUserId() != userId) {
            throw new IllegalStateException("Вы не можете
завершить этот обмен");
        }

        Date actualExchangeDate = new Date();
        exchangeDAO.updateExchangeDate(exchangeId,
actualExchangeDate);
        exchangeDAO.completeRequest(exchangeId);
        bookDAO.transferBookOwnership(exchange.getBookId(),
exchange.getFromUserId());
        bookDAO.removeFromWishlist(exchange.getFromUserId(),
```

```
exchange.getBookId());
    }
}
```

Листинг Б.3 – BookController.java

```
package com.bookexchange.controller;

import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.multipart.MultipartFile;
import
org.springframework.web.servlet.mvc.support.RedirectAttributes;

import javax.servlet.http.HttpSession;
import java.util.List;

import com.bookexchange.model.Book;
import com.bookexchange.model.Owner;
import com.bookexchange.model.Genre;
import com.bookexchange.model.Condition;
import com.bookexchange.service.BookService;

@Controller
public class BookController {

    private final BookService bookService;

    public BookController(BookService bookService) {
        this.bookService = bookService;
    }

    private boolean isUserBlocked(HttpSession session) {
        String status = (String)
session.getAttribute("userStatus");
        return "blocked".equals(status);
    }

    @GetMapping("/book/register.htm")
    public String registerBook(HttpSession session, Model model) {
        Integer userId = (Integer) session.getAttribute("userId");
        if (userId == null) return "redirect:/login.htm";
        if (isUserBlocked(session)) {
            model.addAttribute("error", "Ваш аккаунт
заблокирован.");
            return "redirect:/account.htm";
        }

        List<Genre> genres = bookService.getAllGenres();
        List<Condition> conditions =
```

```
bookService.getAllConditions();
        model.addAttribute("genres", genres);
        model.addAttribute("conditions", conditions);
        return "RegisterBook";
    }

    @GetMapping("/catalog.htm")
    public String catalog(
            @RequestParam(value = "page", defaultValue = "1") int
page,
            @RequestParam(value = "genre", defaultValue = "all")
String genre,
            @RequestParam(value = "query", defaultValue = "")
String query,
            Model model) {

        int pageSize = 12;
        List<Book> books = bookService.getBooks(genre, query,
page, pageSize);
        int totalBooks = bookService.countBooks(genre, query);
        int totalPages = (int) Math.ceil((double) totalBooks /
pageSize);

        model.addAttribute("books", books);
        model.addAttribute("genres", bookService.getAllGenres());
        model.addAttribute("currentPage", page);
        model.addAttribute("totalPages", totalPages);
        model.addAttribute("totalBooks", totalBooks);
        model.addAttribute("selectedGenre", genre);
        model.addAttribute("searchQuery", query);
        return "Catalog";
    }

    @GetMapping("/catalog/search.htm")
    public String searchBooks(@RequestParam("query") String query,
                              @RequestParam(value = "genre",
defaultValue = "all") String genre) {
        return "redirect:/catalog.htm?query=" + query + "&genre="
+ genre;
    }

    @PostMapping("/book/register.htm")
    public String processRegisterBook(
            @RequestParam("title") String title,
            @RequestParam("author") String author,
            @RequestParam("genreId") int genreId,
            @RequestParam(value = "year", required = false)
Integer year,
            @RequestParam(value = "language", required = false)
String language,
            @RequestParam("conditionId") int conditionId,
            @RequestParam(value = "isbn", required = false) String
isbn,
```

```
            @RequestParam(value = "description", required = false)
String description,
            @RequestParam(value = "cover", required = false)
MultipartFile cover,
            HttpSession session) {

        Integer userId = (Integer) session.getAttribute("userId");
        if (userId == null) return "redirect:/login.htm";

        Book book = new Book();
        book.setTitle(title);
        book.setAuthor(author);
        book.setYear(year);
        book.setLanguage(language);
        book.setGenreId(genreId);
        book.setConditionId(conditionId);
        book.setIsbn(isbn);
        book.setDescription(description);
        book.setOwnerId(userId);
        book.setStatus("available");

        if (cover != null && !cover.isEmpty()) {
            try {
                String appPath =
session.getServletContext().getRealPath("/");
                String coverPath =
bookService.saveBookCover(cover, userId, appPath);
                book.setCoverPath(coverPath);
            } catch (Exception e) {
                book.setCoverPath("/resources/images/default-
book.jpg");
            }
        } else {
            book.setCoverPath("/resources/images/default-
book.jpg");
        }

        bookService.saveBook(book);
        return "redirect:/catalog.htm";
    }

    @GetMapping("/book/edit.htm")
    public String editBook(@RequestParam("id") int bookId,
HttpSession session, Model model) {
        Integer userId = (Integer) session.getAttribute("userId");
        if (userId == null) return "redirect:/login.htm";
        if (isUserBlocked(session)) {
            model.addAttribute("error", "Ваш аккаунт
заблокирован.");
            return "redirect:/account.htm";
        }

        Book book = bookService.getBookById(bookId);
```

```
        if (book == null || book.getOwnerId() != userId) return
"redirect:/account.htm";

        model.addAttribute("genres", bookService.getAllGenres());
        model.addAttribute("conditions",
bookService.getAllConditions());
        model.addAttribute("book", book);
        return "RegisterBook";
    }

    @PostMapping("/book/update.htm")
    public String updateBook(
            @RequestParam("bookId") int bookId,
            @RequestParam("title") String title,
            @RequestParam("author") String author,
            @RequestParam("genreId") int genreId,
            @RequestParam(value = "year", required = false)
Integer year,
            @RequestParam(value = "language", required = false)
String language,
            @RequestParam("conditionId") int conditionId,
            @RequestParam(value = "isbn", required = false) String
isbn,
            @RequestParam(value = "description", required = false)
String description,
            @RequestParam(value = "cover", required = false)
MultipartFile cover,
            HttpSession session) {

        Integer userId = (Integer) session.getAttribute("userId");
        if (userId == null) return "redirect:/login.htm";

        Book book = bookService.getBookById(bookId);
        if (book == null || book.getOwnerId() != userId) return
"redirect:/account.htm";

        book.setTitle(title);
        book.setAuthor(author);
        book.setYear(year);
        book.setLanguage(language);
        book.setGenreId(genreId);
        book.setConditionId(conditionId);
        book.setIsbn(isbn);
        book.setDescription(description);

        if (cover != null && !cover.isEmpty()) {
            try {
                String appPath =
session.getServletContext().getRealPath("/");
                bookService.deleteCoverFile(book.getCoverPath(),
appPath);
                String coverPath =
bookService.saveBookCover(cover, userId, appPath);
```

```
                book.setCoverPath(coverPath);
            } catch (Exception e) {
                System.err.println("Ошибка при обновлении обложки:
" + e.getMessage());
            }
        }

        bookService.updateBook(book);
        return "redirect:/account.htm";
    }

@PostMapping("/book/delete.htm")
public String deleteBook(@RequestParam("bookId") int bookId,
                         HttpSession session,
                         Model model,
                         RedirectAttributes redirectAttributes) {
    System.out.println("=== DELETE BOOK START ===");
    System.out.println("bookId: " + bookId);

    Integer userId = (Integer) session.getAttribute("userId");
    System.out.println("userId: " + userId);

    if (userId == null) {
        System.out.println("User not authenticated");
        return "redirect:/login.htm";
    }

    try {
        Book book = bookService.getBookById(bookId);
        System.out.println("Book found: " + (book != null ?
book.getTitle() : "null"));

        if (book == null) {
            System.out.println("Book not found!");
            redirectAttributes.addFlashAttribute("error", "Книга
не найдена");
            return "redirect:/account.htm";
        }

        System.out.println("Book owner: " + book.getOwnerId() + ",
Current user: " + userId);

        if (book.getOwnerId() != userId) {
            System.out.println("Permission denied!");
            redirectAttributes.addFlashAttribute("error", "У вас
нет прав на удаление этой книги");
            return "redirect:/account.htm";
        }

        // Удаляем файл обложки если есть
        if (book.getCoverPath() != null &&
!book.getCoverPath().isEmpty()
            && !book.getCoverPath().contains("default-book.jpg"))
```

```
{
            try {
                String realPath =
session.getServletContext().getRealPath("/");
                bookService.deleteCoverFile(book.getCoverPath(),
realPath);
            } catch (Exception e) {
                System.err.println("Error deleting cover file: " +
e.getMessage());
            }
        }

        // Удаляем книгу
        bookService.deleteBook(bookId);
        System.out.println("Book deleted successfully!");
        redirectAttributes.addFlashAttribute("success", "Книга
успешно удалена");

    } catch (Exception e) {
        System.err.println("Error deleting book: " +
e.getMessage());
        e.printStackTrace();
        redirectAttributes.addFlashAttribute("error", "Ошибка при
удалении книги: " + e.getMessage());
    }

    System.out.println("=== DELETE BOOK END ===");
    return "redirect:/account.htm";
}

    @GetMapping("/book/details.htm")
    public String bookDetails(@RequestParam("id") int bookId,
Model model, HttpSession session) {
        Book book = bookService.getBookById(bookId);
        if (book == null) return "redirect:/catalog.htm";

        Owner owner = bookService.getBookOwner(bookId);
        model.addAttribute("book", book);
        model.addAttribute("owner", owner);

        Integer userId = (Integer) session.getAttribute("userId");
        model.addAttribute("canExchange", userId != null && userId
!= book.getOwnerId());
        return "BookDetails";
    }

    @PostMapping("/wishlist/add.htm")
    public String addToWishlist(@RequestParam("bookId") int
bookId, HttpSession session) {
        Integer userId = (Integer) session.getAttribute("userId");
        if (userId == null) return "redirect:/login.htm";
        bookService.addToWishlist(userId, bookId);
        return "redirect:/book/details.htm?id=" + bookId;
```

```
    }

    @PostMapping("/wishlist/remove.htm")
    public String removeFromWishlist(@RequestParam("bookId") int
bookId, HttpSession session) {
        Integer userId = (Integer) session.getAttribute("userId");
        if (userId == null) return "redirect:/login.htm";
        bookService.removeFromWishlist(userId, bookId);
        return "redirect:/account.htm";
    }
}
```

Листинг Б.4 – Book.java

```
package com.bookexchange.model;
public class Book {
    private int id;
    private String title;
    private String author;
    private Integer year;
    private String language;
    private int genreId;
    private String genreName;
    private int conditionId;
    private String conditionName;
    private String isbn;
    private String description;
    private String coverPath;
    private int ownerId;
    private String ownerName;
    private String ownerEmail;
    private String status;
    private String initials;
    private String coverColor;

    public Book() {
        this.coverColor = "#d9c2b8"; // цвет по умолчанию
    }

    // Геттеры и сеттеры
    public int getId() { return id; }
    public void setId(int id) { this.id = id; }

    public String getTitle() { return title; }
    public void setTitle(String title) {
        this.title = title;
        if (title != null && !title.isEmpty()) {
            this.initials = title.substring(0, 1).toUpperCase();
        }
    }

    public String getAuthor() { return author; }
    public void setAuthor(String author) { this.author = author; }
```

```
    public Integer getYear() { return year; }
    public void setYear(Integer year) { this.year = year; }

    public String getLanguage() { return language; }
    public void setLanguage(String language) { this.language =
language; }

    public int getGenreId() { return genreId; }
    public void setGenreId(int genreId) { this.genreId = genreId;
}

    public String getGenreName() { return genreName; }
    public void setGenreName(String genreName) { this.genreName =
genreName; }

    public String getGenre() {
    return this.genreName;
}

    public int getConditionId() { return conditionId; }
    public void setConditionId(int conditionId) { this.conditionId
= conditionId; }

    public String getConditionName() { return conditionName; }
    public void setConditionName(String conditionName) {
this.conditionName = conditionName; }

    public String getIsbn() { return isbn; }
    public void setIsbn(String isbn) { this.isbn = isbn; }

    public String getDescription() { return description; }
    public void setDescription(String description) {
this.description = description; }

    public String getCoverPath() { return coverPath; }
    public void setCoverPath(String coverPath) { this.coverPath =
coverPath; }

    public int getOwnerId() { return ownerId; }
    public void setOwnerId(int ownerId) { this.ownerId = ownerId;
}

    public String getOwnerName() { return ownerName; }
    public void setOwnerName(String ownerName) { this.ownerName =
ownerName; }

    public String getOwnerEmail() { return ownerEmail; }
    public void setOwnerEmail(String ownerEmail) { this.ownerEmail
= ownerEmail; }

    public String getStatus() { return status; }
    public void setStatus(String status) { this.status = status; }
```

```
    public String getInitials() { return initials; }
    public void setInitials(String initials) { this.initials =
initials; }

    public String getCoverColor() { return coverColor; }
    public void setCoverColor(String coverColor) { this.coverColor
= coverColor; }

    public String getConditionClass() {
        if (conditionName != null) {
            if (conditionName.contains("Отличное") ||
conditionName.contains("Новая"))
                return "good";
            else if (conditionName.contains("Хорошее"))
                return "normal";
            else if (conditionName.contains("Приемлемое"))
                return "satisfactory";
        }
        return "other";
    }
}
```

Приложение В

(рекомендуемое)

Листинг автоматизированных тестов

Листинг В.1 – Добавление 100 книг

```
SELECT MIN(id) as min_id, MAX(id) as max_id, COUNT(*) as
user_count FROM Users;

DO $$
DECLARE
    v_start_time TIMESTAMP;
    v_end_time TIMESTAMP;
    v_time_diff_ms INTEGER;
    v_count INTEGER := 100;
    v_i INTEGER := 1;
    v_end_id INTEGER;
    v_min_user_id INTEGER;
    v_max_user_id INTEGER;
    v_user_count INTEGER;
    v_owner_id INTEGER;
BEGIN

    SELECT MIN(id), MAX(id), COUNT(*) INTO v_min_user_id,
v_max_user_id, v_user_count FROM Users;

    IF v_user_count = 0 THEN
        RAISE EXCEPTION 'Нет пользователей в таблице Users.
Сначала добавьте пользователей.';
    END IF;

    RAISE NOTICE 'Найдено пользователей: %, от ID % до %',
v_user_count, v_min_user_id, v_max_user_id;

    v_start_time := clock_timestamp();
    v_end_id := v_i + v_count - 1;

    WHILE v_i <= v_end_id LOOP

        v_owner_id := v_min_user_id + (v_i % v_user_count);

        INSERT INTO Books (title, author, year, language,
genre_id, condition_id, isbn, description, cover_path, owner_id,
status)
        VALUES (
            'Книга ' || v_i,
            'Автор ' || (v_i % 10 + 1),
            2000 + (v_i % 25),
            CASE v_i % 3
                WHEN 0 THEN 'Русский'
                WHEN 1 THEN 'Английский'
                ELSE 'Немецкий'
            END,
            (v_i % 10) + 1,
            (v_i % 5) + 1,
            '978-5-' || v_i || '-' || (v_i * 2) || '-' || (v_i *
3),
```

```
            'Описание книги номер ' || v_i,
            '/covers/book' || v_i || '.jpg',
            v_owner_id,
            CASE v_i % 3
                WHEN 0 THEN 'available'
                WHEN 1 THEN 'reserved'
                ELSE 'in_exchange'
            END
        );
        v_i := v_i + 1;
    END LOOP;

    v_end_time := clock_timestamp();
    v_time_diff_ms := EXTRACT(MILLISECOND FROM (v_end_time -
v_start_time))::INTEGER;

    RAISE NOTICE '=== Вставка 100 книг ===';
    RAISE NOTICE 'Время выполнения: % мс', v_time_diff_ms;
    RAISE NOTICE 'Количество записей: %', v_count;
    RAISE NOTICE '----------------------------------------';
END $$;

-- Проверка
SELECT COUNT(*) AS total_books FROM Books;

-- Проверка распределения книг по владельцам
SELECT owner_id, COUNT(*) as book_count
FROM Books
GROUP BY owner_id
ORDER BY owner_id;
```

Листинг В.2 – Добавление 500 книг

```
SELECT MIN(id) as min_id, MAX(id) as max_id, COUNT(*) as
user_count FROM Users;

DO $$
DECLARE
    v_start_time TIMESTAMP;
    v_end_time TIMESTAMP;
    v_time_diff_ms INTEGER;
    v_count INTEGER := 500;
    v_i INTEGER := 1;
    v_end_id INTEGER;
    v_min_user_id INTEGER;
    v_max_user_id INTEGER;
    v_user_count INTEGER;
    v_owner_id INTEGER;
BEGIN

    SELECT MIN(id), MAX(id), COUNT(*) INTO v_min_user_id,
v_max_user_id, v_user_count FROM Users;

    IF v_user_count = 0 THEN
```

```
        RAISE EXCEPTION 'Нет пользователей в таблице Users.
Сначала добавьте пользователей.';
    END IF;

    RAISE NOTICE 'Найдено пользователей: %, от ID % до %',
v_user_count, v_min_user_id, v_max_user_id;

    v_start_time := clock_timestamp();
    v_end_id := v_i + v_count - 1;

    WHILE v_i <= v_end_id LOOP

        v_owner_id := v_min_user_id + (v_i % v_user_count);

        INSERT INTO Books (title, author, year, language,
genre_id, condition_id, isbn, description, cover_path, owner_id,
status)
        VALUES (
            'Книга ' || v_i,
            'Автор ' || (v_i % 10 + 1),
            2000 + (v_i % 25),
            CASE v_i % 3
                WHEN 0 THEN 'Русский'
                WHEN 1 THEN 'Английский'
                ELSE 'Немецкий'
            END,
            (v_i % 10) + 1,
            (v_i % 5) + 1,
            '978-5-' || v_i || '-' || (v_i * 2) || '-' || (v_i *
3),
            'Описание книги номер ' || v_i,
            '/covers/book' || v_i || '.jpg',
            v_owner_id,
            CASE v_i % 3
                WHEN 0 THEN 'available'
                WHEN 1 THEN 'reserved'
                ELSE 'in_exchange'
            END
        );
        v_i := v_i + 1;
    END LOOP;

    v_end_time := clock_timestamp();
    v_time_diff_ms := EXTRACT(MILLISECOND FROM (v_end_time -
v_start_time))::INTEGER;

    RAISE NOTICE '=== Вставка 500 книг ===';
    RAISE NOTICE 'Время выполнения: % мс', v_time_diff_ms;
    RAISE NOTICE 'Количество записей: %', v_count;
    RAISE NOTICE '----------------------------------------';
END $$;

-- Проверка
```

```
SELECT COUNT(*) AS total_books FROM Books;

-- Проверка распределения книг по владельцам
SELECT owner_id, COUNT(*) as book_count
FROM Books
GROUP BY owner_id
ORDER BY owner_id;
```

Листинг В.3 – Добавление 1000 книг

```
SELECT MIN(id) as min_id, MAX(id) as max_id, COUNT(*) as
user_count FROM Users;

DO $$
DECLARE
    v_start_time TIMESTAMP;
    v_end_time TIMESTAMP;
    v_time_diff_ms INTEGER;
    v_count INTEGER := 1000;
    v_i INTEGER := 1;
    v_end_id INTEGER;
    v_min_user_id INTEGER;
    v_max_user_id INTEGER;
    v_user_count INTEGER;
    v_owner_id INTEGER;
BEGIN

    SELECT MIN(id), MAX(id), COUNT(*) INTO v_min_user_id,
v_max_user_id, v_user_count FROM Users;

    IF v_user_count = 0 THEN
        RAISE EXCEPTION 'Нет пользователей в таблице Users.
Сначала добавьте пользователей.';
    END IF;

    RAISE NOTICE 'Найдено пользователей: %, от ID % до %',
v_user_count, v_min_user_id, v_max_user_id;

    v_start_time := clock_timestamp();
    v_end_id := v_i + v_count - 1;

    WHILE v_i <= v_end_id LOOP

        v_owner_id := v_min_user_id + (v_i % v_user_count);

        INSERT INTO Books (title, author, year, language,
genre_id, condition_id, isbn, description, cover_path, owner_id,
status)
        VALUES (
            'Книга ' || v_i,
            'Автор ' || (v_i % 10 + 1),
            2000 + (v_i % 25),
            CASE v_i % 3
                WHEN 0 THEN 'Русский'
```

```
                WHEN 1 THEN 'Английский'
                ELSE 'Немецкий'
            END,
            (v_i % 10) + 1,
            (v_i % 5) + 1,
            '978-5-' || v_i || '-' || (v_i * 2) || '-' || (v_i *
3),
            'Описание книги номер ' || v_i,
            '/covers/book' || v_i || '.jpg',
            v_owner_id,
            CASE v_i % 3
                WHEN 0 THEN 'available'
                WHEN 1 THEN 'reserved'
                ELSE 'in_exchange'
            END
        );
        v_i := v_i + 1;
    END LOOP;

    v_end_time := clock_timestamp();
    v_time_diff_ms := EXTRACT(MILLISECOND FROM (v_end_time -
v_start_time))::INTEGER;

    RAISE NOTICE '=== Вставка 1000 книг ===';
    RAISE NOTICE 'Время выполнения: % мс', v_time_diff_ms;
    RAISE NOTICE 'Количество записей: %', v_count;
    RAISE NOTICE '----------------------------------------';
END $$;

-- Проверка
SELECT COUNT(*) AS total_books FROM Books;

-- Проверка распределения книг по владельцам
SELECT owner_id, COUNT(*) as book_count
FROM Books
GROUP BY owner_id
ORDER BY owner_id;
```