

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет

вычислительной техники

Кафедра

«Вычислительная техника»

Направление подготовки: 09.03.01 «Информатика и вычислительная техника»

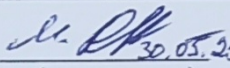
Профиль подготовки: «Программное обеспечение средств вычислительной
техники и автоматизированных систем»

БАКАЛАВРСКАЯ РАБОТА

на тему

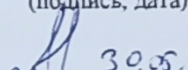
WEB-ПРИЛОЖЕНИЕ «КНИГОВОРОТ»

Студент


(подпись, дата)

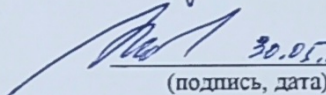
Давкин Максим Александрович
(ФИО полностью)

Руководитель


(подпись, дата)

Акифьев И.В.
(фамилия, инициалы)

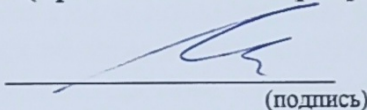
Нормоконтролёр


(подпись, дата)

Бычков А.С.
(фамилия, инициалы)

Работа допущена к защите (протокол зас. кафедры от 28.05.2025 г. № 16)

Заведующий кафедрой

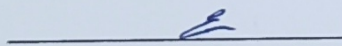

(подпись)

М.А. Митрохин
(фамилия, инициалы)

Работа защищена с отметкой хорошо (протокол заседания ГЭК от

11 июня 2025 г. № 27)

Секретарь ГЭК



Е.И. Гурин
(подпись) (фамилия, инициалы)

Пенза 2025

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

Факультет
вычислительной техники

Кафедра
«Вычислительная техника»

«Утверждаю»
Заведующий кафедрой
М.А. Митрохин
«06» мая 2025 г.

**ЗАДАНИЕ
НА БАКАЛАВРСКУЮ РАБОТУ**

1. Студент Давкин Максим Александрович гр. 21ВВП2 ФВТ

2. Тема работы: web-приложение «Книговорот»

Тема утверждена приказом ПГУ № 601/о от 06.05.2025 г.

3. Руководитель бакалаврской работы: к.э.н., доцент кафедры ВТ ПГУ Акифьев И.В.

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

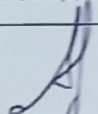
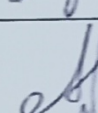
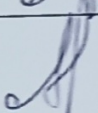
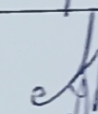
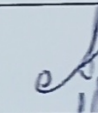
Разработать веб-приложение «Книговорот» для обмена книгами электронными и вещественными. Функционал электронного обмена – каждый пользователь сайта может загрузить свою книгу или скачать том, загруженный другим пользователем. Для обмена вещественного – при загрузке книги пользователю сайта необходимо указать, что книга есть в доступе и договориться об этом в специально созданном общем чате. Пользователь может зарегистрироваться и впоследствии авторизоваться на сайте, чтобы использовать весь функционал приложения. Можно редактировать данные о пользователе, оставлять свое мнение о книгах в комментариях под ними. Должен быть интуитивно понятный интерфейс, реализованы инструменты для удобного поиска нужных книг – поиск и сортировка. Пользователь может создать подборку книг.

Серверная часть приложения должна сохранять всех пользователей, все книги и подборки, сообщения из чата и комментарии. СУБД – PostgreSQL, а все приложение должно быть реализовано на языке Java.

5. Перечень подлежащих разработке вопросов:

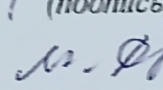
Произвести анализ предметной области: выявить достоинства и недостатки существующих веб-приложений в сфере обмена книгами, сделать выводы, поставить итоговую задачу. Изучить работу с выбранной средой разработки, фреймворком Spring и его дополнениями. Подключить БД и программу администрирования СУБД. Выбрать и реализовать наиболее подходящую архитектуру, разработать и протестировать итоговое веб-приложение.

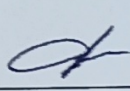
6. Календарный график выполнения работы:

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Изучение предметной области	32	06.05.2025 – 08.05.2025	
2	Анализ требований	32	09.05.2025 – 11.05.2025	
3	Создание макета графического интерфейса	30	12.05.2025 – 14.05.2025	
4	Выбор технологий, применяемых при проектировании и разработке	40	15.05.2025 – 18.05.2025	
5	Реализация графического интерфейса	66	19.05.2025 – 22.05.2025	
6	Разработка основной части программы	70	23.05.2025 – 27.05.2025	
7	Оформление пояснительной записки	54	28.05.2025 – 30.05.2025	

Дата выдачи задания "06" мая 2025 г.

Руководитель бакалаврской работы  06.05.25. Акифьев И.В.
(подпись, дата)

Задание к исполнению принял студент  06.05.25. Давкин М.А.
(подпись, дата)

Работу к защите допустить
Декан факультета  04.06.2025. Фионова Л.Р.
(подпись, дата)

Реферат

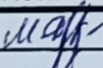
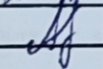
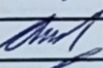
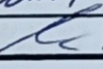
Бакалаврская работа содержит 147 листов, 51 рисунок, 8 таблиц, 6 источников и 3 приложения.

WEB-ПРИЛОЖЕНИЕ «КНИГОВОР», SPRING BOOT, SPRING FRAMEWORK, SPRING SECURITY, INTELLIJ IDEA, POSTGRESQL, PGADMIN4, SQL, CSS, HTML, JAVA

Цель работы – реализовать веб-приложение, позволяющее пользователям обмениваться электронными книгами или предлагать обмен вещественных книг.

Приложение «Книговорот» реализует функционал просмотра книг в браузере и скачивания, создания подборок. После регистрации/авторизации пользователь имеет возможность общаться с другими посетителями сайта в общем чате.

Итогом проведенной работы стало клиент-серверное приложение, полностью удовлетворяющее поставленной цели. Приложение разработано, протестированы его основные функции, реализующие обмен книг между пользователями. Реализация приложения проходила на языке программирования высокого уровня Java, работа с БД проводилась на языке запросов SQL. Использовались такие технологии как, Spring Boot, Spring Framework, Spring Security, интегрированная среда разработки IntelliJ IDEA. Выбран инструмент администрирования PGAdmin4 для СУБД PostgreSQL. Для верстки использовались языки HTML и CSS.

					ПГУ 1.090301.29.001 ПЗ			
Изм.	Лист	№ докум.	Подпись	Дата				
Разраб.		Давкин М.А.		30.05.25	Web-приложение «Книговорот» Пояснительная записка	Лит.	Лист	Листов
Провер.		Акифьев И.В.		30.05.25			2	147
Н. Контр.		Бычков А.С.		30.05.25		ФВТ, каф. ВТ гр. 21ВВП2		
Утверд.		Митрохин М.А.		30.05.25				

Содержание

Перечень сокращений и обозначений	5
Введение	6
1 Анализ предметной области	7
1.1 Анализ актуальности темы	7
1.2 Анализ аналогов	8
1.2.1 Сервис «LiveLib»	8
1.2.2 Сервис «BookCrossing»	9
1.2.3 Сообщества в социальной сети «ВКонтакте»	10
1.2.4 Магазин «Лабиринт»	11
1.2.5 Вывод	12
1.3 Постановка задачи	14
2 Выбор технологий и средств разработки приложения	17
2.1 Spring Framework	17
2.2 Spring Security	18
2.3 Spring JPA	19
2.4 PostgreSQL	20
2.5 PgAdmin4	20
2.6 IntelliJ IDEA	20
2.7 Apache Tomcat	21
2.8 Java	22
2.9 Bootstrap	22
3 Архитектура	24
4 Реализация приложения	27
4.1 Описание работы приложения	27
4.2 Структура БД	28
4.3 Основные компоненты	32
4.3.1 Controller	32
4.3.2 Service	35
4.3.3 Model	40

4.4	Основные алгоритмы взаимодействия клиента и сервера.....	41
4.5	Развертывание приложения	45
5	Тестирование приложения.....	48
5.1	Регистрация.....	48
5.2	Авторизация.....	52
5.3	Редактирование данных пользователя.....	54
5.4	Взаимодействие с книгами авторизованного пользователя	56
5.5	Общение в чате.....	61
5.6	Функционал неавторизованного пользователя.....	62
6	Безопасность жизнедеятельности	64
	Заключение	67
	Список использованных источников.....	68
	Приложение А – UML-диаграммы	69
	Приложение Б – Листинг программы.....	74
	Приложение В – Презентация	140

Перечень сокращений и обозначений

БД – база данных

СУБД – система управления базой данных

AOP (от англ. Aspect-Oriented Programming) – аспектно-ориентированное программирование

API (от англ. Application Programming Interface) – интерфейс прикладного программирования

CSS (от англ. Cascading Style Sheets) – язык декорирования и описания внешнего вида веб-страницы

DI (от англ. Dependency Injection) – механизм внедрения зависимостей

HTML (от англ. HyperText Markup Language) – язык гипертекстовой разметки

HTTP (от англ. HyperText Transfer Protocol) – протокол для передачи данных

IoC (от англ. Inversion of Control) – концепция передачи управления зависимости от кода к системе

JPA (от англ. Java Persistence API) – стандарт языка программирования Java для управления реляционными данными (базами данных)

MVC (от англ. Model-View-Controller) — шаблон программирования, делящий архитектуру приложения на три основных модуля

REST (от англ. Representational State Transfer) – архитектура взаимодействия между клиентом и сервером

SQL (от англ. Structured Query Language) – язык структурированных запросов

UML (от англ. Unified Modeling Language) – язык графического описания для объектного моделирования

URL (от англ. Uniform Resource Locator) – адрес ресурса в Интернет

XML (от англ. eXtensible Markup Language) – расширяемый язык разметки

Введение

В информационном обществе доступ к литературе выступает основным аспектом культурного и интеллектуального развития личности. Несмотря на развитие электронных форматов книг и увеличение количества различных платформ для чтения, существует проблема ограниченного доступа к редким изданиям, отсутствия удобной площадки для обмена как электронными, так и физическими экземплярами книг, а также возможности формирования сообществ читателей со схожими интересами.

Необходимость в создании приложения для эффективного взаимодействия читателей можно обусловить не только потребностью самих книголюбов, но и библиотекарей – людей, чью работу можно заметно облегчить, компьютеризируя и систематизируя практически все их функции.

Разработать одно приложение, объединяющее в себе социальное взаимодействие пользователей, возможность обмена физических экземпляров книг, незатруднительное получение электронной книги, является актуальной проблемой. Она не только отвечает потребностям читателей, но и создает новые возможности для библиотек, обогащая культурную жизнь и повышая уровень доступа к литературе для всех желающих.

					ПГУ 1.090301.29.001 ПЗ	Лист
						6
Изм.	Лист	№ докум.	Подпись	Дата		

1 Анализ предметной области

1.1 Анализ актуальности темы

Потребность в обмене книгами имеет глубокие исторические корни, начиная с первых библиотек Древнего мира. Классические библиотечные системы можно рассматривать как первый институционализированный механизм распространения литературы через временный доступ к физическим экземплярам, а с развитием технологий и появлением интернета концепция книгообмена претерпела значительные трансформации. Первые электронные системы для обмена книгами появились в начале 2000-х гг. и представляли собой преимущественно форумы или доски объявлений, на которых пользователи размещали списки книг для обмена, параллельно с этим развивались первые электронные библиотеки, предоставляющие доступ к оцифрованным версиям произведений.

Последующее развитие технологической инфраструктуры интернета и появление более совершенных программных инструментов привели к формированию специализированных платформ, таких как BookMoosh, PaperbackSwap и BookCrossing, которые автоматизировали процессы поиска, учета и обмена физическими книгами. В России подобные функции частично реализованы на платформах типа LiveLib и в группах различных соцсетей. Одновременно с этим происходило развитие электронных библиотек и сервисов для распространения электронных книг (например, Libgen, Z-Library), которые, однако, часто функционируют в юридически неопределенном статусе из-за проблем с авторскими правами. Так, платформы для обмена книгами зачастую специализируются либо только на электронных форматах, либо только на физических экземплярах, что создает разрозненность в читательском опыте пользователей. Также разработанные системы часто не предоставляют удобные механизмы для коммуникации между пользователями и формирования персонализированных подборок литературы. Разработка веб-приложения для

					ПГУ 1.090301.29.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		7

обмена книгами актуальна также в контексте популяризации чтения и создания доступной среды для обмена книжными ресурсами, участие в сообществах читателей повышает мотивацию к чтению и расширяет литературный кругозор участников.

1.2 Анализ аналогов

В настоящее время на рынке представлены разные платформы для обмена книгами, каждая имеет свои особенности и недочеты. В данном пункте рассмотрены актуальные сервисы по схожей тематике: «LiveLib», «BookCrossing», сообщества в социальной сети «ВКонтакте», «Лабиринт».

1.2.1 Сервис «LiveLib»

LiveLib — крупнейшая в России социальная сеть для любителей книг. Платформа представляет собой гибрид рекомендательного сервиса, каталога книг и социальной сети с элементами обмена физическими книгами. Интерфейс сервиса «LiveLib» представлен на рисунке 1.

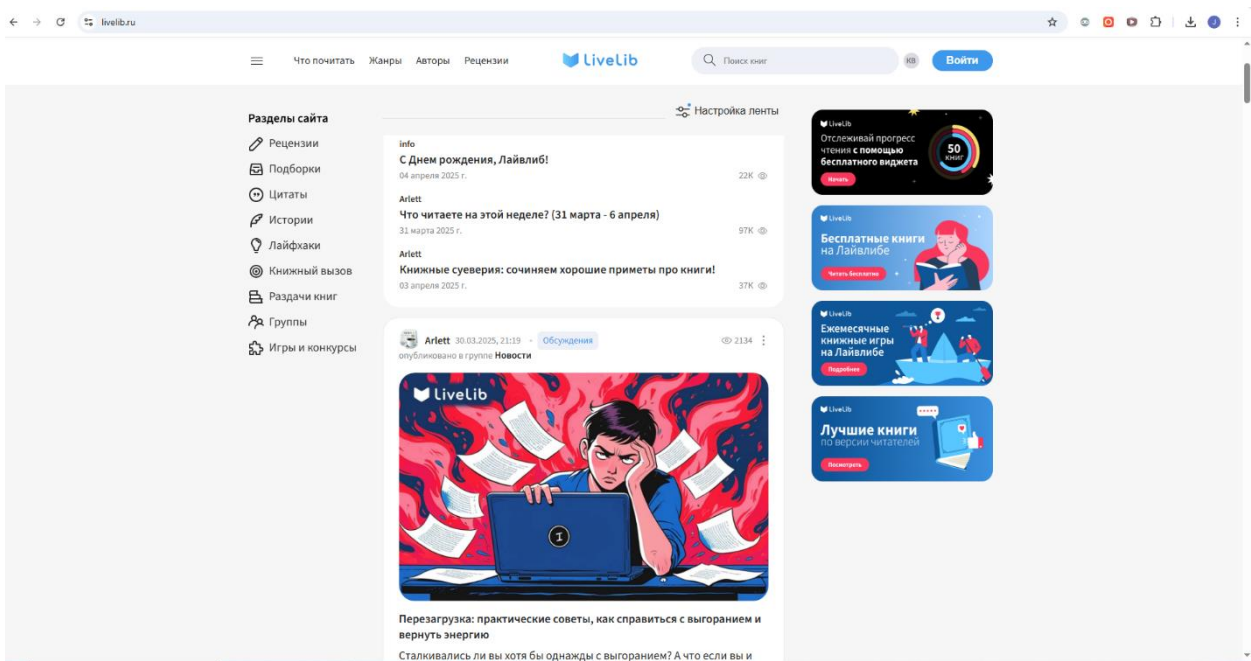


Рисунок 1 – Официальный сайт «LiveLib»

Преимущества «LiveLib»:

– крупнейшее сообщество книголюбов в РФ;

- развитая система рецензий и оценок, помогающая в выборе книг;
- возможность создания персональных книжных полок и подборок;
- функционал «Хочу почитать» и «Читаю», помогающий организовать чтение;
- система мотивации в виде «достижений» и статистики;
- ежегодные конкурсы и книжные челленджи.

Недостатки «LiveLib»:

- обмен книгами реализован как вторичный функционал и не всегда удобен;
- отсутствие встроенной системы сообщений для коммуникации по обмену;
- нет возможности обмена или скачивания электронных книг;
- перегруженный интерфейс с множеством функций;
- отсутствие эффективной системы фильтрации пользователей для обмена;
- нет локального поиска книг (по городам/районам).

1.2.2 Сервис «BookCrossing»

«BookCrossing» — это международное движение по обмену книгами, основанное на принципе «освобождения» книг; пользователи регистрируют книги на сайте, получают уникальный код, оставляют книги в общественных местах и отслеживают их путешествие.

Преимущества «BookCrossing» могут быть выделены следующие:

- уникальная концепция «путешествующих» книг;
- глобальное сообщество с представительством в России;
- отсутствие прямых финансовых затрат;
- система отслеживания перемещения книг через уникальные коды;
- экологический аспект повторного использования книг.

Недостатки «BookCrossing»:

- низкая вероятность получения конкретной желаемой книги;

- слабая активность движения в большинстве регионов России;
- отсутствие прямого обмена между пользователями;
- проблемы с сохранностью «освобожденных» книг;
- отсутствие гарантий качества получаемых книг.

Интерфейс сервиса представлен на рисунке 2.

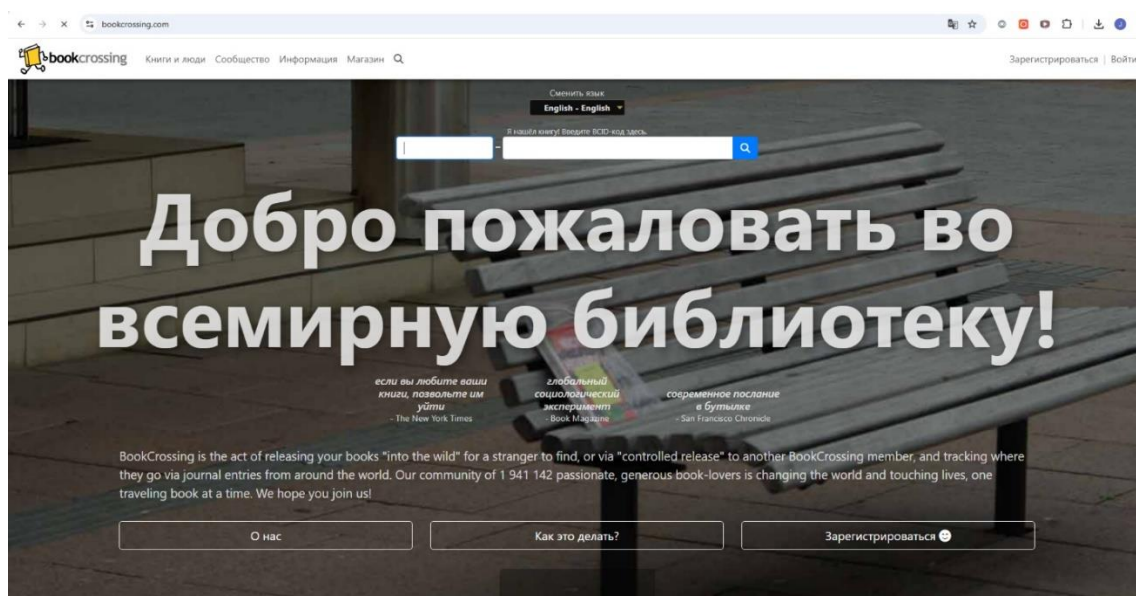


Рисунок 2 – Официальный сайт «BookCrossing»

1.2.3 Сообщества в социальной сети «ВКонтакте»

Сообщества в социальной сети ВКонтакте, созданные для обмена физическими книгами, обычно организованы по географическому принципу (городам, районам) или тематическим интересам.

Преимущества:

- использование популярной и привычной для россиян социальной платформы;
- встроенная система личных сообщений и уведомлений;
- возможность создания локальных групп для удобного офлайн-обмена;
- фотографии книг и публичные профили пользователей;
- формирование тематических сообществ по жанрам и интересам;
- отсутствие необходимости регистрации на дополнительных сервисах.

Недостатки:

- отсутствие единой системы классификации и каталогизации книг;
- нет специализированных инструментов для обмена (отслеживание статуса, рейтинги);
- децентрализация (множество разрозненных групп);
- низкий уровень модерации и отсутствие гарантий безопасного обмена;
- нет системного подхода к организации коллекций;
- обмен ограничен только печатными книгами.

На рисунке 3 показан интерфейс одного из сообществ.

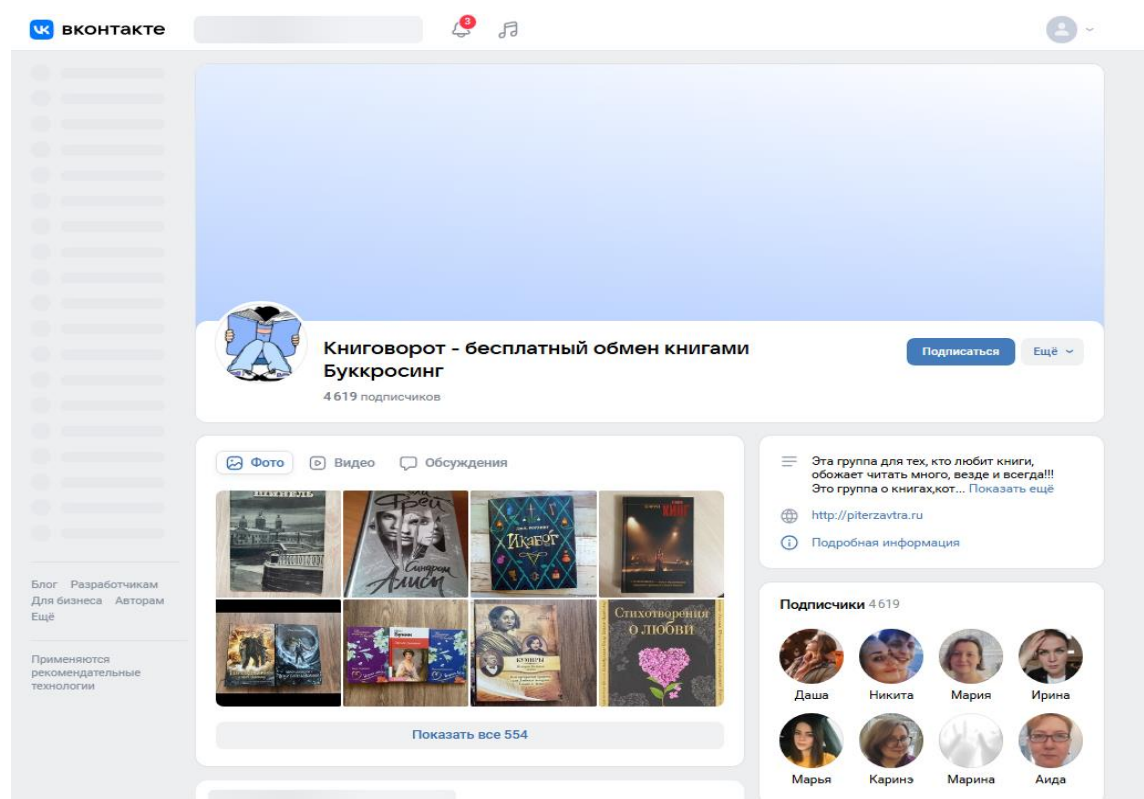


Рисунок 3 – Сообщество в социальной сети ВКонтакте

1.2.4 Магазин «Лабиринт»

Лабиринт (раздел «Обменяю/Подарю») – это крупный российский интернет-магазин книг с дополнительным сервисом, позволяющим пользователям совершать обмен или дарить книги из своих коллекций. Сайт приложения представлен на рисунке 4.

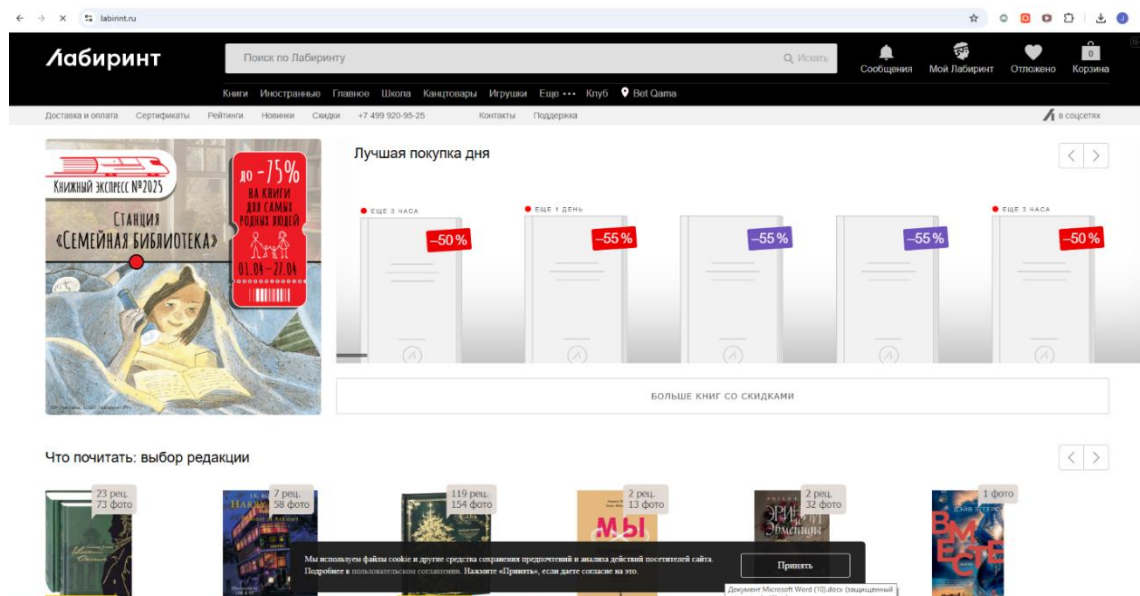


Рисунок 4 – Официальный сайт «Лабиринт»

Преимущества:

- интеграция с популярным книжным магазином;
- надежность и официальный статус платформы;
- большая аудитория активных читателей;
- рейтинговая система пользователей;
- удобный каталог с подробным описанием книг;
- возможность комбинировать покупку и обмен.

Недостатки:

- функция обмена является вторичной для платформы;
- сложная система активации функционала обмена (требуется история покупок);
- отсутствие прямого чата между пользователями;
- нет возможности обмена электронными книгами;
- часто встречаются неактуальные объявления;
- географическая ограниченность (в основном крупные города).

1.2.5 Вывод

Основные выявленные недостатки существующих решений: отсутствие единой платформы для обмена как электронными, так и физическими книгами,

недостаточно развитые инструменты коммуникации между пользователями, ограниченные возможности для создания тематических подборок отсутствие полноценного функционала для чтения электронных книг в браузере. Недостатки подтверждают актуальность разработки универсальной платформы, объединяющей возможности обмена электронными и физическими книгами с функциями социальной сети для читателей. Общий сравнительный анализ сервисов представлен в таблице 1.

Таблица 1 – Сравнительный анализ существующих платформ для обмена книгами в РФ

Название	Основной функционал	Преимущества	Недостатки
LiveLib	Рецензии, рейтинги, подборки, обмен бумажными книгами	<ul style="list-style-type: none"> - Большое сообщество - Богатая база рецензий - Интеграция с соцсетями - Возможность создания подборок 	<ul style="list-style-type: none"> - Ограниченный функционал обмена - Нет электронных книг для скачивания - Сложный интерфейс для новичков
БукКроссинг	Регистрация, отслеживание книг, "освобождение" книг в общественных местах	<ul style="list-style-type: none"> - Международная система - Бесплатное использование 	<ul style="list-style-type: none"> - Нет гарантии получения книги - Отсутствие прямой коммуникации - Нет возможности обмена электронными книгами
Лабиринт (раздел "Обменяю/Подарю")	Покупка, продажа, обмен печатных книг	<ul style="list-style-type: none"> - Большая аудитория - Надежность - Интеграция с магазином - Отзывы о пользователях 	<ul style="list-style-type: none"> - Нет обмена электронными книгами - Часто неактивные объявления - Нет прямого чата
ReadRate	Рецензии, рейтинги, отзывы, ограниченный обмен	<ul style="list-style-type: none"> - Интуитивный интерфейс - Хорошие 	<ul style="list-style-type: none"> - Слабо развитый раздел обмена - Ориентация на

Название	Основной функционал	Преимущества	Недостатки
		рекомендации - Возможность создания коллекций - Интеграция с магазинами	отзывы, а не обмен - Нет обмена электронными книгами - Малая активность пользователей
ВКонтакте (группы обмена книгами)	Объявления обмена/дарения книг, обсуждения	- Большая аудитория - Удобная коммуникация - Возможность создания локальных групп - Бесплатно	- Отсутствие гарантий - Нет структурированной базы книг - Преимущественно офлайн-обмен
Авито (раздел "Книги")	Продажа/обмен печатных книг	- Обширная аудитория - Геолокация - Рейтинг продавцов - Простой интерфейс	- Ориентация на продажу, а не обмен - Нет фильтров для книг - Отсутствие тематического сообщества - Нет обмена электронными книгами

1.3 Постановка задачи

Функционал разрабатываемого приложения «Книговорот» отличен для пользователя авторизованного и неавторизованного.

Неавторизованный пользователь имеет ограниченный набор функций – ему позволено ознакомиться с внешним видом сайта и просмотреть все книги, загруженные на сайт (без скачивания или обмена). Функционал:

- регистрация;
- авторизация;
- просмотр списка книг, загруженных на сайт;
- просмотр электронных книг.

Диаграмма вариантов использования приложения для неавторизованного пользователя представлена на рисунке 5.

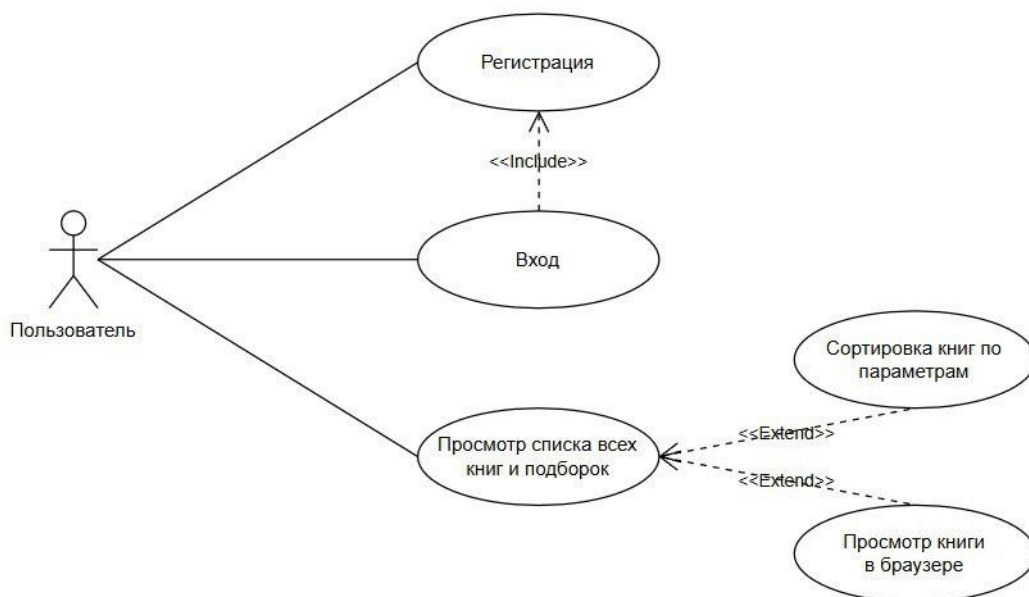


Рисунок 5 – UML-диаграмма вариантов использования неавторизованного пользователя

Функционал авторизованного пользователя значительно шире:

- просмотр списка книг, загруженных на сайт, сортировка;
- просмотр тематических подборок книг;
- просмотр электронных книг, скачивание;
- редактирование данных пользователя;
- создание собственных тематических подборок;
- загрузка своих книг на сайт;
- общение в чате с другими пользователями;
- возможность комментирования книг, загруженных пользователями.

Диаграмма вариантов использования приложения для неавторизованного пользователя представлена на рисунке 6.

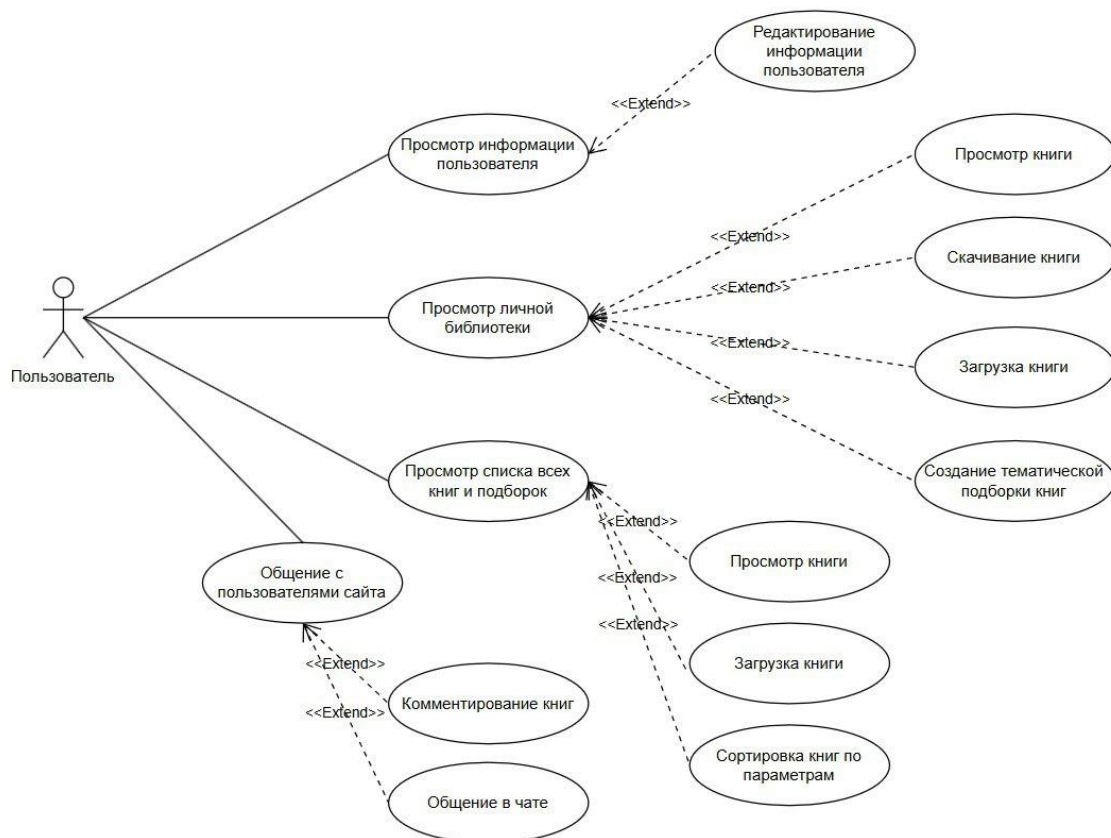


Рисунок 6 – UML-диаграмма вариантов использования авторизованного пользователя

2 Выбор технологий и средств разработки приложения

Для разработки клиент-серверного приложения «Книговорот» были выбраны технологии:

- Spring Framework;
- Spring JPA;
- Spring Security;
- PostgreSQL;
- pgAdmin4;
- IntelliJ IDEA;
- Apache Tomcat;
- Java;
- Bootstrap.

2.1 Spring Framework

Spring Framework — это фреймворк с открытым исходным кодом и основанная на Java платформа, предоставляющая полную поддержку инфраструктуры для создания корпоративных Java-приложений. Таким образом, разработчики не должны думать об инфраструктуре приложения и могут сконцентрироваться на его бизнес-логике, а не конфигурации [1]. С 2003 г. (дата появления фреймворка) Spring эволюционировал от контейнера внедрения зависимостей до комплексной экосистемы, охватывающей различные аспекты разработки приложений. В основе Spring Framework стоит принцип Инверсии управления (IoC), где контроль за созданием объектов, управлением их жизненным циклом осуществляет фреймворк. Spring-контейнер управляет бинами (объектами Java) через внедрение зависимостей — связанность компонентов слабее и, как следствие, улучшена тестируемость.

Контейнер состоит из нескольких модулей:

- основной контейнер предоставляет фундаментальные функции IoC и внедрения зависимостей, включая реализации BeanFactory и ApplicationContext;

– АОР (Аспектно-ориентированное программирование) отделяет сквозную функциональность от бизнес-логики;

– доступ к данным облегчает взаимодействие с технологиями персистентности, такими как JDBC, JPA и Hibernate;

– Web предлагает архитектуру MVC для разработки веб-приложений и RESTful API.

– интеграция с корпоративными системами обмена сообщениями.

Spring Framework выбран для разработки веб-приложений благодаря:

– стабильности, высокой эффективности разработки;

– экосистеме, которая решает различные аспекты разработки приложений;

– функциям корпоративного уровня, поддержке транзакций, безопасности, кэширования;

– фреймворк может использоваться в различных сценариях развертывания;

– фреймворк постоянно совершенствуется и адаптируется к современным практикам разработки.

Spring Framework обеспечивает основу для создания масштабируемых, поддерживаемых и безопасных веб-приложений, благодаря чему был выбран для решения нашей задачи.

2.2 Spring Security

Spring Security — это мощный фреймворк аутентификации и контроля доступа, который осуществляет защиту ресурсов и обеспечение доступа к определенным функциям только авторизованным пользователям.

Фреймворк предоставляет несколько основных компонентов:

– Authentication Manager проверяет учетные данные пользователя и создает токены аутентификации;

– Access Decision Manager определяет, имеет ли пользователь разрешение на доступ к ресурсу;

– UserDetailsService извлекает сведения о пользователе из хранилища данных;

– PasswordEncoder безопасно хеширует и проверяет пароли.

Spring Security является предпочтительным выбором для обеспечения безопасности, потому что будет обеспечена нативная работа с Spring Framework. Благодаря фреймворку мы можем реализовать надежные меры безопасности с минимальными усилиями, защитив приложение от распространенных угроз при сохранении удобства использования и производительности.

2.3 Spring JPA

Spring Data JPA является важным компонентом фреймворка, который облегчает доступ к базе данных и операциям персистентности. Он предоставляет абстрактный слой над Java Persistence API (JPA) и сокращает шаблонный код для доступа к данным.

Spring Data JPA работает на архитектурных уровнях:

– абстракция репозитория определяет высокоуровневый интерфейс для операций с данными;

– методы запросов автоматически генерируют запросы на основе имен методов;

– интеграция с JPA работает с Hibernate, EclipseLink или другими провайдерами JPA;

– управление транзакциями интегрируется с транзакционной инфраструктурой Spring.

Spring Data JPA выбран для персистентности данных из-за:

– повышения продуктивности (сокращает шаблонный код);

– стандартизации шаблонов доступа к данным;

– проверки запросов во время компиляции;

– интеграции с различными СУБД;

– хорошей документации.

С Spring Data JPA, можно сосредоточиться на бизнес-логике, а не на коде доступа к данным, что ускоряет разработку приложения, увеличивает поддерживаемость.

2.4 PostgreSQL

PostgreSQL – это расширенная объектно-реляционная система управления базами данных, которая славится надежностью, производительностью, функциональной устойчивостью. Данные в PostgreSQL хранятся в таблицах, которые могут содержать объекты, такие как строки, столбцы и индексы.

Благодаря технологии, позволяющей работать с данными в формате JSON и JSONB, PostgreSQL сочетает в себе такие преимущества традиционных СУБД, как транзакционность, атомарность изменений и целостность данных, с гибкостью NoSQL без потери производительности [2].

2.5 PgAdmin4

PgAdmin — это платформа управления и администрирования баз данных PostgreSQL. Как наиболее популярный и богатый инструмент администрирования с открытым исходным кодом, он предоставляет мощный интерфейс для прокитировщиков баз данных на PostgreSQL.

PgAdmin предлагает комплексные функции управления:

- создание, редактирование и удаление баз данных, представлений, схем, таблиц и др.;
- просмотр, редактирование, импорт и экспорт данных таблиц;
- запуск произвольных запросов с автодополнением;
- управление пользователями и ролями, создание и управление пользователями базы данных и привилегиями и др.

2.6 IntelliJ IDEA

IntelliJ IDEA — это интегрированная среда разработки (IDE), созданная для разработки на Java, но поддерживающая множество других языков

программирования. Среда разработана компанией JetBrains и на сегодняшний день представляет собой один из лучших инструментов для профессиональной разработки программного обеспечения.

IDE включает анализ кода, компиляцию, отладку, инструменты для разработки пользовательского интерфейса, движок рефакторинга поддерживает операции реструктуризации кода.

IntelliJ IDEA дает возможности комплексной поддержки разработки на Java:

- контекстно-зависимые предложения кода;
- обнаружение ошибок и исправления в реальном времени;
- обширные возможности рефакторинга;
- инструменты для корпоративной разработки Java;
- поддержка Spring, Hibernate и др.;
- встроенный декомпилятор для просмотра библиотек;

Среда поддерживает веб-разработку:

- редактирование HTML/CSS/JavaScript;
- поддержка TypeScript;
- интеграция с веб-фреймворками Angular, React, Vue.js;
- REST-клиент для тестирования API;
- инструменты базы данных;
- мгновенное обновление браузера во время разработки;
- JavaScript-отладчик.

2.7 Apache Tomcat

Apache Tomcat — это легковесный веб-сервер и контейнер сервлетов с открытым исходным кодом, который реализует спецификации Java Servlet, JavaServer Pages (JSP), Java Expression Language и WebSocket. Сейчас Tomcat широко используется и заслужил огромную популярность в силу стабильности и эффективности для развертывания веб-приложений Java.

2.8 Java

Java является одним из наиболее влиятельных и широко принятых языков программирования. Со своего представления компанией Sun Microsystems (сейчас принадлежит Oracle) в 1995 году, Java стал универсальным платформонезависимым языком, который используется во всем: от веб-приложений до корпоративных систем, мобильных устройств и встроенных систем.

Архитектура Java построена на нескольких принципах:

- объектно-ориентированное программирование со всеми его принципами: инкапсуляция, наследование, полиморфизм и абстракция;
- независимость от платформы: «Написано однажды, работает везде» благодаря JVM;
- типизация: тщательная проверка типов при компиляции;
- автоматическое управление памятью: сборка мусора устраняет необходимость ручного выделения и освобождения памяти;
- структурированный подход к управлению ошибками;
- возможности параллельного программирования;
- многоуровневая защита от вредоносного кода.

Платформа Java состоит из трех основных компонентов:

- язык программирования Java: синтаксис и семантика языка;
- Java Virtual Machine (JVM) исполняет байт-код Java;
- Java API (библиотеки) – большой набор написанных классов и интерфейсов.

2.9 Bootstrap

Bootstrap — это ведущий фреймворк с открытым исходным кодом для разработки отзывчивых, ориентированных на мобильные устройства веб-интерфейсов.

Созданный в Twitter и выпущенный в 2011 году, революционизировал разработку пользовательских интерфейсов, предоставляя стандартизированную библиотеку компонентов и утилит, значительно ускоряющую процесс разработки и обеспечивающую согласованность дизайна.

					ПГУ 1.090301.29.001 ПЗ	Лист
						23
Изм.	Лист	№ докум.	Подпись	Дата		

3 Архитектура

В третьем разделе изложена архитектура создаваемого приложения «Книговорот». Паттерном разработки выбрана достаточно распространенная ныне архитектура MVC.

Архитектура MVC (Model, View, Controller) – шаблон проектирования приложения, при котором функционал делится на три компонента: модель (Model), представление (View), контроллер (Controller). Данный паттерн упрощает разработку приложения, позволяя изменять его различные части независимо друг от друга, не нарушая при этом работоспособности всей системы. Взаимодействие пользователя с MVC приложением следует естественному циклу: пользователь совершает действие, в ответ на это приложение меняет свою модель данных и предоставляет пользователю обновленный вид. А затем цикл повторяется [3].

В модели заложена бизнес-логика приложения. Контроллер посылает команды модели, как та обрабатывает данные, размещает записи в БД, читает из БД, выдает нужную информацию. В «Книговороте» модель отвечает за хранение данных о книгах, пользователях, сохраняет сообщения из чата и т.д. После произведенных действий, модель передает сформированные данные в представление.

Представление – воспроизведение информации от модели в виде пользовательского интерфейса (User Interface, UI). Представление отражает HTML-код, с помощью которого приложение взаимодействует непосредственно с пользователем.

Контроллер – связующее звено между всеми компонентами системы. Классы контроллера получают данные запроса к серверу и отправляют их в модель для дальнейшей обработки и сохранения в БД. Далее контроллер с помощью представления выбирает, каким именно способом выдать данные клиенту, после чего формируется HTML-ответ.

Архитектура приложения «Книговорот» показана на рисунке 7.

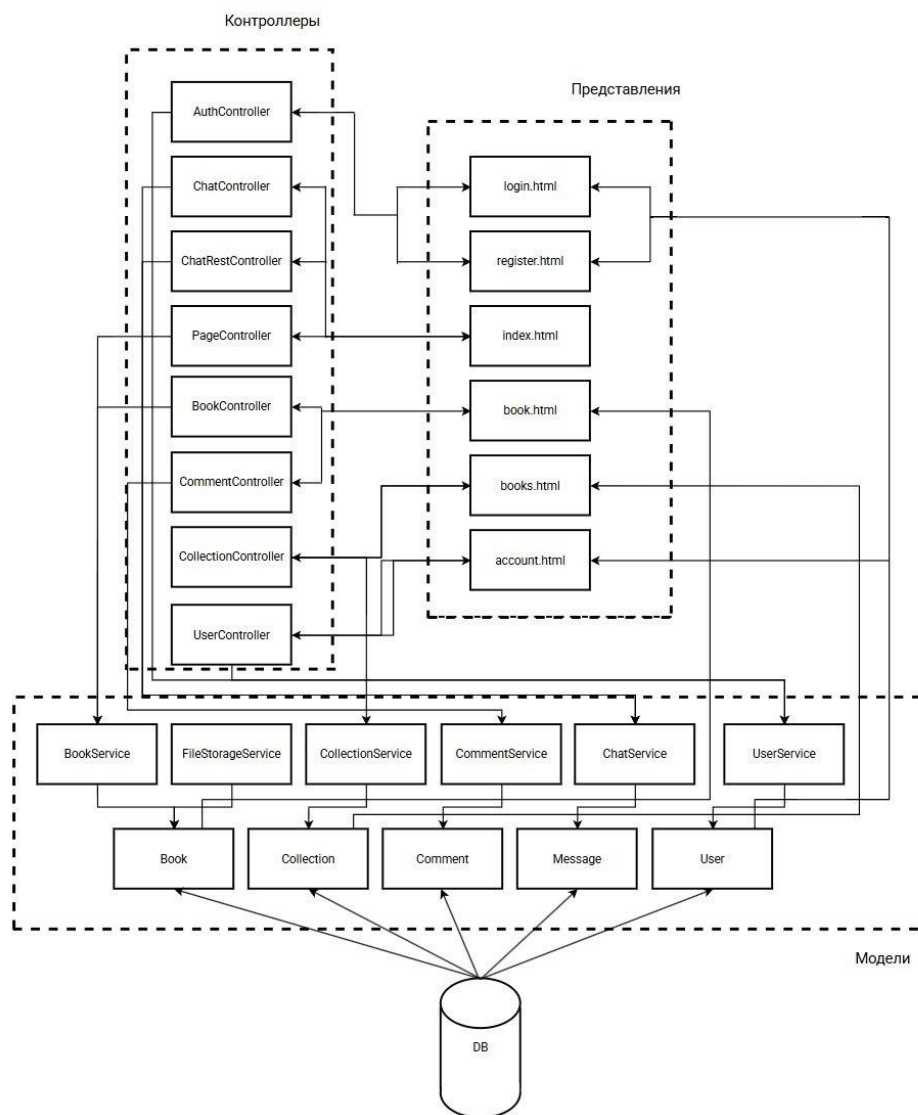


Рисунок 7 – Архитектура приложения «Книговорот»

AuthController берет информацию о данных пользователя (логин, пароль при авторизации; фамилия, имя, логин, пароль при регистрации) с слоя представления – login.html и register.html, после чего передает их в слой модели и бизнес-логики для обработки – User и UserService.

ChatController получает сообщения от клиентов в чате со слоя представления index.html и отдает их на сохранение в БД модели Message и сервису ChatService. ChatRestController возвращает историю последних сообщений чата, используя те же модель и сервис.

CommentController получает запросы на добавление нового комментария и подгружает оставленные ранее (представление – book.html). Сервис CommentService и модель Comment обрабатывают нужный запрос.

PageController использует подгружает полные данные о книге и кладет их в модель Book через сервис BookService. BookController, получая данные об операции, запрашиваемой пользователем с book.html (например, скачать книгу), обращается к модели Book, выполняя нужные действия в BookService. Для сохранения файла необходима реализация отдельного сервиса – FileStorageService.

CollectionController отвечает за операции над подборками книг – возвращает подборки пользователя и выводит все загруженные на сайт коллекции. Представление коллекций реализовано в файле books.html, а управление из контроллера передается в модель Collection, она отдает запрос в сервисный слой – CollectionService.

UserController обрабатывает запросы на получение и изменение данных пользователя (фамилия, имя, пароль). Контроллер получает их с представления – account.html и передает в модель User. Для выполнения операций использует UserService.

С базой данных приложение взаимодействует через модели – Book, User, Collection, Comment, Message – так же называются основные таблицы БД.

4 Реализация приложения

В данном разделе рассмотрен процесс разработки веб-приложения «Книговорот»: детально описаны основные компоненты приложения, создание и взаимодействие с сущностями в БД, реализация заданного функционала приложения для авторизованных и неавторизованных пользователей (гостей). Здесь полные сведения о том, как разработано клиент-серверное приложение «Книговорот».

4.1 Описание работы приложения

Последовательность выполняемых действий в приложении представлена на рисунке 8.

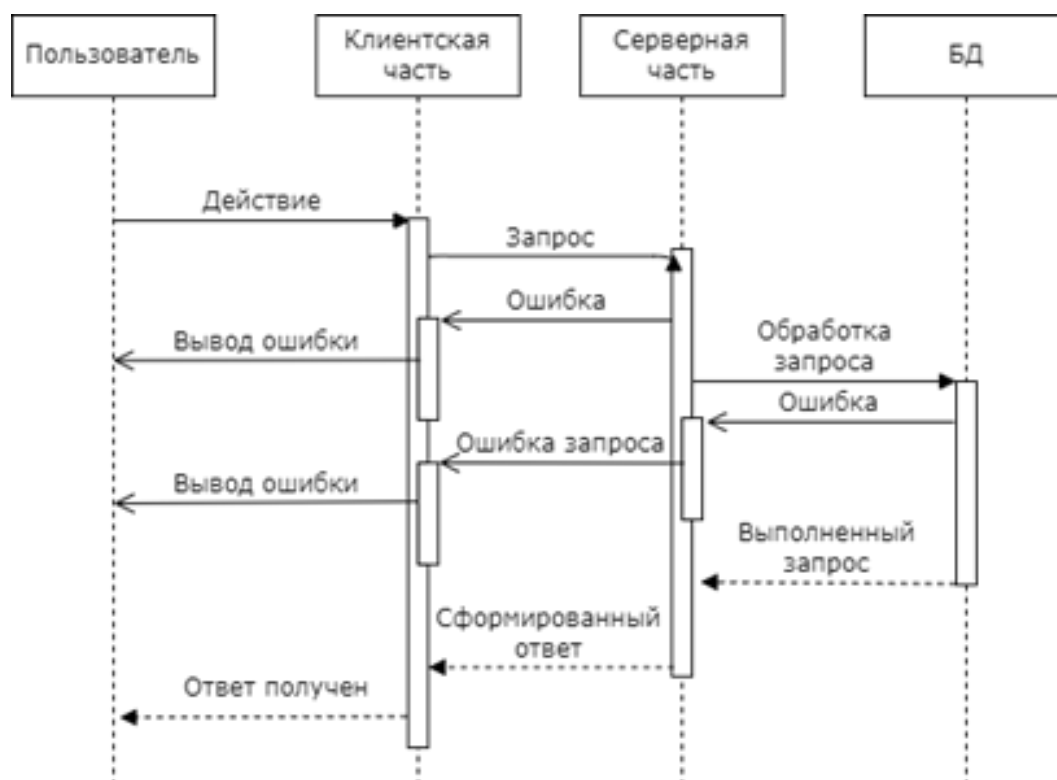


Рисунок 8 – UML-диаграмма последовательностей

В простейшем виде алгоритм работы приложения состоит в следующем: на клиентской части (HTML, CSS, Thymeleaf) пользователь производит некоторые действия, после чего сервер получает запрос на выполнение. Если по какой-либо причине операцию сервер выполнить не может, клиенту

возвращается информация в виде уведомления/сообщения о том, что операция прервана. Если же запрос успешно принят, то после обращения к БД и выполнения необходимых действий, сервер может вернуть пользователю данные, создать или изменить их. Ошибки могут появиться на любом этапе выполнения запроса (при создании сущности в БД, например), но всегда они будут обработаны сервером, а пользователь получит уведомление об ошибке, возможно с пояснением причин ее возникновения.

4.2 Структура БД

Для хранения информации в разрабатываемом веб-приложении «Книговорот» необходимо сконструировать и проанализировать базу данных, в которой будет находиться вся информация о зарегистрированных пользователях, загруженных книгах, комментариях, сообщениях, созданных подборках. В данном подразделе будут подробно рассмотрены таблицы нашей БД, их атрибуты и связи.

Основные таблицы:

- books (структура представлена в таблице 2) хранит всю информацию о загружаемых книгах: название, автор, год издания и др.;
- collections (структура представлена в таблице 3) содержит информацию о создаваемых пользователями подборках;
- users (структура представлена в таблице 4) содержит в себе всю информацию о пользователях сайта;
- messages (структура представлена в таблице 5) хранит отправленные в чате сообщения, их отправителя и время отправки;
- comments (структура представлена в таблице 6) располагает в себе оставленные пользователями комментарии под книгами (соответственно, информацию о самой книге, о пользователе, который высказал свое мнение).

Таблица 2 – Атрибуты сущности «books» (Книги)

Поле	Тип данных	Описание	Ключ	Пример
id	BIGINT	Идентификатор книги	PK	5
author	VARCHAR(255)	Автор		Федор Достоевский
created_at	TIMESTAMP(6)	Время загрузки		2025-05-11 19:47:02.641721
file_name	VARCHAR(255)	Название файла с книгой		Иванов
physical	BOOLEAN	Метка о книге «офлайн»		true
title	VARCHAR(255)	Название		Преступление и наказание
year_written	INTEGER	Год издания		1866
uploader_id	BIGINT	Id пользователя	FK	4

Таблица 3 – Атрибуты сущности «collections» (Подборки)

Поле	Тип данных	Описание	Ключ	Пример
id	BIGINT	Идентификатор книги	PK	2
name	VARCHAR(255)	Название		Русские книги
owner_id	BIGINT	Id пользователя	FK	3

Таблица 4 – Атрибуты таблицы «users» (Пользователи)

Поле	Тип данных	Описание	Ключ	Пример
id	BIGINT	Идентификатор пользователя	PK	19
first_name	VARCHAR(255)	Фамилия		Егоров
last_name	VARCHAR(255)	Имя		Егор

Поле	Тип данных	Описание	Ключ	Пример
password	VARCHAR(255)	Пароль		Egor999!)
username	VARCHAR(255)	Логин		egor2002

Таблица 5 – Атрибуты сущности «messages» (Сообщения)

Поле	Тип данных	Описание	Ключ	Пример
id	BIGINT	Идентификатор сообщения	PK	9
send_at	TIMESTAMP(6)	Время отправки		2025-05-11 19:47:02.641721
text	TEXT	Текст		ну здорова
author_id	BIGINT	Id отправителя	FK	8

Таблица 6 – Атрибуты сущности «comments» (Комментарии)

Поле	Тип данных	Описание	Ключ	Пример
id	BIGINT	Идентификатор комментария	PK	3
created_at	TIMESTAMP(6)	Время отправки		2025-05-11 19:47:02.641721
text	TEXT	Текст		ну здорова
user_id	BIGINT	Id отправителя	FK	2
book_id	BIGINT	Id книги	FK	4

БД содержит и «вспомогательные» таблицы:

- collection_books (структура представлена в таблице 7) соотносит идентификатор книги с идентификатором коллекции;
- user_roles (структура представлена в таблице 8) содержит информацию о роли посетителя сайта.

Таблица 7 – Атрибуты сущности «collection_books» (Подборка книг)

Поле	Тип данных	Описание	Ключ	Пример
collection_id	BIGINT	Идентификатор подборки	FK	14
book_id	BIGINT	Идентификатор книги	FK	9

Таблица 8 – Атрибуты сущности «user_roles» (Роли пользователей)

Поле	Тип данных	Описание	Ключ	Пример
user_id	BIGINT	Идентификатор пользователя	FK	14
roles	VARCHAR(255)	Роль		ROLE_USER

Ниже на рисунке 9 отражена схема БД приложения «Книговорот».

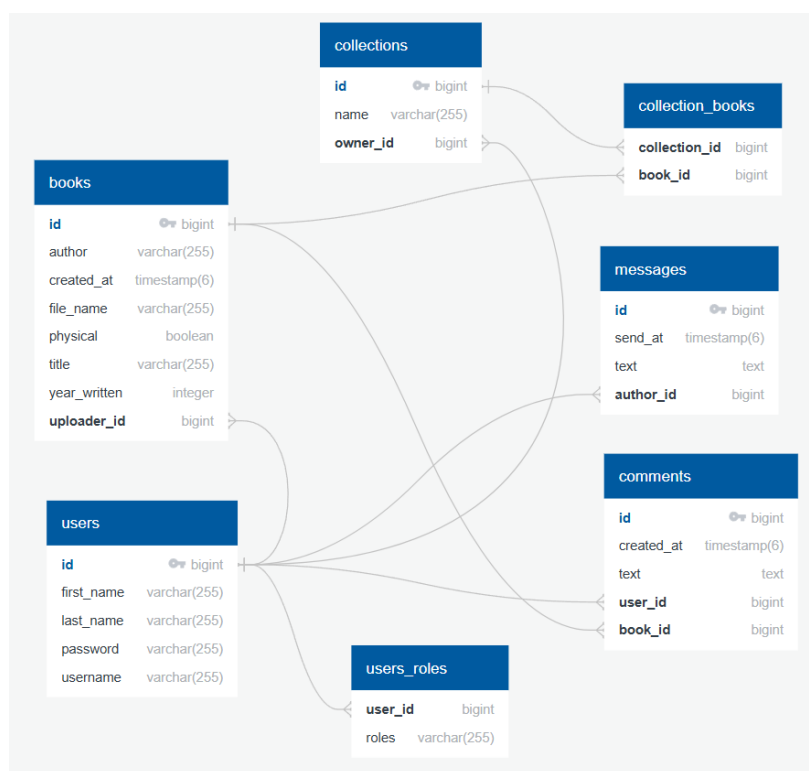


Рисунок 9 – Схема БД

4.3 Основные компоненты

Ранее уже было рассмотрено, что основной архитектурой приложения была выбрана MVC (модель-представление-контроллер). Однако приложения, разработанные на Spring, добавляют еще несколько слоев для упрощения программирования: Service и Repository (контроллер отдает данные клиента сервису, чтобы тот произвел необходимые операции, а репозиторий уже получает результаты от сервиса и отправляет их в БД). В данном разделе подробно описывается, каким образом были реализованы слой контроллера, сервис и модель.

4.3.1 Controller

Контроллер устанавливает, как компонент реагирует на действия пользователя. Скажем, когда пользователь щелкает на флажке, контроллер реагирует изменением модели, отражая выбор пользователя (отмечен или не отмечен), что приводит к обновлению представления [4]. В «Книговороте» таким примером может служить проставление флажка «книга офлайн», если пользователь загружает книгу для физического обмена.

Возьмем BookController, который работает с моделью Book; ее назначение – сохранить всю информацию о томах в БД. Класс содержит методы:

- getBooks();
- getLatest();
- downloadBook();
- getBookById();
- addBook().

Первый метод getBooks() по сути является get-запросом, который получает книгу из БД по названию, автору и свойству – электронный том или вещественный. за получение книги из библиотеки пользователя по её идентификатору. Функция отвечает за поиск по критериям (название или автор), сортировку (только «офлайн» или «онлайн» книги) и вывод списка на вкладке «Книги».

Листинг 1. getBooks():

```
@GetMapping
public List<BookDto> getBooks(
    @RequestParam(required = false) String title,
    @RequestParam(required = false) String author,
    @RequestParam(required = false) Boolean physical
) {
    return bookService.searchBooks(title, author, physical);
}
```

Метод getLatest() также реализует get-запрос, который получает последние загруженные книги на сайт. Метод возвращает функцию, реализованную в модели, с заданным числом томов, необходимых для отображения – 10.

Листинг 2. getLatest():

```
@GetMapping("/latest")
public List<BookDto> getLatest(
    @RequestParam(defaultValue = "10") int count
) {
    return bookService.findLatest(count);
}
```

Метод downloadBook() являет собой обработчик get-запроса, отвечающий за скачивание электронной книги с сайта по ее уникальному идентификатору. Если файл будет найден, то данный метод создаст объект ResponseEntity <Resource> (из Spring Framework), в теле которого будет ответ в виде массива байтов с заголовком и именем файла для загрузки.

Листинг 3. downloadBook():

```
@GetMapping("/{id}/download")
public ResponseEntity<Resource> downloadBook(@PathVariable Long
id) {
    Resource file = bookService.loadBookFile(id);
    return ResponseEntity.ok()
        .header(HttpHeaders.CONTENT_DISPOSITION,
            "attachment; filename=\"" + file.getFilename()
+ "\"")
}
```

```

        .body(file);
    }

```

Метод `getBookById()` отвечает за получение подробностей по конкретной книге (данный метод вспомогательный). Аннотация `@PathVariable` перед параметром метода указывает, что идентификатор для поиска извлекается из пути URL (пример для идентификатора 5: <https://booklover.ru/5>).

Листинг 4. `getBookById()`:

```

@GetMapping("/{id}")
public BookDto getBookById(@PathVariable Long id) {
    return bookService.getBookById(id);
}

```

Метод `addBook()` обрабатывает post-запрос, отвечающий за загрузку книги на сайт. Метод отмечен аннотацией `@ResponseStatus`, определяющей код состояния в ответе, отправляемом клиенту. Возвращается код состояния HTTP 201 (создано), свидетельствующий, что ресурс был успешно создан [5]. Аннотация `@RequestPart` позволяет преобразовать часть запроса в объект (например, файл книги). Метод отдает управление в сервисный слой (слой модели), передавая параметрами вызов функции получения идентификатора пользователя, данные о книге и файл (который может отсутствовать, если подразумевается вещественная книга).

Листинг 5. `addBook()`:

```

@PostMapping(consumes = "multipart/form-data")
@ResponseStatus(HttpStatus.CREATED)
public BookDto addBook(
    @AuthenticationPrincipal UserDetails user,
    @RequestPart("metadata") CreateBookRequest request,
    @RequestPart(value = "file", required = false) MultipartFile
    file
) {
    return bookService.createBook(user.getUsername(), request,
    file);
}

```

4.3.2 Service

Сервис, в сущности, представляет собой обработчик данных, которые поступают к нему из контроллера. Он отвечает за взаимодействие сущностей проекта, обрабатывает исключения, работает с DTO (Data Transfer Object – элемент передачи данных между частями приложения). Здесь рассмотрен класс `BookServiceImpl` (реализация интерфейса `BookService`; в данном проекте она одна, потому она не является обязательной, но если потребуется изменить логику класса или метода, который использует объект интерфейса, то имплементация станет необходимой), содержащий методы:

- `searchBooks()`;
- `findLatest()`;
- `loadBookFile()`;
- `createBook()`;
- `getBookById()`;
- `getHtmlBook()`;
- `toDto()`.

Метод `searchBooks()` принимает параметрами название, автора и пометку, физическая книжка или электронная. Создается выборка, определяется корневая таблица (объект запроса), объявляется список условий (`predicates`). Далее идет набор условий в зависимости от запроса пользователя – требуется найти книгу по названию, автору, вещественную или электронную; после чего все набранные условия добавляются в запрос к БД. Результатом будет список объектов класса `Book`.

Листинг 6. `searchBooks()`:

```
public List<BookDto> searchBooks(String title, String author,
Boolean physical) {
    CriteriaBuilder cb = entityManager.getCriteriaBuilder();
    CriteriaQuery<Book> query = cb.createQuery(Book.class);
    Root<Book> book = query.from(Book.class);
    List<Predicate> predicates = new ArrayList<>();
```

```

if (title != null && !title.isEmpty()) {
    predicates.add(cb.like(cb.lower(book.get("title")), "%" +
title.toLowerCase() + "%"));
}
if (author != null && !author.isEmpty()) {
    predicates.add(cb.like(cb.lower(book.get("author")),
"% " + author.toLowerCase() + "%"));
}
if (physical != null) {
    predicates.add(cb.equal(book.get("physical"),
physical));
}
query.where(cb.and(predicates.toArray(new Predicate[0])));
return entityManager.createQuery(query).getResultList()
    .stream()
    .map(b -> toDto(b,
isDownloadAllowed()))).collect(Collectors.toList());
}

```

В метод `findLatest()` приходит один параметр – количество требуемых последних загруженных книг (установлено 10). Метод передает управление `bookRepository`, который вернет список книг в виде объектов (при помощи функции `toDto`), отсортированных по убыванию по полю `createdAt` (дата загрузки).

Листинг 7. `findLatest()`:

```

public List<BookDto> findLatest(int count) {
return bookRepository
    .findAllOrderByCreatedAtDesc(PageRequest.of(0,
count))
    .stream()
    .map(b -> toDto(b, isDownloadAllowed()))
    .collect(Collectors.toList());
}

```

Метод loadBookFile() используется для скачивания книги с сайта. Сначала метод обращается к bookRepository для поиска книги по идентификатору, проверяет возможно ли ее загрузить (является ли электронной), далее создается путь к файлу с использованием uploadDir и именем файла. Затем создается объект UrlResource, который представляет ресурс, доступный по URL.

Листинг 8. loadBookFile():

```
public Resource loadBookFile(Long id) {
    Book book = bookRepository.findById(id)
        .orElseThrow(() -> new NotFoundException("Book not
found: " + id));
    if (book.isPhysical() || book.getFileName() == null) {
        throw new NotFoundException("No file for book: " + id);
    }
    try {
        Path file = uploadDir.resolve(book.getFileName());
        return new UrlResource(file.toUri());
    } catch (MalformedURLException e) {
        throw new RuntimeException("Error loading file", e);
    }
}
```

Метод createBook() принимает параметры: объект Request (содержит все данные о книге – автор, название и др.), имя пользователя, разместившего том на сайте, файл с книгой. Для получения данных пользователя метод обращается к UserRepository, откуда берет его логин. Получает через геттеры из объекта Request автора, название, год издания, свойство – вещественная или электронная книга и идентификатор пользователя; устанавливает их в объект book. В конце код загружает файл и сохраняет объект book в репозиторий bookRepository (впоследствии в БД), трансформируя его в DTO с помощью метода toDto() и передавая информацию о том, разрешена ли загрузка.

Листинг 9. addBook():

```
public BookDto createBook(String username, CreateBookRequest
```

```

request, MultipartFile file) {
    User user = userRepository.findByUsername(username)
        .orElseThrow(() -> new NotFoundException("User not
found: " + username));
    Book book = new Book();
    book.setTitle(request.getTitle());
    book.setAuthor(request.getAuthor());
    book.setYear(request.getYear());
    book.setPhysical(request.isPhysical());
    book.setUploader(user);

    if (!request.isPhysical() && file != null && !file.isEmpty())
    {
        String filename = UUID.randomUUID() + "_" +
file.getOriginalFilename();
        try {
            Files.copy(file.getInputStream(),
uploadDir.resolve(filename));
        } catch (Exception e) {
            throw new RuntimeException("File upload failed", e);
        }
        book.setFileName(filename);
    }

    Book saved = bookRepository.save(book);
    return toDto(saved, isDownloadAllowed());
}

```

Вспомогательный метод `getBookById()` берет параметр – идентификатор пользователя – и с помощью метода `findById()` ищет необходимую книгу в репозитории `bookRepository`. Управление передает в `toDto()`, преобразующий объект в объект передачи данных.

Листинг 10. `getBookById()`:

```

public BookDto getBookById(Long id) {
    Book book = bookRepository.findById(id)

```

```

        .orElseThrow(() -> new NotFoundException("Book not
found: " + id));
    return toDto(book, isDownloadAllowed());
}

```

В метод `getBookById()` приходит идентификатор книги, с помощью которого загружается файл. Возвращаемый метод `converter.convert(book.getInputStream())` принимает входной поток файла книги и преобразует его в строку HTML. Класс `Fb2ToHtmlConverter()`, реализованный в отдельном модуле, читает файл `.fb2` из `InputStream` и возвращает простой HTML.

Листинг 11. `getHtmlBook()`:

```

public String getHtmlBook(Long id) throws IOException {
    Resource book = loadBookFile(id);
    return converter.convert(book.getInputStream());
}

return toDto(book, isDownloadAllowed());
}

```

Метод `toDto()` преобразует объект `Book` в объект `BookDto` – это достаточно распространенный паттерн, используемый для передачи данных между слоями приложения. Метод принимает два параметра – `Book` (объект книги для преобразования) и булевый флаг `isDownloadAllowed`, указывающий, разрешена ли загрузка книги.

Листинг 12. `toDto()`:

```

private BookDto toDto(Book b, Boolean isDownloadAllowed) {
    return new BookDto(
        b.getId(),
        b.getTitle(),
        b.getAuthor(),
        b.getYear(),
        b.isPhysical(),
        b.getFileName(),
        b.getUploader().getUsername(),
        b.getCreatedAt(),

```

```

        isDownloadAllowed
    );
}

```

4.3.3 Model

В «Книговороте» модель отвечает за хранение данных о пользователях, книгах, сохраняет сообщения из чата и др. После произведенных действий, модель передаст сформированные данные в слой представления. Сохранение данных, например, о книгах, производится в таблицу book, в модели эту информацию хранит класс Book. Он содержит две аннотации: @Entity (класс есть сущность, связанная с таблицей БД) и @Table (имя таблицы). Поле идентификатора является первичным ключом, для его обозначения выставлена аннотация @Id. Поле uploader является внешним ключом, для чего использована аннотация @ManyToOne, которая указывает, что пользователь может загрузить несколько книг, образуя связь многие-к-одному между таблицами Books и Users.

Листинг 13. Book.java:

```

@Entity
@Table(name = "books")
public class Book {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(nullable = false)
    private String title;

    @Column(nullable = false)
    private String author;

    @Column(name = "year_written", nullable = false)
    private int year;
}

```

```

@Column(nullable = false)
private boolean physical;

@Column(name = "file_name")
private String fileName;

@ManyToOne(fetch = FetchType.LAZY)
@JoinColumn(name = "uploader_id", nullable = false)
private User uploader;

@Column(name = "created_at", updatable = false)
private LocalDateTime createdAt;

public Book() {}

@PrePersist
protected void onCreate() {
    this.createdAt = LocalDateTime.now();
}

// --- getters & setters ---
}

```

4.4 Основные алгоритмы взаимодействия клиента и сервера

В начале подробно рассмотрим механизм регистрации и авторизации пользователя – это его первые действия. Посетитель сайта может продолжить работу с сайтом как гость, без авторизации, и пользоваться ограниченным функционалом сайта, либо зарегистрироваться/зайти под уже существующими логином и паролем и использовать полные возможности приложения. При разработке учтено, что неавторизованный посетитель сайта не сможет опробовать функции зарегистрированного пользователя – формы для отправки сообщения, комментария, вкладки личного кабинета для гостя не отображаются. При начальном входе предлагается выбор – зарегистрироваться или войти с логином и паролем, при выборе регистрации будет предложена форма, в которой

необходимо ввести данные нового пользователя. Логин будет проверен на уникальность – если он уже есть в базе, то клиенту будет отправлено уведомление, а форма будет сброшена. Если введенные данные уникальны – в БД регистрируется новый пользователь и происходит перенаправление на начальную страницу с выбором авторизации или регистрации нового пользователя. Авторизация может не произойти, если логин или пароль будут введены неверно, пользователь об этом так же будет уведомлен, а форма сброшена. При успешной авторизации юзер может воспользоваться всеми услугами приложения. Диаграмма последовательности авторизации и регистрации представлена на рисунке 10.

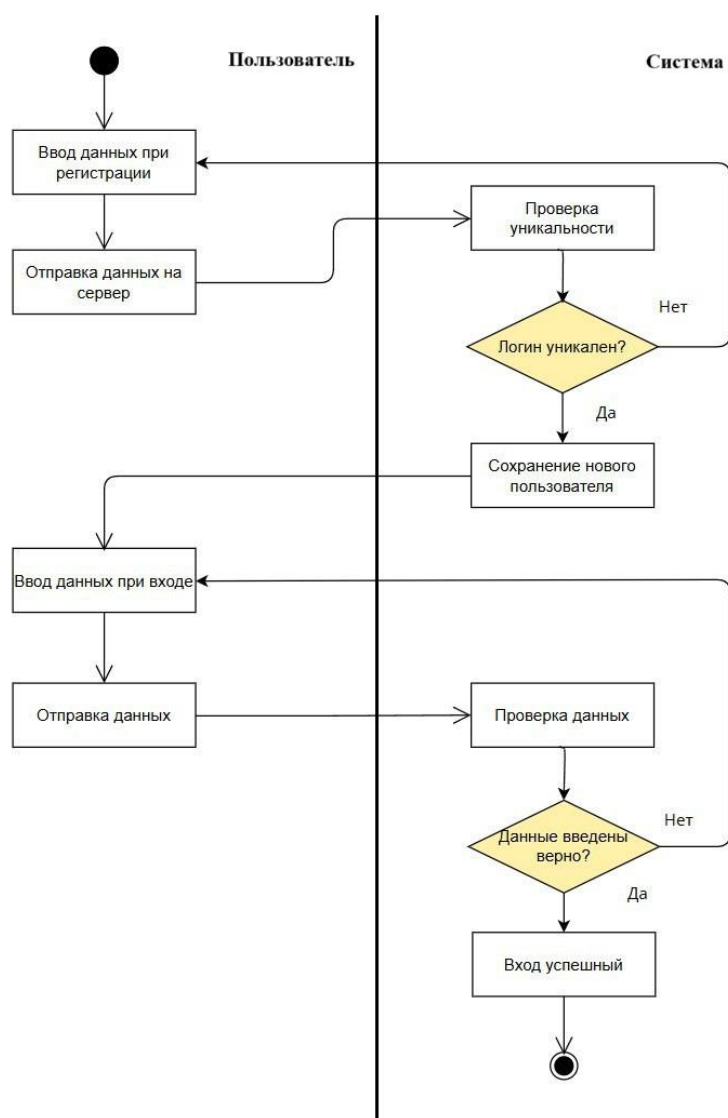


Рисунок 10 – UML-диаграмма деятельности авторизации и регистрации

Рассмотрим алгоритм, описывающий основную цель сайта – добавление книг и подборок (рисунок 11).

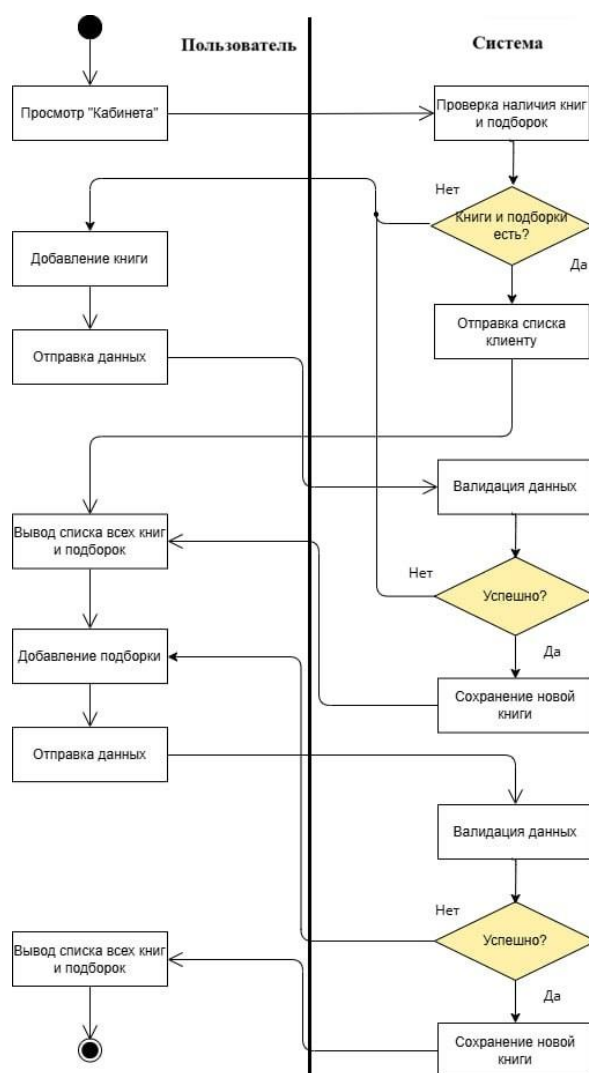


Рисунок 11 – UML-диаграмма деятельности загрузки книг и подборок

Личная библиотека пользователей находится на вкладке «Кабинет» – здесь выходит список всех книг, подборок, предлагается изменить личные данные (имя, фамилия, пароль). Вкладка доступна только авторизованным пользователям, гостям она не отображается. Если пользователь еще не загружал книги и не создавал подборки, он может добавить их – для этого предназначены кнопки «Добавить книгу» и «Создать подборку». Добавление книги происходит через специальную форму – в ней клиент должен указать название, автора, год издания, прикрепить файл, если книга в электронном виде или поставить флажок рядом с меткой «книга офлайн», если книга вещественная. Если загрузка книги

в БД произошла успешно, то форма закрывается, пользователь перенаправляется в личный кабинет, а в списке книг добавился новый объект. При некорректном заполнении формы клиент получит уведомление, а форма будет сброшена.

Создание подборки происходит аналогично – при нажатии на кнопку «Создать подборку» выходит форма, в которой нужно указать название и установить флажки рядом с теми книгами, которые будут составлять подборку. Если проблем с созданием не возникло, форма закроется, пользователь останется на вкладке «Кабинет», а раздел «Мои подборки» будет содержать новый объект с полным списком томов, включенных в подборку.

Работа с книгой (рисунок 12) для пользователя не ограничивается добавлением тома на сайт.

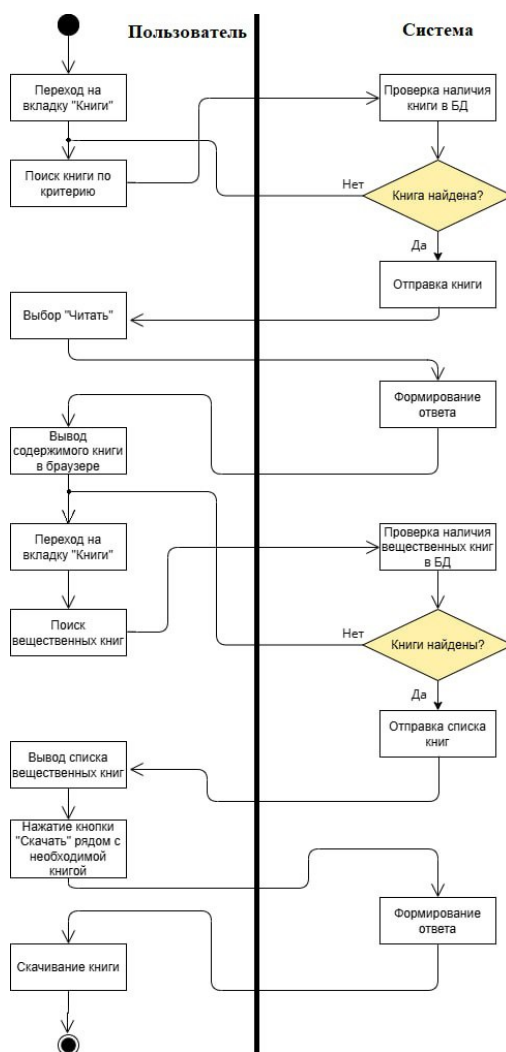


Рисунок 12 – Диаграмма деятельности чтения, скачивания и поиска книг

Авторизованный посетитель сайта может читать в браузере, скачать на устройство, найти нужную книгу в поиске, отсортировать список по критерию. Для того, чтобы открыть книгу для чтения в браузере, пользователю необходимо перейти на вкладку «Книги», набрать название или автора нужного тома в поисковой строке; если нужный экземпляр будет найден, то надо только кликнуть рядом с ним на кнопку «Читать». Аналогично следует поступить, если необходимо скачать содержимое книги, отличие будет только в кнопке «Скачать». Список книг можно отсортировать по критерию «вещественная» или «электронная», при таком запросе сервер выдаст все тома, удовлетворяющие заданному условию.

4.5 Развертывание приложения

Система приложения «Книговорот» состоит из следующих частей:

- клиентская часть, отвечающая за пользовательский интерфейс (поля ввода, кнопки, списки), разработана на HTML; для взаимодействия с сервером используется REST. REST предоставляет набор архитектурных ограничений, который в случае применения целиком выделяет масштабируемость взаимодействий компонентов, их независимое развертывание, универсальность интерфейсов и промежуточные компоненты для сокращения задержки взаимодействий, повышения безопасности и инкапсуляции устаревших систем [6];
- серверная часть, обрабатывающая данные пользователя, реализована на языке Java с фреймворком Spring, который помогает просто и быстро разработать нужное приложение;
- БД из приложения управляется Spring JPA, который предоставляет доступ к PostgreSQL;
- Spring Security используется для обеспечения безопасности при регистрации и авторизации в приложении;
- развертывание обеспечивается веб-сервером Apache Tomcat.

Распределение компонентов системы по физическим узлам представлено на рисунке 13.

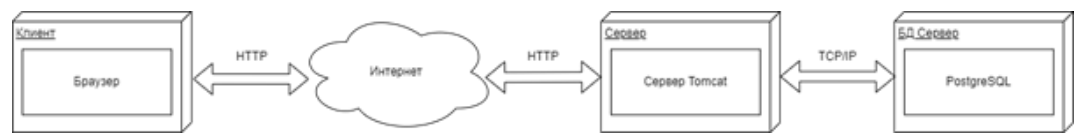


Рисунок 13 – UML-диаграмма развертывания приложения

Для подключения БД скачано приложение PgAdmin4 v.9.2, с ним в пакете установился PostgreSQL версий 16 и 17. Для работы с веб-приложением «Книговорот» в программе была создана БД booklover, добавлены описанные ранее таблицы (рисунок 14):

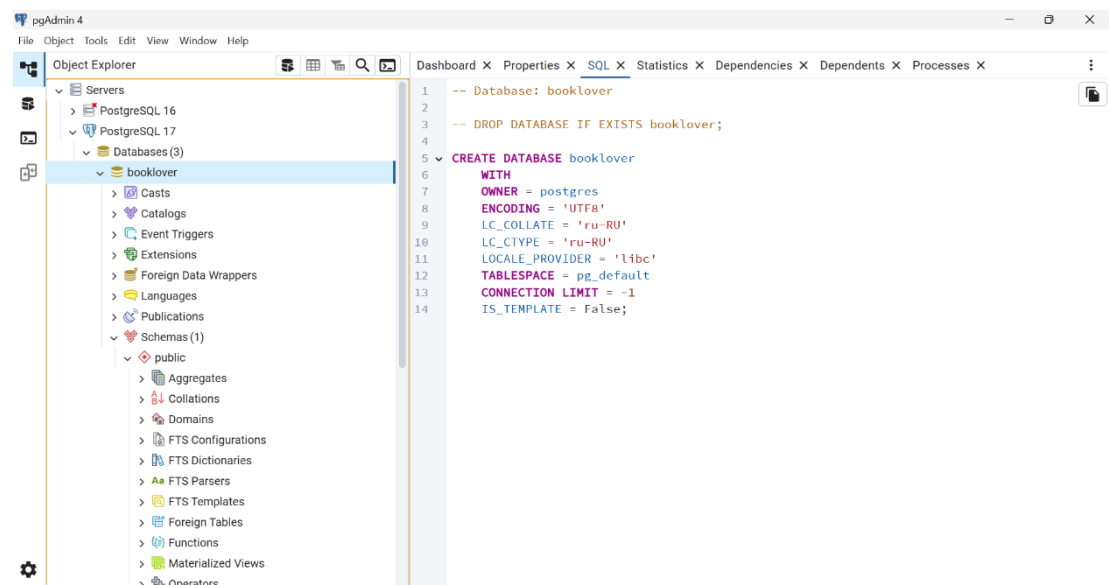


Рисунок 14 – База данных приложения в pgAdmin

Что касается сервера Tomcat, то он устанавливается автоматически. При запуске приложения, созданного на Spring Boot, тот обнаруживает контроллер Spring MVC и, по умолчанию, запускает встроенный Apache Tomcat 10.1.40. Сервер по умолчанию функционирует на порту 8080, но возможна перенастройка в файле application.properties. На рисунке 15 отображен автоматический запуск Tomcat при компиляции приложения:

```

Debug Application x
Threads & Variables Console
3:41:31.524+03:00 INFO 11184 --- [ restartedMain] .e.DevToolsPropertyDefaultsPostProcessor : For additional web related logging consider setti
3:41:34.758+03:00 INFO 11184 --- [ restartedMain] .s.d.r.c.RepositoryConfigurationDelegate : Bootstrapping Spring Data JPA repositories in DEF
3:41:34.981+03:00 INFO 11184 --- [ restartedMain] .s.d.r.c.RepositoryConfigurationDelegate : Finished Spring Data repository scanning in 201 ms
3:41:37.238+03:00 INFO 11184 --- [ restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port 8080 (http)
3:41:37.286+03:00 INFO 11184 --- [ restartedMain] o.apache.catalina.core.StandardService : Starting service [Tomcat]
3:41:37.287+03:00 INFO 11184 --- [ restartedMain] o.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/10.1.40]
3:41:37.434+03:00 INFO 11184 --- [ restartedMain] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationContext
booklover > src > main > java > com > example > booklover > Application 10:14 CRLF UTF-8 4 spaces

```

Рисунок 15 – Запуск Tomcat

5 Тестирование приложения

Тестирование работоспособности созданного приложения «Книговорот» состоит из следующих компонентов:

- регистрация;
- авторизация;
- редактирование данных пользователя;
- взаимодействие с книгами авторизованного пользователя (загрузка книг, чтение, скачивание, поиск и сортировка книг, создание подборок, комментирование);
- общение в чате;
- тест функционала неавторизованных пользователей.

5.1 Регистрация

Чтобы провести регистрацию нового пользователя, необходимо перейти на стартовую страницу веб-приложения и выбрать окно «Регистрация», где необходимо ввести данные: имя, фамилия, логин, пароль. Для теста был создан пользователь тах, с аналогичным имени логином и паролем (рисунок 16).

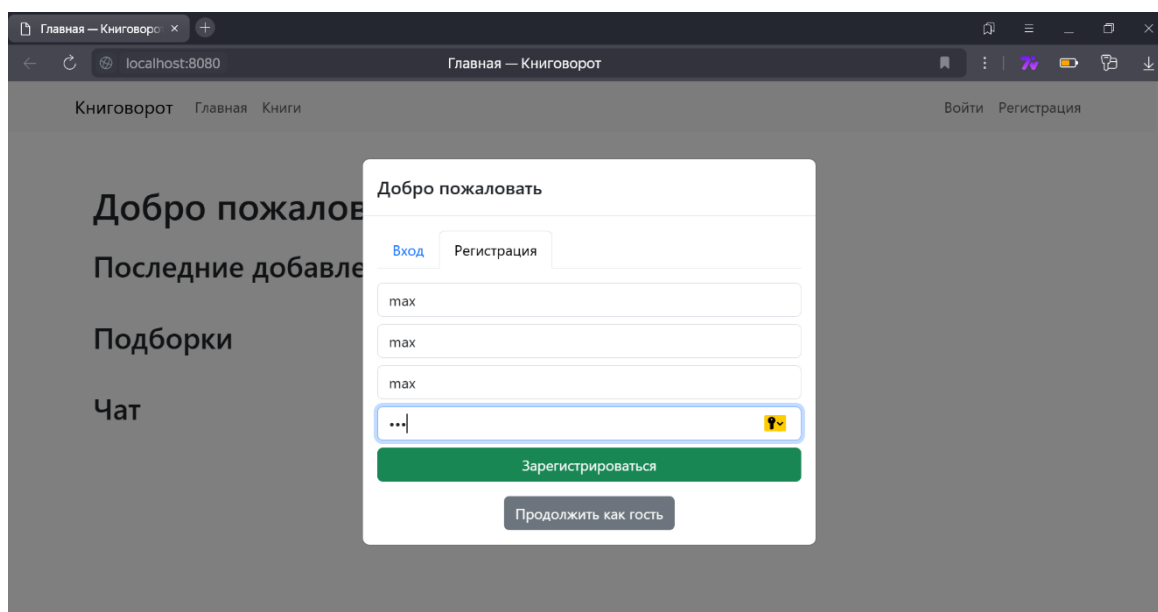


Рисунок 16 – Регистрация нового пользователя «тах»

Состояние базы данных до регистрации нового пользователя (она пустая, рисунок 17):

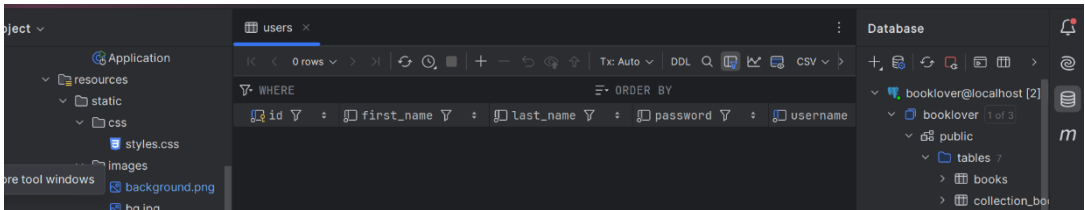


Рисунок 17 – База данных до регистрации пользователя

После успешной регистрации база данных пополнилась новой записью, а на главной странице предлагается совершить вход в систему (рисунки 18-19).

Вход

Успешная регистрация. Теперь войдите.

>Login

>Password

Войти

Продолжить как гость

Нет аккаунта? [Зарегистрироваться](#)

© 2025 Книговорот. Все права защищены.

Рисунок 18 – Переход на форму «Вход» после регистрации

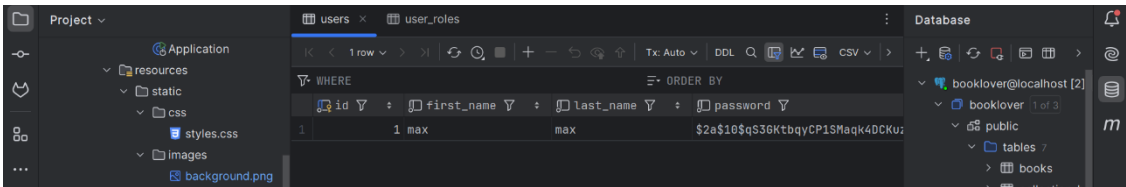


Рисунок 19 – Новая запись в БД о пользователе «max»

Выйдя из созданного аккаунта и заново введя те же данные, приложение выбрасывает сообщение о том, что пользователь с данным логином уже существует, и перенаправляет снова на страницу регистрации (рисунки 20-21).

Рисунок 20 – Регистрация пользователя с уже существующим логином

Регистрация

Рисунок 21 – Сообщение «Пользователь с таким логином уже существует»

Для будущих тестов было создано еще несколько пользователей (рисунки 22-24):

Регистрация

Имя

danil

Фамилия

danilov

Логин

danilov1992

Пароль

....



Зарегистрироваться

Уже зарегистрированы? [Войти](#)

Рисунок 22 – Регистрация нового пользователя «danil»

Регистрация

Имя

egor

Фамилия

egorov

Логин

egor2002

Пароль

....



Зарегистрироваться

Уже зарегистрированы? [Войти](#)

Рисунок 23 – Регистрация нового пользователя «egor»

id	first_name	last_name	password
1	max	max	\$2a\$10\$q\$3GKtbqyCP1SMaQk4DCKuz
2	daniil	daniilov	\$2a\$10\$x6Zw/xbMiIr0KXadKqF0Q0v
3	egor	egorov	\$2a\$10\$0y/nNJC9cCk0FR2S9U5x/uf

Рисунок 24 – Записи о созданных пользователях в БД

5.2 Авторизация

Тестирование авторизации разделим на два сценария – ввод корректных логина и пароля (успешная авторизация) и ошибочных данных (сообщение об ошибке).

Для успешного прохождения авторизации необходимо ввести данные, которые уже есть в БД – в данном примере введем логин и пароль пользователя «egor» (рисунки 25-26).

Вход

Логин

egor2002

Пароль

••••

🔑

Войти

Продолжить как гость

Нет аккаунта?

[Зарегистрироваться](#)

Рисунок 25 – Ввод корректных данных

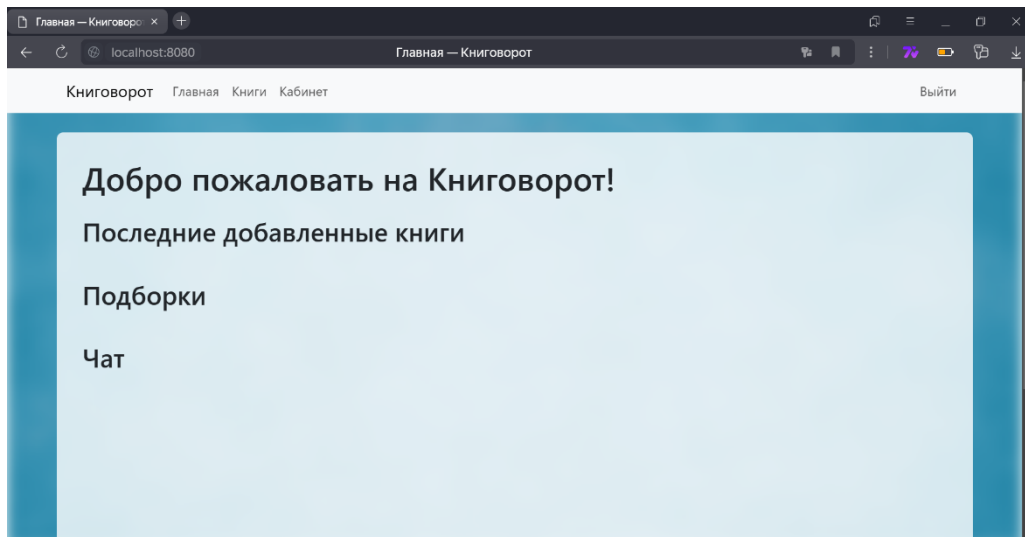


Рисунок 26 – Вход на сайт после успешной авторизации

На рисунках 27-28 представлен тест авторизации с вводом некорректных данных:

Рисунок 27 – Ввод несуществующего в БД логина

Вход

Неверный логин или пароль

Логин

Введите логин

Пароль

Введите пароль

Войти

Продолжить как гость

Нет аккаунта? [Зарегистрироваться](#)

Рисунок 28 – Сообщение «Неверный логин или пароль»

5.3 Редактирование данных пользователя

Приложением предусмотрено, что пользователь всегда может изменить свои данные. Для обновления данных необходимо вверху сайта выбрать вкладку «Кабинет», где можно ввести новые фамилию, имя или пароль, после чего кликнуть на «Сохранить профиль» (рисунки 29-33).

WHERE					ORDER BY				
id	first_name	last_name	password						
1	max	max	\$2a\$10\$qS36KtbqyCP1Smaqk4DCKuz						
2	daniil	daniilov	\$2a\$10\$x6Zw/xbMiIr0KXadKqF0Q0v						
3	egor	egorov	\$2a\$10\$0y/nNJC9cCk0FR2S9U5x/uH						

Рисунок 29 – Записи в БД до изменения данных

Рисунок 30 – Профиль пользователя «max» до изменения данных

Рисунок 31 – Редактирование фамилии пользователя

Рисунок 32 – Уведомление об успешном изменении данных

	id	first_name	last_name	password
1	2	daniil	daniilov	\$2a\$10\$x6Zw/xbMiIr0KXadKqF0Q0v
2	3	egor	egorov	\$2a\$10\$0y/nNJC9cCk0FR2S9U5x/ut
3	1	max	maximov	\$2a\$10\$qS3GKtbqyCP1SMaqk4DCKuz

Рисунок 33 – БД после завершения изменения данных

5.4 Взаимодействие с книгами авторизованного пользователя

Авторизованный пользователь обладает функционалом загрузки книги, просмотра, скачивания, а также создания тематической подборки книг.

Простейшая функция – загрузка книги на сайт – доступна в личном кабинете пользователя. Загрузить книгу можно как в электронном виде, так и «сообщить» другим пользователям о том, что книга есть офлайн; оба варианта загрузки отображены на рисунках 34-38.

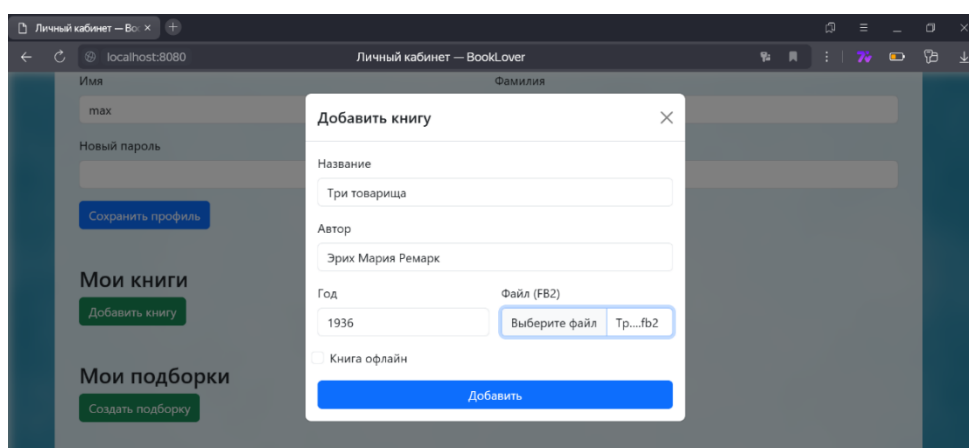


Рисунок 34 – Загрузка электронной книги

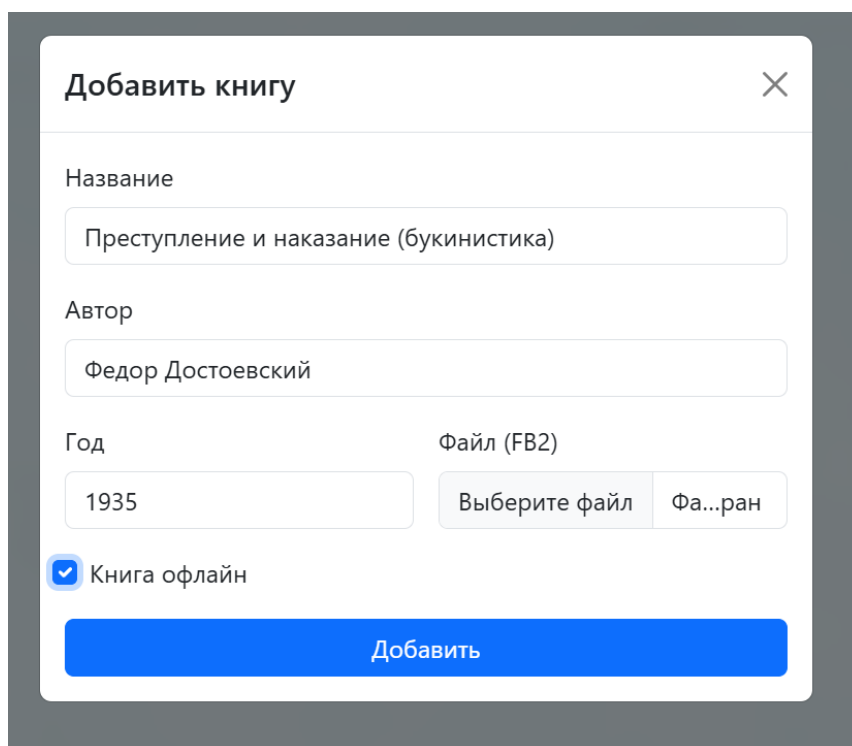


Рисунок 35 – Загрузка книги «офлайн»

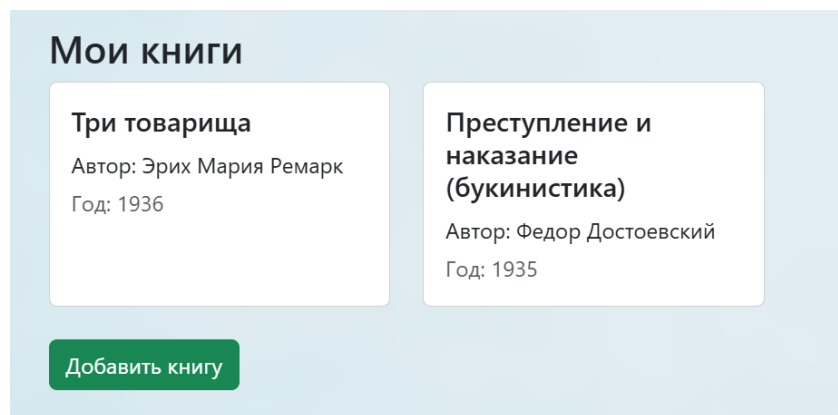


Рисунок 36 – Загруженные книги в личном кабинете

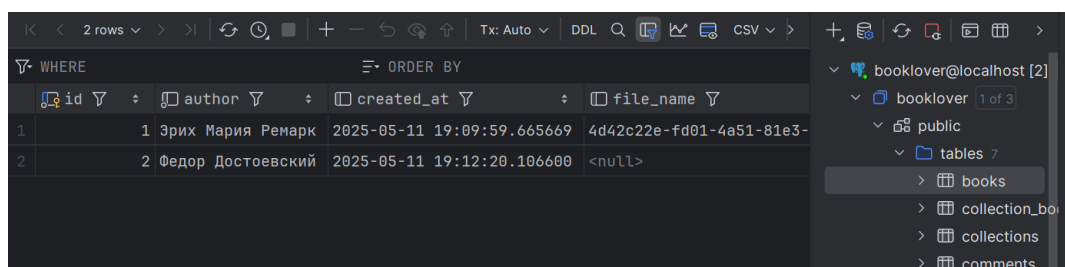


Рисунок 37 – Загруженные книги в БД

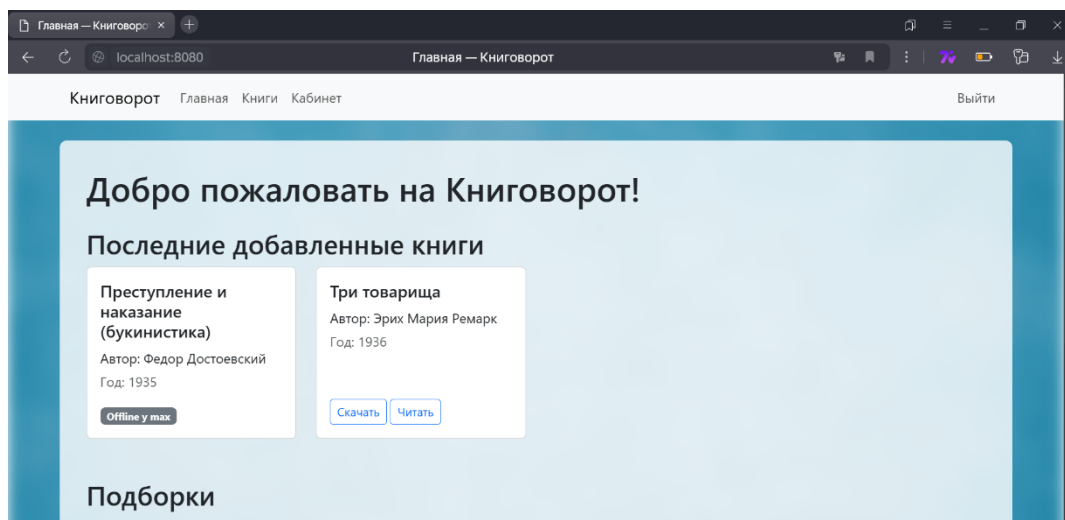


Рисунок 38 – Главная страница сайта после загрузки книг

Загруженные книги авторизованный пользователь сможет прочитать в электронном виде, или скачать (рисунки 39-40). Для тестирования произведен вход с другого пользователя.

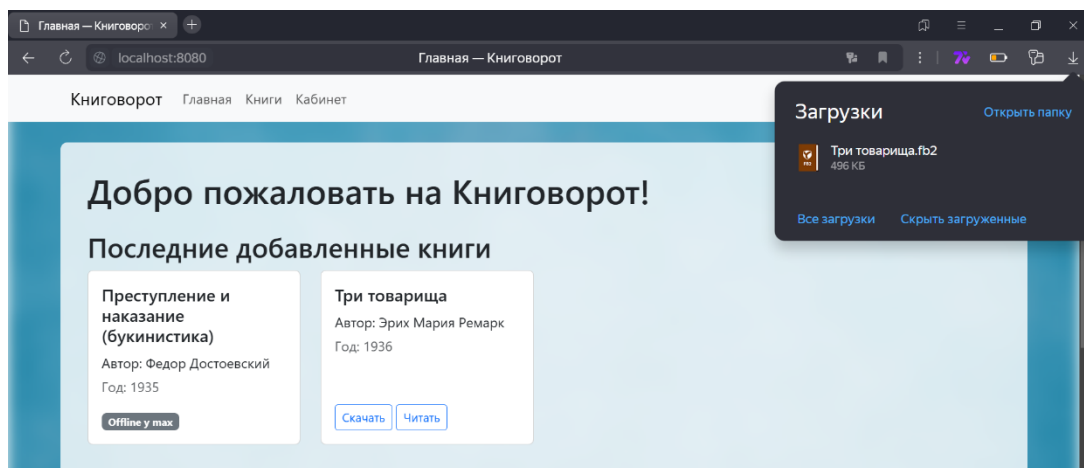


Рисунок 39 – Скачивание книги с сайта

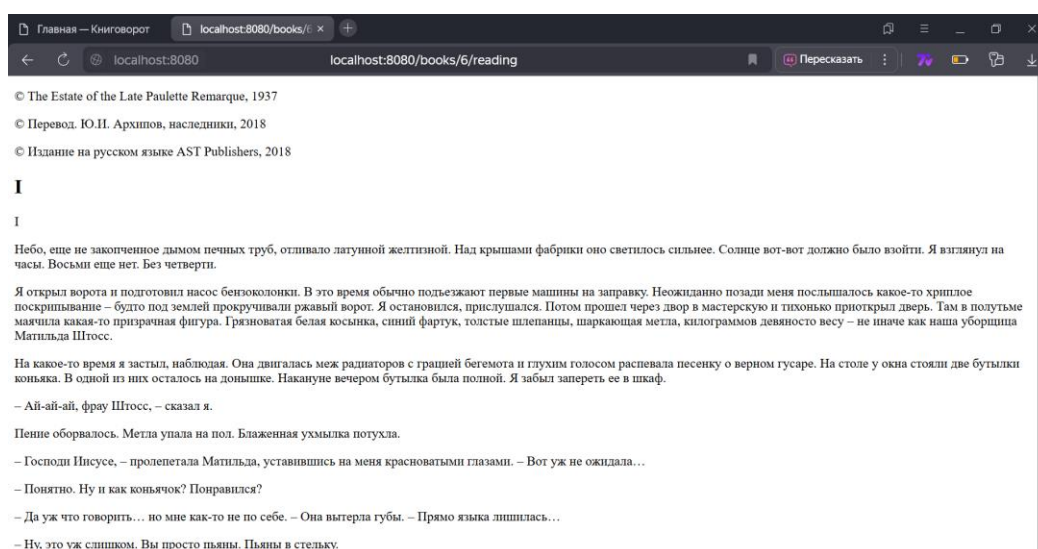


Рисунок 40 – Чтение книги в браузере

С других аккаунтов добавлено еще несколько книг (рисунок 41).

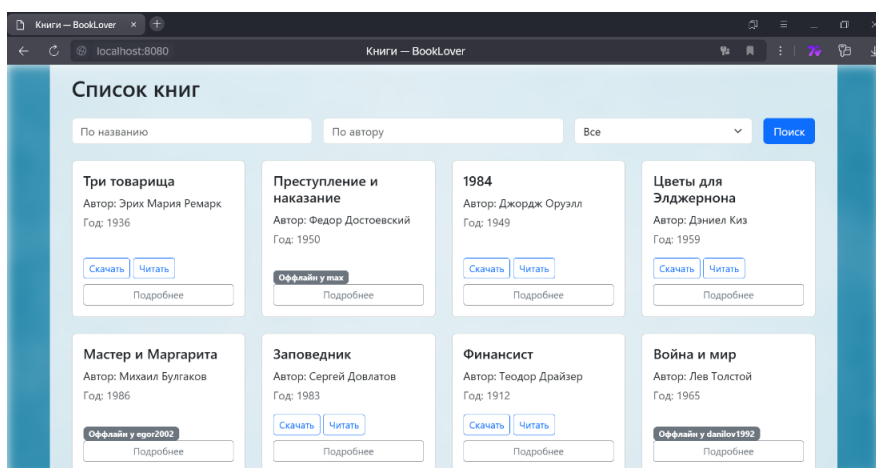


Рисунок 41 – Добавленные пользователями книги

На вкладке «Книги» помимо просмотра всех книг, есть возможность отсортировать книги (вывести только вещественные или только электронные), а также найти нужный том по названию или автору (рисунки 42-44).

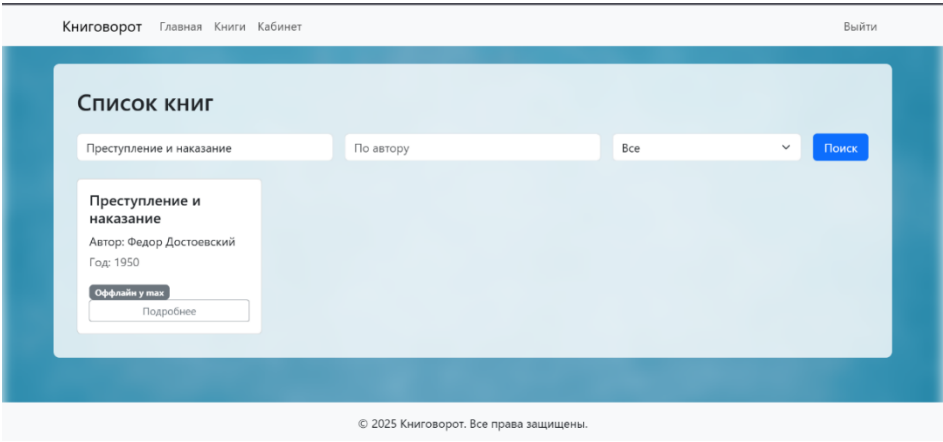


Рисунок 42 – Поиск книги по названию

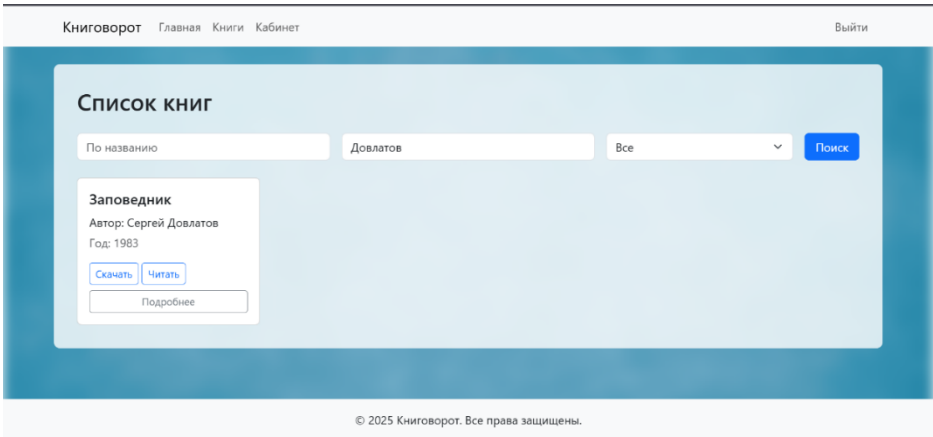


Рисунок 43 – Поиск книг по автору

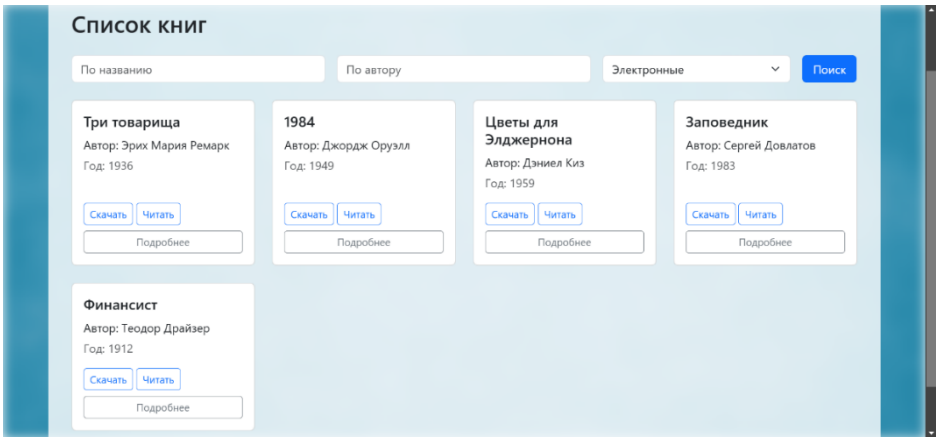


Рисунок 44 – Сортировка книг по критерию «электронные»

Авторизованный пользователь имеет возможность комментировать книги – оставить свое мнение можно о каждой книге – как электронной, так и вещественной (рисунок 45). Для комментирования нужно нажать в окошке с книгой кнопку «Подробнее».

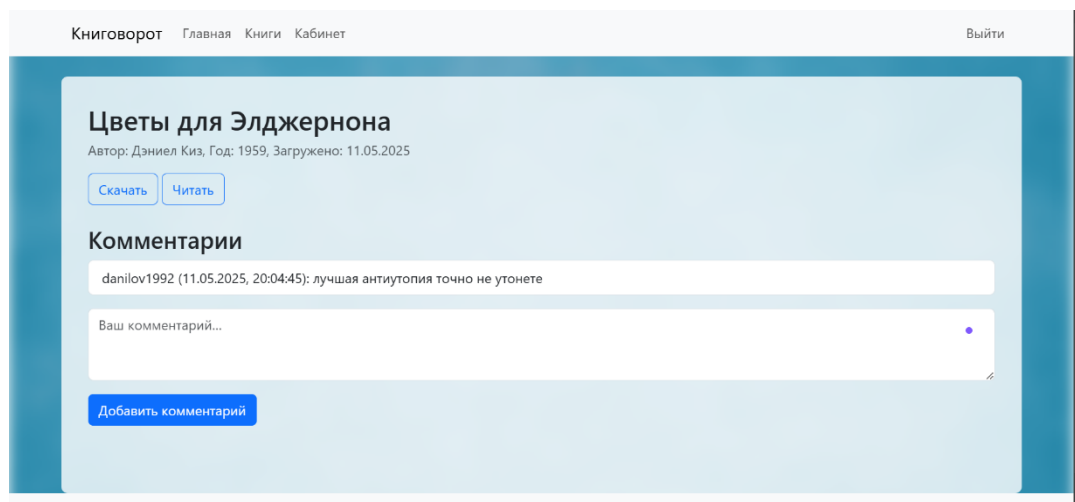


Рисунок 45 – Комментирование книги

Книги можно объединять в подборки – данный функционал реализован в «Кабинете» (рисунки 46-47). Создавать подборку можно из томов всего сайта.

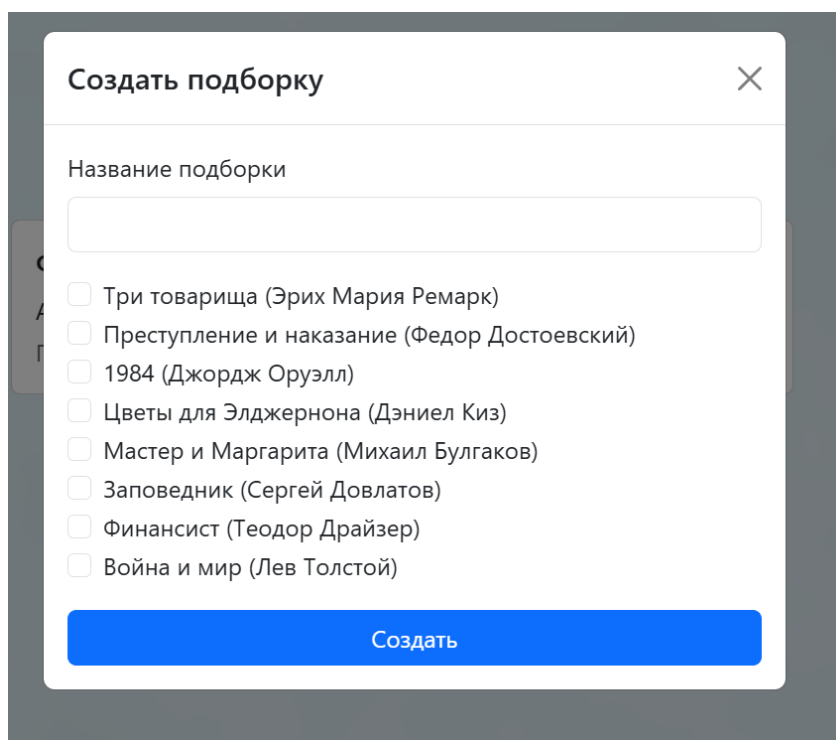


Рисунок 46 – Создание подборки

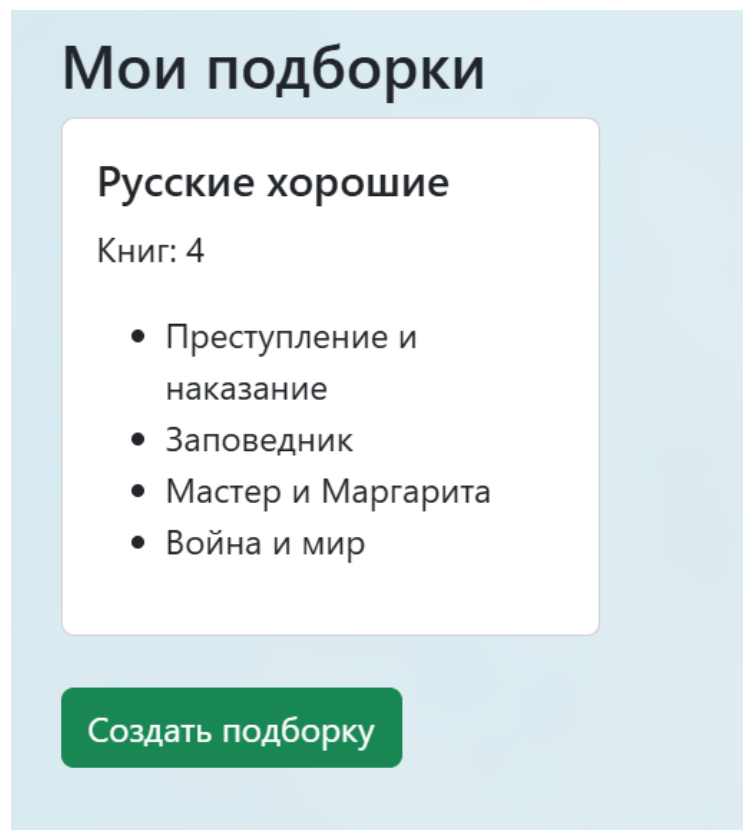


Рисунок 47 – Результат создания подборки

5.5 Общение в чате

Одна из главных функций, доступных только зарегистрированным читателям, это возможность общаться в чате – тут люди могут обсудить возможный обмен, книжные интересы, или просто пообщаться на сторонние темы. Чат реализован на главной странице сайта (рисунки 48-49).

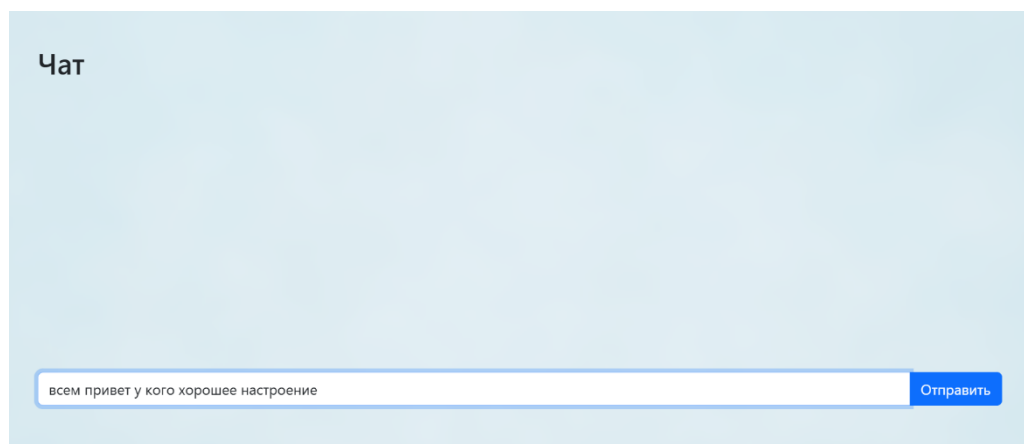


Рисунок 48 – Отправление сообщения в чате

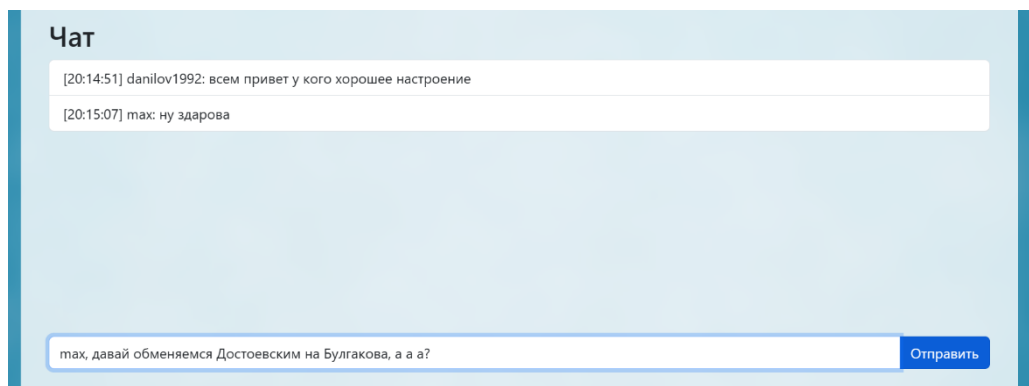


Рисунок 49 – Общение пользователей в чате

5.6 Функционал неавторизованного пользователя

Функционал неавторизованного пользователя ограничен только просмотром того, что создал пользователь авторизованный. Гость не может ничего менять, комментировать, оставлять сообщения, загружать – он может читать в браузере книгу, просматривать и сортировать список книг, может увидеть сообщения в чате и комментарии.

У гостя нет кнопок «Скачать», «Отправить сообщение» (как и поля для ввода сообщения), нет вкладки «Кабинет» (рисунок 50).

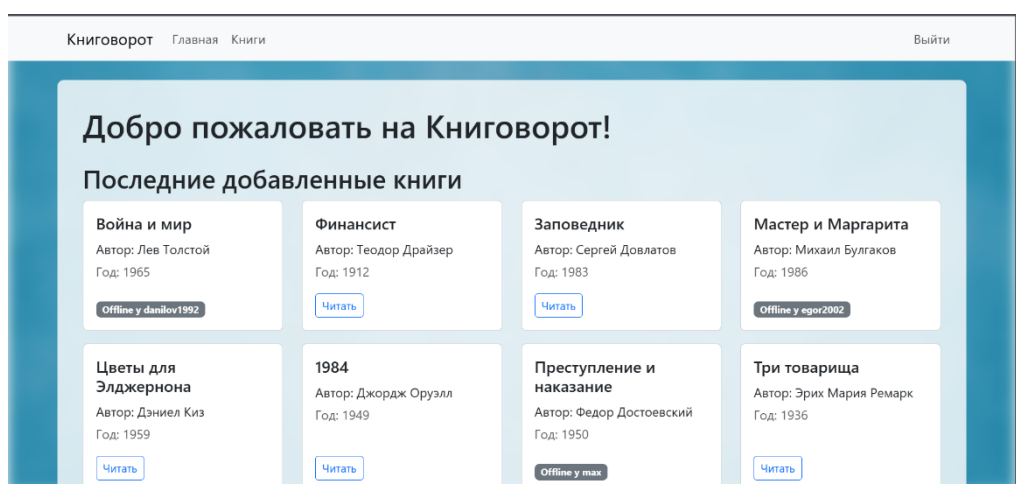


Рисунок 50 – Основная страница сайта для неавторизованного пользователя

При попытке оставить комментарий под книгой выходит уведомление (рисунок 51).

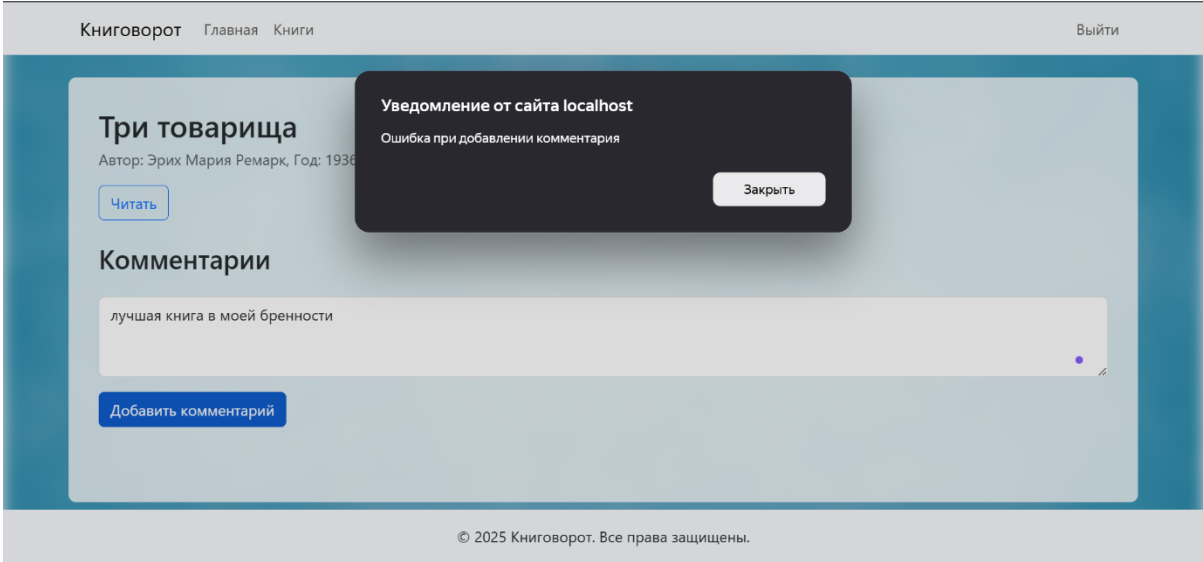


Рисунок 51 – Уведомление при комментировании

6 Безопасность жизнедеятельности

В современной профессиональной среде, в частности, в области разработки программного обеспечения, длительная работа за компьютером становится нормой. Однако, пренебрежение принципами безопасности жизнедеятельности может негативно сказаться на здоровье сотрудников, приводя к проблемам со зрением, нарушениям опорно-двигательного аппарата и другим заболеваниям. Ниже рассмотрены ключевые аспекты безопасности при работе за компьютером, а также представлены рекомендации, направленные на минимизацию рисков для здоровья программистов.

Ключевую роль в обеспечении БЖД при работе за компьютером играет правильное освещение. Плохая организация может повлечь неприятные последствия: головные боли, усталость, напряжение, снижается продуктивность, возможны серьезные глазные заболевания. Среди требований к освещению рабочего места:

- достаточная освещенность: нужно четко различать детали на мониторе и документах, но освещенность не должна быть и избыточной (могут появиться блики). Освещенность экрана должна быть ниже, чем рабочей поверхности, для последней значение освещенности – 300-500 люкс;

- равномерность освещения: не должно быть резких перепадов между освещенностью монитора и окружающего пространства, это может повлечь сильную и быструю утомляемость глаз;

- необходимо избегать попадания прямых солнечных лучей в глаза, на экране не должно быть бликов и отражений (рекомендация – использовать матовые экраны);

- правильный выбор источников света: предпочтительна нейтральная цветовая температура источника (около 4000 К), хорошо подойдут светодиодные (LED) и люминесцентные лампы;

- регулируемость освещения под индивидуальные особенности человека;

– естественное освещение: максимально эффективно нужно использовать дневной свет, лучше расположить рабочее место с ПК рядом с окном (важно избегать попадания солнечных лучей на монитор), необходимы шторы или жалюзи для регулировки интенсивности света;

– искусственное освещение: помимо общего комнатного освещения нужно использовать и местное – настольная лампа, фонарь конкретно под рабочее место; свет должен падать сбоку и немного спереди от монитора (важно избегать бликов).

Эргономика рабочего места:

– расположение монитора: верхний край должен находиться чуть ниже или на уровне глаз, расстояние до монитора примерно равно длине вытянутой руки;

– кресло: предпочтительно с регулируемой высотой, подлокотниками и поясничной поддержкой. Ступни должны полностью касаться пола, а колени согнуты под прямым углом;

– периферия: клавиатура с мышью должны располагаться на одном уровне, чтобы предплечья располагались параллельны полу, а локти лежали на столе или подлокотниках, желательна подставка для запястий;

– осанка: правильно сидеть прямо, не сутулясь, опираясь на спинку кресла; плечи в расслабленном состоянии; необходимо каждый час работы за компьютером делать перерыв и разминаться в целях избегания затекания.

Правила и рекомендации по заботе по зрению при работе с ПК:

– расстояние до монитора: оптимальное расстояние около 50-70 см, так нагрузка на глаза значительно ниже;

– регулировка яркости и контрастности монитора, экран не должен быть слишком ярким и слишком тусклым;

– перерывы: можно делать упражнение 20-20-20 – каждые 20 минут смотреть на объект на расстоянии 20 футов (около 6 метров) в течение 20 секунд;

– можно выполнять простейшие упражнения для глаз для снятия напряжения – смотреть вблизи и вдаль, вверх-вниз, влево-вправо;

– необходимо регулярно проверять зрение, посещать окулиста для профилактического осмотра.

Профилактика туннельного синдрома и заболеваний опорно-двигательного аппарата:

– правильное расположение рук: запястья прямо во время работы с периферией компьютера;

– эргономичные клавиатура и мышь, которые поддерживают естественное положение рук, запястий и локтей;

– регулярные разминки рук и кистей – сжимать-разжимать кулаки, вращение кистей;

– регулярные перерывы, возможно физическая активность – небольшие прогулки помогают расслабиться и снять накопившееся напряжение.

Помимо аспектов, связанных с ПК и возможным вредом для здоровья от работы с ним, важно обратить внимание на организацию рабочего времени и отдыха. Необходимо правильно планировать рабочий день, чтобы избежать перегрузок и стресса; делать перерывы – желательно короткие каждые полчаса и перерывы с разминкой и физической активностью каждые 2-3 часа. Нужно соблюдать баланс между работой и личной жизнью, не перегружая себя работой, нанося при этом себе вред. Достаточное количество сна (7-8 часов), правильное питание и спорт также помогают в снижении негативного влияния при работе с компьютером. Важно уделять время своему психическому благополучию: поддерживать свой настрой, при возможном стрессе освоить техники релаксации и медитации.

Заключение

При выполнении данной выпускной квалификационной работы были реализованы задачи: проанализирована актуальность темы разрабатываемого приложения, проведен анализ и выбор среды разработки и инструментов, выбрана наиболее подходящая архитектура, разработано веб-приложение «Книговорот» с базой данных к нему, произведены тесты.

Получены принципы и навыки разработки веб-приложений, освоен фреймворк Spring Boot, закреплены умения в программировании на языке Java. Изучена и проведена работа с базой данных PSQL. Тесты приложения показывают, что клиент-серверное приложение «Книговорот» разработано в соответствии со всеми заранее описанными требованиями, функционал реализован аккуратно и корректно.

Приложение может улучшаться – может появиться более интерактивная система общения между пользователями сайта, система физического обмена книг может быть уточнена и доработана, внешний вид приложения и интерфейс может усовершенствоваться, а безопасность пользователей увеличена.

Список использованных источников

1. Раджпут Д. Spring. Все паттерны проектирования. — СПб.: «Питер», 2019. — 320 с.
2. Панченко Иван. PostgreSQL: вчера, сегодня, завтра. Открытые системы. СУБД, № 03, 2015 (20 июля 2015). [Электронный ресурс] / URL: <https://www.osp.ru/os/2015/03/13046900> (дата обращения: 15 мая 2025).
3. Адам Фримен. ASP.NET MVC 4 с примерами на C# 5.0 для профессионалов, 4-е издание. — М.: «Вильямс», 2013. — 688 с.
4. Герберт Шилдт. Java. Полное руководство, 12-е изд.: Пер. с англ. — СПб.: ООО «Диалектика», 2023. — 1344 с.
5. Уоллс К. Spring в действии. — М.: ДМК Пресс, 2013. — 752 с.
6. Амундсен М. RESTful Web API: Паттерны и практики. — СПб.: Питер, 2025. — 464 с.

Приложение А

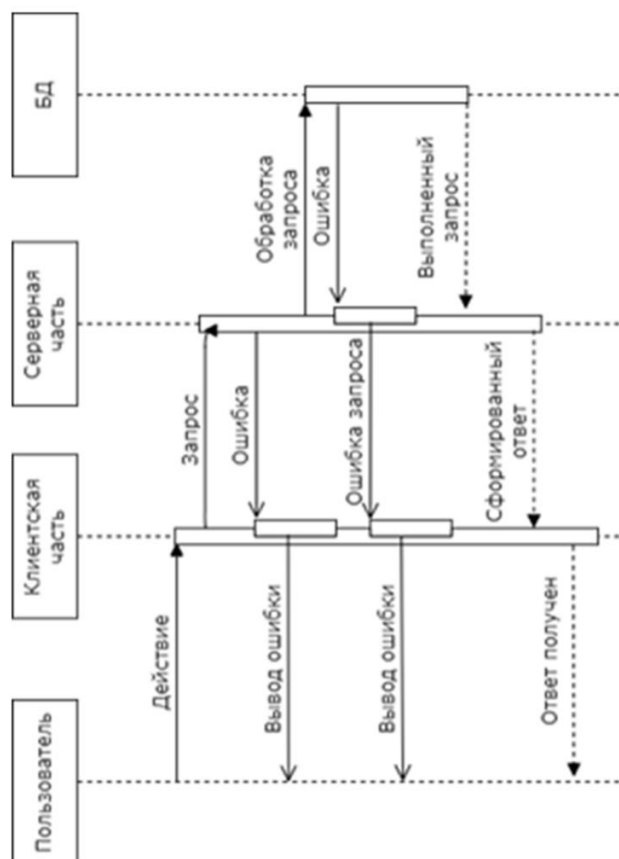
(Обязательное)

Web-приложение «Книговорот»

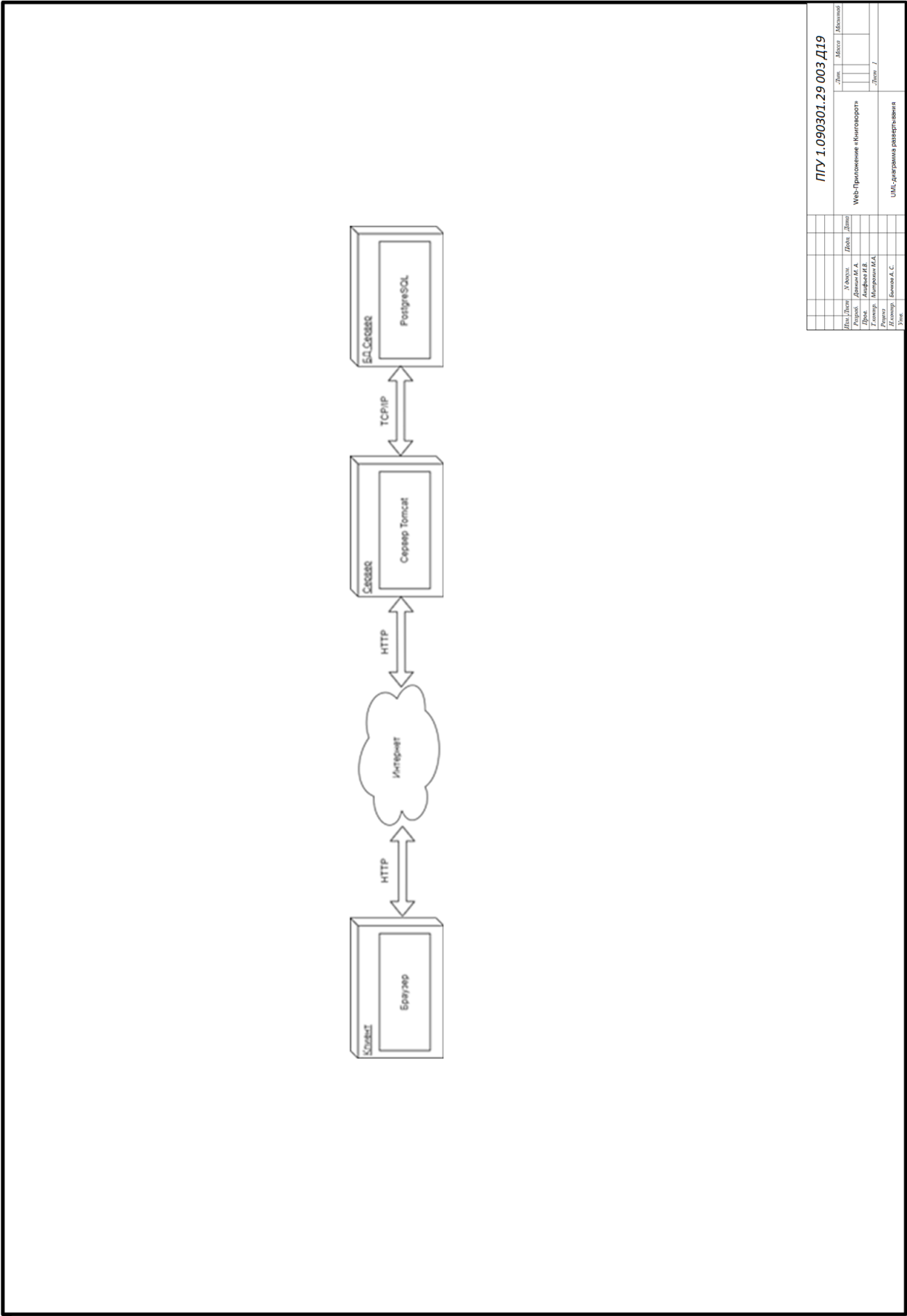
UML-диаграммы

					ПГУ 1.090301.29.001 ПЗ	Лист
						69
Изм.	Лист	№ докум.	Подпись	Дата		

Приложение А.1 – Диаграмма последовательностей

[illegible]

Приложение А.2 – Диаграмма развертывания



ПГУ 1.090301.29.003 Д19

Имя	Уровень	Роль	Диагн.	Масштаб
Браузер	Клиент	Клиент	1	1
Сервер Tomcat	Сервер	Сервер	1	1
БД PostgreSQL	База данных	База данных	1	1
Интернет	Сеть	Сеть	1	1

Web-Приложение «Книгопорт»

UML-диаграмма развертывания

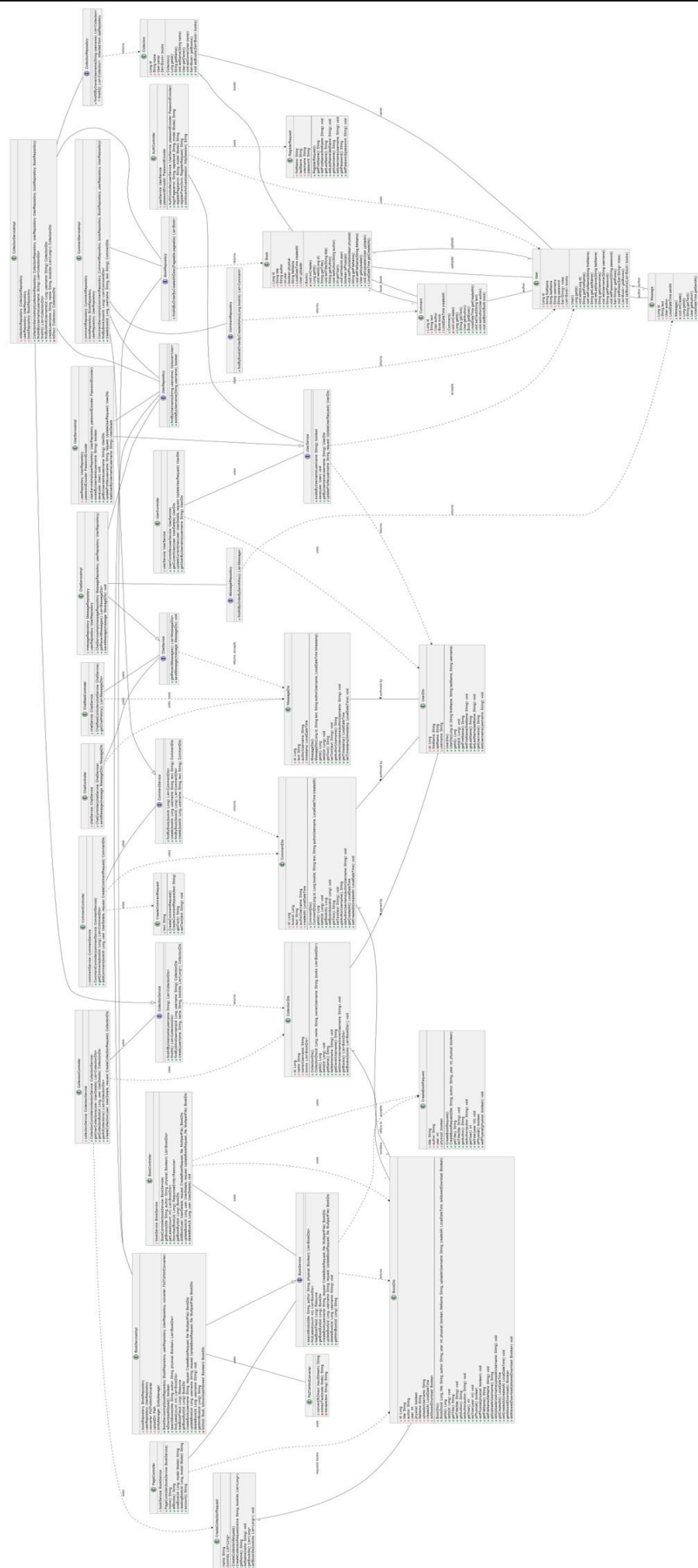
Имя	Уровень	Роль	Диагн.	Масштаб
Браузер	Клиент	Клиент	1	1
Сервер Tomcat	Сервер	Сервер	1	1
БД PostgreSQL	База данных	База данных	1	1
Интернет	Сеть	Сеть	1	1

ПГУ 1.090301.29.001 ПЗ

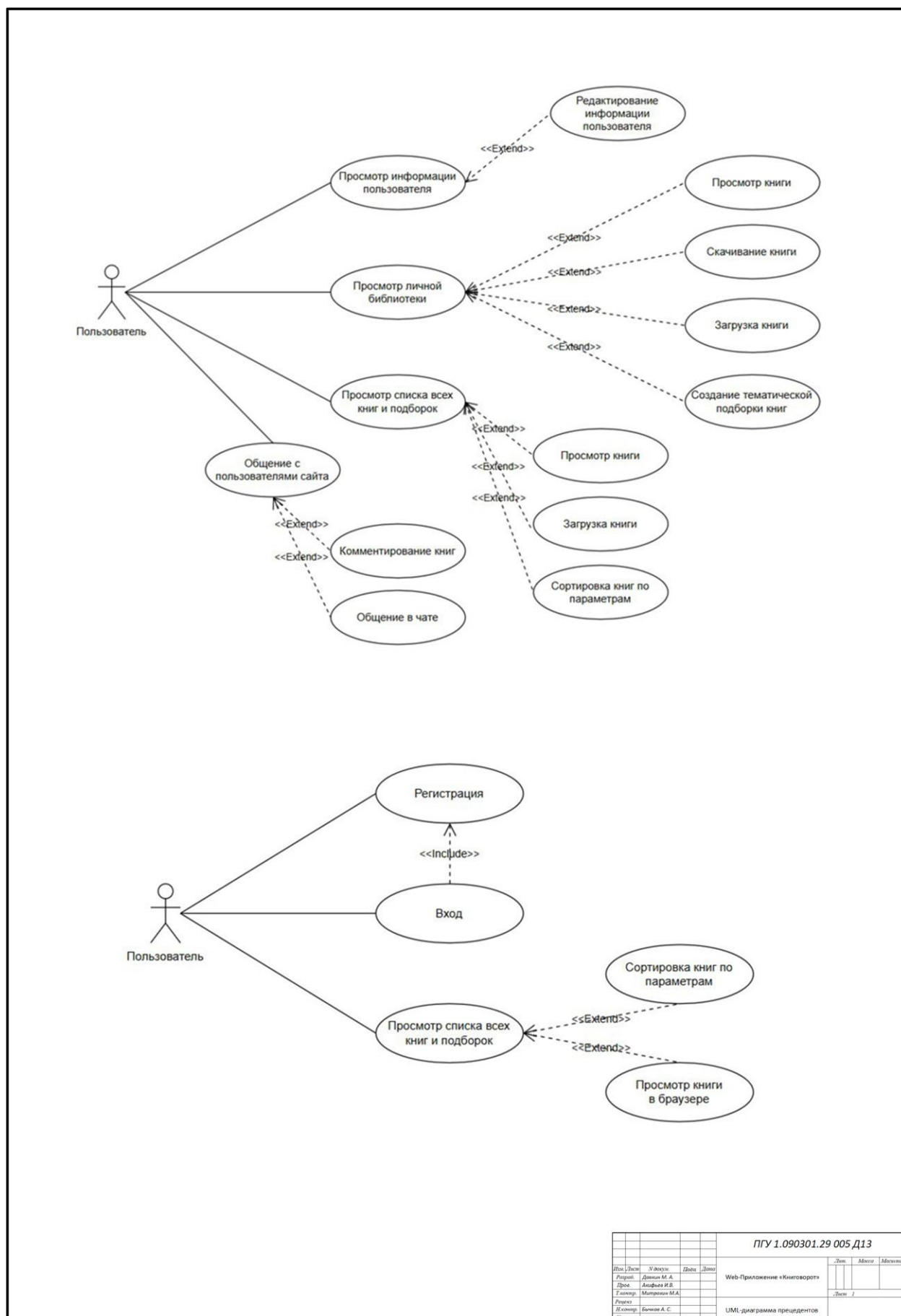
Лист

71

Приложение А.3 – Диаграмма классов

[illegible]

Приложение А.4 – Диаграммы вариантов использования



Приложение Б

(Обязательное)

Web-приложение «Книговорот»

Листинг программы

					ПГУ 1.090301.29.001 ПЗ	Лист
						74
Изм.	Лист	№ докум.	Подпись	Дата		

Файл Book.java

```
package com.example.booklover.model.entity;

import jakarta.persistence.Column;
import jakarta.persistence.Entity;
import jakarta.persistence.FetchType;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;
import jakarta.persistence.JoinColumn;
import jakarta.persistence.ManyToOne;
import jakarta.persistence.PrePersist;
import jakarta.persistence.Table;

import java.time.LocalDateTime;

@Entity
@Table(name = "books")
public class Book {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(nullable = false)
    private String title;

    @Column(nullable = false)
    private String author;

    @Column(name = "year_written", nullable = false)
    private int year;

    @Column(nullable = false)
    private boolean physical;

    @Column(name = "file_name")
    private String fileName;

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "uploader_id", nullable = false)
    private User uploader;

    @Column(name = "created_at", updatable = false)
    private LocalDateTime createdAt;

    public Book() {}

    @PrePersist
    protected void onCreate() {
        this.createdAt = LocalDateTime.now();
    }

    // --- getters & setters ---
```

```

public Long getId() {
    return id;
}

public void setId(Long id) {
    this.id = id;
}

public String getTitle() {
    return title;
}

public void setTitle(String title) {
    this.title = title;
}

public String getAuthor() {
    return author;
}

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

public int getYear() {
    return year;
}

public void setYear(int year) {
    this.year = year;
}

public boolean isPhysical() {
    return physical;
}

public void setPhysical(boolean physical) {
    this.physical = physical;
}

public String getFileName() {
    return fileName;
}

public void setFileName(String fileName) {
    this.fileName = fileName;
}

public User getUploader() {
    return uploader;
}

public void setUploader(User uploader) {
    this.uploader = uploader;
}

```

```

        public LocalDateTime getCreatedAt() {
            return createdAt;
        }
    }
}

```

Файл Collection.java

```

package com.example.booklover.model.entity;

import jakarta.persistence.*;
import java.util.HashSet;
import java.util.Set;

@Entity
@Table(name = "collections")
public class Collection {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    /** Название подборки */
    @Column(nullable = false)
    private String name;

    /** Владелец подборки */
    @ManyToOne(fetch = FetchType.LAZY, optional = false)
    @JoinColumn(name = "owner_id", nullable = false)
    private User owner;

    /** Книги, входящие в подборку */
    @ManyToMany
    @JoinTable(
        name = "collection_books",
        joinColumns = @JoinColumn(name = "collection_id"),
        inverseJoinColumns = @JoinColumn(name = "book_id")
    )
    private Set<Book> books = new HashSet<>();

    public Collection() {}

    public Long getId() {
        return id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public User getOwner() {

```

```

        return owner;
    }

    public void setOwner(User owner) {
        this.owner = owner;
    }

    public Set<Book> getBooks() {
        return books;
    }

    public void setBooks(Set<Book> books) {
        this.books = books;
    }
}

```

Файл Comment.java

```

package com.example.booklover.model.entity;

import jakarta.persistence.*;
import java.time.LocalDateTime;

@Entity
@Table(name = "comments")
public class Comment {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    /** Текст комментария */
    @Column(nullable = false, columnDefinition = "TEXT")
    private String text;

    /** Пользователь, оставивший комментарий */
    @ManyToOne(fetch = FetchType.LAZY, optional = false)
    @JoinColumn(name = "user_id", nullable = false)
    private User author;

    /** Книга, к которой относится комментарий */
    @ManyToOne(fetch = FetchType.LAZY, optional = false)
    @JoinColumn(name = "book_id", nullable = false)
    private Book book;

    /** Время создания комментария */
    @Column(name = "created_at", updatable = false, nullable =
false)
    private LocalDateTime createdAt;

    public Comment() {}

    @PrePersist
    protected void onCreate() {
        this.createdAt = LocalDateTime.now();
    }
}

```

```

    }

    // --- getters ---

    public Long getId() {
        return id;
    }

    public String getText() {
        return text;
    }

    public User getAuthor() {
        return author;
    }

    public Book getBook() {
        return book;
    }

    public LocalDateTime getCreatedAt() {
        return createdAt;
    }

    // --- setters ---

    public void setText(String text) {
        this.text = text;
    }

    public void setAuthor(User author) {
        this.author = author;
    }

    public void setBook(Book book) {
        this.book = book;
    }
}

```

Файл Message.java

```

package com.example.booklover.model.entity;

import jakarta.persistence.*;
import java.time.LocalDateTime;

@Entity
@Table(name = "messages")
public class Message {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    /** Текст сообщения */

```

```

@Column(nullable = false, columnDefinition = "TEXT")
private String text;

/** Пользователь-автор сообщения */
@ManyToOne(fetch = FetchType.LAZY, optional = false)
@JoinColumn(name = "author_id", nullable = false)
private User author;

/** Время отправки */
@Column(name = "sent_at", updatable = false, nullable = false)
private LocalDateTime sentAt;

public Message() {}

@PrePersist
protected void onCreate() {
    this.sentAt = LocalDateTime.now();
}

// --- getters ---

public Long getId() {
    return id;
}

public String getText() {
    return text;
}

public User getAuthor() {
    return author;
}

public LocalDateTime getSentAt() {
    return sentAt;
}

// --- setters ---

public void setText(String text) {
    this.text = text;
}

public void setAuthor(User author) {
    this.author = author;
}
}

```

Файл User.java

```

package com.example.booklover.model.entity;

import com.fasterxml.jackson.annotation.JsonIgnore;
import jakarta.persistence.CollectionTable;
import jakarta.persistence.Column;

```

```

import jakarta.persistence.ElementCollection;
import jakarta.persistence.Entity;
import jakarta.persistence.FetchType;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;
import jakarta.persistence.JoinColumn;
import jakarta.persistence.OneToMany;
import jakarta.persistence.Table;
import jakarta.persistence.CascadeType;

import java.util.ArrayList;
import java.util.HashSet;
import java.util.List;
import java.util.Set;

@Entity
@Table(name = "users")
public class User {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(name = "first_name", nullable = false)
    private String firstName;

    @Column(name = "last_name", nullable = false)
    private String lastName;

    @Column(nullable = false, unique = true)
    private String username;

    @Column(nullable = false)
    @JsonIgnore
    private String password;

    @ElementCollection(fetch = FetchType.EAGER)
    @CollectionTable(
        name = "user_roles",
        joinColumns = @JoinColumn(name = "user_id")
    )
    @Column(name = "role")
    private Set<String> roles = new HashSet<>();

    @OneToMany(
        mappedBy = "uploader",
        cascade = CascadeType.ALL,
        orphanRemoval = true
    )
    @JsonIgnore
    private List<Book> books = new ArrayList<>();

    public User() {}

    // --- getters & setters ---

```

```

public Long getId() {
    return id;
}

public void setId(Long id) {
    this.id = id;
}

public String getFirstName() {
    return firstName;
}

public void setFirstName(String firstName) {
    this.firstName = firstName;
}

public String getLastName() {
    return lastName;
}

public void setLastName(String lastName) {
    this.lastName = lastName;
}

public String getUsername() {
    return username;
}

public void setUsername(String username) {
    this.username = username;
}

public String getPassword() {
    return password;
}

public void setPassword(String password) {
    this.password = password;
}

public Set<String> getRoles() {
    return roles;
}

public void setRoles(Set<String> roles) {
    this.roles = roles;
}

public List<Book> getBooks() {
    return books;
}

public void setBooks(List<Book> books) {
    this.books = books;
}

```

```
}
```

Файл BookDto.java

```
package com.example.booklover.model.entity.dto;

import java.time.LocalDateTime;

public class BookDto {

    private Long id;
    private String title;
    private String author;
    private int year;
    private boolean physical;
    private String fileName;           // имя файла для скачивания,
если электронная
    private String uploaderUsername;   // логин пользователя-
загрузчика
    private LocalDateTime createdAt;   // дата загрузки
    private Boolean isAllowedDownload; // дата загрузки

    public BookDto() {
    }

    public BookDto(Long id, String title, String author, int year,
boolean physical,
                    String fileName, String uploaderUsername,
LocalDateTime createdAt, Boolean isAllowedDownload) {
        this.id = id;
        this.title = title;
        this.author = author;
        this.year = year;
        this.physical = physical;
        this.fileName = fileName;
        this.uploaderUsername = uploaderUsername;
        this.createdAt = createdAt;
        this.isAllowedDownload = isAllowedDownload;
    }

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public String getTitle() {
        return title;
    }

    public void setTitle(String title) {
        this.title = title;
    }
}
```

```

public String getAuthor() {
    return author;
}

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

public int getYear() {
    return year;
}

public void setYear(int year) {
    this.year = year;
}

public boolean isPhysical() {
    return physical;
}

public void setPhysical(boolean physical) {
    this.physical = physical;
}

public String getFileName() {
    return fileName;
}

public void setFileName(String fileName) {
    this.fileName = fileName;
}

public String getUploaderUsername() {
    return uploaderUsername;
}

public void setUploaderUsername(String uploaderUsername) {
    this.uploaderUsername = uploaderUsername;
}

public LocalDateTime getCreatedAt() {
    return createdAt;
}

public void setCreatedAt(LocalDateTime createdAt) {
    this.createdAt = createdAt;
}

public Boolean getAllowedDownload() {
    return isAllowedDownload;
}

public void setAllowedDownload(Boolean allowedDownload) {
    isAllowedDownload = allowedDownload;
}

```

```
}
```

Файл CollectionDto.java

```
package com.example.booklover.model.entity.dto;

import java.util.List;

public class CollectionDto {

    private Long id;
    private String name;
    private String ownerUsername;
    private List<BookDto> books;

    public CollectionDto() {
    }

    public CollectionDto(Long id, String name, String
ownerUsername, List<BookDto> books) {
        this.id = id;
        this.name = name;
        this.ownerUsername = ownerUsername;
        this.books = books;
    }

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getOwnerUsername() {
        return ownerUsername;
    }

    public void setOwnerUsername(String ownerUsername) {
        this.ownerUsername = ownerUsername;
    }

    public List<BookDto> getBooks() {
        return books;
    }

    public void setBooks(List<BookDto> books) {
```

```

        this.books = books;
    }
}

```

Файл CommentDto.java

```

package com.example.booklover.model.entity.dto;

import java.time.LocalDateTime;

public class CommentDto {

    private Long id;
    private Long bookId;
    private String text;
    private String authorUsername;
    private LocalDateTime createdAt;

    public CommentDto() {
    }

    public CommentDto(Long id, Long bookId, String text, String
authorUsername, LocalDateTime createdAt) {
        this.id = id;
        this.bookId = bookId;
        this.text = text;
        this.authorUsername = authorUsername;
        this.createdAt = createdAt;
    }

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public Long getBookId() {
        return bookId;
    }

    public void setBookId(Long bookId) {
        this.bookId = bookId;
    }

    public String getText() {
        return text;
    }

    public void setText(String text) {
        this.text = text;
    }

    public String getAuthorUsername() {

```

```

        return authorUsername;
    }

    public void setAuthorUsername(String authorUsername) {
        this.authorUsername = authorUsername;
    }

    public LocalDateTime getCreatedAt() {
        return createdAt;
    }

    public void setCreatedAt(LocalDateTime createdAt) {
        this.createdAt = createdAt;
    }
}

```

Файл CreateBookRequest.java

```

package com.example.booklover.model.entity.dto;

public class CreateBookRequest {

    private String title;
    private String author;
    private int year;
    private boolean physical;

    public CreateBookRequest() {
    }

    public CreateBookRequest(String title, String author, int
year, boolean physical) {
        this.title = title;
        this.author = author;
        this.year = year;
        this.physical = physical;
    }

    public String getTitle() {
        return title;
    }

    public void setTitle(String title) {
        this.title = title;
    }

    public String getAuthor() {
        return author;
    }

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

    public int getYear() {

```

```

        return year;
    }

    public void setYear(int year) {
        this.year = year;
    }

    public boolean isPhysical() {
        return physical;
    }

    public void setPhysical(boolean physical) {
        this.physical = physical;
    }
}

```

Файл CreateCollectionRequest.java

```

package com.example.booklover.model.entity.dto;

import java.util.List;

public class CreateCollectionRequest {

    private String name;
    private List<Long> bookIds;

    public CreateCollectionRequest() {
    }

    public CreateCollectionRequest(String name, List<Long>
bookIds) {
        this.name = name;
        this.bookIds = bookIds;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public List<Long> getBookIds() {
        return bookIds;
    }

    public void setBookIds(List<Long> bookIds) {
        this.bookIds = bookIds;
    }
}

```

Файл CreateCommentRequest.java

					ПГУ 1.090301.29.001 ПЗ	Лист
						88
Изм.	Лист	№ докум.	Подпись	Дата		

```

package com.example.booklover.model.entity.dto;

public class CreateCommentRequest {

    private String text;

    public CreateCommentRequest() {
    }

    public CreateCommentRequest(String text) {
        this.text = text;
    }

    public String getText() {
        return text;
    }

    public void setText(String text) {
        this.text = text;
    }
}

```

Файл LoginRequest.java

```

package com.example.booklover.model.entity.dto;

public class LoginRequest {

    private String username;
    private String password;

    public LoginRequest() {
    }

    public String getUsername() {
        return username;
    }

    public void setUsername(String username) {
        this.username = username;
    }

    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }
}

```

Файл MessageDto.java

```

package com.example.booklover.model.entity.dto;

```

```

import java.time.LocalDateTime;

public class MessageDto {

    private Long id;
    private String text;
    private String authorUsername;
    private LocalDateTime timestamp;

    public MessageDto() {
    }

    public MessageDto(Long id, String text, String
authorUsername, LocalDateTime timestamp) {
        this.id = id;
        this.text = text;
        this.authorUsername = authorUsername;
        this.timestamp = timestamp;
    }

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public String getText() {
        return text;
    }

    public void setText(String text) {
        this.text = text;
    }

    public String getAuthorUsername() {
        return authorUsername;
    }

    public void setAuthorUsername(String authorUsername) {
        this.authorUsername = authorUsername;
    }

    public LocalDateTime getTimestamp() {
        return timestamp;
    }

    public void setTimestamp(LocalDateTime timestamp) {
        this.timestamp = timestamp;
    }
}

```

Файл RegisterRequest.java

					ПГУ 1.090301.29.001 ПЗ	Лист
						90
Изм.	Лист	№ докум.	Подпись	Дата		

```

package com.example.booklover.model.entity.dto;

public class RegisterRequest {

    private String firstName;
    private String lastName;
    private String username;
    private String password;

    public RegisterRequest() {
    }

    public String getFirstName() {
        return firstName;
    }

    public void setFirstName(String firstName) {
        this.firstName = firstName;
    }

    public String getLastName() {
        return lastName;
    }

    public void setLastName(String lastName) {
        this.lastName = lastName;
    }

    public String getUsername() {
        return username;
    }

    public void setUsername(String username) {
        this.username = username;
    }

    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }
}

```

Файл UploadBookRequest.java

```

package com.example.booklover.model.entity.dto;

public class UpdateBookRequest {

    private String title;
    private String author;
    private int year;

```

```

        private boolean physical;

        public UpdateBookRequest() {
        }

        public UpdateBookRequest(String title, String author, int
year, boolean physical) {
            this.title = title;
            this.author = author;
            this.year = year;
            this.physical = physical;
        }

        public String getTitle() {
            return title;
        }

        public void setTitle(String title) {
            this.title = title;
        }

        public String getAuthor() {
            return author;
        }

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

        public int getYear() {
            return year;
        }

        public void setYear(int year) {
            this.year = year;
        }

        public boolean isPhysical() {
            return physical;
        }

        public void setPhysical(boolean physical) {
            this.physical = physical;
        }
    }

```

Файл UpdateUserRequest.java

```

package com.example.booklover.model.entity.dto;

public class UpdateUserRequest {

    private String firstName;
    private String lastName;
    private String password;

```

```

        public UpdateUserRequest() {
        }

        public UpdateUserRequest(String firstName, String lastName,
String password) {
            this.firstName = firstName;
            this.lastName = lastName;
            this.password = password;
        }

        public String getFirstName() {
            return firstName;
        }

        public void setFirstName(String firstName) {
            this.firstName = firstName;
        }

        public String getLastName() {
            return lastName;
        }

        public void setLastName(String lastName) {
            this.lastName = lastName;
        }

        public String getPassword() {
            return password;
        }

        public void setPassword(String password) {
            this.password = password;
        }
    }

```

Файл UserDto.java

```

package com.example.booklover.model.entity.dto;

public class UserDto {

    private Long id;
    private String firstName;
    private String lastName;
    private String username;

    public UserDto() {
    }

    public UserDto(Long id, String firstName, String lastName,
String username) {
        this.id = id;
        this.firstName = firstName;
        this.lastName = lastName;
    }

```

```

        this.username = username;
    }

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public String getFirstName() {
        return firstName;
    }

    public void setFirstName(String firstName) {
        this.firstName = firstName;
    }

    public String getLastName() {
        return lastName;
    }

    public void setLastName(String lastName) {
        this.lastName = lastName;
    }

    public String getUsername() {
        return username;
    }

    public void setUsername(String username) {
        this.username = username;
    }
}

```

Файл BookRepository.java

```

package com.example.booklover.repository;

import com.example.booklover.model.entity.Book;
import org.springframework.data.domain.Pageable;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;

import java.util.List;

public interface BookRepository extends JpaRepository<Book, Long>
{
    // для получения ленты последних добавленных книг
    List<Book> findAllByOrderByCreatedAtDesc(Pageable pageable);
}

```

Файл CollectionRepository.java

```

package com.example.booklover.repository;

import com.example.booklover.model.entity.Collection;
import org.springframework.data.jpa.repository.JpaRepository;

import java.util.List;

public interface CollectionRepository extends
JpaRepository<Collection, Long> {
    // подборки конкретного пользователя
    List<Collection> findAllByOwnerUsername(String username);
    // все подборки – метод findAll() уже предоставляется
JpaRepository
}

```

Файл CommentRepository.java

```

package com.example.booklover.repository;

import com.example.booklover.model.entity.Comment;
import org.springframework.data.jpa.repository.JpaRepository;
import java.util.List;

public interface CommentRepository extends JpaRepository<Comment,
Long> {

    /**
     * Возвращает все комментарии для указанной книги,
     * отсортированные по времени создания (от самых ранних к
поздним) .
     */
    List<Comment> findByBookIdOrderByCreatedAtAsc(Long bookId);
}

```

Файл MessageRepository.java

```

package com.example.booklover.repository;

import com.example.booklover.model.entity.Message;
import org.springframework.data.jpa.repository.JpaRepository;
import java.util.List;

public interface MessageRepository extends JpaRepository<Message,
Long> {

    /**
     * Возвращает всю историю чата,
     * отсортированную по времени отправки (от самых ранних к
поздним) .
     */
    List<Message> findAllByOrderBySentAtAsc();
}

```

Файл UserRepository.java

					ПГУ 1.090301.29.001 ПЗ	Лист
						95
Изм.	Лист	№ докум.	Подпись	Дата		

```

package com.example.booklover.repository;

import com.example.booklover.model.entity.User;
import org.springframework.data.jpa.repository.JpaRepository;

import java.util.Optional;

public interface UserRepository extends JpaRepository<User, Long>
{
    Optional<User> findByUsername(String username);
    boolean existsByUsername(String username);
}

```

Файл BookService.java

```

package com.example.booklover.service;

import com.example.booklover.model.entity.dto.BookDto;
import com.example.booklover.model.entity.dto.CreateBookRequest;
import com.example.booklover.model.entity.dto.UpdateBookRequest;
import org.springframework.core.io.Resource;
import org.springframework.web.multipart.MultipartFile;

import java.io.IOException;
import java.util.List;

public interface BookService {
    List<BookDto> searchBooks(String title, String author,
Boolean physical);
    List<BookDto> findLatest(int count);
    Resource loadBookFile(Long id);
    BookDto getBookById(Long id);
    BookDto createBook(String username, CreateBookRequest
request, MultipartFile file);
    BookDto updateBook(Long id, String username,
UpdateBookRequest request, MultipartFile file);
    void deleteBook(Long id, String username);

    String getHtmlBook(Long id) throws IOException;
}

```

Файл ChatService.java

```

package com.example.booklover.service;

import com.example.booklover.model.entity.dto.MessageDto;

import java.util.List;

public interface ChatService {
    List<MessageDto> getRecentMessages();
    void saveMessage(MessageDto message);
}

```

Файл CollectionService.java

```
package com.example.booklover.service;

import com.example.booklover.model.entity.dto.CollectionDto;

import java.util.List;

public interface CollectionService {
    List<CollectionDto> findAllByUsername(String username);
    List<CollectionDto> findAll();
    CollectionDto findByIdAndUsername(Long id, String username);
    CollectionDto create(String username, String name, List<Long>
bookIds);
}
```

Файл CommentService.java

```
package com.example.booklover.service;

import com.example.booklover.model.entity.dto.CommentDto;

import java.util.List;

public interface CommentService {
    List<CommentDto> findByBook(Long bookId);
    CommentDto create(Long bookId, String username, String text);
}
```

Файл FileStorageService.java

```
package com.example.booklover.service;

import org.springframework.core.io.Resource;
import org.springframework.web.multipart.MultipartFile;

public interface FileStorageService {
    /**
     * Сохраняет файл и возвращает сгенерированное имя.
     */
    String storeFile(MultipartFile file);

    /**
     * Загружает файл как Resource.
     */
    Resource loadFileAsResource(String filename);
}
```

Файл UserService.java

```
package com.example.booklover.service;

import com.example.booklover.model.entity.dto.UpdateUserRequest;
```

```

import com.example.booklover.model.entity.dto.UserDto;
import com.example.booklover.model.entity.User;

public interface UserService {
    boolean existsByUsername(String username);
    void save(User user);
    UserDto getByUsername(String username);
    UserDto updateProfile(String username, UpdateUserRequest
request);
}

```

Файл BookServiceImpl.java

```

package com.example.booklover.service.impl;

import com.example.booklover.exception.AccessDeniedException;
import com.example.booklover.exception.NotFoundException;
import com.example.booklover.model.entity.Book;
import com.example.booklover.model.entity.User;
import com.example.booklover.model.entity.dto.BookDto;
import com.example.booklover.model.entity.dto.CreateBookRequest;
import com.example.booklover.model.entity.dto.UpdateBookRequest;
import com.example.booklover.repository.BookRepository;
import com.example.booklover.repository.UserRepository;
import com.example.booklover.service.BookService;
import com.example.booklover.util.Fb2ToHtmlConverter;
import jakarta.persistence.EntityManager;
import jakarta.persistence.PersistenceContext;
import jakarta.persistence.criteria.CriteriaBuilder;
import jakarta.persistence.criteria.CriteriaQuery;
import jakarta.persistence.criteria.Predicate;
import jakarta.persistence.criteria.Root;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.core.io.UrlResource;
import org.springframework.core.io.Resource;
import org.springframework.data.domain.PageRequest;
import
org.springframework.security.core.authority.SimpleGrantedAuthority;
import
org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.stereotype.Service;
import org.springframework.web.multipart.MultipartFile;

import java.io.IOException;
import java.net.MalformedURLException;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.util.ArrayList;
import java.util.List;
import java.util.UUID;
import java.util.stream.Collectors;

```

```

@Service
public class BookServiceImpl implements BookService {

    private final BookRepository bookRepository;
    private final UserRepository userRepository;
    private final Fb2ToHtmlConverter converter;
    private final Path uploadDir = Paths.get("uploads");

    @PersistenceContext
    private EntityManager entityManager;

    @Autowired
    public BookServiceImpl(BookRepository bookRepository,
                           UserRepository userRepository,
                           Fb2ToHtmlConverter converter) {
        this.bookRepository = bookRepository;
        this.userRepository = userRepository;
        this.converter = converter;
        try {
            Files.createDirectories(uploadDir);
        } catch (Exception e) {
            throw new RuntimeException("Could not create upload
directory", e);
        }
    }

    public boolean isDownloadAllowed() {
        return
SecurityContextHolder.getContext().getAuthentication()
                           .getAuthorities().contains(new
SimpleGrantedAuthority("ROLE_USER"));
    }

    @Override
    public List<BookDto> searchBooks(String title, String author,
Boolean physical) {

        // 1. Получаем CriteriaBuilder
        CriteriaBuilder cb = entityManager.getCriteriaBuilder();

        // 2. Создаем CriteriaQuery для сущности Book
        CriteriaQuery<Book> query = cb.createQuery(Book.class);

        // 3. Определяем корневую таблицу (основной объект
запроса)
        Root<Book> book = query.from(Book.class);

        // 4. Создаем список для условий (predicates)
        List<Predicate> predicates = new ArrayList<>();

        // 5. Добавляем условия фильтрации в зависимости от
параметров

        // Условие для совпадения названия (title)
        if (title != null && !title.isEmpty()) {

```

```

        predicates.add(cb.like(cb.lower(book.get("title")),
"%\" + title.toLowerCase() + \"%"));
    }

    // Условие для совпадения автора (author)
    if (author != null && !author.isEmpty()) {
        predicates.add(cb.like(cb.lower(book.get("author")),
"%\" + author.toLowerCase() + \"%"));
    }

    // Условие для физического состояния книги (physical)
    if (physical != null) {
        predicates.add(cb.equal(book.get("physical"),
physical));
    }
    // 6. Собираем все условия вместе и добавляем их в запрос
    query.where(cb.and(predicates.toArray(new
Predicate[0]))));

    return entityManager.createQuery(query).getResultList()
        .stream()
        .map(b -> toDto(b, isDownloadAllowed()))
        .collect(Collectors.toList());
}

@Override
public List<BookDto> findLatest(int count) {
    return bookRepository
        .findAllByOrderByCreatedAtDesc(PageRequest.of(0,
count))
        .stream()
        .map(b -> toDto(b, isDownloadAllowed()))
        .collect(Collectors.toList());
}

@Override
public Resource loadBookFile(Long id) {
    Book book = bookRepository.findById(id)
        .orElseThrow(() -> new NotFoundException("Book
not found: \" + id));
    if (book.isPhysical() || book.getFileName() == null) {
        throw new NotFoundException("No file for book: \" +
id);
    }
    try {
        Path file = uploadDir.resolve(book.getFileName());
        return new UrlResource(file.toUri());
    } catch (MalformedURLException e) {
        throw new RuntimeException("Error loading file", e);
    }
}

@Override
public BookDto getBookById(Long id) {
    Book book = bookRepository.findById(id)

```

```

        .orElseThrow(() -> new NotFoundException("Book
not found: " + id));
        return toDto(book, isDownloadAllowed());
    }

    @Override
    public BookDto createBook(String username, CreateBookRequest
request, MultipartFile file) {
        User user = userRepository.findByUsername(username)
        .orElseThrow(() -> new NotFoundException("User
not found: " + username));
        Book book = new Book();
        book.setTitle(request.getTitle());
        book.setAuthor(request.getAuthor());
        book.setYear(request.getYear());
        book.setPhysical(request.isPhysical());
        book.setUploader(user);

        if (!request.isPhysical() && file != null &&
!file.isEmpty()) {
            String filename = UUID.randomUUID() + "_" +
file.getOriginalFilename();
            try {
                Files.copy(file.getInputStream(),
uploadDir.resolve(filename));
            } catch (Exception e) {
                throw new RuntimeException("File upload failed",
e);
            }
            book.setFileName(filename);
        }

        Book saved = bookRepository.save(book);
        return toDto(saved, isDownloadAllowed());
    }

    @Override
    public BookDto updateBook(Long id, String username,
UpdateBookRequest request, MultipartFile file) {
        Book book = bookRepository.findById(id)
        .orElseThrow(() -> new NotFoundException("Book
not found: " + id));
        if (!book.getUploader().getUsername().equals(username))
        {
            throw new AccessDeniedException("Not owner of book:
" + id);
        }
        book.setTitle(request.getTitle());
        book.setAuthor(request.getAuthor());
        book.setYear(request.getYear());
        book.setPhysical(request.isPhysical());

        if (!request.isPhysical() && file != null &&
!file.isEmpty()) {
            String filename = UUID.randomUUID() + "_" +
file.getOriginalFilename();

```

```

        try {
            Files.copy(file.getInputStream(),
uploadDir.resolve(filename));
            book.setFileName(filename);
        } catch (Exception e) {
            throw new RuntimeException("File upload failed",
e);
        }
    } else if (request.isPhysical()) {
        book.setFileName(null);
    }

    Book updated = bookRepository.save(book);
    return toDto(updated, isDownloadAllowed());
}

@Override
public void deleteBook(Long id, String username) {
    Book book = bookRepository.findById(id)
        .orElseThrow(() -> new NotFoundException("Book
not found: " + id));
    if (!book.getUploader().getUsername().equals(username))
    {
        throw new AccessDeniedException("Not owner of book:
" + id);
    }
    bookRepository.delete(book);
}

@Override
public String getHtmlBook(Long id) throws IOException {
    Resource book = loadBookFile(id);
    return converter.convert(book.getInputStream());
}

private BookDto toDto(Book b, Boolean isDownloadAllowed) {
    return new BookDto(
        b.getId(),
        b.getTitle(),
        b.getAuthor(),
        b.getYear(),
        b.isPhysical(),
        b.getFileName(),
        b.getUploader().getUsername(),
        b.getCreatedAt(),
        isDownloadAllowed
    );
}
}

```

Файл ChatServiceImpl.java

```

package com.example.booklover.service.impl;

import com.example.booklover.exception.NotFoundException;

```

```

import com.example.booklover.model.entity.Message;
import com.example.booklover.model.entity.User;
import com.example.booklover.model.entity.dto.MessageDto;
import com.example.booklover.repository.MessageRepository;
import com.example.booklover.repository.UserRepository;
import com.example.booklover.service.ChatService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

import java.util.List;
import java.util.stream.Collectors;

@Service
public class ChatServiceImpl implements ChatService {

    private final MessageRepository messageRepository;
    private final UserRepository userRepository;

    @Autowired
    public ChatServiceImpl(MessageRepository messageRepository,
                           UserRepository userRepository) {
        this.messageRepository = messageRepository;
        this.userRepository = userRepository;
    }

    @Override
    public List<MessageDto> getRecentMessages() {
        return messageRepository
            .findAllByOrderBySentAtAsc()
            .stream()
            .map(m -> new MessageDto(
                m.getId(),
                m.getText(),
                m.getAuthor().getUsername(),
                m.getSentAt()
            ))
            .collect(Collectors.toList());
    }

    @Transactional
    @Override
    public void saveMessage(MessageDto message) {
        User user = userRepository.findByUsername(message.getAuthorUsername())
            .orElseThrow(() -> new NotFoundException("User not found: " + message.getAuthorUsername()));

        Message m = new Message();
        m.setText(message.getText());
        m.setAuthor(user);
        // sentAt будет установлен через @PrePersist
        messageRepository.save(m);
    }
}

```

Файл CollectionServiceImpl.java

```
package com.example.booklover.service.impl;

import com.example.booklover.exception.NotFoundException;
import com.example.booklover.model.entity.Book;
import com.example.booklover.model.entity.Collection;
import com.example.booklover.model.entity.User;
import com.example.booklover.model.entity.dto.BookDto;
import com.example.booklover.model.entity.dto.CollectionDto;
import com.example.booklover.repository.BookRepository;
import com.example.booklover.repository.CollectionRepository;
import com.example.booklover.repository.UserRepository;
import com.example.booklover.service.CollectionService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.util.List;
import java.util.Set;
import java.util.stream.Collectors;

@Service
public class CollectionServiceImpl implements CollectionService
{

    private final CollectionRepository collectionRepository;
    private final UserRepository userRepository;
    private final BookRepository bookRepository;

    @Autowired
    public CollectionServiceImpl(CollectionRepository collectionRepository,
                                UserRepository userRepository,
                                BookRepository bookRepository)
    {
        this.collectionRepository = collectionRepository;
        this.userRepository = userRepository;
        this.bookRepository = bookRepository;
    }

    @Override
    public List<CollectionDto> findAllByUsername(String
username) {
        return
collectionRepository.findAllByOwnerUsername(username)
                .stream()
                .map(this::toDto)
                .collect(Collectors.toList());
    }

    @Override
    public List<CollectionDto> findAll() {
        return collectionRepository.findAll()
                .stream()
                .map(this::toDto)
    }
}
```

```

        .collect(Collectors.toList());
    }

    @Override
    public CollectionDto findByIdAndUsername(Long id, String
username) {
        Collection coll = collectionRepository.findById(id)
            .orElseThrow(() -> new
NotFoundException("Collection not found: " + id));
        if (!coll.getOwner().getUsername().equals(username)) {
            throw new NotFoundException("Collection not found: "
+ id);
        }
        return toDto(coll);
    }

    @Override
    public CollectionDto create(String username, String name,
List<Long> bookIds) {
        User user = userRepository.findByUsername(username)
            .orElseThrow(() -> new NotFoundException("User
not found: " + username));
        Set<Book> books =
bookRepository.findAllById(bookIds).stream().collect(Collectors.toSet
());

        Collection coll = new Collection();
        coll.setName(name);
        coll.setOwner(user);
        coll.setBooks(books);
        Collection saved = collectionRepository.save(coll);
        return toDto(saved);
    }

    private CollectionDto toDto(Collection c) {
        List<BookDto> bookDtos = c.getBooks().stream()
            .map(b -> new BookDto(
                b.getId(),
                b.getTitle(),
                b.getAuthor(),
                b.getYear(),
                b.isPhysical(),
                b.getFileName(),
                b.getUploader().getUsername(),
                b.getCreatedAt(),
                false
            ))
            .collect(Collectors.toList());
        return new CollectionDto(
            c.getId(),
            c.getName(),
            c.getOwner().getUsername(),
            bookDtos
        );
    }
}

```

Файл CommentServiceImpl.java

```
package com.example.booklover.service.impl;

import com.example.booklover.exception.NotFoundException;
import com.example.booklover.model.entity.Book;
import com.example.booklover.model.entity.Comment;
import com.example.booklover.model.entity.User;
import com.example.booklover.model.entity.dto.CommentDto;
import com.example.booklover.repository.BookRepository;
import com.example.booklover.repository.CommentRepository;
import com.example.booklover.repository.UserRepository;
import com.example.booklover.service.CommentService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.util.List;
import java.util.stream.Collectors;

@Service
public class CommentServiceImpl implements CommentService {

    private final CommentRepository commentRepository;
    private final BookRepository bookRepository;
    private final UserRepository userRepository;

    @Autowired
    public CommentServiceImpl(CommentRepository commentRepository,
                              BookRepository bookRepository,
                              UserRepository userRepository) {
        this.commentRepository = commentRepository;
        this.bookRepository = bookRepository;
        this.userRepository = userRepository;
    }

    @Override
    public List<CommentDto> findByBook(Long bookId) {
        return commentRepository
            .findByBookIdOrderByCreatedAtAsc(bookId)
            .stream()
            .map(c -> new CommentDto(
                c.getId(),
                c.getBook().getId(),
                c.getText(),
                c.getAuthor().getUsername(),
                c.getCreatedAt()
            ))
            .collect(Collectors.toList());
    }

    @Override
    public CommentDto create(Long bookId, String username, String
text) {
        Book book = bookRepository.findById(bookId)
```

```

        .orElseThrow(() -> new NotFoundException("Book
not found: " + bookId));
        User user = userRepository.findByUsername(username)
        .orElseThrow(() -> new NotFoundException("User
not found: " + username));

        Comment comment = new Comment();
        comment.setBook(book);
        comment.setAuthor(user);
        comment.setText(text);

        Comment saved = commentRepository.save(comment);
        return new CommentDto(
            saved.getId(),
            saved.getBook().getId(),
            saved.getText(),
            saved.getAuthor().getUsername(),
            saved.getCreatedAt()
        );
    }
}

```

Файл FileStorageServiceImpl.java

```

package com.example.booklover.service.impl;

import com.example.booklover.service.FileStorageService;
import jakarta.annotation.PostConstruct;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.core.io.UrlResource;
import org.springframework.core.io.Resource;
import org.springframework.stereotype.Service;
import org.springframework.util.StringUtils;
import org.springframework.web.multipart.MultipartFile;

import java.io.IOException;
import java.net.MalformedURLException;
import java.nio.file.*;
import java.util.UUID;

@Service
public class FileStorageServiceImpl implements FileStorageService
{
    @Value("${file.upload-dir}")
    private String uploadDir;

    private Path fileStorageLocation;

    @PostConstruct
    public void init() {
        fileStorageLocation =
Paths.get(uploadDir).toAbsolutePath().normalize();
        try {
            Files.createDirectories(fileStorageLocation);
        }
    }
}

```

```

        } catch (IOException ex) {
            throw new RuntimeException("Could not create upload
directory", ex);
        }
    }

    @Override
    public String storeFile(MultipartFile file) {
        String original =
StringUtils.cleanPath(file.getOriginalFilename());
        String filename = UUID.randomUUID().toString() + "_" +
original;

        try {
            if (original.contains("..")) {
                throw new RuntimeException("Filename contains
invalid path sequence " + original);
            }
            Path target = fileStorageLocation.resolve(filename);
            Files.copy(file.getInputStream(), target,
StandardCopyOption.REPLACE_EXISTING);
            return filename;
        } catch (IOException ex) {
            throw new RuntimeException("Could not store file " +
original, ex);
        }
    }

    @Override
    public Resource loadFileAsResource(String filename) {
        try {
            Path file =
fileStorageLocation.resolve(filename).normalize();
            Resource resource = new UrlResource(file.toUri());
            if (resource.exists() && resource.isReadable()) {
                return resource;
            } else {
                throw new RuntimeException("File not found: " +
filename);
            }
        } catch (MalformedURLException ex) {
            throw new RuntimeException("File not found: " +
filename, ex);
        }
    }
}

```

Файл UserServiceImpl.java

```

package com.example.booklover.service.impl;

import com.example.booklover.exception.NotFoundException;
import com.example.booklover.model.entity.User;
import com.example.booklover.model.entity.dto.UpdateUserRequest;
import com.example.booklover.model.entity.dto.UserDto;
import com.example.booklover.repository.UserRepository;

```

```

import com.example.booklover.service.UserService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.core.GrantedAuthority;
import
org.springframework.security.core.authority.SimpleGrantedAuthority;
import org.springframework.security.core.userdetails.*;
import
org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.stereotype.Service;

import java.util.Collections;
import java.util.List;
import java.util.stream.Collectors;

@Service
public class UserServiceImpl implements UserService,
UserDetailsService {

    private final UserRepository userRepository;
    private final PasswordEncoder passwordEncoder;

    @Autowired
    public UserServiceImpl(UserRepository userRepository,
                           PasswordEncoder passwordEncoder) {
        this.userRepository = userRepository;
        this.passwordEncoder = passwordEncoder;
    }

    // --- UserService methods ---

    @Override
    public boolean existsByUsername(String username) {
        return userRepository.existsByUsername(username);
    }

    @Override
    public void save(User user) {
        userRepository.save(user);
    }

    @Override
    public UserDto getByUsername(String username) {
        User user = userRepository.findByUsername(username)
            .orElseThrow(() -> new NotFoundException("User
not found: " + username));
        return new UserDto(
            user.getId(),
            user.getFirstName(),
            user.getLastName(),
            user.getUsername()
        );
    }

    @Override
    public UserDto updateProfile(String username,
UpdateUserRequest request) {

```

```

        User user = userRepository.findByUsername(username)
            .orElseThrow(() -> new NotFoundException("User
not found: " + username));

        if (request.getFirstName() != null) {
            user.setFirstName(request.getFirstName());
        }
        if (request.getLastName() != null) {
            user.setLastName(request.getLastName());
        }
        if (request.getPassword() != null &&
!request.getPassword().isBlank()) {
user.setPassword(passwordEncoder.encode(request.getPassword()));
        }
        userRepository.save(user);

        return new UserDto(
            user.getId(),
            user.getFirstName(),
            user.getLastName(),
            user.getUsername()
        );
    }

    // --- UserDetailsService method ---

    /**
     * Загружает пользователя для Spring Security по username.
     * Преобразует роли в GrantedAuthority.
     */
    @Override
    public UserDetails loadUserByUsername(String username) throws
UsernameNotFoundException {
        User user = userRepository.findByUsername(username)
            .orElseThrow(() ->
                new UsernameNotFoundException("User not
found: " + username));

        List<GrantedAuthority> authorities =
user.getRoles().stream()
            .map(SimpleGrantedAuthority::new)
            .collect(Collectors.toList());

        return
org.springframework.security.core.userdetails.User.builder()
            .username(user.getUsername())
            .password(user.getPassword())
            .authorities(authorities)
            .build();
    }
}

```

Файл Fb2ToHtmlConverter.java

					ПГУ 1.090301.29.001 ПЗ	Лист
						110
Изм.	Лист	№ докум.	Подпись	Дата		

```

package com.example.booklover.util;

import org.springframework.stereotype.Component;
import org.w3c.dom.*;
import javax.xml.parsers.*;
import java.io.InputStream;

@Component
public class Fb2ToHtmlConverter {

    /**
     * Читает FB2 из InputStream и возвращает простой HTML.
     */
    public String convert(InputStream fb2Input) {
        try {
            DocumentBuilderFactory factory =
DocumentBuilderFactory.newInstance();
            factory.setNamespaceAware(false);
            DocumentBuilder builder =
factory.newDocumentBuilder();
            Document doc = builder.parse(fb2Input);

            StringBuilder html = new StringBuilder();
            html.append("<!DOCTYPE html><html><head><meta
charset=\"UTF-8\"></head><body>");

            NodeList sections =
doc.getElementsByTagName("section");
            for (int i = 0; i < sections.getLength(); i++) {
                Element section = (Element) sections.item(i);

                // заголовок раздела
                NodeList titles =
section.getElementsByTagName("title");
                if (titles.getLength() > 0) {
                    String header = extractText(titles.item(0));

                    html.append("<h2>").append(escape(header)).append("</h2>");
                }

                // параграфы
                NodeList paras =
section.getElementsByTagName("p");
                for (int j = 0; j < paras.getLength(); j++) {
                    String text =
paras.item(j).getTextContent();

                    html.append("<p>").append(escape(text)).append("</p>");
                }

                html.append("</body></html>");
                return html.toString();
            } catch (Exception ex) {
                throw new RuntimeException("Failed to convert FB2 to
HTML", ex);
            }
        }
    }
}

```

```

    }
}

private String extractText(Node node) {
    StringBuilder sb = new StringBuilder();
    NodeList children = node.getChildNodes();
    for (int k = 0; k < children.getLength(); k++) {
        Node c = children.item(k);
        if (c.getNodeType() == Node.TEXT_NODE) {
            sb.append(c.getNodeValue());
        } else if (c.getNodeType() == Node.ELEMENT_NODE) {
            sb.append(extractText(c));
        }
    }
    return sb.toString();
}

private String escape(String text) {
    return text
        .replace("&", "&amp;")
        .replace("<", "&lt;")
        .replace(">", "&gt;");
}
}

```

Файл AuthController.java

```

package com.example.booklover.controller;

import com.example.booklover.model.entity.User;
import com.example.booklover.model.entity.dto.RegisterRequest;
import com.example.booklover.service.UserService;
import jakarta.servlet.http.HttpSession;
import org.springframework.beans.factory.annotation.Autowired;
import
org.springframework.security.authentication.UsernamePasswordAuthentic
ationToken;
import org.springframework.security.core.Authentication;
import
org.springframework.security.core.authority.SimpleGrantedAuthority;
import
org.springframework.security.core.context.SecurityContext;
import
org.springframework.security.core.context.SecurityContextHolder;
import
org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.*;

import java.util.Collections;

@Controller
@RequestMapping("/auth")
public class AuthController {

```

```

private final UserService userService;
private final PasswordEncoder passwordEncoder;

@Autowired
public AuthController(UserService userService,
                      PasswordEncoder passwordEncoder) {
    this.userService = userService;
    this.passwordEncoder = passwordEncoder;
}

/** GET /auth/login – форма входа */
@GetMapping("/login")
public String loginPage(@RequestParam(value = "error",
required = false) String error,
                      @RequestParam(value = "registered",
required = false) String registered,
                      Model model) {
    model.addAttribute("error", error != null);
    model.addAttribute("registered", registered != null);
    return "auth/login";
}

/** GET /auth/register – форма регистрации */
@GetMapping("/register")
public String registerPage(@RequestParam(value = "error",
required = false) String error,
                      Model model) {
    model.addAttribute("error", error != null);
    return "auth/register";
}

/** POST /auth/register – регистрация через HTML-форму */
@PostMapping("/register")
public String registerForm(RegisterRequest form) {
    if (userService.existsByUsername(form.getUsername())) {
        return "redirect:/auth/register?error";
    }
    User user = new User();
    user.setFirstName(form.getFirstName());
    user.setLastName(form.getLastName());
    user.setUsername(form.getUsername());

    user.setPassword(passwordEncoder.encode(form.getPassword()));
    user.setRoles(Collections.singleton("ROLE_USER"));
    userService.save(user);
    return "redirect:/auth/login?registered";
}

/** POST /auth/guest – войти как гость (ROLE_GUEST) */
@PostMapping("/guest")
public String continueAsGuest(
    HttpSession session
) {
    Authentication guestAuth = new
UsernamePasswordAuthenticationToken(

```

```

        "guest",
        null,
        Collections.singletonList(new
SimpleGrantedAuthority("ROLE_GUEST"))
    );
    // Устанавливаем Authentication в SecurityContext
    SecurityContext securityContext =
SecurityContextHolder.createEmptyContext();
    securityContext.setAuthentication(guestAuth);
    SecurityContextHolder.setContext(securityContext);

    // Сохраняем SecurityContext в сессии
    session.setAttribute("SPRING_SECURITY_CONTEXT",
securityContext);

SecurityContextHolder.getContext().setAuthentication(guestAuth);
    return "redirect:/";
}
}

```

Файл BookController.java

```

package com.example.booklover.controller;

import com.example.booklover.model.entity.dto.BookDto;
import com.example.booklover.model.entity.dto.CreateBookRequest;
import com.example.booklover.model.entity.dto.UpdateBookRequest;
import com.example.booklover.service.BookService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.core.io.Resource;
import org.springframework.http.HttpHeaders;
import org.springframework.http.ResponseEntity;
import org.springframework.http.HttpStatus;
import
org.springframework.security.core.annotation.AuthenticationPrincipal;
import
org.springframework.security.core.userdetails.UserDetails;
import org.springframework.web.bind.annotation.*;
import org.springframework.web.multipart.MultipartFile;

import java.util.List;

@RestController
@RequestMapping("/api/books")
public class BookController {

    private final BookService bookService;

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

    /**

```

```

    * GET /api/books
    * Поиск и список книг.
    */
@GetMapping
public List<BookDto> getBooks(
    @RequestParam(required = false) String title,
    @RequestParam(required = false) String author,
    @RequestParam(required = false) Boolean physical
) {
    return bookService.searchBooks(title, author, physical);
}

/**
 * GET /api/books/latest
 * Лента последних добавленных книг.
 */
@GetMapping("/latest")
public List<BookDto> getLatest(
    @RequestParam(defaultValue = "10") int count
) {
    return bookService.findLatest(count);
}

/**
 * GET /api/books/{id}/download
 * Скачивание файла книги (электронной версии).
 */
@GetMapping("/{id}/download")
public ResponseEntity<Resource> downloadBook(@PathVariable
Long id) {
    Resource file = bookService.loadBookFile(id);
    return ResponseEntity.ok()
        .header(HttpHeaders.CONTENT_DISPOSITION,
            "attachment; filename=\""
            + file.getFilename() + "\"")
        .body(file);
}

/**
 * GET /api/books/{id}
 * Получение подробностей по одной книге.
 */
@GetMapping("/{id}")
public BookDto getBookById(@PathVariable Long id) {
    return bookService.getBookById(id);
}

/**
 * POST /api/books
 * Добавление новой книги (multipart/form-data).
 */
@PostMapping(consumes = "multipart/form-data")
@ResponseStatus(HttpStatus.CREATED)
public BookDto addBook(
    @AuthenticationPrincipal UserDetails user,
    @RequestPart("metadata") CreateBookRequest request,

```

```

        @RequestParam(value = "file", required = false)
        MultipartFile file
    ) {
        return bookService.createBook(user.getUsername(),
request, file);
    }

    /**
     * PUT /api/books/{id}
     * Обновление данных книги.
     */
    @PutMapping(value =("/{id}", consumes = "multipart/form-
data")
    public BookDto updateBook(
        @PathVariable Long id,
        @AuthenticationPrincipal UserDetails user,
        @RequestParam("metadata") UpdateBookRequest request,
        @RequestParam(value = "file", required = false)
        MultipartFile file
    ) {
        return bookService.updateBook(id, user.getUsername(),
request, file);
    }

    /**
     * DELETE /api/books/{id}
     * Удаление книги (только владелец).
     */
    @DeleteMapping("/{id}")
    @ResponseStatus(HttpStatus.NO_CONTENT)
    public void deleteBook(
        @PathVariable Long id,
        @AuthenticationPrincipal UserDetails user
    ) {
        bookService.deleteBook(id, user.getUsername());
    }
}

```

Файл ChatController.java

```

package com.example.booklover.controller;

import com.example.booklover.model.entity.dto.MessageDto;
import com.example.booklover.service.ChatService;
import org.springframework.beans.factory.annotation.Autowired;
import
org.springframework.messaging.handler.annotation.MessageMapping;
import org.springframework.messaging.handler.annotation.SendTo;
import org.springframework.stereotype.Controller;

import java.time.LocalDateTime;

@Controller
public class ChatController {

```

```

private final ChatService chatService;

@Autowired
public ChatController(ChatService chatService) {
    this.chatService = chatService;
}

/**
 * Получает сообщение от клиента по адресу /app/chat
 * Сохраняет через ChatService, устанавливает метку времени
 * и рассылает всем подписчикам /topic/messages
 */
@RequestMapping("/chat")
@SendTo("/topic/messages")
public MessageDto sendMessage(MessageDto message) {
    message.setTimestamp(LocalDateTime.now());
    chatService.saveMessage(message);
    return message;
}
}

```

Файл ChatRestController.java

```

package com.example.booklover.controller;

import com.example.booklover.model.entity.dto.MessageDto;
import com.example.booklover.service.ChatService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController
@RequestMapping("/api/chat")
public class ChatRestController {

    private final ChatService chatService;

    @Autowired
    public ChatRestController(ChatService chatService) {
        this.chatService = chatService;
    }

    /**
     * GET /api/chat/messages
     * Возвращает историю последних сообщений общего чата.
     */
    @GetMapping("/messages")
    public List<MessageDto> getChatHistory() {
        return chatService.getRecentMessages();
    }
}

```

Файл CollectionController.java

```

package com.example.booklover.controller;

import com.example.booklover.model.entity.dto.CollectionDto;
import
com.example.booklover.model.entity.dto.CreateCollectionRequest;
import com.example.booklover.service.CollectionService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import
org.springframework.security.core.annotation.AuthenticationPrincipal;
import
org.springframework.security.core.userdetails.UserDetails;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController
@RequestMapping("/api/collections")
public class CollectionController {

    private final CollectionService collectionService;

    @Autowired
    public CollectionController(CollectionService
collectionService) {
        this.collectionService = collectionService;
    }

    /**
     * GET /api/collections
     * Возвращает все подборки текущего пользователя.
     */
    @GetMapping
    public List<CollectionDto> getUserCollections(
        @AuthenticationPrincipal UserDetails user
    ) {
        return
collectionService.findAllByUsername(user.getUsername());
    }

    /**
     * GET /api/collections/{id}
     * Возвращает одну подборку по ID (только если она принадлежит
пользователю).
     */
    @GetMapping("/{id}")
    public CollectionDto getCollectionById(
        @PathVariable Long id,
        @AuthenticationPrincipal UserDetails user
    ) {
        return collectionService.findByIdAndUsername(id,
user.getUsername());
    }

    /**
     * GET /api/collections/all

```

```

        * Возвращает все подборки всех пользователей.
        */
@GetMapping("/all")
public List<CollectionDto> getAllCollections() {
    return collectionService.findAll();
}

/**
 * POST /api/collections
 * Создаёт новую подборку: имя + список ID книг.
 * Тело запроса:
 * {
 *     "name": "Название подборки",
 *     "bookIds": [1, 2, 5, 10]
 * }
 */
@PostMapping
@ResponseStatus(HttpStatus.CREATED)
public CollectionDto createCollection(
    @AuthenticationPrincipal UserDetails user,
    @RequestBody CreateCollectionRequest request
) {
    return collectionService.create(
        user.getUsername(),
        request.getName(),
        request.getBookIds()
    );
}

// при необходимости можно добавить методы update/delete
}

```

Файл CommentController.java

```

package com.example.booklover.controller;

import com.example.booklover.model.entity.dto.CommentDto;
import
com.example.booklover.model.entity.dto.CreateCommentRequest;
import com.example.booklover.service.CommentService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import
org.springframework.security.core.annotation.AuthenticationPrincipal;
import
org.springframework.security.core.userdetails.UserDetails;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController
@RequestMapping("/api/books/{bookId}/comments")
public class CommentController {

    private final CommentService commentService;

```

```

    @Autowired
    public CommentController(CommentService commentService) {
        this.commentService = commentService;
    }

    /**
     * GET /api/books/{bookId}/comments
     * Возвращает все комментарии для указанной книги.
     */
    @GetMapping
    public List<CommentDto> getComments(@PathVariable Long
bookId) {
        return commentService.findByBook(bookId);
    }

    /**
     * POST /api/books/{bookId}/comments
     * Добавляет новый комментарий к указанной книге.
     * Требуется аутентификации.
     */
    @PostMapping
    @ResponseStatus(HttpStatus.CREATED)
    public CommentDto addComment(
        @PathVariable Long bookId,
        @AuthenticationPrincipal UserDetails user,
        @RequestBody CreateCommentRequest request
    ) {
        return commentService.create(bookId, user.getUsername(),
request.getText());
    }
}

```

Файл PageController.java

```

package com.example.booklover.controller;

import com.example.booklover.model.entity.dto.BookDto;
import com.example.booklover.service.BookService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.MediaType;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.*;

import java.io.IOException;

@Controller
public class PageController {

    private final BookService bookService;

    @Autowired
    public PageController(BookService bookService) {
        this.bookService = bookService;
    }
}

```

```

    }

    @GetMapping("/")
    public String home() {
        return "index";
    }

    @GetMapping("/books")
    public String allBooks() {
        return "books";
    }

    @GetMapping("/books/{id}")
    public String oneBook(@PathVariable Long id, Model model) {
        // подгружаем полные данные книги и кладём в модель
        BookDto book = bookService.getBookById(id);
        model.addAttribute("book", book);
        return "book";
    }

    @GetMapping(value = "/books/{id}/reading", produces =
MediaType.TEXT_HTML_VALUE)
    @ResponseBody
    public String readingBook(@PathVariable Long id, Model model)
throws IOException {
        // подгружаем полные данные книги и кладём в модель
        return bookService.getHtmlBook(id);
    }

    @GetMapping("/account")
    public String account() {
        return "account";
    }
}

```

Файл UserController.java

```

package com.example.booklover.controller;

import com.example.booklover.model.entity.dto.UpdateUserRequest;
import com.example.booklover.model.entity.dto.UserDto;
import com.example.booklover.service.UserService;
import org.springframework.beans.factory.annotation.Autowired;
import
org.springframework.security.core.annotation.AuthenticationPrincipal;
import
org.springframework.security.core.userdetails.UserDetails;
import org.springframework.web.bind.annotation.*;

@RestController
@RequestMapping("/api/users")
public class UserController {

    private final UserService userService;

```

```

@Autowired
public UserController(UserService userService) {
    this.userService = userService;
}

/**
 * GET /api/users/me
 * Возвращает данные текущего пользователя.
 */
@GetMapping("/me")
public UserDto getCurrentUser(@AuthenticationPrincipal
UserDetails user) {
    return userService.getByUsername(user.getUsername());
}

/**
 * PUT /api/users/me
 * Обновление профиля текущего пользователя.
 * Тело запроса:
 * {
 *     "firstName": "НовоеИмя",
 *     "lastName": "НоваяФамилия",
 *     "password": "НовыйПароль" // необязательно
 * }
 */
@PostMapping("/me")
public UserDto updateCurrentUser(
    @AuthenticationPrincipal UserDetails user,
    @RequestBody UpdateUserRequest request
) {
    return userService.updateProfile(user.getUsername(),
request);
}

/**
 * GET /api/users/{username}
 * Возвращает публичные данные пользователя по логину.
 */
@GetMapping("/{username}")
public UserDto getUserByUsername(@PathVariable String
username) {
    return userService.getByUsername(username);
}
}

```

Файл SecurityConfig.java

```

package com.example.booklover.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.http.HttpMethod;
import
org.springframework.security.authentication.AuthenticationManager;
import

```

```

org.springframework.security.authentication.dao.DaoAuthenticationProv
ider;
    import
org.springframework.security.config.annotation.authentication.configu
ration.AuthenticationConfiguration;
    import
org.springframework.security.config.annotation.web.builders.HttpSecur
ity;
    import
org.springframework.security.config.annotation.web.configuration.Enab
leWebSecurity;
    import
org.springframework.security.core.userdetails.UserDetailsService;
    import
org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
    import
org.springframework.security.crypto.password.PasswordEncoder;
    import org.springframework.security.web.SecurityFilterChain;

@Configuration
@EnableWebSecurity
public class SecurityConfig {

    private final UserDetailsService userDetailsService;

    public SecurityConfig(UserDetailsService userDetailsService)
{
        this.userDetailsService = userDetailsService;
    }

    /**
     * BCrypt-энкодер. static чтобы его создание не требовало
инстанса SecurityConfig,
     * и тем самым разрывает циклическую зависимость.
     */
    @Bean
    public static PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }

    /** Провайдер аутентификации на базе DAO (UserDetailsService
+ PasswordEncoder) */
    @Bean
    public DaoAuthenticationProvider authenticationProvider() {
        var provider = new DaoAuthenticationProvider();
        provider.setUserDetailsService(userDetailsService);
        provider.setPasswordEncoder(passwordEncoder());
        return provider;
    }

    /** Экспонируем AuthenticationManager (если нужно вручную) */
    @Bean
    public AuthenticationManager authenticationManager(
        AuthenticationConfiguration cfg
    ) throws Exception {
        return cfg.getAuthenticationManager();
    }

```

```

    }

    /** Основные правила безопасности */
    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity
http) throws Exception {
        http
            .csrf(csrf -> csrf.disable())

            .authenticationProvider(authenticationProvider())
                .authorizeHttpRequests(auth -> auth
                    // публичные ресурсы и страницы
входа/регистрации
                    .requestMatchers("/", "/css/**",
"/js/**", "/auth/**").permitAll()
                    // WebSocket/SockJS
                    .requestMatchers("/ws-chat/**",
"/app/**", "/topic/**").permitAll()
                    // скачивание книги и отправка в чат –
только для ROLE_USER
                    .requestMatchers(HttpMethod.GET,
"/api/books/{id}/download").hasRole("USER")
                    .requestMatchers(HttpMethod.POST,
"/app/chat/**").hasRole("USER")
                    // REST API и личный кабинет – только для
ROLE_USER
                    .requestMatchers("/api/**",
"/account/**").hasAnyRole("USER", "GUEST")
                    .anyRequest().authenticated()
                )
            .formLogin(form -> form
                .loginPage("/auth/login")
                .loginProcessingUrl("/auth/login")
                .defaultSuccessUrl("/", true)
                .permitAll()
            )
            .logout(logout -> logout
                .logoutUrl("/auth/logout")
                .logoutSuccessUrl("/")
                .permitAll()
            );
        return http.build();
    }
}

```

Файл WebSocketConfig.java

```

package com.example.booklover.config;

import org.springframework.context.annotation.Configuration;
import
org.springframework.messaging.simp.config.MessageBrokerRegistry;
import
org.springframework.web.socket.config.annotation.EnableWebSocketMessa
geBroker;

```

					ПГУ 1.090301.29.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		124

```

import
org.springframework.web.socket.config.annotation.StompEndpointRegistr
y;

import
org.springframework.web.socket.config.annotation.WebSocketMessageBrok
erConfigurer;

@Configuration
@EnableWebSocketMessageBroker
public class WebSocketConfig implements
WebSocketMessageBrokerConfigurer {

    /**
     * Регистрирует точку подключения SockJS для WebSocket.
     * Клиенты будут подключаться к /ws-chat.
     */
    @Override
    public void registerStompEndpoints(StompEndpointRegistry
registry) {
        registry
            .addEndpoint("/ws-chat")
            .setAllowedOriginPatterns("*") // можно уточнить
домены при необходимости
            .withSockJS();
    }

    /**
     * Настраивает брокер сообщений:
     * - Клиенты могут подписываться на топики /topic
     * - Сообщения от @MessageMapping ожидаются с префиксом /app
     */
    @Override
    public void configureMessageBroker(MessageBrokerRegistry
registry) {
        registry.enableSimpleBroker("/topic");
        registry.setApplicationDestinationPrefixes("/app");
    }
}

```

Файл AccessDeniedException.java

```

package com.example.booklover.exception;

public class AccessDeniedException extends RuntimeException {
    public AccessDeniedException(String message) {
        super(message);
    }
}

```

Файл ErrorResponse.java

```

package com.example.booklover.exception;

import com.fasterxml.jackson.annotation.JsonFormat;
import java.time.LocalDateTime;

```

					ПГУ 1.090301.29.001 ПЗ	Лист
						125
Изм.	Лист	№ докум.	Подпись	Дата		

```

public class ErrorResponse {

    @JsonFormat(shape = JsonFormat.Shape.STRING, pattern = "yyyy-MM-dd'T'HH:mm:ss")
    private final LocalDateTime timestamp = LocalDateTime.now();

    private final int status;
    private final String error;
    private final String message;
    private final String path;

    public ErrorResponse(int status, String error, String message, String path) {
        this.status = status;
        this.error = error;
        this.message = message;
        this.path = path;
    }

    public LocalDateTime getTimestamp() {
        return timestamp;
    }

    public int getStatus() {
        return status;
    }

    public String getError() {
        return error;
    }

    public String getMessage() {
        return message;
    }

    public String getPath() {
        return path;
    }
}

```

Файл GlobalExceptionHandler.java

```

package com.example.booklover.exception;

import jakarta.servlet.http.HttpServletRequest;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.ExceptionHandler;
import org.springframework.web.bind.annotation.RestControllerAdvice;

@RestControllerAdvice
public class GlobalExceptionHandler {

```

					ПГУ 1.090301.29.001 ПЗ	Лист
						126
Изм.	Лист	№ докум.	Подпись	Дата		

```

        @ExceptionHandler (NotFoundException.class)
        public                               ResponseEntity<ErrorResponse>
handleNotFound(NotFoundException ex,

HttpServletRequest request) {
    ErrorResponse body = new ErrorResponse(
        HttpStatus.NOT_FOUND.value(),
        HttpStatus.NOT_FOUND.getReasonPhrase(),
        ex.getMessage(),
        request.getRequestURI()
    );
    return new ResponseEntity<>(body, HttpStatus.NOT_FOUND);
}

        @ExceptionHandler (AccessDeniedException.class)
        public                               ResponseEntity<ErrorResponse>
handleAccessDenied(AccessDeniedException ex,

HttpServletRequest request) {
    ErrorResponse body = new ErrorResponse(
        HttpStatus.FORBIDDEN.value(),
        HttpStatus.FORBIDDEN.getReasonPhrase(),
        ex.getMessage(),
        request.getRequestURI()
    );
    return new ResponseEntity<>(body, HttpStatus.FORBIDDEN);
}

        @ExceptionHandler (Exception.class)
        public ResponseEntity<ErrorResponse> handleAll (Exception ex,

HttpServletRequest request) {
    ErrorResponse body = new ErrorResponse(
        HttpStatus.INTERNAL_SERVER_ERROR.value(),

HttpStatus.INTERNAL_SERVER_ERROR.getReasonPhrase(),
        ex.getMessage(),
        request.getRequestURI()
    );
    return                               new                               ResponseEntity<>(body,
HttpStatus.INTERNAL_SERVER_ERROR);
}
}

```

Файл NotFoundException.java

```

package com.example.booklover.exception;

public class NotFoundException extends RuntimeException {
    public NotFoundException(String message) {
        super(message);
    }
}

```

Файл Application.java

					ПГУ 1.090301.29.001 ПЗ	Лист
						127
Изм.	Лист	№ докум.	Подпись	Дата		

```

package com.example.booklover;

import org.springframework.boot.SpringApplication;
import
org.springframework.boot.autoconfigure.SpringBootApplication;

/**
 * Main class for starting the BookLover application.
 */
@SpringBootApplication
public class Application {

    public static void main(String[] args) {
        // Launch the Spring Boot application.
        SpringApplication.run(Application.class, args);
    }
}

```

Файл login.html

```

<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org"
      lang="ru">
<head>
    <meta charset="UTF-8"/>
    <title>Вход – BookLover</title>
    <link
href="https://cdn.jsdelivrivr.net/npm/bootstrap@5.3.2/dist/css/bootstrap
.min.css"
      rel="stylesheet"/>
</head>
<body class="d-flex flex-column min-vh-100">

<!-- шапка -->
<div th:replace="fragments/header :: header"></div>

<main class="flex-fill">
    <div class="container mt-5" style="max-width: 400px;">

        <h2 class="mb-4">Вход</h2>

        <!-- если был ?error -->
        <div th:if="${error}" class="alert alert-danger">
            Неверный логин или пароль
        </div>
        <!-- если был ?registered -->
        <div th:if="${registered}" class="alert alert-success">
            Успешная регистрация. Теперь войдите.
        </div>

        <form th:action="@{/auth/login}" method="post">
            <div class="mb-3">
                <label for="username" class="form-label">Логин</label>
                <input id="username" name="username" type="text"
                    class="form-control" placeholder="Введите логин"

```

```

required/>
    </div>
    <div class="mb-3">
        <label for="password" class="form-label">Пароль</label>
        <input id="password" name="password" type="password"
            class="form-control" placeholder="Введите пароль"
required/>
    </div>
    <button type="submit" class="btn btn-primary w-
100">Войти</button>
</form>

    <form th:action="@{/auth/guest}" method="post" class="mt-3">
        <button type="submit" class="btn btn-secondary w-100">
            Продолжить как гость
        </button>
    </form>

    <p class="text-center mt-3">
        Нет аккаунта?
        <a th:href="@{/auth/register}">Зарегистрироваться</a>
    </p>
</div>
</main>

<!-- подвал -->
<div th:replace="fragments/footer :: footer"></div>

<script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/js/bootstrap.b
undle.min.js"></script>
</body>
</html>

```

Файл register.html

```

<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org"
    lang="ru">
<head>
    <meta charset="UTF-8"/>
    <title>Регистрация — BookLover</title>
    <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/css/bootstrap
.min.css"
        rel="stylesheet"/>
</head>
<body class="d-flex flex-column min-vh-100">

    <!-- шапка -->
    <div th:replace="fragments/header :: header"></div>

    <main class="flex-fill">
        <div class="container mt-5" style="max-width: 400px;">

```

					ПГУ 1.090301.29.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		129

```

<h2 class="mb-4">Регистрация</h2>

<!-- если был ?error -->
<div th:if="{error}" class="alert alert-danger">
    Пользователь с таким логином уже существует!
</div>

<form th:action="@{/auth/register}" method="post">
    <div class="mb-3">
        <label          for="firstName"          class="form-
label">Имя</label>
        <input          id="firstName"          name="firstName"
type="text"
                                class="form-control" placeholder="Введите
имя" required/>
    </div>
    <div class="mb-3">
        <label          for="lastName"          class="form-
label">Фамилия</label>
        <input id="lastName" name="lastName" type="text"
                                class="form-control" placeholder="Введите
фамилию" required/>
    </div>
    <div class="mb-3">
        <label          for="username"          class="form-
label">Логин</label>
        <input id="username" name="username" type="text"
                                class="form-control"
placeholder="Придумайте логин" required/>
    </div>
    <div class="mb-3">
        <label          for="password"          class="form-
label">Пароль</label>
        <input          id="password"          name="password"
type="password"
                                class="form-control"
placeholder="Придумайте пароль" required/>
    </div>
    <button type="submit" class="btn btn-success w-100">
        Зарегистрироваться
    </button>
</form>

<p class="text-center mt-3">
    Уже зарегистрированы?
    <a th:href="@{/auth/login}">Войти</a>
</p>
</div>
</main>

<!-- подвал -->
<div th:replace="fragments/footer :: footer"></div>

<script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/js/bootstrap.b
undle.min.js"></script>

```

```
</body>
</html>
```

Файл account.html

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org"
      xmlns:sec="http://www.thymeleaf.org/thymeleaf-extras-
springsecurity6"
      lang="ru">
  <head>
    <meta charset="UTF-8"/>

    <meta                                     name="user"
th:content="${#authentication.getName()}" />
    <title>Личный кабинет – BookLover</title>
    <!-- Bootstrap CSS из CDN -->
    <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/css/bootstrap
.min.css"
      rel="stylesheet"/>
    <link th:href="@{/css/styles.css}" rel="stylesheet"/>
  </head>
  <body class="d-flex flex-column min-vh-100">
    <!-- статичный фон с размытием -->
    <div class="page-bg "></div>
    <!-- header -->
    <div th:replace="fragments/header :: header"></div>

    <main class="flex-fill">
      <div class="container mt-4 content-overlay">

        <!-- Профиль -->
        <h2>Мой профиль</h2>
        <form id="profileForm" class="row g-3 mb-5">
          <div class="col-md-6">
            <label                for="firstName"                class="form-
label">Имя</label>
            <input                id="firstName"                name="firstName"
class="form-control" required/>
          </div>
          <div class="col-md-6">
            <label                for="lastName"                class="form-
label">Фамилия</label>
            <input                id="lastName"                name="lastName"
class="form-control" required/>
          </div>
          <div class="col-12">
            <label for="password" class="form-label">Новый
пароль</label>
            <input                id="password"                name="password"
type="password" class="form-control"/>
          </div>
          <div class="col-12">
            <button                type="submit"                class="btn                btn-
```

```

primary">Сохранить профиль</button>
    </div>
</form>

<!-- Мои книги -->
<h3>Мои книги</h3>
<div id="myBooks" class="row row-cols-1 row-cols-md-3 g-
4 mb-4">
    <!-- JS подгрузит карточки -->
</div>
<button class="btn btn-success mb-5" data-bs-
toggle="modal" data-bs-target="#addBookModal">
    Добавить книгу
</button>

<!-- Мои подборки -->
<h3>Мои подборки</h3>
<div id="myCollections" class="row row-cols-1 row-cols-
md-3 g-4 mb-4">
    <!-- JS подгрузит карточки -->
</div>
<button class="btn btn-success mb-5" data-bs-
toggle="modal" data-bs-target="#createCollectionModal">
    Создать подборку
</button>

</div>
</main>

<!-- Модалка: Добавить книгу -->
<div class="modal fade" id="addBookModal" tabindex="-1" aria-
hidden="true">
    <div class="modal-dialog">
        <div class="modal-content">
            <div class="modal-header">
                <h5 class="modal-title">Добавить книгу</h5>
                <button type="button" class="btn-close" data-bs-
dismiss="modal"></button>
            </div>
            <div class="modal-body">
                <form id="addBookForm" enctype="multipart/form-
data" class="row g-3">
                    <div class="col-12">
                        <label class="form-
label">Название</label>
                        <input name="title" class="form-control"
required/>
                    </div>
                    <div class="col-12">
                        <label class="form-label">Автор</label>
                        <input name="author" class="form-
control" required/>
                    </div>
                    <div class="col-6">
                        <label class="form-label">Год</label>
                        <input name="year" type="number"

```

```

class="form-control" required/>
        </div>
        <div class="col-6">
            <label                class="form-label">Файл
(FB2)</label>
            <input                name="file"                type="file"
accept=".fb2" class="form-control"/>
        </div>
        <div class="col-12 form-check">
            <input                id="physicalCheck"
name="physical" type="checkbox" class="form-check-input"/>
            <label for="physicalCheck" class="form-
check-label">Книга офлайн</label>
        </div>
        <div class="col-12">
            <button type="submit" class="btn btn-
primary w-100">Добавить</button>
        </div>
    </form>
</div>
</div>
</div>
</div>
</div>

<!-- Модалка: Создать подборку -->
<div class="modal fade" id="createCollectionModal" tabindex="-1"
aria-hidden="true">
    <div class="modal-dialog">
        <div class="modal-content">
            <div class="modal-header">
                <h5 class="modal-title">Создать подборку</h5>
                <button type="button" class="btn-close" data-bs-
dismiss="modal"></button>
            </div>
            <div class="modal-body">
                <form id="createCollectionForm" class="row g-3">
                    <div class="col-12">
                        <label                class="form-label">Название
подборки</label>
                        <input id="collectionName" class="form-
control" required/>
                    </div>
                    <div id="collectionBooksList" class="col-12"
style="max-height:250px; overflow-y:auto;">
                        <!-- JS подгрузит чекбоксы -->
                    </div>
                    <div class="col-12">
                        <button type="submit" class="btn btn-
primary w-100">Создать</button>
                    </div>
                </form>
            </div>
        </div>
    </div>
</div>
</div>

```

```

<!-- footer -->
<div th:replace="fragments/footer :: footer"></div>

<!-- Скрипты -->
<script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/js/bootstrap.b
undle.min.js"></script>
<script th:src="@{/js/account.js}"></script>
</body>
</html>

```

Файл book.html

```

<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org"
lang="ru">
<head>
<meta charset="UTF-8"/>
<meta name="book-id" th:content="${book.id}"/>
<title>Детали книги – BookLover</title>
<link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/css/bootstrap
.min.css"
rel="stylesheet"/>
<link th:href="@{/css/styles.css}" rel="stylesheet"/>
</head>
<body class="d-flex flex-column min-vh-100"
th:attr="data-book-id=${book.id}">
<!-- статичный фон с размытием -->
<div class="page-bg "></div>
<!-- шапка -->
<div th:replace="fragments/header :: header"></div>

<main class="flex-fill">
<div class="container mt-4 content-overlay">

<!-- Заголовок и метаданные книги -->
<h2 id="bookTitle" class="mb-1"></h2>
<p id="bookMeta" class="text-muted mb-3"></p>

<!-- Действия: скачать или "офлайн" -->
<div id="bookActions" class="mb-4"></div>

<!-- Комментарии -->
<h3>Комментарии</h3>
<ul id="commentList" class="list-group mb-3" style="max-
height:300px; overflow-y:auto;"></ul>

<div class="mb-3">
<textarea id="commentText" class="form-control"
rows="3" placeholder="Ваш комментарий..."></textarea>
</div>
<button id="addCommentBtn" class="btn btn-primary mb-
5">Добавить комментарий</button>

```

```

        </div>
</main>

<!-- подвал -->
<div th:replace="fragments/footer :: footer"></div>

<script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/js/bootstrap.b
undle.min.js"></script>
<script th:src="@{/js/book.js}"></script>
</body>
</html>

```

Файл books.html

```

<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org" lang="ru">
<head>
    <meta charset="UTF-8"/>
    <title>Книги – BookLover</title>
    <!-- Bootstrap CSS -->
    <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/css/bootstrap
.min.css"
        rel="stylesheet"/>

    <link th:href="@{/css/styles.css}" rel="stylesheet"/>
</head>
<body class="d-flex flex-column min-vh-100">
<!-- статичный фон с размытием -->
<div class="page-bg "></div>

<!-- шапка -->
<div th:replace="fragments/header :: header"></div>

<main class="flex-fill">
    <div class="container mt-4 content-overlay">

        <h2 class="mb-4">Список книг</h2>

        <!-- Форма поиска -->
        <form id="searchForm" class="row g-3 mb-4">
            <div class="col-md-4">
                <input id="titleInput" class="form-control"
placeholder="По названию"/>
            </div>
            <div class="col-md-4">
                <input id="authorInput" class="form-control"
placeholder="По автору"/>
            </div>
            <div class="col-md-3">
                <select id="physicalInput" class="form-select">
                    <option value="">Все</option>
                    <option value="false">Электронные</option>
                    <option value="true">Вещественные</option>
                </select>
            </div>
        </form>
    </div>
</main>

```

					ПГУ 1.090301.29.001 ПЗ	Лист
						135
Изм.	Лист	№ докум.	Подпись	Дата		

```

        </select>
    </div>
    <div class="col-md-1">
        <button type="submit" class="btn btn-primary w-
100">Поиск</button>
    </div>
</form>

<!-- Карточки книг -->
<div id="booksList" class="row row-cols-1 row-cols-md-3
g-4">
    <!-- Заполняется books.js -->
</div>
</div>
</main>

<!-- подвал -->
<div th:replace="fragments/footer :: footer"></div>

<!-- скрипты -->
<script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/js/bootstrap.b
undle.min.js"></script>
<script th:src="@{/js/books.js}"></script>
</body>
</html>

```

Файл index.html

```

<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org"
      xmlns:sec="http://www.thymeleaf.org/thymeleaf-extras-
springsecurity6"
      lang="ru">
    <head>
        <meta charset="UTF-8"/>
        <meta
                                name="user"
th:content="${#authentication.getName()}" />
        <title>Главная – Книговорот</title>
        <!-- Bootstrap CSS -->
        <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/css/bootstrap
.min.css"
                                rel="stylesheet"/>
        <!-- Ваши стили: blur-фон и оверлей -->
        <link th:href="@{/css/styles.css}" rel="stylesheet"/>
    </head>
    <body class="d-flex flex-column min-vh-100">

        <!-- статичный фон с размытием -->
        <div class="page-bg"></div>

        <!-- header -->
        <div th:replace="fragments/header :: header"></div>

```

```

<!-- Welcome modal для анонимных -->
<div sec:authorize="isAnonymous()">
    <div class="modal fade" id="welcomeModal" tabindex="-1" aria-
labelledby="welcomeModalLabel" aria-hidden="true">
        <div class="modal-dialog modal-dialog-centered">
            <div class="modal-content">
                <div class="modal-header">
                    <h5 class="modal-title"
id="welcomeModalLabel">Добро пожаловать</h5>
                </div>
                <div class="modal-body">
                    <ul class="nav nav-tabs mb-3"
id="welcomeTabs" role="tablist">
                        <li class="nav-item"
role="presentation">
                            <button class="nav-link active"
                                data-bs-toggle="tab"
                                data-bs-target="#login-pane"
                                type="button">Вход</button>
                        </li>
                        <li class="nav-item"
role="presentation">
                            <button class="nav-link"
                                data-bs-toggle="tab"
                                data-bs-target="#register-
pane"
                                type="button">Регистрация</button>
                        </li>
                    </ul>
                    <div class="tab-content">
                        <div class="tab-pane fade show active"
id="login-pane" role="tabpanel">
                            <form th:action="@{/auth/login}"
method="post">
                                <input name="username"
class="form-control mb-2" placeholder="Логин" required/>
                                <input name="password"
type="password" class="form-control mb-2" placeholder="Пароль"
required/>
                                <button class="btn btn-primary
w-100">Войти</button>
                            </form>
                        </div>
                        <div class="tab-pane fade" id="register-
pane" role="tabpanel">
                            <form th:action="@{/auth/register}"
method="post">
                                <input name="firstName"
class="form-control mb-2" placeholder="Имя" required/>
                                <input name="lastName"
class="form-control mb-2" placeholder="Фамилия" required/>
                                <input name="username"
class="form-control mb-2" placeholder="Логин" required/>
                                <input name="password"

```

```

type="password"      class="form-control mb-2"      placeholder="Пароль"
required/>
                                <button class="btn btn-success
w-100">Зарегистрироваться</button>
                                </form>
                                </div>
                            </div>
                            <div class="text-center mt-3">
                                <form
                                    th:action="@{/auth/guest}"
method="post">
                                    <button
                                        class="btn
                                        btn-
secondary">Продолжить как гость</button>
                                    </form>
                                </div>
                            </div>
                        </div>
                    </div>
                </div>
            </div>
        <main class="flex-fill">
            <div class="container mt-4 content-overlay">

                <!-- Приветствие -->
                <h1 class="mb-4">Добро пожаловать на Книговорот!</h1>

                <!-- Последние добавленные книги -->
                <h2>Последние добавленные книги</h2>
                <div id="latestBooks" class="row gy-3 mb-5"></div>

                <!-- Подборки -->
                <h2>Подборки</h2>
                <div id="collections" class="row gy-3 mb-5"></div>

                <!-- Чат -->
                <h2>Чат</h2>
                <ul
                    id="messages"
                    class="list-group
                    mb-3"
style="height:300px;overflow-y:auto;"></ul>

                <!-- Форма чата для авторизованных -->
                <div sec:authorize="hasRole('USER')" class="input-group
mb-5">
                    <input
                        id="chatInput"
                        class="form-control"
placeholder="Введите сообщение..."/>
                    <button
                        id="sendBtn"
                        class="btn
                        btn-
primary">Отправить</button>
                </div>

                <!-- Подсказка для гостей -->
                <div sec:authorize="isAnonymous()" class="alert alert-
secondary mb-5">
                    Чтобы писать в чате, пожалуйста,
                    <a th:href="@{/auth/login}">войдите</a> или
                    <a
th:href="@{/auth/register}">зарегистрируйтесь</a>.
                </div>

```

```

        </div>
    </main>

    <!-- footer -->
    <div th:replace="fragments/footer :: footer"></div>

    <script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/js/bootstrap.b
undle.min.js"></script>
    <!-- автопоказ модалки -->
    <script>
        document.addEventListener('DOMContentLoaded', () => {
            const el = document.getElementById('welcomeModal');
            if (el) new bootstrap.Modal(el).show();
        });
    </script>
    <script
src="https://cdn.jsdelivr.net/npm/sockjs-
client@1.6.1/dist/sockjs.min.js"></script>
    <script
src="https://cdnjs.cloudflare.com/ajax/libs/stomp.js/2.3.3/stomp.min.
js"></script>

    <script th:src="@{/js/main.js}"></script>
    <script th:src="@{/js/chat.js}"></script>

</body>
</html>

```

					ПГУ 1.090301.29.001 ПЗ	Лист
						139
Изм.	Лист	№ докум.	Подпись	Дата		

Приложение В

(Обязательное)

Web-приложение «Книговорот»

Презентация

					ПГУ 1.090301.29.001 ПЗ	Лист
						140
Изм.	Лист	№ докум.	Подпись	Дата		



WEB-ПРИЛОЖЕНИЕ «КНИГОВОРОТ»

Автор работы:

студент группы 21ВВП2
Давкин Максим Александрович

Научный руководитель:

к.э.н., доцент каф. ВТ ПГУ
Акифьев И.В.

Цель и требования

- Цель работы – разработать web-приложение для обмена книгами электронными и вещественными.
- Требования к приложению:
 - чтение и скачивание, добавление и удаление книг;
 - поиск и сортировка книг;
 - создание и просмотр подборок;
 - регистрация и авторизация пользователя;
 - общение в чате, комментирование книг.

Актуальность темы

- В современном мире большинство людей привыкло читать книги в электронном формате. Однако существует огромная часть книг, которые невозможно по разным причинам перенести в любое мобильное и не только устройство – это редкие книги, эксклюзивные издания, старые и древние книги, которые могут быть не в лучшем состоянии. Их ценность для книголюбителей нельзя приуменьшать.
- Идея создания приложения, систематизирующего издания любого формата и образующего свое сообщество людей по книгообмену и не только, выглядит актуальной.

Стек технологий

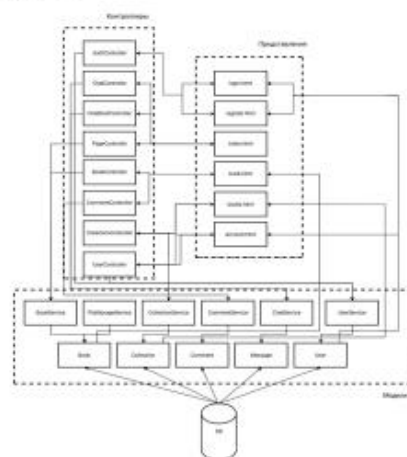


					ПГУ 1.090301.29.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		142

Диаграммы вариантов использования



Архитектура



Структура базы данных



Заключение

- При выполнении данной выпускной квалификационной работы было разработано веб-приложение «Книговорот».
- Приложение может улучшаться – может появиться более интерактивная система общения между пользователями сайта, система физического обмена книг может быть уточнена и доработана, внешний вид приложения и интерфейс может усовершенствоваться, а безопасность пользователей увеличена.

ВСЕМ СПАСИБО ЗА ВНИМАНИЕ!



					ПГУ 1.090301.29.001 ПЗ	Лист
						145
Изм.	Лист	№ докум.	Подпись	Дата		