

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

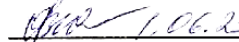
КЛИЕНТ-СЕРВЕРНОЕ ПРИЛОЖЕНИЕ

«БИБЛИОТЕКА»

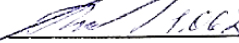
Студент

 1.06.23 Серов Даниил Валерьевич
(подпись, дата)

Руководитель

 1.06.23 Юрова Ольга Викторовна
(подпись, дата)

Нормоконтролёр

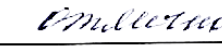
 1.06.23 Бычков Андрей Станиславович
(подпись, дата)

Работа допущена к защите (протокол зас. кафедры от 02.06.2023г. № 14)

Заведующий кафедрой


(подпись)

М.А. Митрохин
(фамилия, инициалы)

Работа защищена с отметкой  (протокол заседания ГЭК от

06.06.2023г. № 6)

Секретарь ГЭК


(подпись)

Е.И. Гурин
(фамилия, инициалы)

Пенза 2023

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

Факультет
вычислительной техники

Кафедра
«Вычислительная техника»

«Утверждаю»
Заведующий кафедрой
М.А. Митрохин
«26» 04 2023 г.

**ЗАДАНИЕ
НА БАКАЛАВРСКУЮ РАБОТУ**

1. Студент Серов Даниил Валерьевич гр.19ВВП1 факультета вычислительной техники

2. Тема работы: клиент серверное приложение «Библиотека»

Тема утверждена приказом ПГУ № 357/о от "26" апреля 2023 г.

3. Руководитель бакалаврской работы к.т.н., доцент кафедры ВТ ПГУ Юрова О.В.

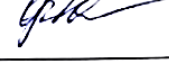
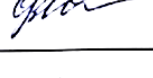
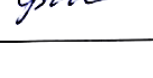
4. Задание на работу (назначение разработки, исходные данные и т.п.)

Разработать клиент-серверное приложение «Библиотека» обмена электронными книгами между пользователями. Приложение должно позволять пользователям выполнять просмотр электронных книг, как собственных, так и других пользователей, а также осуществлять возможность загрузки и удаления книг. Также приложение должно позволять осуществлять регистрацию и авторизацию пользователя, редактировать информацию о пользователе. Приложение должно быть спроектировано с использованием клиент-серверной архитектуры. Клиентская часть программы должна обладать понятным графическим интерфейсом. Сервер должен включать в себя сохранение данных в базу данных, передачу, редактирование и удаление их из нее. Клиентская программа должна позволять пользователю добавлять, скачивать и удалять электронные книги, получать список этих книг и просматривать их.

5. Перечень подлежащих разработке вопросов

Требуется произвести анализ предметной области: изучить существующие аналоги, изучить способы проектирования и разработки клиент-серверного приложения. Также необходимо изучить технологий для разработки приложения, такие как Spring Framework, PostgreSQL, Spring Security и т.д.

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1.	Изучение предметной области	50	26.04.2023-30.04.2023	
2.	Анализ требований	10	01.05.2023-02.05.2023	
3.	Выбор технологий, применяемых при проектировании и разработке	50	03.05.2023-07.05.2023	
4.	Описание алгоритмов	40	08.05.2023-11.05.2023	
5.	Разработка клиент серверного приложения «Библиотека»	150	12.05.2023-29.05.2023	
6.	Оформление пояснительной записки	24	30.05.2023-31.05.2023	

Дата выдачи задания "26" апреля 2023 г.

Руководитель бакалаврской работы _____  Юрова О.В.
(подпись, дата)

Задание к исполнению принял студент _____  Серов Д.В.
(подпись, дата)

Работу к защите допустить
Декан факультета _____  02.03.2023 Фионова Л.Р.

Реферат

Бакалаврская работа содержит 97 лист, 43 рисунка, 4 таблицы, 6 источников и 3 приложения.

КЛИЕНТ-СЕРВЕРНОЕ ПРИЛОЖЕНИЕ «БИБЛИОТЕКА», *SPRING FRAMEWORK, INTELLIJ IDEA, SPRING SECURITY, POSTGRESQL, PGADMIN4, HTML, CSS, JAVA*

Цель работы – разработать клиент-серверное приложение, позволяющую совершать обмен электронными книгами между пользователями.

Разработанные приложение должно позволять пользователям выполнять просмотр электронных книг, как собственных, так и других пользователей, а также осуществлять возможность загрузки и удаления книг. Также приложение должно позволять осуществлять регистрацию и авторизацию пользователя, редактировать информацию о пользователе.

Результатом разработки является проект, удовлетворяющий поставленной задаче. А именно, разработано, реализовано и протестировано приложение для обмена электронными книгами между пользователями. Разработка производилась с использованием масштабного стека технологий, к которым относятся: язык программирования *Java*, язык структурированных запросов *SQL*, *Spring Framework*, *Spring Security*, язык гипертекстовой разметки *HTML*, язык каскадных таблиц стилей *CSS*, среда разработки *IntelliJ IDEA*. Для хранения данных выбрана СУБД *PostgreSQL*, а для её администрирования применялся инструмент *PGAdmin4*.

					ПГУ1.090301.21.001 ПЗ			
Изм.	Лист	№ докум.	Подпись	Дата				
Разраб.		Серов Д.В.			Клиент-сервёрное приложение «Библиотека» Пояснительная записка	Лит.	Лист	Листов
Провер.		Юрова О.В.					2	97
Н. Контр.		Бычков А. С.				ФВТ, каф. ВТ гр. 19ВВП1		
Утверд.		Митрохин М.А.						

Содержание

Перечень сокращений и обозначений.....	5
Введение.....	6
1 Клиент серверное приложение «Библиотека»	7
1.1 Анализ предметной области	7
1.1.1 Анализ актуальности темы	7
1.1.2 Клиент-серверная архитектура.....	8
1.1.3 Анализ аналогов клиент-серверных приложений библиотек	9
1.1.4 Постановка задачи	14
1.2 Выбор технологий и средств разработки приложения	15
1.2.1 Spring framework	16
1.2.2 Spring security	16
1.2.3 Spring JPA	17
1.2.4 PostgreSQL	18
1.2.5 PgAdmin4	18
1.2.6 IntelliJ IDEA	19
1.2.7 Apache Tomcat	19
1.2.8 Java.....	19
1.2.9 Bootstrap	20
1.3 Архитектура клиент-серверного приложения «Библиотека»	20
1.4 Реализация приложения	23
1.4.1 Описание работы приложения.....	23
1.4.2 Структура БД.....	25
1.4.3 Функционал приложения	27

1.4.4 Основные компоненты	29
1.4.5 Основные алгоритмы взаимодействия клиента и сервера	37
1.4.6 Тестирование приложения	42
2 Безопасность жизнедеятельности.....	54
2.1 Требования к помещению и рабочему месту.....	54
2.2 Требования к освещению	55
2.3 Требования к режиму труда и отдыха	56
2.4 Рекомендации по использованию программного обеспечения	57
Заключение	59
Список используемых источников.....	60
Приложение А	61
Приложение Б	66
Приложение В.....	87

Перечень сокращений и обозначений

В настоящей работе применяют следующие сокращения и обозначения

БД – база данных

ПО – программное обеспечение

СУБД – система управления базой данных

CSS – (от англ. *Cascading Style Sheets*) – формальный язык описания внешнего вида документа.

CSRF (от англ. *Cross-Site Request Forgery*) – вид атак на сайт, при которой злоумышленник с помощью мошеннического сайта заставляет браузер пользователя выполнять на доверенном сайте действия от его имени

HTML (от англ. *Hyper Text Markup Language*) – язык гипертекстовой разметки

HTTP – (от англ. *Hypertext Transfer Protocol*) – протокол прикладного уровня передачи данных

JPA – (от англ. *Java Persistence API*) – спецификация, описывающая систему управления сохранением java-объектов в таблицы реляционных баз данных в удобном виде

MVC (от англ. *Model-View-Controller*) — шаблон (паттерн) программирования, разделяющий архитектуру приложения на три модуля

REST – (от англ. *Representational State Transfer*) – передача состояния представления

SOAP – (от англ. *Simple Object Access Protocol*) – протокол обмена сообщениями

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

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

XSS (от англ. *Cross-Site Scripting*) — распространенный тип веб-атаки, заключающийся во внедрении на страницу сайта вредоносного кода

Введение

Современный мир становится все более зависимым от технологий, и книги не являются исключением. Онлайн-библиотеки становятся все более популярными, потому что они предлагают удобный способ получения доступа к книгам и другой информации. Кроме того, онлайн-библиотеки обеспечивают доступ к книгам из любой точки мира, что является большим преимуществом для людей, которые живут в удаленных или малонаселенных районах.

Клиент-серверное приложение онлайн библиотека не только удобно для пользователей, но и облегчает работу библиотекарей. Такое приложение позволяет библиотекарям легко управлять книгами, подписками и пользователями, а также предоставлять различную статистическую информацию.

Таким образом, разработка клиент-серверного приложения онлайн библиотека является актуальной и важной задачей для современного общества, поскольку позволяет улучшить доступность книг и упростить управление библиотечным фондом.

					<i>ПГУ1.090301.21.001 ПЗ</i>	Лист
						6
Изм.	Лист	№ докум.	Подпись	Дата		

1 Клиент серверное приложение «Библиотека»

1.1 Анализ предметной области

Начало разработки программного продукта всегда связано с анализом предметной области. Необходимо определить кто будет пользоваться продуктом, выбрать актуальные технологии, которые позволят в будущем поддерживать и развивать продукт. В данном подразделе рассматриваются общие теоретические вопросы, связанные с проектированием и реализацией клиент-серверного приложения с возможностью загрузки и скачиванием книг. Изучение данных вопросов поможет определить основные проблемы и задачи для разработки программного продукта.

1.1.1 Анализ актуальности темы

В современном мире библиотеки все еще остаются важными источниками информации и знаний для многих людей. Однако, с развитием информационных технологий и доступности интернета, читатели стали все больше использовать электронные ресурсы и онлайн-библиотеки.

Клиент-серверные приложения для библиотек могут быть очень полезными для улучшения доступности и комфортного использования библиотечных ресурсов. Они позволяют пользователям быстро и удобно искать, заказывать книги, а также получать доступ к электронным версиям книг и журналов.

Одним из главных преимуществ библиотечных клиент-серверных сервисов является возможность удаленного доступа к электронным книжным ресурсам, что позволяет клиентам получать необходимую информацию в любое время и из любого места. Также, такие сервисы могут помочь библиотекам улучшить свою эффективность и повысить уровень обслуживания.

Следует отметить, что клиент-серверные приложения для библиотек имеют свои недостатки, такие как сложность внедрения и высокие затраты на разработку и поддержку. Но учитывая растущую потребность пользователей в быстром и удобном доступе к информации, использование таких приложений может стать необходимостью для современных библиотек.

					ПГУ1.090301.21.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		7

1.1.2 Клиент-серверная архитектура

Клиент-серверная архитектура – это модель распределенных систем, которая предполагает наличие двух типов узлов: клиентов и серверов.

Клиенты — это программы, которые запускаются на компьютере пользователя и предоставляют интерфейс для взаимодействия с сервером. Клиенты могут быть как установленными приложениями, так и веб-браузерами, которые работают с веб-серверами.

Серверы — это программы, которые запускаются на специально выделенном компьютере и предоставляют некоторые услуги клиентам. Серверы могут предоставлять различные сервисы, такие как веб-серверы, базы данных, почтовые серверы и т.д [1].

Клиент и сервер взаимодействуют по определенному протоколу, который определяет соглашение между ними. Клиент отправляет запросы на сервер, а сервер возвращает ответы на эти запросы.

Преимущества клиент-серверной архитектуры:

- централизованное управление и хранение данных на сервере;
- удобство совместной работы и обмена данными между пользователями;
- легкость сопровождения и модификации системы;
- способность разделения прав доступа к различным ресурсам информационной системы.

Недостатки клиент-серверной архитектуры:

- необходимость поддерживать и обновлять серверную инфраструктуру;
- возможность отказа работы всей системы при сбое сервера;
- необходимость обеспечения безопасности и защиты данных на сервере;
- дороговизна серверного устройства и его содержание.

Взаимодействие клиента и сервера происходит по определенному протоколу, который определяет соглашение между ними. Обычно протоколы взаимодействия бывают стандартизированы и поддерживаются множеством различных программ и

устройств. Примерами могут служить такие протоколы и технологии как *HTTP*, *SOAP*, *REST* и т.д.

REST - это стиль архитектуры, который используется для создания распределенных систем. Его основная идея заключается в том, что клиент и сервер должны обмениваться представлением ресурсов, которые хранятся на сервере. В данной работе будет использоваться *REST* стиль, так как он является простым, гибким и эффективным стилем архитектуры [2].

1.1.3 Анализ аналогов клиент-серверных приложений библиотек

Каждый год на рынок появляются новые сервисы и приложения, предлагающие новые возможности для комфортного использования продукта пользователями. Одними из таких приложений является библиотечные приложения, которые позволяют пользователям искать, заказывать и читать книги в электронном виде.

Для того чтобы реализовать такое приложения необходимо провести анализ аналогов. Это позволит выявить преимущества и недостатки существующих решений, а также определить требования к разрабатываемому приложению. В данном пункте будут рассмотрены три популярных сервиса для чтения электронных книг:

- онлайн-библиотека «*Mybook.ru*»;
- онлайн-сервис «*Bookmate*»;
- онлайн-библиотека «Самиздат».

1.1.3.1 Онлайн-библиотека «*Mybook.ru*»

Интерфейс онлайн сервиса «*Mybook.ru*» представлен на рисунках 1 и 2.

Особенностями данного аналога является возможность оформления платной подписки, которая позволяет пользователю читать все книги, имеющиеся в данной библиотеке.

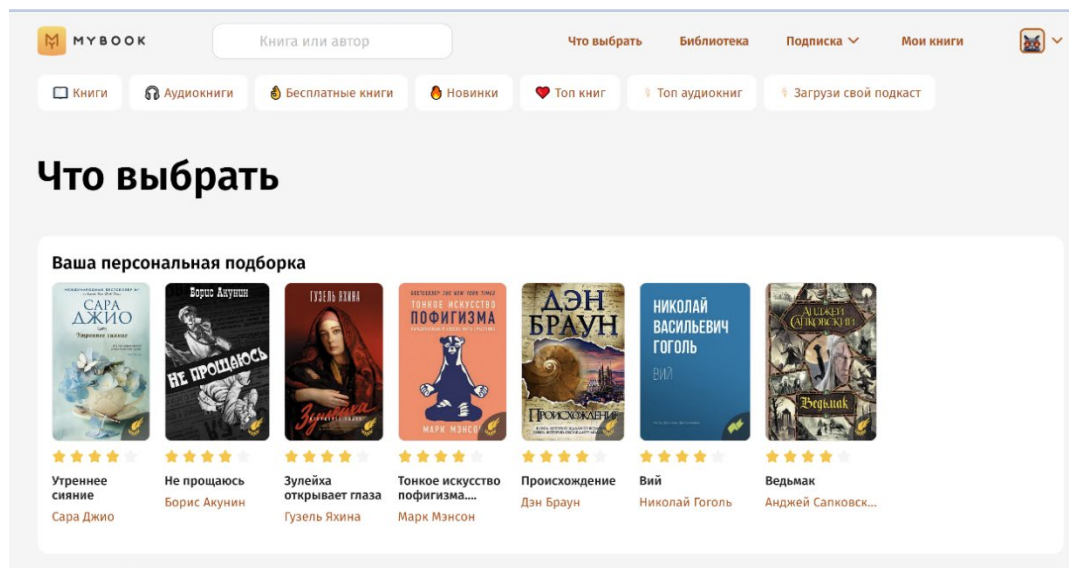


Рисунок 1 – Главная страница сервиса

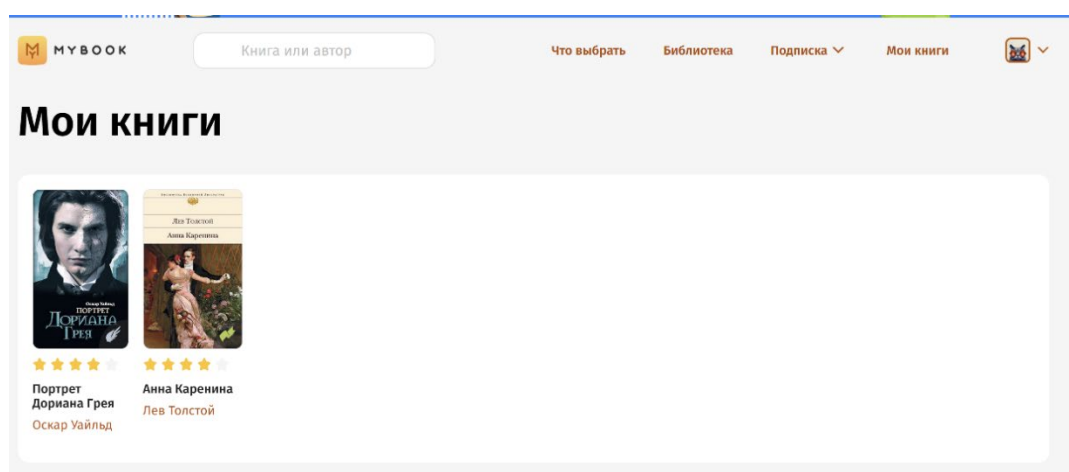


Рисунок 2 – Библиотека клиента

Сервис онлайн-библиотеки «Mybook.ru» предоставляет своим пользователям широкий спектр услуг для чтения книг. Ниже перечислены основные функции и возможности, которые доступны на сайте:

- поиск книг. На сайте «Mybook.ru» пользователь может найти книги по автору, названию, жанру, тегам, ISBN и другим параметрам;
- чтение книг онлайн. Данный сервис позволяет читать книги прямо на сайте без необходимости их скачивания. Для этого необходимо зарегистрироваться на сайте и выбрать нужную книгу в каталоге;

– покупка подписки. Пользователь может оформить подписку на сервис, которая дает возможность просматривать все книги, находящиеся в библиотеке;

– создание библиотеки. Пользователь может создать собственную библиотеку на сайте, добавляя книги в избранное.

1.1.3.2 Онлайн-сервис «Bookmate»

Интерфейс онлайн сервиса «Bookmate» представлен на рисунках 3 и 4.

Особенностями данного сервиса является возможность загружать свои книги и хранить их на сайте, создавая тем самым собственную библиотеку.

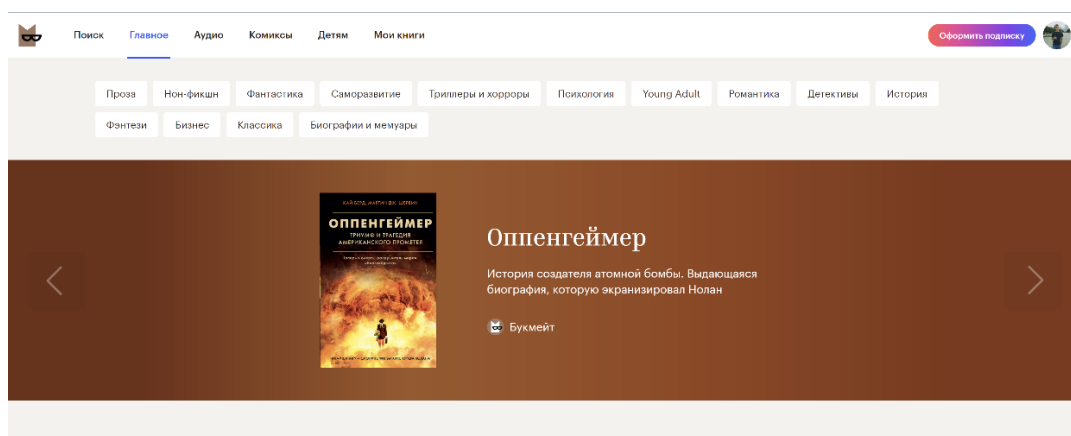


Рисунок 3 – Главная страница сервиса

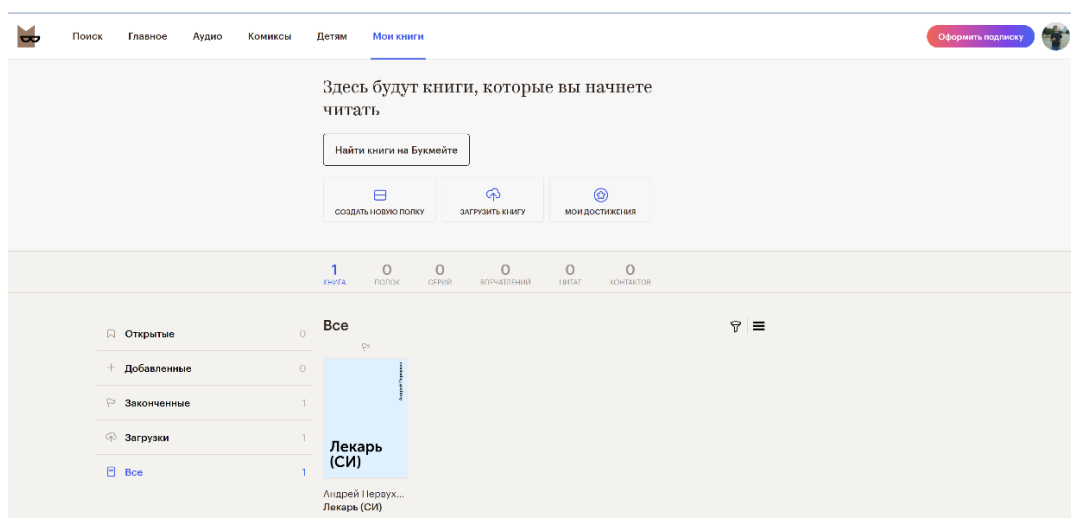


Рисунок 4 – Библиотека клиента

Онлайн-библиотека «Bookmate» дает своим клиентам возможность чтения, просмотра книг и прослушивания аудиокниг. Ниже приведены основные функции, которые доступны пользователям:

– библиотека книг и аудиокниг. Пользователи могут выбрать из более чем 1 миллиона книг и аудиокниг на различных языках;

– персонализированные рекомендации. «Bookmate» предлагает рекомендации книг и аудиокниг на основе прошлых прочитанных книг;

– чтение и прослушивание в режиме онлайн. Пользователи имеют возможность читать и слушать книги прямо в браузере, что позволяет не скачивать книги на устройство;

– разделение книг с друзьями. Читатели могут рекомендовать книги своим друзьям и делиться своими отзывами и оценками книг;

– создание собственных коллекций. Сервис предоставляет функционал, который позволяет создавать свои собственные коллекции книг и аудиокниг, добавляя их с сайта либо загружая с устройства;

– поиск. Пользователи могут искать книги и аудиокниги по автору, названию, жанру и другим параметрам.

1.1.3.3 Онлайн-библиотека «Самиздат»

Интерфейс сайта «Самиздат» представлен на рисунках 5 и 6.

Главным отличием данной библиотеки является предоставление возможности молодым авторам публиковать свои книги на сайте, а также отслеживать полезную информацию для пользователей и авторов.

Онлайн-библиотека «Самиздат» представляет собой сайт, на котором пользователи могут бесплатно читать и выкладывать свои книги. В библиотеке представлены произведения различных жанров и авторов, включая классику, фантастику, детективы и другие.

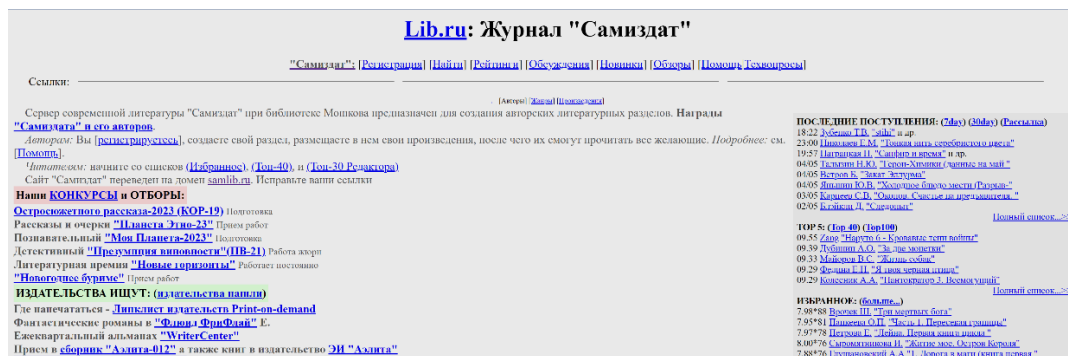


Рисунок 5 – Главная страница сервиса.

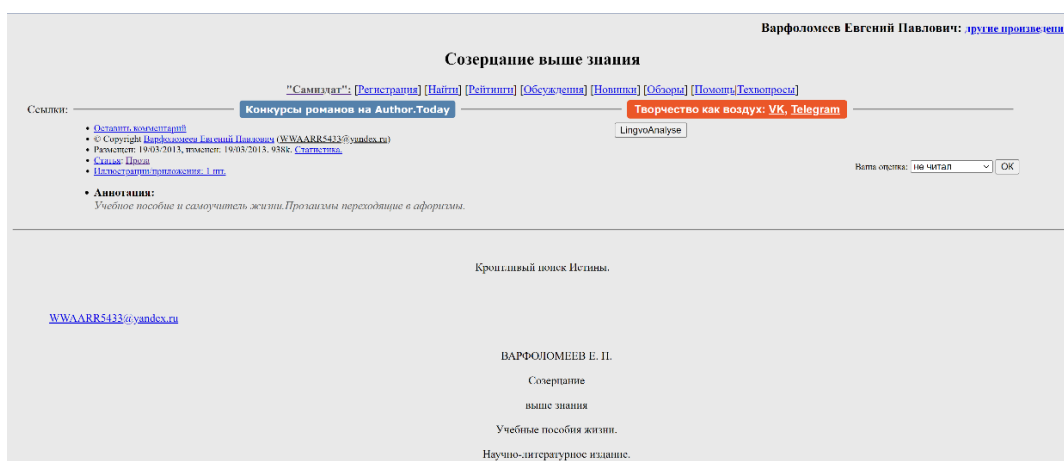


Рисунок 6 – Просмотр книги

Для поиска нужной книги пользователи могут использовать различные параметры, такие как название, автор и другие. Также на сайте можно оставлять комментарии к книгам, выражать свое мнение и обсуждать произведения с другими читателями.

Зарегистрированные пользователи могут добавлять свои книги на сайт и делиться ими с другими читателями.

Онлайн-библиотека «Самиздат» предоставляет пользователям удобный и бесплатный доступ к богатой коллекции книг, а также возможность общения с другими читателями и создания своей собственной библиотеки.

1.1.3.4 ВЫВОД

Результаты анализа существующих аналогов можно представить в таблице 1, показывающие минусы и плюсы рассматриваемых сервисов.

					ПГУ1.090301.21.001 ПЗ	Лист
						13
Изм.	Лист	№ докум.	Подпись	Дата		

Исходя из полученных результатов, было решено разработать клиент-серверное приложение «Библиотека», которое будет предоставлять пользователям удобный интерфейс, все основные функции, необходимые для комфортного использования, также обеспечить возможность авторизованным пользователям просматривать книги онлайн или скачивать на устройство.

Таблица 1 – Достоинства и недостатки онлайн-библиотек

Функционал	<i>Bookmate</i>	<i>Mybook.ru</i>	Самиздат
Поиск по критериям	+	+	+
Чтение книги онлайн	+	+	+
Возможность общения между читателями	+	-	+
Создание собственной библиотеки	+	+	-
Возможность загружать книги	+	-	+
Возможность скачивать книги	-	-	-
Бесплатный сервис	-	-	+
Понятный интерфейс	+	+	-

1.1.4 Постановка задачи

Клиент-серверное приложение «Библиотека» должно иметь следующий функционал, который представлен ниже.

Со стороны сервера:

- предоставление данных о пользователе, книгах;
- проверка корректности вводимых данных в базу данных;
- редактирование, добавление и удаление пользователя, экземпляра книги в базе данных;
- обеспечивать безопасность, позволяя загружать в базу данных книги,

только авторизованным пользователям.

Со стороны клиента:

- регистрация и авторизация с удобным графическим интерфейсом;
- загрузка и скачивание книги, хранимой на сервере;
- редактирование, добавление и удаление данных о книги, пользователи;
- просмотр списка книг других пользователей и своей библиотеки;
- возможность просмотра книг перед скачиванием.

Минимальные системные требования:

- операционная система: *Windows* (7, 8, 10);
- браузер: *Google Chrome, Mozilla Firefox, Opera* (с поддержкой *HTML5, CSS3*);
- минимальная скорость интернета: 2-3 Мбит/сек;
- минимальный объем оперативной памяти: 1 ГБ (рекомендуется 2 ГБ и выше);
- минимальный объем свободного места на жестком диске: 100 МБ.

1.2 Выбор технологий и средств разработки приложения

В этом подразделе будут рассмотрены технологии, язык программирования и среда разработки, которые были выбраны для написания клиент-серверного приложения «Библиотека».

Технологии и средства разработки:

- *Spring framework*;
- *Spring JPA*;
- *Spring Security*;=
- *PostgreSQL*;
- *pgAdmin4*;
- *IntelliJ IDEA*;
- *Apache Tomcat*;
- *Java*;

– *Bootstrap*.

1.2.1 *Spring framework*

Spring Framework – это один из самых популярных и широко используемых фреймворков для разработки приложений на *Java*. Он предоставляет множество функциональных возможностей для упрощения и ускорения процесса разработки, а также для улучшения качества и надежности приложения.

Основные возможности *Spring Framework* включают:

– управление зависимостями: *Spring Framework* предоставляет механизмы для управления зависимостями между компонентами приложения. Это позволяет упростить процесс разработки и улучшить его качество;

– внедрение зависимостей: *Spring Framework* предоставляет механизмы для внедрения зависимостей между компонентами приложения. Это делает разработку приложения гибкой и расширяемой;

– управление транзакциями: *Spring Framework* предоставляет механизмы для управления транзакциями в приложении. Данная возможность упрощает работу с базами данных, а также повышает надежность и безопасность.

Spring Framework также поддерживает различные стандарты и спецификации, такие как *Java EE*, *JPA* и другие. Тем самым можно использовать этот инструмент в различных средах и с различными технологиями. В целом, *Spring Framework* является мощным и гибким фреймворком для разработки приложений на *Java* [3].

1.2.2 *Spring security*

Spring security – это фреймворк для обеспечения безопасности веб-приложений на основе *Spring Framework*. Он предоставляет мощные механизмы для аутентификации и авторизации пользователей, защиты от атак и обработки ошибок безопасности.

Основные возможности *Spring Security*:

– аутентификация: *Spring Security* предоставляет механизмы для аутентификации пользователей, включая поддержку различных методов

аутентификации, таких как базовая аутентификация, форма входа и *OAuth*;

- авторизация: *Spring Security* предоставляет механизмы для авторизации пользователей, включая поддержку различных методов авторизации, таких как ролевая авторизация и правила авторизации;

- защита от атак: *Spring Security* предоставляет механизмы для защиты приложения от различных типов атак, включая атаки на сессии, *CSRF*-атаки и *XSS*-атаки.

Основные компоненты *Spring Security* включают:

- *authentication*: компонент, который предоставляет механизмы для аутентификации пользователей;

- *authorization*: компонент, который предоставляет механизмы для авторизации пользователей;

- *web Security*: компонент, который предоставляет механизмы для защиты веб-приложения;

- *config*: компонент, который предоставляет механизмы для настройки *Spring Security*.

1.2.3 *Spring JPA*

Spring JPA – это модуль *Spring Framework*, который предоставляет простой и удобный способ работы с базами данных в приложениях на *Java*. Он предоставляет возможность работать с различными СУБД и облегчает процесс работы с данными.

Основные возможности *Spring JPA*:

- аннотации: *Spring JPA* предоставляет множество аннотаций, которые могут быть использованы для маппинга объектов *Java* на таблицы в базе данных. Позволяет уменьшить количество кода;

- репозитории: *Spring JPA* предоставляет механизмы для создания репозитория, которые предоставляют удобный интерфейс для работы с данными. Это позволяет упростить процесс работы с данными и повысить расширяемость;

- транзакции: *Spring JPA* предоставляет механизмы для управления транзакциями в приложении. Это позволяет повысить надежность и безопасность.

1.2.4 PostgreSQL

PostgreSQL – это мощная объектно-реляционная система управления базами данных, которая предоставляет широкий спектр функций и возможностей для хранения, организации, управления и анализа данных. Рассматриваемая база данных является клиент-серверной системой, где клиентское приложение подключается к серверу *PostgreSQL* для выполнения операций с базой данных. Сервер *PostgreSQL* состоит из нескольких процессов, включая процесс главного сервера, процесс управления соединениями и несколько рабочих процессов, которые обрабатывают запросы на операции с базой данных. Ниже представлено подробное описание основных компонентов и функций *PostgreSQL*.

– язык запросов. Поддерживает множество языков запросов, включая *SQL*, которая является стандартным языком для взаимодействия с большинством СУБД. *PostgreSQL* также поддерживает *Procedural Language/SQL (PL/SQL)*, который предоставляет возможность написания процедур и функций на *SQL*, а также другие языки, такие как *Perl, Python, Tcl, Java* и *C*;

– хранение данных. Используется объектно-реляционную модель хранения данных, что означает, что данные хранятся в таблицах, которые могут содержать объекты, такие как строки, столбцы и индексы. *PostgreSQL* также поддерживает хранение данных в виде *JSON*-документов, что позволяет хранить данные с более сложной структурой;

– транзакции. *PostgreSQL* поддерживает транзакции, которые позволяют выполнять группу операций с базой данных как единое целое. Если одна из операций в транзакции завершается неудачно, все операции отменяются.

1.2.5 PgAdmin4

PgAdmin 4 - это бесплатное кросс-платформенное приложение для управления базами данных *PostgreSQL*. Оно предоставляет удобный и интуитивно понятный интерфейс для работы с базами данных, администрирования и мониторинга.

1.2.6 *IntelliJ IDEA*

IntelliJ IDEA - это интегрированная среда разработки для языков программирования *Java*, *Kotlin*, *Groovy*, *Scala* и других. Она разработана компанией *JetBrains* и предоставляет множество инструментов и функций для удобной и эффективной разработки программного обеспечения.

Данная среда предоставляет удобный рефакторинг кода. Он позволяет легко и быстро изменять структуру кода, переименовывать переменные и методы, извлекать методы, инкапсулировать поля и многое другое. Это помогает упростить код и повысить его читабельность. Также она интегрируется с популярными сборщиками проектов, такими как *Maven*, *Gradle* и *Ant*, что позволяет облегчить процесс сборки и управления зависимостями проектов.

IntelliJ интегрируется с популярными серверами приложений, такими как *Tomcat*, *GlassFish*, *JBoss* и другими.

1.2.7 *Apache Tomcat*

Apache Tomcat представляет собой веб-сервер и контейнер сервлетов *Java*, разработанный *Apache Software Foundation*. *Tomcat* является открытым и бесплатным программным обеспечением и является одним из самых популярных контейнеров сервлетов *Java*.

Tomcat используется для выполнения *Java*-сервлетов и страниц *JSP* веб-приложений. Он поддерживает стандарты *Java Servlet*, *JavaServer Pages*, *Java Expression Language* и *Java WebSocket*. *Tomcat* также может использоваться как самостоятельный веб-сервер или как промежуточный сервер, например, совместно с *Apache HTTP Server*.

Tomcat является легким и простым в использовании сервлет-контейнером, который может работать на различных операционных системах, включая *Windows*, *Linux* и *macOS*.

1.2.8 *Java*

Java – широко используемый язык программирования для написания интернет-приложений. Миллионы приложений *Java* используются сегодня. *Java* –

это многоплатформенный, объектно-ориентированный и сетевцентрический язык, который сам по себе может использоваться как платформа. Это быстрый, безопасный и надежный язык программирования для всего: от мобильных приложений и корпоративного ПО до приложений для работы с большими данными и серверных технологий.

Java строго типизированный язык программирования, что означает, что каждая переменная должна быть определена как определенный тип данных. Он также имеет автоматическое управление памятью, что означает, что программа сама заботится о выделении и освобождении памяти. Еще одним достоинством данного языка программирования является богатая стандартная библиотека, которая включает в себя множество классов и методов для работы с различными аспектами программирования, такими как работа с сетями, базами данных, графикой и т.д.

1.2.9 *Bootstrap*

Bootstrap - это бесплатный фреймворк для разработки веб-сайтов, который был создан *Twitter*. Он использует *HTML*, *CSS* и *JavaScript* для создания красивых и отзывчивых веб-сайтов.

Данный фреймворк предоставляет разработчикам множество готовых компонентов, таких как кнопки, формы, таблицы, навигационные панели, модальные окна и т.д. Кроме того, он предоставляет множество классов *CSS*, которые позволяют быстро и легко настроить стиль веб-сайта.

Одним из основных преимуществ *Bootstrap* является его отзывчивый дизайн. Это означает, что веб-сайт, созданный с использованием *Bootstrap*, будет автоматически адаптироваться к различным экранам и устройствам, таким как мобильные телефоны и планшеты.

1.3 Архитектура клиент-серверного приложения «Библиотека»

Данный подраздел рассматривает архитектуру клиент-серверного приложения и взаимодействие выбранных технологий внутри продукта.

					ПГУ1.090301.21.001 ПЗ	Лист
						20
Изм.	Лист	№ докум.	Подпись	Дата		

Архитектурным паттерном для данного приложения была выбрана *MVC* архитектура.

Архитектура *MVC* (*Model-View-Controller*) - это популярный подход к разработке приложений, который разделяет приложение на три компонента: модель (*Model*), представление (*View*) и контроллер (*Controller*) [4]. Архитектура этого приложения представлена на рисунке 7.

Модель – это набор классов, реализующих всю бизнес-логику приложения. Эти классы отвечают за обработку данных (сущностей), размещение их в БД, чтение из БД, а также за взаимодействие между самими объектами, составляющими такие данные. В данном случае модель будет отвечать за хранение информации о книгах, авторах, пользователях и т.д. Также модель будет отвечать за проверку прав доступа пользователей к определенным ресурсам.

Представление – набор классов и шаблонов, отвечающих за интерфейс взаимодействия с пользователями (*User Interface, UI*). Представление будет содержать *HTML*-код, который будет взаимодействовать с пользователем.

Контроллер – это связующее звено между первыми двумя компонентами. Классы данного компонента получают данные, содержащиеся в запросе к серверу и передают их в модель для обработки и сохранения. После этого контроллер выбирает, каким способом показать их клиенту с помощью использования некоторого представления, и передает ему данные для формирования *HTML*-ответа.

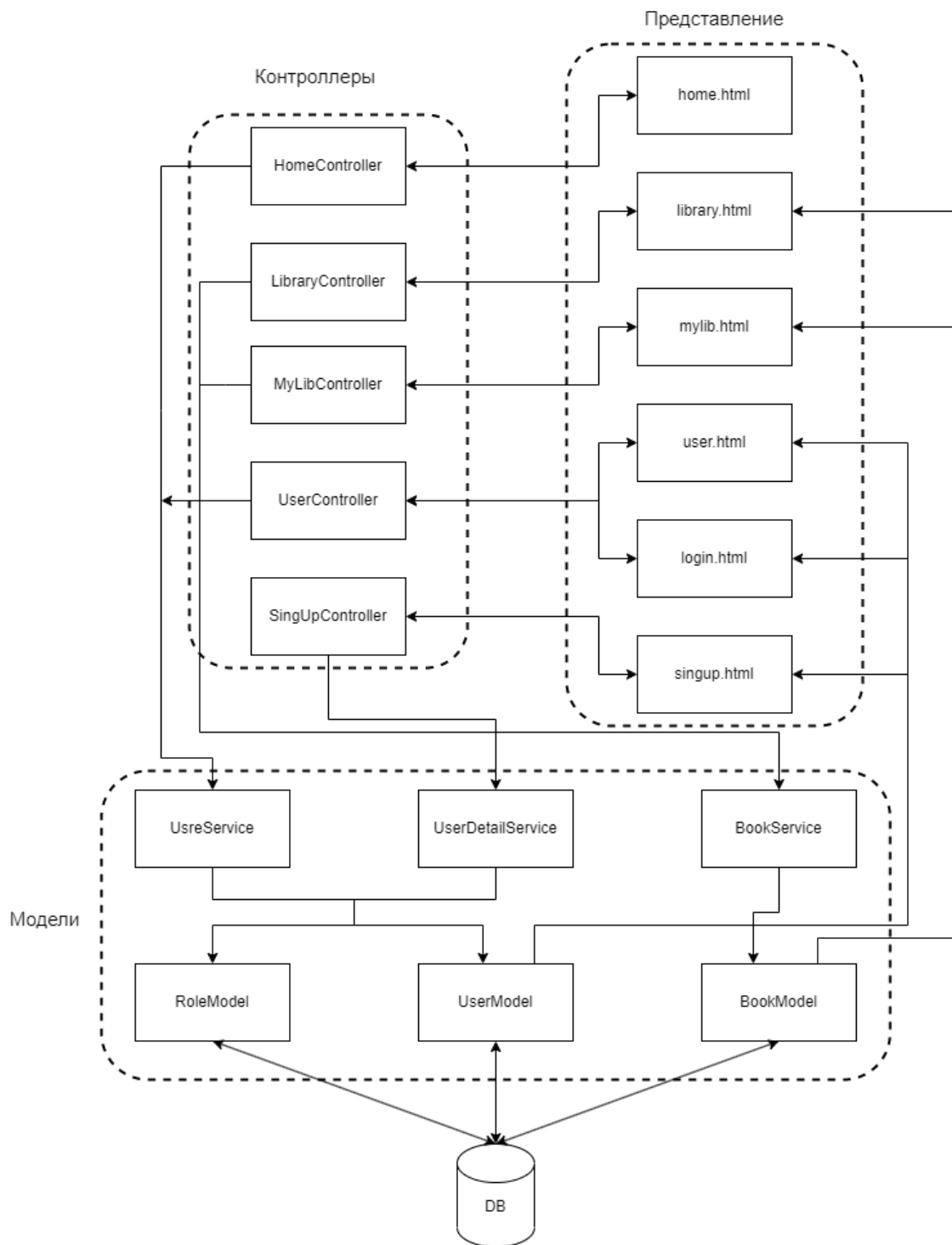


Рисунок 7 – Архитектура приложения

Инструменты реализации архитектуры приложения:

- клиентская часть приложения разработана на *HTML* и *Bootstrap* с использованием *REST* для взаимодействия с сервером;
- серверная часть приложения реализована на *Java* с использованием *Spring*

Framework и *Spring Boot*. *Spring Framework* использован для управления зависимостями и внедрения зависимостей, а *Spring Boot* для быстрой настройки приложения и автоматической настройки;

– взаимодействие с базой данных осуществляется через *Spring JPA*, который предоставляет удобный доступ к базе данных *Postgresql*. Для управления базой данных может использоваться *pg4admin*;

– аутентификация и авторизация пользователей осуществляется через *Spring Security*, который обеспечивает безопасность приложения и контроль доступа;

– для развертывания приложения используется веб-сервер *Apache Tomcat*, который является контейнером сервлетов Java и обеспечивает быстрое и надежное развертывание приложения.

1.4 Реализация приложения

В данном подразделе должен быть описан процесс реализации клиент-серверного приложения "Библиотека", предназначенного для пользователей, которые хотят в удобной форме читать, загружать и скачивать книги в свободном доступе. Будут рассмотрены основные компоненты, функционал приложения и сущности, находящиеся в базе данных, также будет выполнено тестирование приложения. В конечном итоге, цель этого раздела - предоставить полную информацию о разработке и реализации приложения "Библиотека".

1.4.1 Описание работы приложения

Алгоритм работы заключается в следующем: пользователь совершает действия на клиентской части приложения, которая написана на *HTML* с использованием *CSS* стилей и *Thymeleaf* - механизм шаблонов *Java*, в результате совершенных действий сервер получает соответствующий запрос на выполнение задачи. При возникновении ошибки, сервер возвращает клиенту информацию об ошибке, и операция прерывается. В случае успешно принятого запроса, сервер производит обращение к базе данных и в зависимости от запрошенной операции может вернуть пользователю данные, создать, редактировать или удалить их. Когда

запрос к базе данных прошел успешно, сервер вернет пользователю данные, либо выполнит запрашиваемую операцию. Если в ходе обращении к базе данных возникнет ошибка, сервер выполнит обработку данной ошибки и вернет в клиентскую часть информацию об ошибке.

Последовательность выполняемых действий представлена на рисунке 8.

Топология системы и распределение ее компонентов по физическим узлам представлены на рисунке 9.

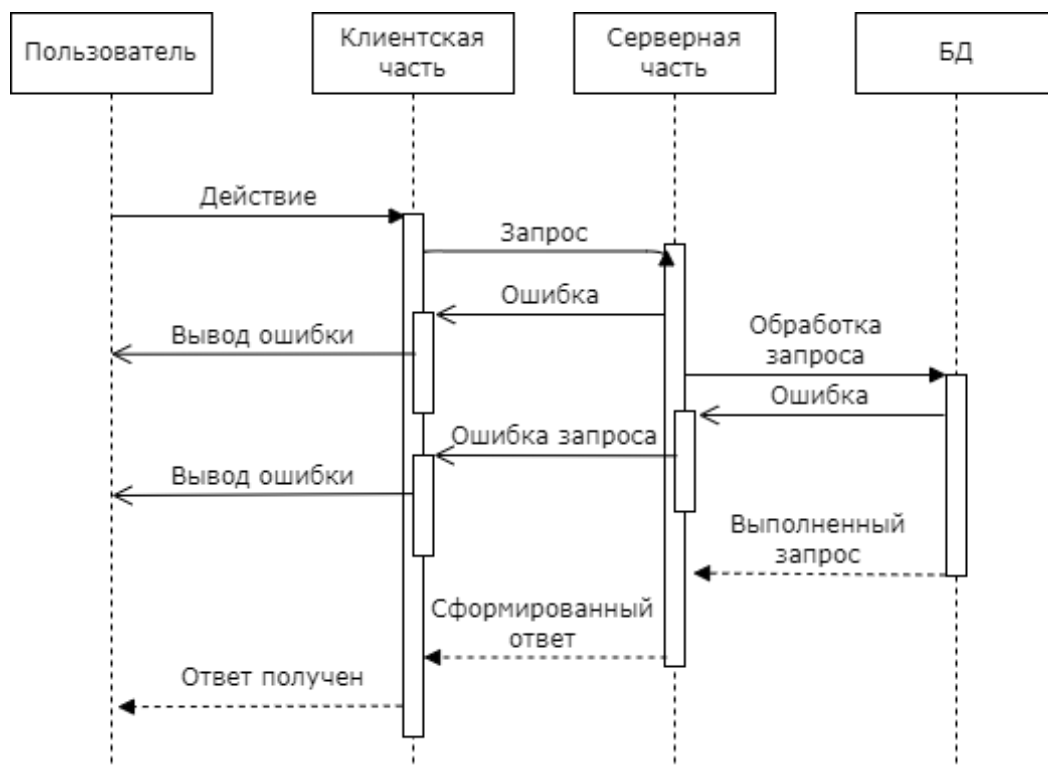


Рисунок 8 – *UML* диаграмма последовательности функционирования приложения

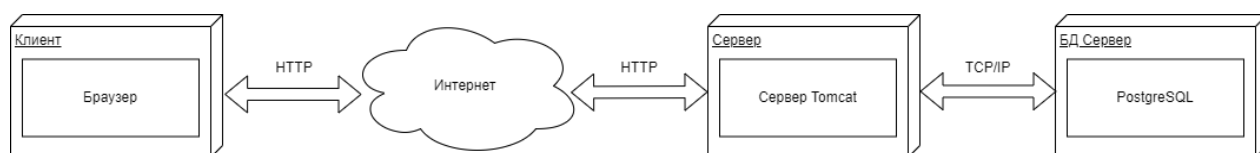


Рисунок 9 – *UML* диаграмма развертывания системы

1.4.2 Структура БД

Для реализации клиент-серверного приложения, способного хранить информацию о пользователях и книгах, необходимо провести анализ и решить какие сущности и связи будут присутствовать в базе данных. Поскольку структура базы данных является ключевым аспектом ее эффективности и функциональности, в данном разделе рассматриваются основные таблицы, атрибуты и связи между ними, которые были разработаны в рамках данной работы [5].

Проведя анализ, было решено, что в разрабатываемом приложении будут использоваться следующие таблицы:

– таблица *users*, представленная в таблице 2. В ней будет содержаться основная информация о пользователях, которая будет использоваться для авторизации и идентификации пользователя в приложении;

– таблица *book*, представленная в таблице 3. Будет хранить основную информацию о книгах, загружаемых пользователями;

– таблица *user_role*, представленная в таблице 4. Здесь содержится роли, которые выдаются пользователям при регистрации, для определения их прав доступа в приложении.

Таблица 2 – Атрибуты сущности «*users*» (Пользователи)

Поле	Тип данных	Описание	Ключ	Пример
<i>id</i>	<i>SERIAL</i>	Идентификатор пользователя	РК	1
<i>email</i>	<i>VARCHAR(100)</i>	Логин пользователя		<i>user@mail.ru</i>
<i>first_name</i>	<i>VARCHAR(60)</i>	Имя пользователя		Иван
<i>last_name</i>	<i>VARCHAR(60)</i>	Фамилия пользователя		Иванов
<i>password</i>	<i>VARCHAR(255)</i>	Пароль пользователя		<i>pass123</i>

Таблица 3 – Атрибуты сущности «*book*» (Книги)

Поле	Тип данных	Описание	Ключ	Пример
<i>id</i>	<i>SERIAL</i>	Идентификатор книги	PK	1
<i>author</i>	<i>VARCHAR(80)</i>	Автор книги		Пушкин
<i>book_name</i>	<i>VARCHAR(80)</i>	Название книги		Онегин
<i>bytes</i>	<i>BYTEA</i>	Содержимое книги		
<i>content_type</i>	<i>VARCHAR(30)</i>	Тип книги		<i>fb2</i>
<i>file_size</i>	<i>INTEGER</i>	Размер книги		804012
<i>original_name_file</i>	<i>VARCHAR(100)</i>	Название файла		<i>onegin.fb2</i>
<i>user_id</i>	<i>INTEGER</i>	Идентификатор пользователя	FK	1

Таблица 4 – Атрибуты таблицы «*user_role*» (Роли пользователей)

Поле	Тип данных	Описание	Ключ	Пример
<i>user_id</i>	<i>INTEGER</i>	Идентификатор пользователя	FK	1
<i>roles</i>	<i>VARCHAR(50)</i>	Роль		<i>USER</i>

На рисунке 10 описана схема базы данных, разрабатываемого приложения, для наглядного рассмотрения взаимодействия таблиц.

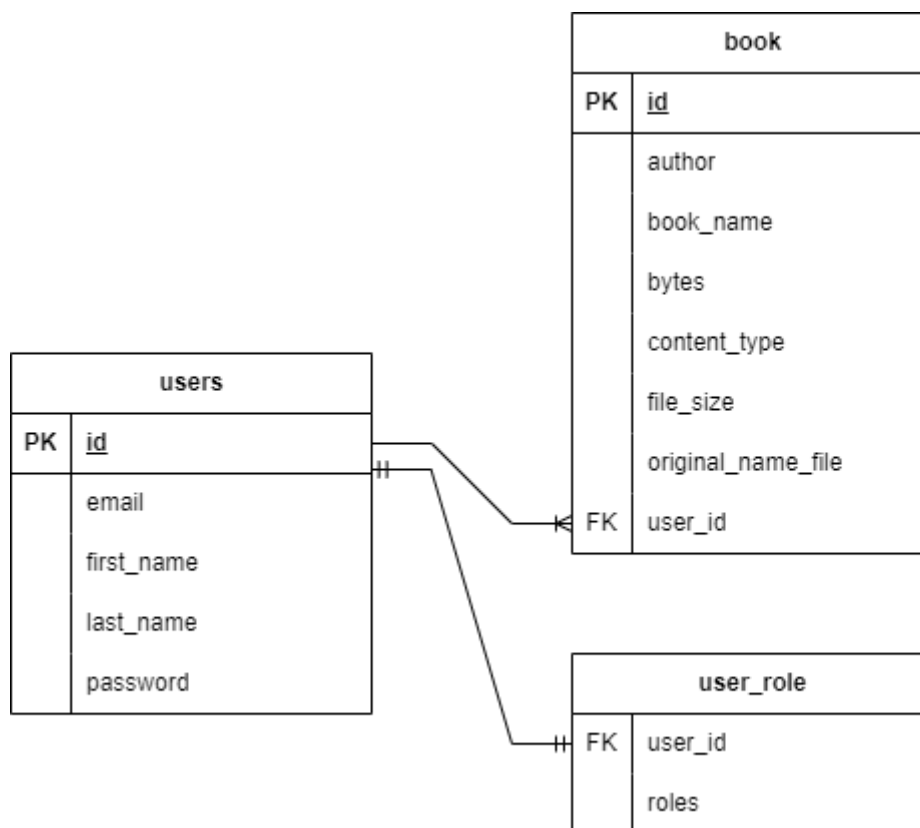


Рисунок 10 – Схема БД

1.4.3 Функционал приложения

Функции данного клиент-серверного приложения можно разделить на две группы:

- для неавторизованного пользователя;
- для авторизованного пользователя.

Неавторизованный пользователь обладает неполным объемом функций, которые позволяют ему ознакомиться с начальным интерфейсом приложения и увидеть список книг, которые загружены другими пользователями. Доступный функционал:

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

Диаграмма вариантов использования приложения для неавторизованного пользователя представлена на рисунке 11.

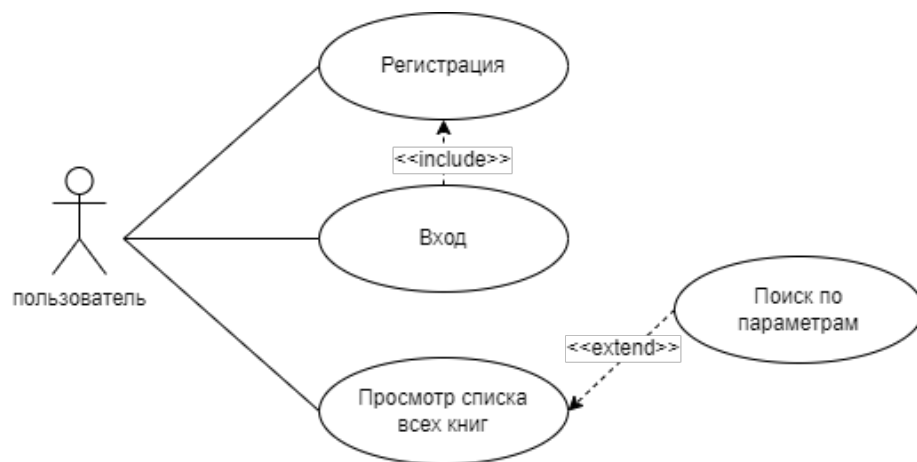


Рисунок 11 – *UML* диаграмма вариантов использования неавторизованного пользователя

Для авторизованного пользователя будет расширенный объем функций, которые позволяют ему помимо просмотра списка книг других пользователей, прочесть в браузере или скачать интересующую книгу, загрузить на сайт свои книги, которые может читать онлайн, скачивать или удалять, также имеет возможность редактировать информацию о себе. Доступный функционал:

- просмотр списка, чтение и скачивание, всех загруженных на сайт книг;
- загрузка своих книг на сайт;
- чтение онлайн и удаление загруженных книг;
- редактирование своих данных или удаление аккаунта;
- выход из аккаунта.

Диаграмма вариантов использования приложения для авторизованного пользователя представлена на рисунке 12.

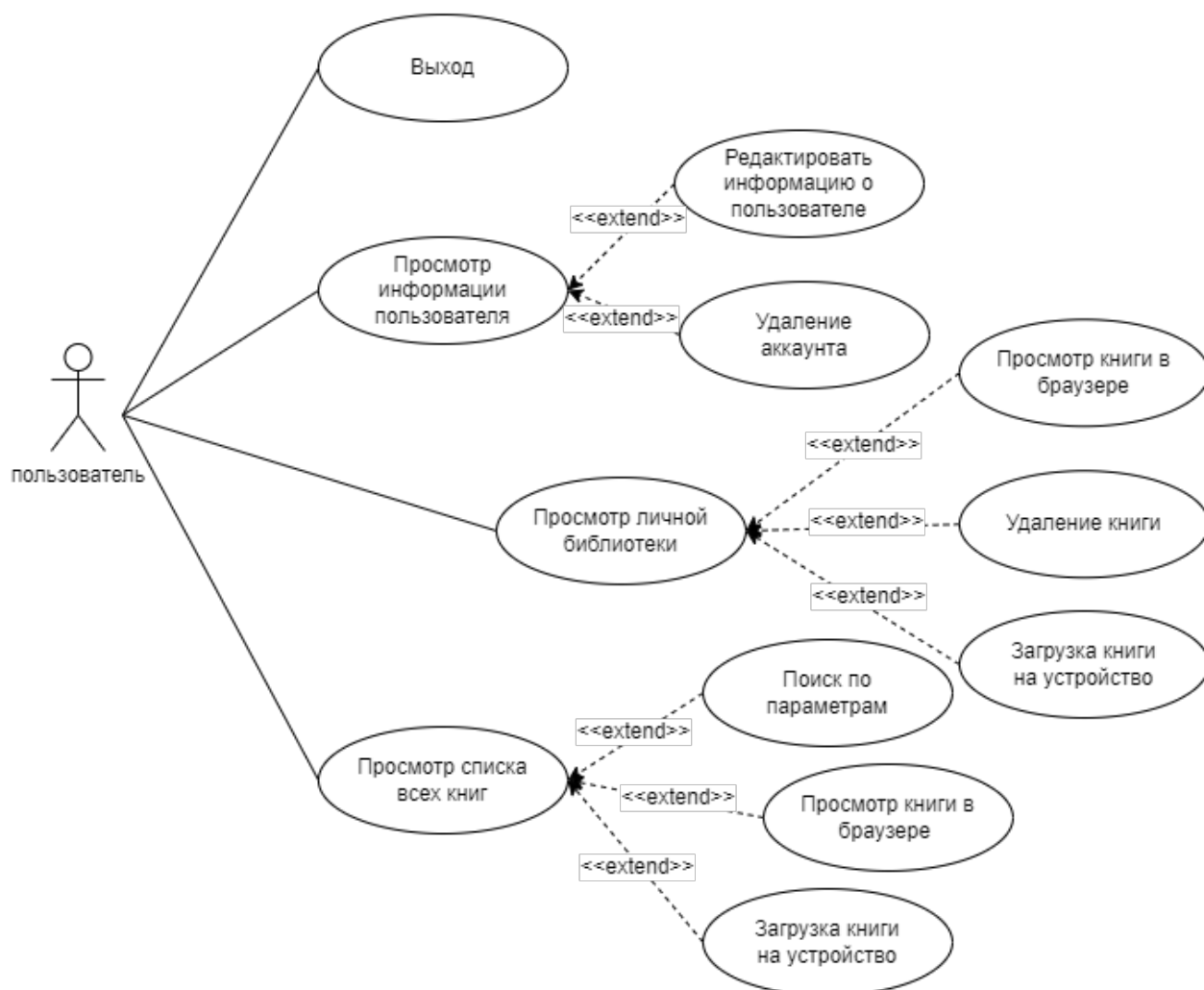


Рисунок 12 – UML диаграмма вариантов использования авторизованного пользователя

1.4.4 Основные компоненты

Реализуемое клиент-серверное приложение содержит на три основные пакета:

- *controller*;
- *service*;
- *model*.

Controller - это связующее звено между первыми двумя компонентами. Классы данного компонента получают данные, содержащиеся в запросе к серверу и передают их в модель для обработки и сохранения.

Service - набор классов, методы которых выполняют различные операции,

такие как чтение, запись или изменение данных в базе данных, обработка и анализ данных. Полученные данные *service* слой передает в *model*.

Model - набор классов, реализующих всю бизнес-логику приложения. Эти классы отвечают за обработку данных (сущностей), размещение их в БД, чтение из БД, а также за взаимодействие между самими объектами, составляющими такие данные.

1.4.4.1 *Controller*

Рассмотрим один из классов контроллеров на примере *MyLibController*. Данный класс отвечает за работу с моделью *Book*, которая получает и сохраняет данные о книгах в базе данных. Контроллер выбирает каким способом отобразить полученные данные пользователю. Для этого в нем реализованы следующие методы:

- *mylib()*;
- *uploadBook()*;
- *downloadBook()*;
- *getBook()*;
- *deleteBook()*.

Метод *mylib()* обрабатывает *get*-запрос, в котором производится поиск всех загруженных книг авторизованного пользователя. Для проверки авторизованного пользователя используется объект *Principal*, получая с его помощью пользователя *User*, происходит поиск книг, которые были загружены этим пользователем.

Ниже приведен листинг функции *mylib()*.

```
@GetMapping("/mylib")
public String myLib(Model model, Principal principal){
    model.addAttribute("mybooks",
bookService.findByUser(principal));
    model.addAttribute("user",
bookService.getUserByPrincipal(principal));
    return "mylib";
}
```

Метод *uploadBook()* обрабатывает *post*-запрос, который загружает книгу в базу данных. Для загрузки книги и определения какой пользователь совершает действие, также используется объект *Principal*, данные о книге контроллер получает с *html* страницы. Все полученные данные, контроллер передает в сервисный слой *BookService*, в нем данные используя модель *Book* сохраняется в базу данных. После успешного выполнения происходит перенаправление на *get*-метод *mylib()*.

Ниже приведен листинг функции *uploadBook ()*.

```
@PostMapping("/mylib/add")
@Transactional
public String uploadBook(@RequestParam("file") MultipartFile file,
Book book, Principal principal) throws IOException {
    bookService.addBook(principal, book, file);
    return "redirect:/mylib";
}
```

Метод *downloadBook()* представляет собой обработчик *get*-запроса, который отвечает за скачивание файла из приложения по его идентификатору. Аннотация *@ResponseBody* указывает на то, что данный метод будет возвращать тело ответа в виде массива байтов, которое будет передано клиенту для скачивания файла. Если файл не найден в базе данных, метод возвращает ошибку 404 с помощью метода *ResponseEntity.notFound().build()*. В случае, если файл найден, метод создает объект *ResponseEntity<byte[]>*, который содержит тело ответа в виде массива байтов файла, и заголовок, указывающий на то, что это прикрепленный файл (*attachment*), а также имя файла для скачивания.

Ниже приведен листинг функции *downloadBook()*.

```
@GetMapping("/download/{id}")
@ResponseBody
public ResponseEntity<byte[]> downloadBook(@PathVariable Long id) {
    Book file = bookService.getBook(id);
    if (file == null) {
        return ResponseEntity.notFound().build();
    }
}
```

```

    }
    return ResponseEntity.ok()
        .header(HttpHeaders.CONTENT_DISPOSITION, "attachment;
filename=\"\" + file.getOriginalNameFile() + \"\")
        .body(file.getBytes());
}

```

Метод *getBook()* представляет собой обработчик *get*-запроса, который отвечает за получение книги из библиотеки пользователя по её идентификатору. Функция по идентификатору книги ищет её в базе данных. Если книга найдена, метод получает её содержимое, которое представлено в виде *XML*-документа. Затем метод использует объект класса *Convertor*, чтобы преобразовать *XML*-документ в *html* формат.

Ниже приведен листинг функции *getBook()*.

```

@GetMapping("/mylib/{id}")
public ResponseEntity<String> getBook(@PathVariable Long id) throws
TransformerException {
    Optional<Book> bookOptional = bookRepository.findById(id);

    if (bookOptional.isPresent()) {
        Book book = bookOptional.get();

        String xml = new String(book.getBytes(),
StandardCharsets.UTF_8);
        String output = convertor.transform(xml);

        HttpHeaders headers = new HttpHeaders();
        headers.setContentType(MediaType.TEXT_HTML);

        return new ResponseEntity<>(output, headers, HttpStatus.OK);
    }
    else {
        return new ResponseEntity<>(HttpStatus.NOT_FOUND);
    }
}

```

```

    }
}

```

Метод *deleteBook()* является обработчиком *post*-запроса, который отвечает за удаление книги из библиотеки пользователя. Метод начинается с вызова сервисного слоя, который по идентификатору книги удаляет её из базы данных. После успешного удаления возвращает строку *redirect:/mylib*, которая указывает на то, нужно перенаправить пользователя на страницу *mylib*.

Ниже приведен листинг функции *deleteBook()*.

```

@PostMapping("/mylib/{id}/delete")
public String deleteBook(@PathVariable Long id){
    bookService.deleteBook(id);
    return "redirect:/mylib";
}

```

1.4.4.2 Service

Сервисный слой является посредниками между контроллерами и репозиториями, обрабатывая данные и выполняя операции над ними. В данном пункте будет рассмотрен такой класс, *BookService*. Он имеет следующие методы:

- *getBook()*;
- *findByUser()*;
- *addBook()*;
- *getUserByPrincipal()*;
- *deleteBook()*.

Метод *getBook()* принимает один параметр - *Long id*, который является идентификатором книги в базе данных. При вызове метода, он использует *bookRepository* для поиска книги в базе данных по указанному идентификатору. Если книга с указанным идентификатором найдена в базе данных, метод возвращает объект класса *Book*. Если же книга не найдена, метод возвращает *null*.

Ниже приведен листинг функции *getBook()*.

```
public Book getBook(Long id){
    return bookRepository.findById(id).orElse(null);
}
```

Метод *findByUser()* принимает параметр *Principal*, который представляет аутентифицированного пользователя в системе. Функция использует другой метод класса *UserService* - *getUserByPrincipal(principal)*, чтобы получить объект пользователя, который выполнил вход в систему на основе переданного параметра *principal*. Затем метод использует объект *bookRepository* для поиска книг, которые принадлежат указанному пользователю. Метод *findByUser* возвращает список книг, которые принадлежат указанному пользователю. Если у пользователя нет книг, метод вернет пустой список.

Ниже приведен листинг функции *findByUser()*.

```
public List<Book> findByUser(Principal principal){
    return bookRepository
        .findByUser(getUserByPrincipal(principal));
}
```

Метод *addBook()* принимает параметры: объект *Principal*, объект класса *Book*, который содержит информацию о добавляемой книге, и объект класса *MultipartFile*, который содержит информацию о загружаемом файле. В данной функции аннотация *@Transactional* используется для обозначения транзакционной операции. Транзакция - это последовательность операций, которые должны быть выполнены как единое целое. Если какая-либо операция не может быть выполнена, транзакция должна быть отменена, чтобы вернуть базу данных в исходное состояние.

Здесь операции сохранения книги в базу данных должны быть выполнены как единое целое. Поэтому, аннотация *@Transactional* используется, чтобы обеспечить корректное выполнение транзакционной операции. Внутри метода, сначала устанавливаются свойства книги, такие как байты файла, имя файла, тип файла, размер файла и пользователь, который добавил книгу. Затем метод *bookRepository.save(book)* используется для сохранения новой книги в базу данных.

Ниже приведен листинг функции *addBook()*.

```
@Transactional
public void addBook(Principal principal, Book book, MultipartFile
file) throws IOException {
    book.setBytes(file.getBytes());
    book.setOriginalNameFile(file.getOriginalFilename());
    book.setContentType(file.getContentType());
    book.setFileSize(file.getSize());
    book.setUser(getUserByPrincipal(principal));
    bookRepository.save(book);
}
```

Метод *deleteBook()* принимает параметр - *Long id*. При вызове метода, он использует *bookRepository* для удаления книги из базы данных по указанному идентификатору.

Ниже приведен листинг функции *deleteBook()*.

```
public void deleteBook(Long id){
    bookRepository.deleteById(id);
}
```

1.4.4.3 Model

Модель представляет собой сущность, которая содержит данные, необходимые для отображения на пользовательском интерфейсе. Она отвечает за хранение и управление данными, используемыми в приложении. Для сохранения данных модели в базе данных создан класс *Book*, который отображает таблицу *book* в *Postgresql*. Аннотация *@Entity* указывает, что класс является сущностью базы данных, а аннотация *@Table* указывает имя таблицы, которая отображается этой сущностью. Аннотация *@Id* указывает, что поле *id* является первичным ключом таблицы, *@Column* указывают имена столбцов таблицы, которые отображаются соответствующими полями класса. Аннотация *@ManyToOne* указывает на отношение между сущностями, в нашем случае - связь многие-к-одному между сущностями *Book* и *User*.

Ниже приведен листинг функции *Book.java*.

```

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

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id")
    private long id;

    @Column(name = "book_name")
    private String bookName;
    @Column(name = "author")
    private String author;
    @Column(name = "original_name_file")
    private String originalNameFile;
    @Column(name = "content_type")
    private String contentType;
    @Column(name = "file_size")
    private Long fileSize;
    @Lob
    @Type(type="org.hibernate.type.BinaryType")
    private byte[] bytes;
    @ManyToOne(cascade = CascadeType.REFRESH, fetch =
FetchType.LAZY)
    @JoinColumn
    private User user;

    public Book() {}

    public Book(long id,
                String bookName,
                String author,
                String originalNameFile,
                String contentType,
                Long fileSize,

```

```

        byte[] bytes,
        User user){

    this.id = id;
    this.bookName = bookName;
    this.author = author;
    this.originalNameFile = originalNameFile;
    this.contentType = contentType;
    this.fileSize = fileSize;
    this.bytes = bytes;
    this.user = user;
}

//...геттеры и сеттеры
}

```

1.4.5 Основные алгоритмы взаимодействия клиента и сервера

Для наглядности алгоритмов взаимодействий были разработаны *UML* диаграммы деятельности, которые отображают последовательность действий между клиентской частью и серверной.

Первым рассматриваемым алгоритмом будет регистрация и авторизация. Для использования всех возможностей, предлагаемых приложением, необходимо выполнить регистрацию и дальнейший вход в систему. При попытке пользователем, не осуществившем вход, использовать функцию, которая доступна только авторизованным пользователям, ему будет предложено войти в приложение. Если клиент не имеет аккаунта в приложении, он может выполнить регистрацию, перейдя на соответствующую страницу и заполнив необходимые поля. Веденные данные отправляются на сервер, где происходит обработка и проверка уникальности информации, если такие данные уже занесены с БД пользователю будет отправлено соответствующее оповещение об этом. В противном случае данные будут сохранены в БД и пользователь будет перенаправлен на страницу авторизации.

Если при выполнении авторизации, введенные данные будут отличны от тех, что хранятся в БД, пользователю будет выведено сообщение о некорректных данных. В случае правильно отправленных данных, будет выполнен вход в приложение, после которого у пользователя будет полный доступ ко всем функциям приложения.

Последовательность описанных действий представлена на рисунке 13.

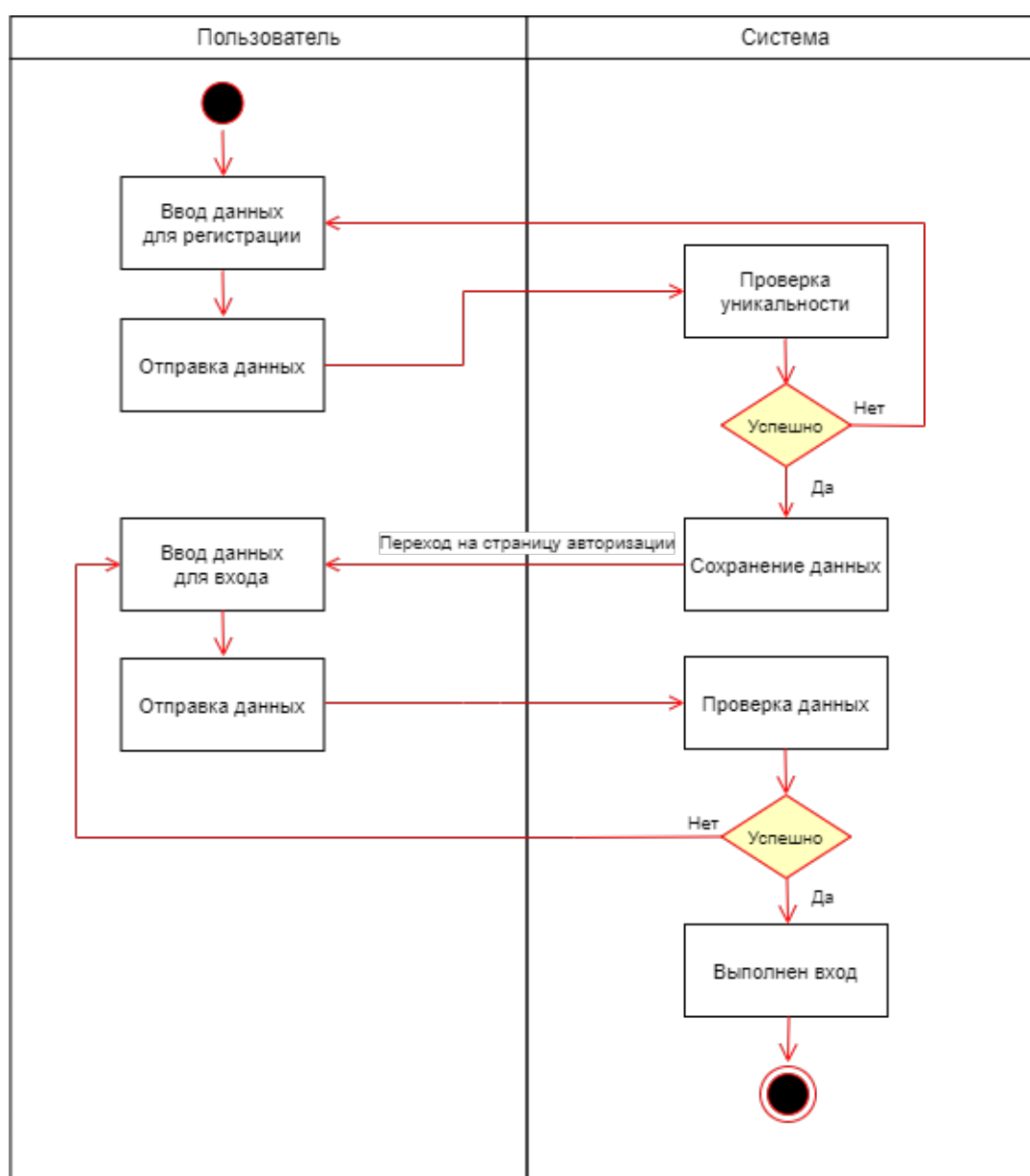


Рисунок 13 – UML диаграмма деятельности регистрации и авторизации

Вторым алгоритмом, который будет рассмотрен, это процесс просмотра списка добавленных книг пользователем, а также непосредственно добавление

книг в личную библиотеку. Просмотр добавленных книг осуществляется пользователем путем перехода на страницу «Моя библиотека». При переходе, системой проходит проверка авторизации пользователя в приложении, если он не выполнил вход, система перенаправляет его на страницу входа и после завершения авторизации, он может перейти на страницу личной библиотеки.

На этой странице проверяется наличие загруженных пользователем в БД книг. Если у него отсутствуют личные книги, он имеет возможность добавить их со своего устройства в приложение.

Для добавления книги пользователь должен ввести всю необходимую информацию и выбрать загружаемый файл с устройства. После выполнения этих действий в системе происходит валидация данных и если проверка проходит успешно, то книга сохраняется в БД, а пользователя перенаправляет обратно в личную библиотеку, где снова проходит проверка на наличие книг и будет показана добавленная книга. В том случае если данные не прошли валидацию, пользователь должен ввести корректную информацию и повторить попытку добавления.

Последовательность действий по просмотру личной библиотеки и добавлению книг представлены на рисунке 14.

Еще один из основных алгоритмов, который будет описан в этом пункте, это алгоритм действий, выполняемых над добавленной книгой. После того, как книга загружена в БД, и она отображается в приложении, с ней можно производить такие действия, как просмотр содержания книги в браузере, скачивание ее из БД на устройство и удаление книги.

Для просмотра, пользователю нужно выбрать книгу, совершить необходимые действия на сайте, после чего в системе будет произведен поиск, выбранной книги. Если книга найдена, будет сформирован ответ, ее содержимое будет показано пользователю в браузере. В противном случае пользователь получит сообщение об ошибке и ему будет предложено выбрать другую книгу.

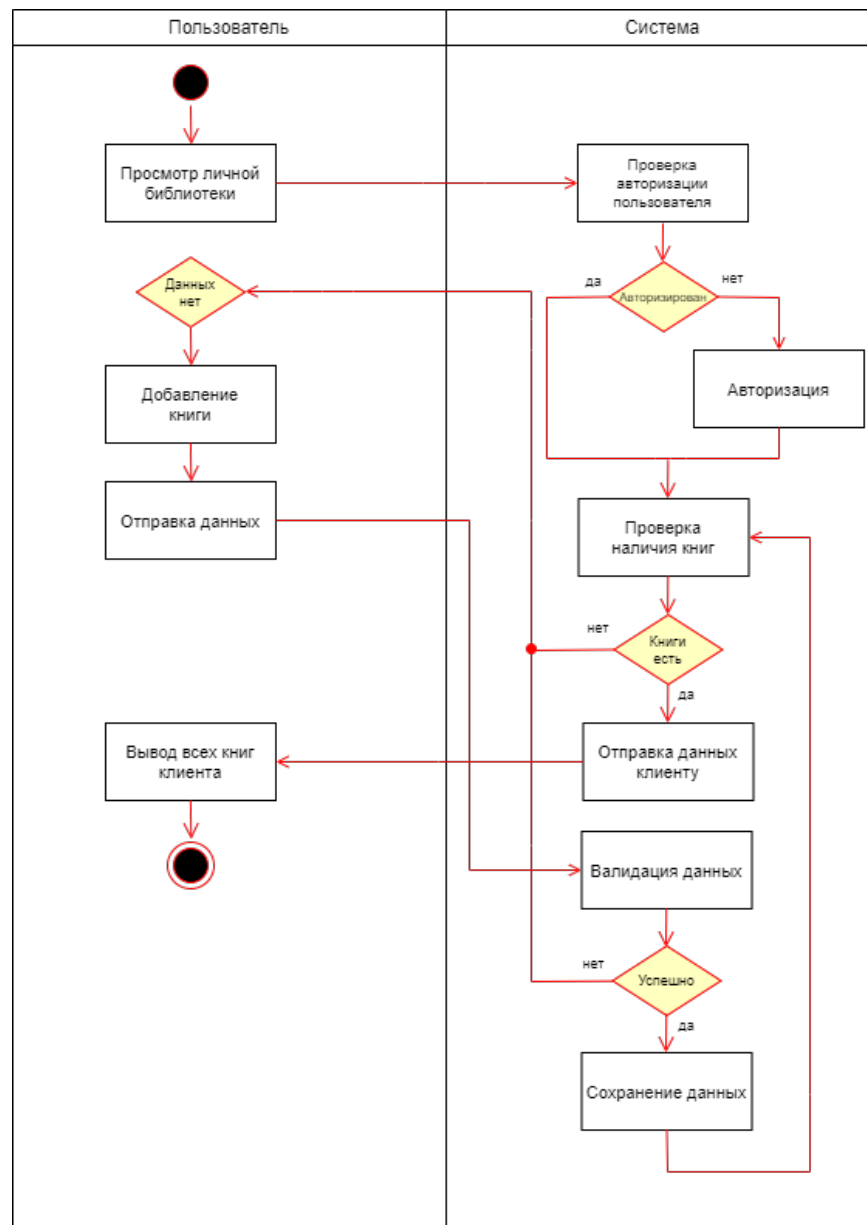


Рисунок 14 – *UML* диаграмма деятельности просмотра личной библиотеки и добавления книг

При скачивании книги на устройство, в системе также производится ее поиск и, если поиск прошел успешно сервером формируется ответ, после чего происходит скачивание. В том случае, когда книга не была найдена, пользователю будет отправлено сообщение об ошибке и пользователь может выбрать другую книгу для скачивания. Если пользователь желает удалить книгу из БД, ему необходимо подтвердить свои действия, после чего серверу будет направлен запрос на

удаление. Сервер также ищет книгу в БД, если книга найдена, она удаляется и пользователю будет показан обновленный список.

На рисунке 15 представлена диаграмма деятельности работы с книгой.

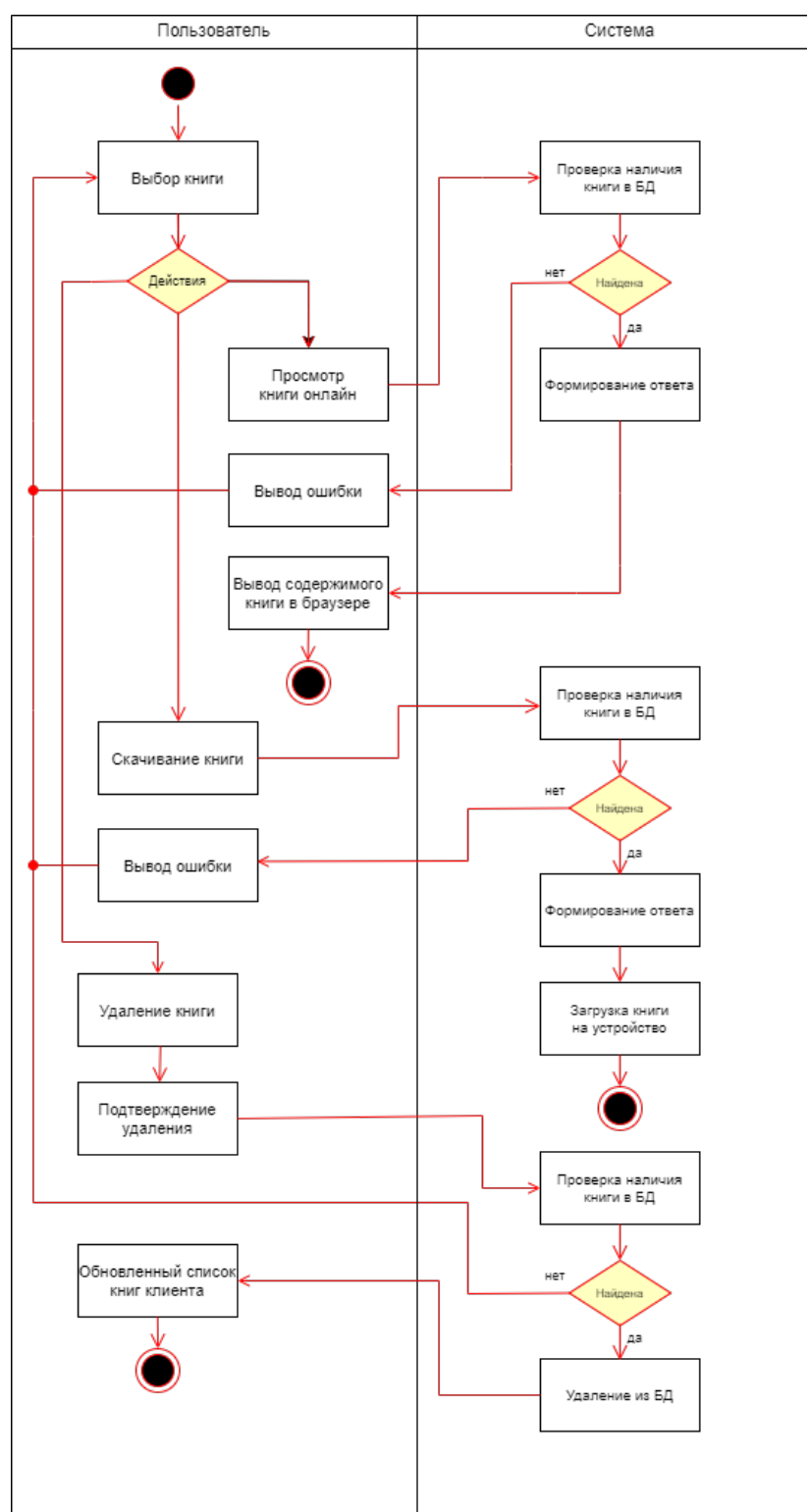


Рисунок 15 – UML диаграмма деятельности работы с книгой

1.4.6 Тестирование приложения

Для проверки работоспособности клиент-серверного приложения «Библиотека» необходимо провести его тестирование.

Тестирование будет состоять из следующих действий:

- регистрация пользователя;
- авторизация пользователя;
- просмотр, загрузка, скачивание и удаление книги;
- просмотр книг других пользователей;
- редактирование и удаление аккаунта пользователя;
- проверка функционала неавторизованных пользователей.

1.4.6.1 Регистрация пользователя

Для проведения тестирования регистрации пользователя, необходимо перейти на страницу с формой регистрацией, и ввести следующие запрашиваемые данные: имя и фамилию пользователя, логин и пароль. После введения данных и выполнения запроса на добавление пользователя, в базе данных должна появиться запись о новом пользователе.

Проверка начального состояния сущности users в базе данных представлена на рисунке 16.

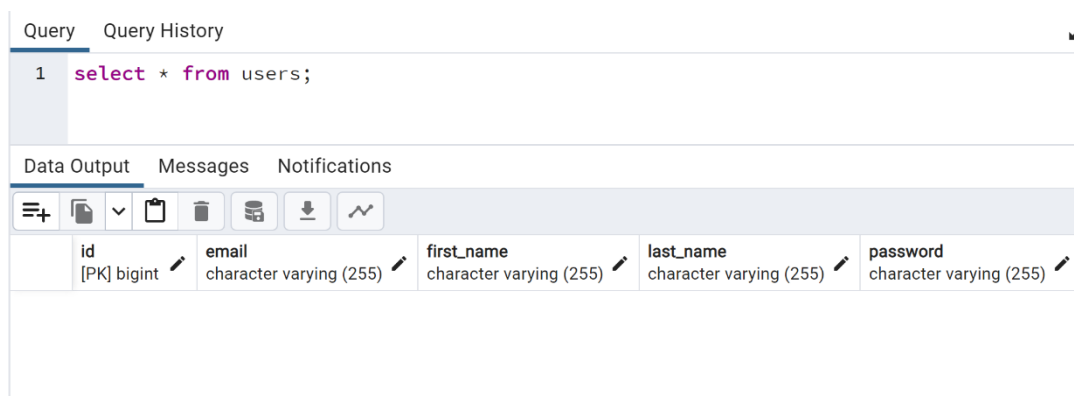


Рисунок 16 – База данных до добавления пользователя

Ввод данных в форму для регистрации приведен на рисунке 17.



Регистрация

Имя Даниил
Фамилия Серов
Логин serov@mail.ru
Пароль ••••
Регистрация

Рисунок 17 – Регистрация пользователя

Проверка состояния сущности *users* в базе данных после регистрации представлена на рисунке 18.

The screenshot shows a database query interface with a 'Query History' tab. The query entered is 'select * from users;'. The 'Output' tab is active, displaying a table with 5 columns: id, email, first_name, last_name, and password. The first row contains the data for the user 'Даниил Серов' with email 'serov@mail.ru' and a long password hash.

id	email	first_name	last_name	password
1	serov@mail.ru	Даниил	Серов	\$2a\$08\$D7Wr6H6Wp4thBc28YxuGuhww8BUkb51/FXPBdkUSBLUOnZ9Apc...

Рисунок 18 – База данных после добавления пользователя

После проведения регистрации, мы видим, что данные введенные на сайте успешно были сохранены в БД.

Если попытаться зарегистрировать пользователя с логином, который уже существует в БД, пользователь получит сообщение об этом и будет предложено ввести корректные данные. Попытка регистрации с логином, который уже есть в БД, и вывод соответствующего сообщения представлены на рисунках 19 и 20.



Регистрация

Имя
Максим

Фамилия
Максимов

Логин
serov@mail.ru

Пароль

Регистрация

Рисунок 19 – Регистрация пользователя с существующим логином в БД



Регистрация

Имя

Фамилия

Логин

Пароль

Регистрация

Пользователь с логином: serov@mail.ru
уже существует

Рисунок 20 – Вывод сообщения

1.4.6.2 Авторизация пользователя

При авторизации пользователь вводит ранее добавленные данные в БД. Если введенные данные и данные, которые хранятся в базе данных совпадают, авторизация проходит успешно. В противном случае пользователю будет показано сообщение об ошибке.

На рисунке 21 представлен сценарий тестирования с ошибочными данными, результат входа с неправильными данными показан на рисунке 22.

Вход

Логин
login

Пароль
•

Вход

Регистрация

Рисунок 21 – Ввод неправильных данных

Вход

Логин

Пароль

Неправильный логин или пароль

Вход

Регистрация

Рисунок 22 – Сообщение об ошибке

Далее введем достоверные данные. На рисунке 23 представлен сценарий тестирования с достоверными данными, результат входа с верными данными показан на рисунке 24.

Вход

Логин
 serov@mail.ru

Пароль

Неправильный логин или пароль

Вход

Регистрация

Рисунок 23 – Ввод верных данных

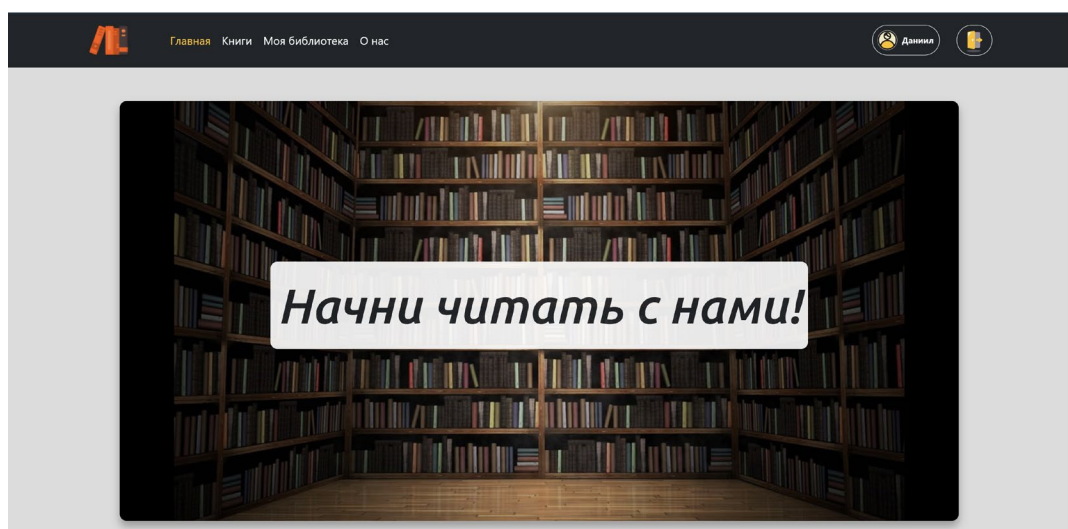


Рисунок 24 – Результат авторизации

В ходе тестирования все сценарии прошли успешно. При неправильных данных пользователю была выведена ошибка. С верно введенными данными произошел переход на главную страницу где отображено имя пользователя, который выполнил вход.

1.4.6.3 Тестирование функционала взаимодействия с книгами

Авторизованным пользователям доступен функционал чтения книги онлайн, загрузки, скачивания и удаления книги. Тестовый сценарий состоит из перехода на станицу «Личной библиотеки», где будет добавлена книга в базу данных, после чего можно будет читать ее онлайн. Также есть функционал для скачивания книги из базы данных на устройство пользователя.

На рисунке 25 отображено добавление книги в базу данных.

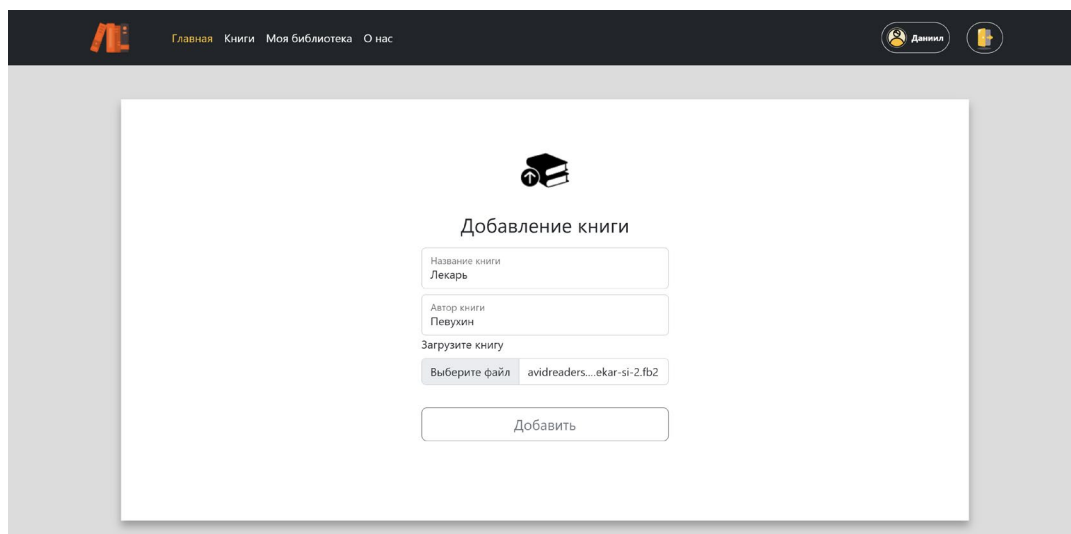


Рисунок 25 – Загрузка книги в базу данных

На рисунке 26 представлен результат добавления книги в базу данных.

[Главная](#) [Книги](#) [Моя библиотека](#) [О нас](#)

Даниил

Личная библиотека: Даниил

Добавить

ID	Название	Автор	Файл	Размер	Действие
4	Лекарь	Певухин	application/octet-stream	804012	<div><div></div><div></div><div></div></div>

Рисунок 26 – Результат загрузки книги в базу данных

После загрузки книги есть возможность читать книгу онлайн. На рисунке 27 показано просмотра книги в браузере.

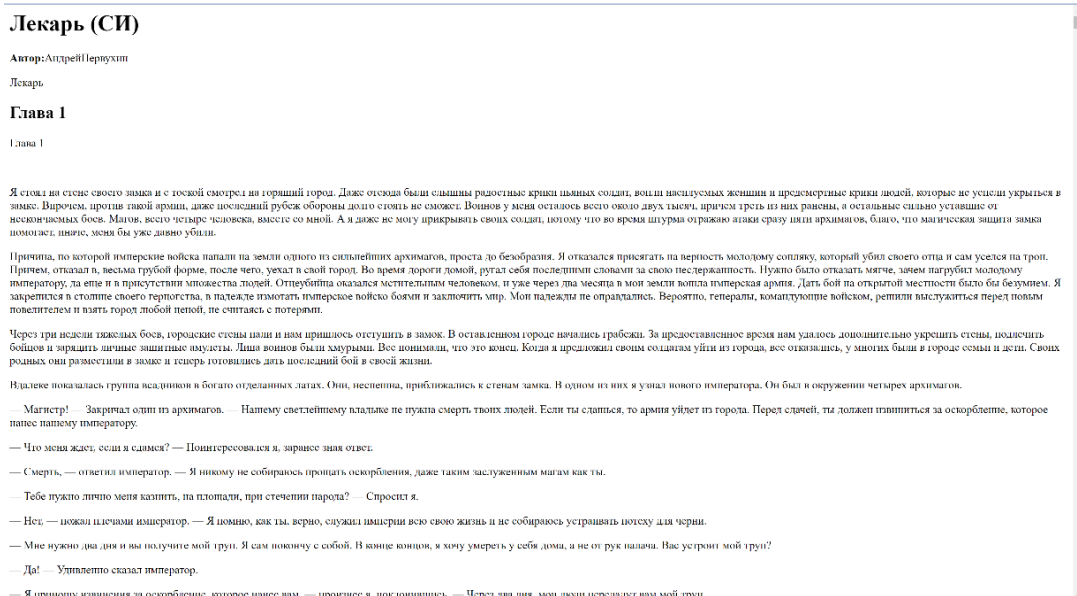


Рисунок 27 – Вывод содержимого книги в браузере

После нажатие на кнопку скачивания, пользователь загрузил книгу из базы данных на свое устройство. На рисунке 28 представлен результат скачивания.

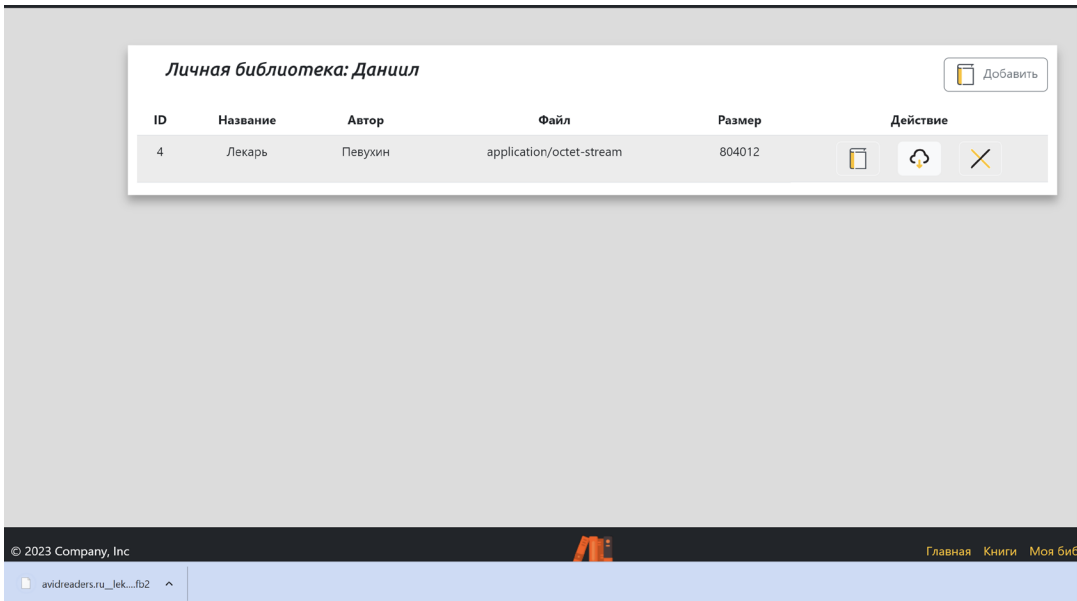


Рисунок 28 – Результат скачивания

Для того, чтобы убедиться в возможности просматривать книги всех пользователей необходимо зарегистрировать в системе еще один аккаунт и добавить в его личную библиотеку книгу. Результат регистрации нового аккаунта представлен на рисунке 29.

1

select * from users;

Data Output

Messages

Notifications

	id [PK] bigint	email character varying (255)	first_name character varying (255)	last_name character varying (255)	password character varying (255)
1	1	serov@mail.ru	Даниил	Серов	\$2a\$08\$мPJbSB3H1AYhj9g14.TRJum/MJlCHBKvdnrHVcfWPcngSZ...
2	3	egorov@mail.ru	Егор	Егоров	\$2a\$08\$FMfj9K6s22fyIKIXmgfE.gSgdGP/Rj.DjH4340hy1vYIN/1taT...

Рисунок 29 – Все пользователи в БД

Все загруженные книги, которые хранятся в базе данных отображены на рисунке 30.

Query

Query History

Scratch Pad

1

select * from book;

Data Output

Messages

Notifications

	id [PK] bigint	author character varying (255)	book_name character varying (255)	bytes bytea	content_type character varying (255)	file_size bigint	original_name_file character varying (255)	user_id bigint
1	4	Певухин	Лекарь	[binary dat...	application/octet-stream	804012	avidreaders.ru_lekar-si-2.fb2	1
2	5	Булгаков	Мастер и Маргарита	[binary dat...	application/octet-stream	2124503	avidreaders.ru_master-i-margarita.fb2	3

Рисунок 30 – Все книги в БД

Для просмотра всех книг, которые есть в приложении, пользователь должен перейти на страницу «Книги». На ней отображены все книги, которые хранятся в БД, с указанием владельцев и возможностью просмотра и скачивания данных книг.

На рисунке 31 представлен список всех доступных книг разных пользователей.

<div> Главная Книги Моя Библиотека О нас </div> <div> Егор </div>					
Всегда найдется, что почитать!					
<div> <input type="text" value="Введите название книг"/> <input type="button" value="Найти"/> </div>					
ID	Название	Автор	Владелец	Размер	Действие
4	Лекарь	Певухин	Даниил Серов	804012	
5	Мастер и Маргарита	Булгаков	Егор Егоров	2124503	

Рисунок 31 – Просмотр книг разных пользователей

Удаления книги происходит после нажатия на кнопку «Удалить» и подтверждения своих действий. Результат удаления показаны на рисунках 32, 33 и 34.

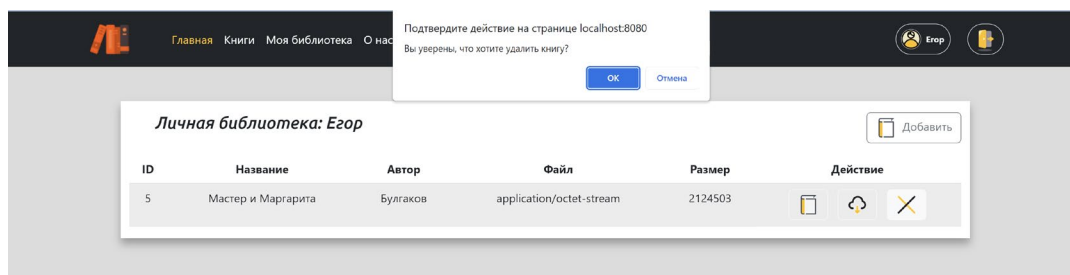


Рисунок 32 – Нажатия на кнопку удалить

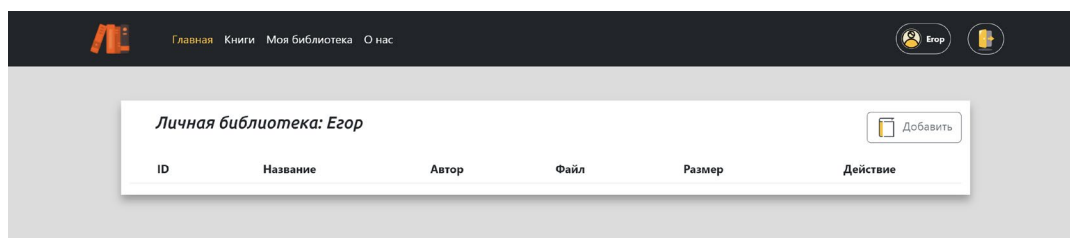


Рисунок 33 – Результат удаления на сайте

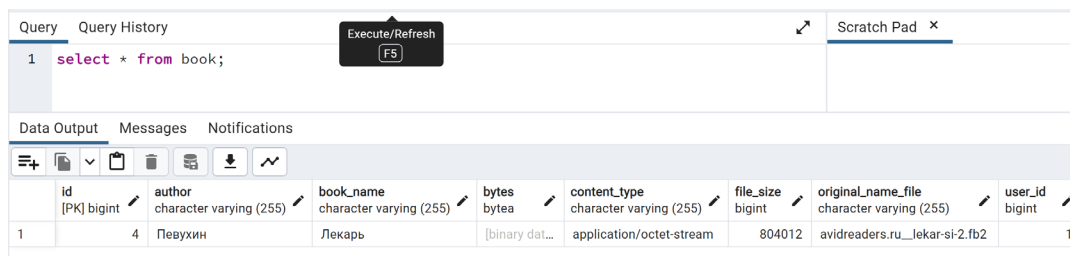


Рисунок 34 – Результат удаления в БД

После проделанных действий, ожидаемый результат совпал с действительным, тестирование данных функций прошли успешно.

1.4.6.4 Редактирование и удаление пользователя

Пользователь может обновить свои данные после перехода в профиль, изменения нужных данных и нажатия кнопки изменить.

Результат редактирования личных данных представлены на рисунках 35-38.

Query

Query History

1

select * from users;

Scratch Pad

Data Output

Messages

Notifications

	id [PK] bigint	email character varying (255)	first_name character varying (255)	last_name character varying (255)	password character varying (255)
1	1	serov@mail.ru	Даниил	Серов	\$2a\$08\$mPjSB3H1AYhj9g14.TRJum/MJICHBKvdnrHVcfWPcngSZ...
2	3	egorov@mail.ru	Егор	Егоров	\$2a\$08\$FMfj9K6s22fYIKlIXmgfE.gSGdGP/Rj.DjH4340hy1vYIN/1taT...

Рисунок 35 – БД до редактирования информации

Главная Книги Моя библиотека О нас

Даниил

Редактирование

Имя
Максим

Фамилия
Максимов

Логин
serov@mail.ru

Новый пароль

Изменить

Удалить аккаунт

Рисунок 36 – Изменение данных в аккаунте «Даниил»

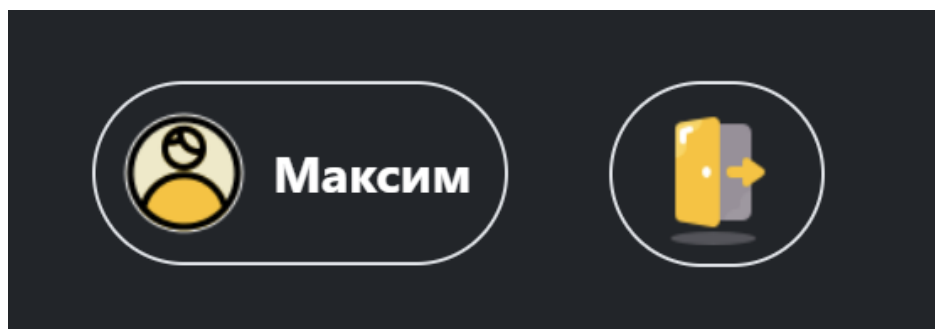


Рисунок 37 – Результат редактирования

Query

Query History

1

select * from users;

Execute/Refresh

F5

Scratch Pad

x

Data Output

Messages

Notifications

	id [PK] bigint	email character varying (255)	first_name character varying (255)	last_name character varying (255)	password character varying (255)
1	3	egorov@mail.ru	Егор	Егоров	\$2a\$08\$FMfj9K6s22fYIKlIXmgfE.gSgdGP/Rj.DjH4340hy1vYIN/1taTUy
2	1	serov@mail.ru	Максим	Максимов	\$2a\$08\$FQTuXdmOPCXZFkxs6i8VOZKcaw8vIDmlF3Mtt/XzuzpvCewasZ...

Рисунок 38 – Результат редактирования

Удаление авторизованного пользователя происходит на той же странице где и редактирование. Для удаления необходимо подтвердить действия.

Результат удаления представлены на рисунках 39 и 40.

Рисунок 39 – Удаление аккаунта

id	email	first_name	last_name	password
3	egorov@mail.ru	Egor	Egorov	\$2a\$08\$FMfj9K6s22fYiKIXmgfE.gSgdGP/Rj.DjH4340hy1vYIN/1taT...

Рисунок 40 –БД после удаления аккаунта

1.4.6.5 Функционал неавторизованного пользователя

Пользователь, который не выполнил вход может просматривать только страницы «Главная», «Книги», «О нас». На странице «Книги» ограниченный функционал, он может только видеть список книг других пользователей, но не может просматривать их содержимое и скачивать их на устройство. При попытке нажать на кнопку «Читать» или «Скачать» на экран выводится оповещение, которое информирует о необходимости авторизации или регистрации для использования этих функций.

На рисунках 41 и 42 представлены результаты вывода оповещений для неавторизованного пользователя.

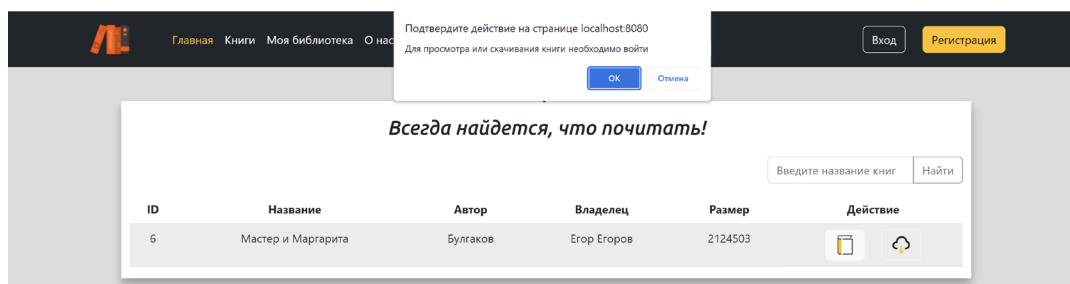


Рисунок 41 – Оповещение при нажатии на кнопку «Читать»

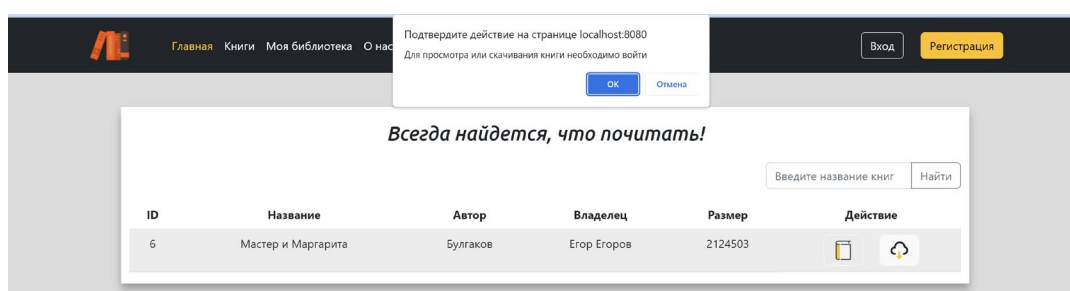


Рисунок 42 – Оповещение при нажатии на кнопку «Скачать»

Также при попытке, неавторизованным пользователем, перейти на страницу «Моя библиотека», он будет перенаправлен на страницу входа, на которой он может либо авторизоваться, либо выполнить регистрацию.

Результат попытки совершить переход на страницу, которая не входит в функционал неавторизованного пользователя, представлена на рисунке 43.

Name	Status	Type	Initiator
mylib	302	document / Redirect	Other
login	200	document	mylib
bootstrap.min.css	200	stylesheet	login

Рисунок 43 – Результат перехода на страницу «Моя библиотека» без авторизации

2 Безопасность жизнедеятельности

Безопасность жизнедеятельности при работе за компьютером является важным фактором для разработчиков и других профессионалов, которые проводят много времени за компьютером. Несоблюдение мер безопасности может привести к различным проблемам со здоровьем, включая напряжение глаз, нарушения осанки, повышенный риск развития карпального туннельного синдрома и других заболеваний.

В данном разделе будут рассмотрены основные аспекты безопасности жизнедеятельности и рекомендации при работе за компьютером для разработчиков.

2.1 Требования к помещению и рабочему месту

При выборе рабочего места, оборудованного компьютером, предпочтение следует отдавать помещениям, ориентированным на север или северо-восток. Площадь на одно рабочее место с ПК с жидкокристаллическими мониторами должна составлять не менее - 4,5 м. кв. Помещения, предусмотренные для работы с ПК, обязательно должны иметь искусственное и естественное освещение. Должна быть достаточная вентиляция, чтобы обеспечить свежий воздух и уменьшить концентрацию вредных веществ, таких как формальдегид, которые могут выделяться из компьютера. Содержание вредных химических веществ в воздухе таких помещений не должно превышать предельно допустимых концентраций вредных веществ в воздухе рабочей зоны в соответствии с действующими гигиеническими нормативами. Рекомендуемый уровень шума в помещении не должен превышать 50 дБ, чтобы избежать отвлекающих звуков и повышенного стресса [6].

Помещение должно быть оборудовано защитным заземлением. Поскольку одним из вредных факторов влияния на человека при использовании компьютера являются электромагнитные излучения, создаваемые монитором, клавиатурой, соединительными кабелями. Превышение нормы такого излучения случается при

использовании устаревших моделей ЭВМ, но наиболее частая причина – неправильно выполненное подключение, отсутствие или некачественное выполнение заземления. При использовании компьютера в таком помещении особенно в сочетании с высоким зрительным и нервно-эмоциональным напряжением, вызывает существенные изменения со стороны центральной нервной и сердечно-сосудистой системы (головные боли, раздражительность, нарушение сна, неадекватные психические реакции).

Рабочий стол должен размещаться таким образом, чтобы монитор был ориентирован боковой стороной к окнам, а естественный свет падал преимущественно слева. В целях защиты от прямых солнечных лучей на окнах должны быть солнцезащитные жалюзи или однотонные шторы из плотной ткани.

Также должны соблюдаться следующие требования:

- температура 22 °С (допустимая – 20-24 °С);
- оптимальная относительная влажность – 40-60% (допустимая – не более 75%);
- скорость движения воздуха не более 0.1 м/с.

2.2 Требования к освещению

Соблюдение норм освещенности при работе за ПК это необходимое условие, так как недостаточное или неправильное освещение может привести к различным проблемам со здоровьем. Можно выделить следующую часто возникающую проблему это напряжение глаз, при ее наличии увеличивается усталость, что приводит к понижению производительности. С другой стороны, избыточное освещение также вызывает усталость глаз и головную боль.

Важно понимать, что освещение соответствует не только общей яркости, но также и равномерности распределения света в помещении. Если свет распределен неправильно, то это приводит к неравномерной яркости на поверхности рабочего стола, бликам на экране монитора и другим проблемам. Для предотвращения этих

проблем в помещении должно быть, как искусственное освещение, так и естественное.

Искусственное освещение в помещениях для эксплуатации ПК должно осуществляться системой общего равномерного освещения, такой системой могут служить люстры и потолочные светильники. При необходимости к общему освещению дополнительно можно устанавливать светильники местного освещения. В качестве источников света при искусственном освещении могут использоваться лампы накаливания или люминесцентные лампы типа желтого свечения. Лампы голубого свечения не используются. Не следует использовать в одном помещении люминесцентные лампы и лампы накаливания. Люминесцентные лампы должны быть защищены экранирующими решетками или рассеивателями для предотвращения слепящего действия ламп на глаза и возникновения бликов на поверхности экранов мониторов.

Рекомендации по требованиям к освещению при работе с ПК:

- общая освещенность была на уровне 300 лк;
- яркость на поверхности рабочего стола была равномерной и не превышала 500 лк;
- рабочее место должно находиться возле окна, чтобы использовать естественный свет;
- цветовая температура искусственного освещения должна быть в диапазоне от 4000 до 6500 К, так как это диапазон наиболее близкий к естественному свету.

2.3 Требования к режиму труда и отдыха

Требования к режиму труда и отдыха при работе с ПК условно могут быть разделены на три категории. Эти категории включают в себя физические, психологические и организационные требования.

Физические требования:

- правильное положение тела при работе с ПК;
- регулярные перерывы для растяжки и разминки;

- установка рабочего места на оптимальной высоте и расстоянии от глаз;
- правильное освещение рабочего места.

Психологические требования:

- регулярные перерывы для предотвращения усталости и переутомления;
- управление стрессом и снятие напряжения.

Организационные требования:

- обеспечение комфортной температуры и влажности в помещении;
- продуманный график рабочего времени;
- отдых в выходные и праздничные дни.

Рекомендуется делать перерывы каждые 60-90 минут работы и на 10-15 минут выйти из помещения или сделать небольшую физическую зарядку. Соблюдение этих требований снижает риск возникновения усталости и уменьшает уровень стресса. Правильный режим труда и отдыха может также повысить производительность и эффективность работы.

2.4 Рекомендации по использованию программного обеспечения

Программное обеспечение, которое помогает программисту следить за требованиями по обеспечению безопасного использования компьютера, предотвращает напряжение глаз, несоблюдение режима работы и другие проблемы, связанные с работой за ПК.

Рекомендуется использовать следующее программное обеспечение для улучшения безопасности жизнедеятельности:

- фильтры синего света. Программы, которые фильтруют синий свет, могут помочь уменьшить напряжение глаз, связанное с работой за компьютером. Некоторые из таких программ: *Night Shift* (для *Mac*), *Night Light* (для *Windows 10*);
- приложения для управления временем. Они помогают управлять временем, тем самым повышая производительность. Также эти приложения, могут уведомлять о необходимости делать перерывы, что помогает уменьшить усталость

глаз и предотвратить перенапряжение мышц. Некоторые из таких программ: *Workrave* или *EyeLeo*;

– программы для управления шрифтами. Позволяют управлять шрифтами, могут помочь уменьшить усталость глаз при чтении. Некоторые из таких программ: *ClearType Tuner* (для *Windows*), *HiDPI Renderer* (для *Mac*);

– программы для управления звуком. Могут помочь уменьшить стресс и улучшить настроение. Некоторые из таких программ: *Noisli*, *Rainy Mood*;

– программы управления задачами. Такие как *Trello* или *ToDoist*, помогают планировать работу и управлять задачами, что может снизить уровень стресса и повысить производительность.

Таким образом, использование выше перечисленного программного обеспечения обеспечит комфортные и безопасные условия работы за компьютером.

Заключение

В ходе написания данной выпускной квалификационной работы были выполнены следующие задачи: анализ актуальности темы, анализ и выбор среды разработки и инструментов, а также анализ архитектуры приложения, разработано клиент серверное приложение «Библиотека», разработана база данных, выполнено тестирование приложения.

Были получены навыки разработки *WEB*-приложений с использованием фреймворка *Spring boot* на языке *Java*. Также улучшены навыки работы с системой управления базами данных *PostgreSQL*. Проведенные тестирование и оценка функционала показали, что клиент серверное приложение «Библиотека» полностью соответствует всем поставленным требованиям и работает корректно.

В дальнейшем приложение можно усовершенствовать, добавив функционал администратора с целью обеспечения большей безопасности пользователей, а также усовершенствовать интерфейс клиентской части приложения для более комфортного использования.

Опыт, полученный в ходе решения поставленных задач и разработке клиент серверного приложения, однозначно поможет в дальнейшем трудоустройстве и развитии себя.

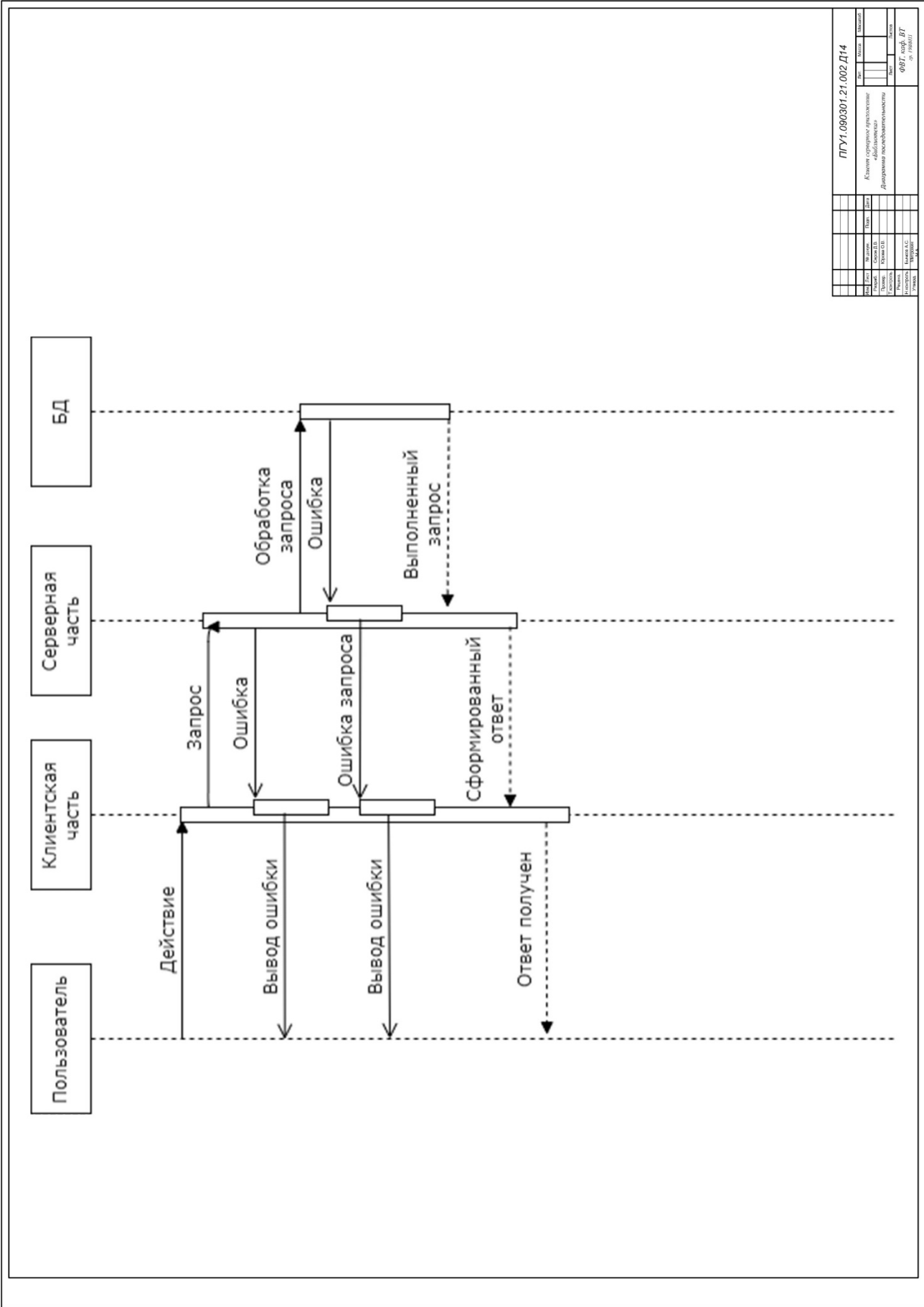
Список используемых источников

1. Клиент-серверная архитектура. [Электронный ресурс] / Режим доступа: https://ru.hexlet.io/courses/internet-fundamentals/lessons/client-server/theory_unit, свободный. (дата обращения 26.04.2023).
2. Марк Массе. Свод правил проектирования *REST API*. - 1-е изд. О'Рейли Медиа, 2011. - 5 с.
3. Документация к фреймворку *Spring Framework*. [Электронный ресурс] / Режим доступа: <https://docs.spring.io/spring-framework/docs/4.2.x/spring-framework-reference/html>, свободный. (дата обращения 04.05.2023).
4. Сысолетин Е. Г. Проектирование интернет-приложений : учеб.- метод. пособие / Е. Г. Сысолетин, С. Д. Ростунцев. — Екатеринбург : Изд-во Урал.ун-та, 2015. — 41 с.
5. Ржеуцкая С.Ю. Базы данных. Язык *SQL*: учеб. пособие / С.Ю. Ржеуцкая. – Вологда: ВоГТУ, 2010. – 17-22 с.
6. Санитарно-гигиенические рекомендации для организации рабочего места пользователя персональным компьютером. [Электронный ресурс] / Режим доступа: <https://sakhgig.ru/main/info/664-sanitarno-gigienicheskie-rekomendacii-dlya-organizacii-rabochego-mesta-polzovatelya-personalnym-kompyuterom.html>, свободный. (дата обращения 30.05.2023).

Приложение А
(Обязательное)
Клиент серверное приложение «Библиотека»
UML-диаграммы

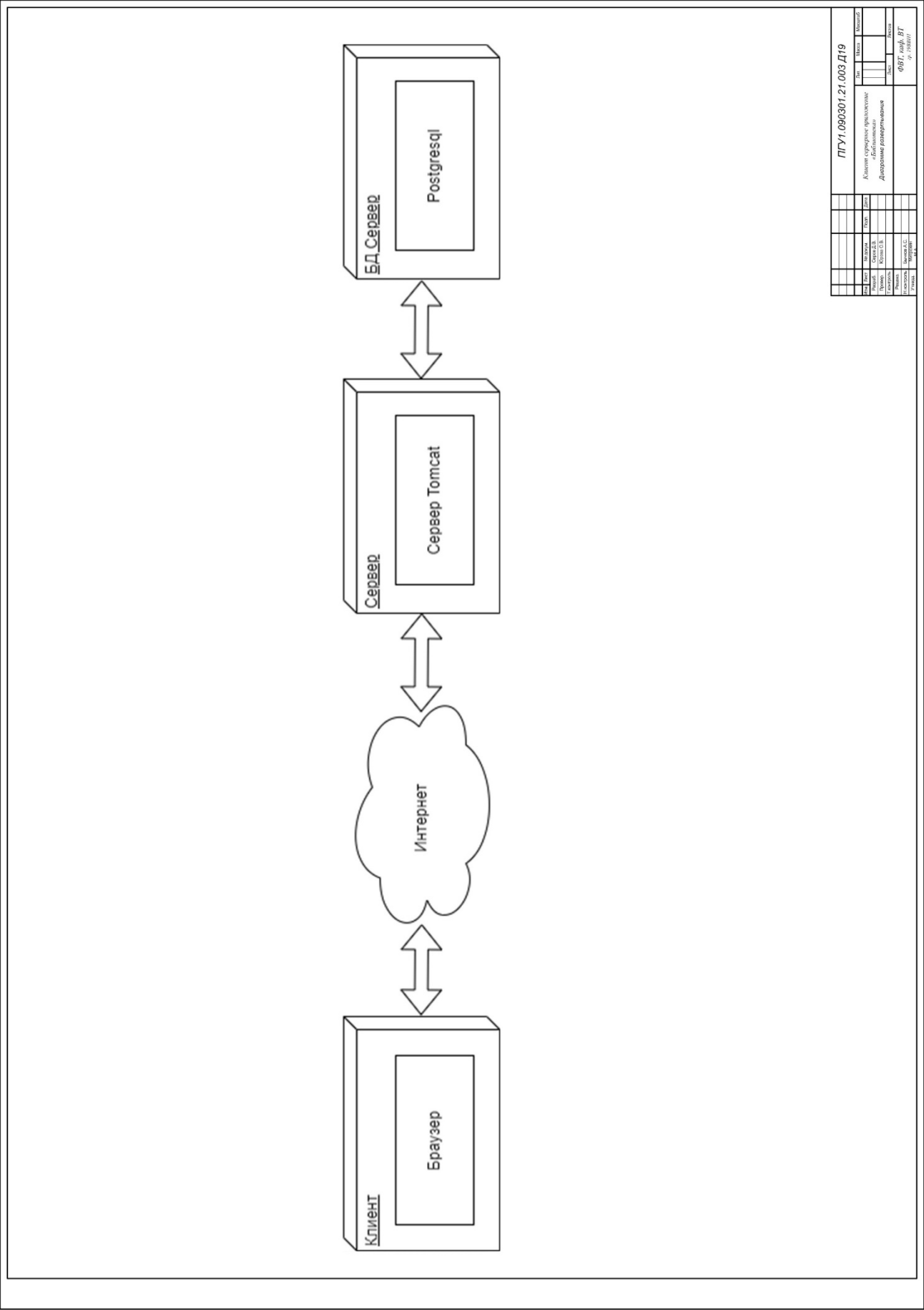
					<i>ПГУ1.090301.21.001 ПЗ</i>	<i>Лист</i>
						<i>61</i>
<i>Изм.</i>	<i>Лист</i>	<i>№ докум.</i>	<i>Подпись</i>	<i>Дата</i>		

Приложение А.1 - Диаграмма последовательностей



ПГУ1.090301.21.002 Д14									
Классификация информации									
Диаграмма последовательности									
Изм.	Лист	№ докум.	Подпись	Дата	Изм.	Лист	№ докум.	Подпись	Дата
ФОРМ. ИТ									
в (милл.)									

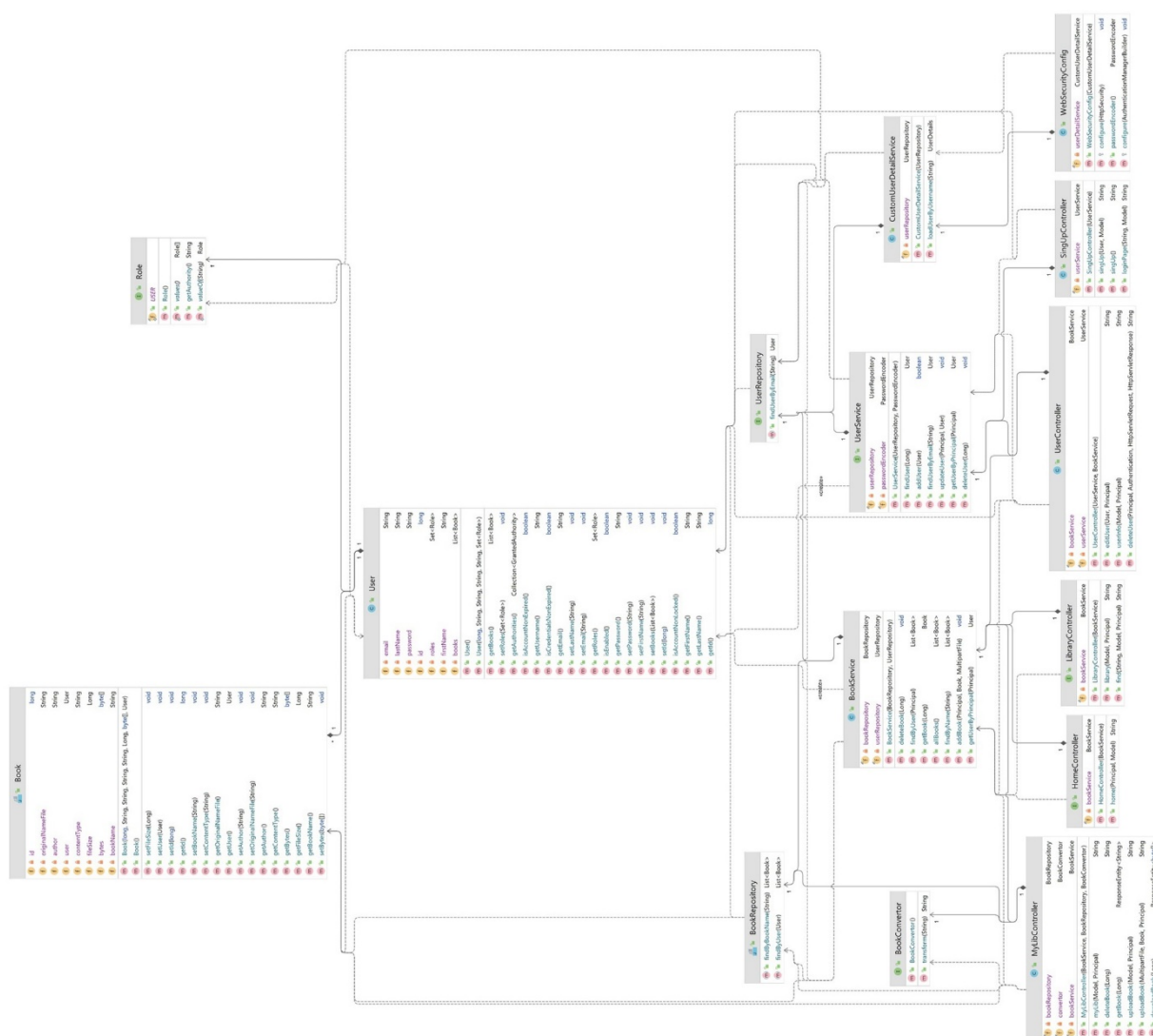
Приложение А.2 - Диаграмма развертывания



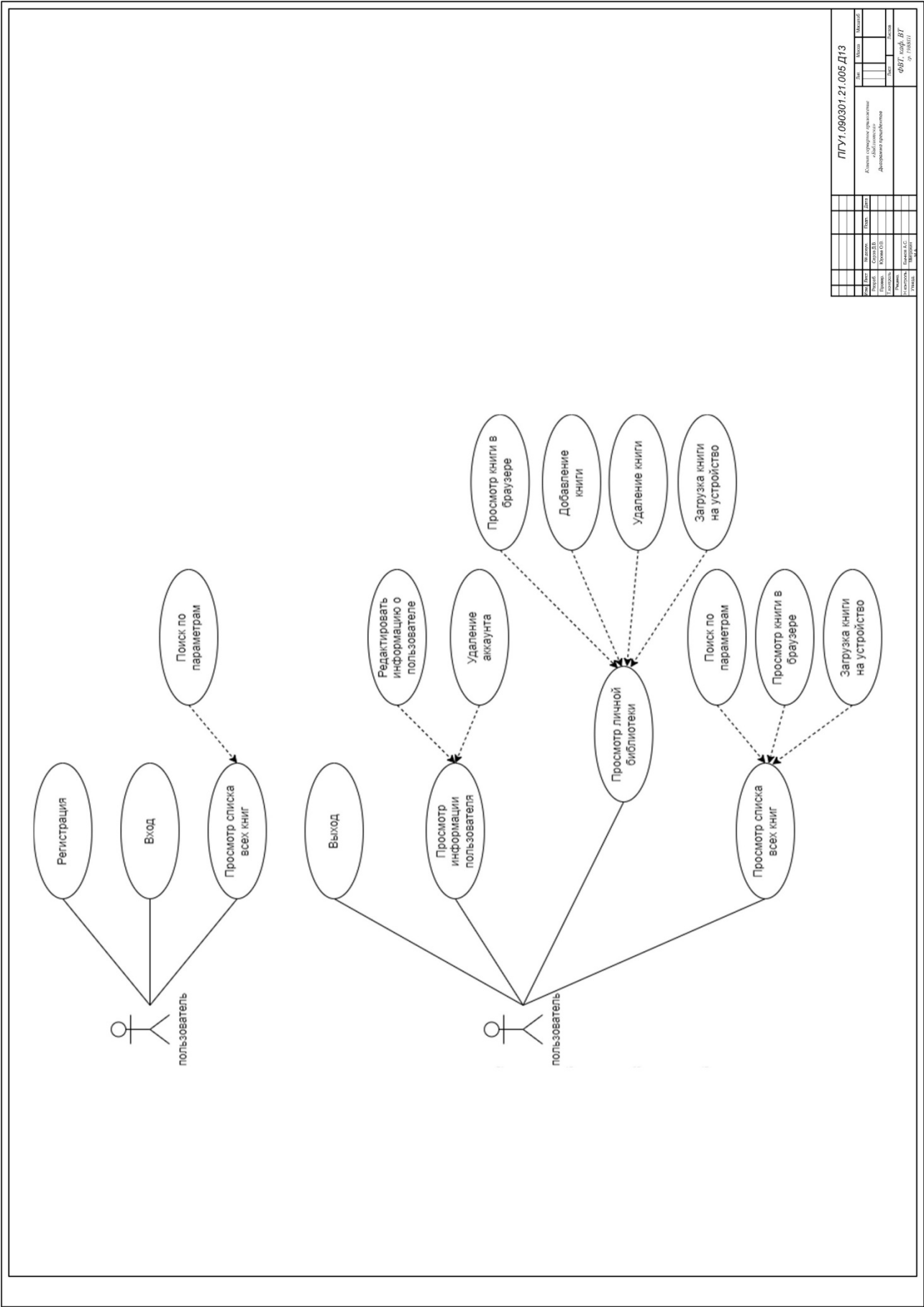
ПГУ1.090301.21.003 Д19									
Изм.	Лист	№ докум.	Подпись	Дата	Исполнитель	Проверенный	Утвержденный	Согласованный	Согласованный
					Исполнитель	Проверенный	Утвержденный	Согласованный	Согласованный
					Клиент сервера, приложение				
					Диаграмма развертывания				
					ФОРМ. табл. ИТ				
					от 10.01.2019				

Изм.	Лист	№ докум.	Подпись	Дата

Приложение А.3 - Диаграмма классов

[illegible]

Приложение А.4 - Диаграмма прецедентов



ПГУ1.090301.21.005 Д13									
Классификация информации									
Дополнительная информация									
ФОРМ. инф. ИТ									
от 19.06.2011									

Приложение Б
(Обязательное)
Клиент серверное приложение «Библиотека»
Листинг программы

					<i>ПГУ1.090301.21.001 ПЗ</i>	Лист
						66
Изм.	Лист	№ докум.	Подпись	Дата		

Файл User.java

```

@Entity
@Table(name = "users")
public class User implements UserDetails {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id")
    private long id;

    @Column(name = "first_name")
    private String firstName;
    @Column(name = "last_name")
    private String lastName;
    @Column(name = "email")
    private String email;
    @Column(name = "password")
    private String password;
    @ElementCollection(targetClass = Role.class, fetch =
FetchType.EAGER)
    @CollectionTable(name = "user_role", joinColumns =
@JoinColumn(name = "user_id"))
    @Enumerated(EnumType.STRING)
    private Set<Role> roles;
    @OneToMany(cascade = CascadeType.ALL, fetch = FetchType.EAGER,
mappedBy = "user")
    private List<Book> books = new ArrayList<>();

    public User(){}

    public User(long id,
                String firstName,
                String lastName,
                String email,
                String password,
                Set<Role> roles){

        this.id = id;
        this.firstName = firstName;
        this.lastName = lastName;
        this.email = email;
        this.password = password;
        this.roles = roles;
    }

    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 getEmail() {
    return email;
}

public void setEmail(String email) {
    this.email = email;
}

public String getPassword() {
    return password;
}

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

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

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

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

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

@Override
public Collection<? extends GrantedAuthority> getAuthorities() {
    return roles;
}

@Override
public String getUsername() {
    return email;
}

@Override
public boolean isAccountNonExpired() {
    return true;
}

```

```

    }

    @Override
    public boolean isAccountNonLocked() {
        return true;
    }

    @Override
    public boolean isCredentialsNonExpired() {
        return true;
    }

    @Override
    public boolean isEnabled() {
        return true;
    }
}

```

Файл Book.java

```

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

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id")
    private long id;

    @Column(name = "book_name")
    private String bookName;
    @Column(name = "author")
    private String author;
    @Column(name = "original_name_file")
    private String originalNameFile;
    @Column(name = "content_type")
    private String contentType;
    @Column(name = "file_size")
    private Long fileSize;
    @Lob
    @Type(type="org.hibernate.type.BinaryType")
    private byte[] bytes;
    @ManyToOne(cascade = CascadeType.REFRESH, fetch = FetchType.LAZY)
    @JoinColumn
    private User user;

    public Book(){}

    public Book(long id,
                String bookName,
                String author,
                String originalNameFile,
                String contentType,
                Long fileSize,
                byte[] bytes,
                User user){
    }
}

```

```

        this.id = id;
        this.bookName = bookName;
        this.author = author;
        this.originalNameFile = originalNameFile;
        this.contentType = contentType;
        this.fileSize = fileSize;
        this.bytes = bytes;
        this.user = user;
    }

    public long getId() {
        return id;
    }

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

    public String getBookName() {
        return bookName;
    }

    public void setBookName(String bookName) {
        this.bookName = bookName;
    }

    public String getAuthor() {
        return author;
    }

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

    public String getOriginalNameFile() {
        return originalNameFile;
    }

    public void setOriginalNameFile(String originalNameFile) {
        this.originalNameFile = originalNameFile;
    }

    public Long getFileSize() {
        return fileSize;
    }

    public void setFileSize(Long fileSize) {
        this.fileSize = fileSize;
    }

    public String getContentType() {
        return contentType;
    }

    public void setContentType(String contentType) {
        this.contentType = contentType;
    }

```

```

    }

    public byte[] getBytes() {
        return bytes;
    }

    public void setBytes(byte[] bytes) {
        this.bytes = bytes;
    }

    public User getUser() {
        return user;
    }

    public void setUser(User user) {
        this.user = user;
    }
}

```

Файл Role.java

```

public enum Role implements GrantedAuthority {
    USER;

    @Override
    public String getAuthority() {
        return name();
    }
}

```

Файл UserRepository.java

```

public interface UserRepository extends JpaRepository<User, Long> {
    User findUserByEmail(String email);
}

```

Файл BookRepository.java

```

public interface BookRepository extends JpaRepository<Book, Long> {
    List<Book> findByBookName(String bookName);
    List<Book> findByUser(User user);
}

```

Файл UserService.java

```

@Service
public class UserService {

    @Autowired
    private final UserRepository userRepository;
    @Autowired
    private final PasswordEncoder passwordEncoder;

    public UserService(UserRepository userRepository,
        PasswordEncoder passwordEncoder) {
    }
}

```

```

        this.userRepository = userRepository;
        this.passwordEncoder =passwordEncoder;
    }

    public User findUser(Long id){
        return userRepository.findById(id).orElse(null);
    }

    public User findUserByEmail(String email){
        return userRepository.findUserByEmail(email);
    }

    public boolean addUser(User user){
        String email = user.getEmail();

        if(userRepository.findUserByEmail(email) != null){
            return false;
        }
        user.setRoles(Collections.singleton(Role.USER));
        user.setPassword(passwordEncoder.encode(user.getPassword()));
        userRepository.save(user);

        return true;
    }

    public void updateUser(Principal principal, User user){
        User oldUser = getUserByPrincipal(principal);

        if(user.getFirstName() != null ||
!user.getFirstName().equals(oldUser.getFirstName())){
            oldUser.setFirstName(user.getFirstName());
        }

        if(user.getLastName() != null ||
!user.getLastName().equals(oldUser.getLastName())){
            oldUser.setLastName(user.getLastName());
        }

        if(user.getPassword() != null){
oldUser.setPassword(passwordEncoder.encode(user.getPassword()));
        }

        userRepository.save(oldUser);
    }

    public void deleteUser(Long id){
        userRepository.deleteById(id);
    }

    public User getUserByPrincipal(Principal principal){
        if(principal == null) return new User();
        return userRepository.findUserByEmail(principal.getName());
    }
}

```

Файл BookService.java

```

@Service
public class BookService {

    @Autowired
    private final BookRepository bookRepository;
    @Autowired
    private final UserRepository userRepository;

    public BookService(BookRepository bookRepository,
                       UserRepository userRepository) {
        this.bookRepository = bookRepository;
        this.userRepository = userRepository;
    }

    public Book getBook(Long id) {
        return bookRepository.findById(id).orElse(null);
    }

    public List<Book> findByName(String bookName) {
        return bookRepository.findByName(bookName);
    }

    public List<Book> findByUser(Principal principal) {
        return
bookRepository.findByUser(getUserByPrincipal(principal));
    }

    public List<Book> allBooks() {
        return bookRepository.findAll();
    }

    @Transactional
    public void addBook(Principal principal, Book book, MultipartFile
file) throws IOException {
        book.setBytes(file.getBytes());
        book.setOriginalNameFile(file.getOriginalFilename());
        book.setContentType(file.getContentType());
        book.setFileSize(file.getSize());
        book.setUser(getUserByPrincipal(principal));
        bookRepository.save(book);
    }

    public void deleteBook(Long id) {
        bookRepository.deleteById(id);
    }

    public User getUserByPrincipal(Principal principal) {
        if(principal == null) return new User();
        return userRepository.findUserByEmail(principal.getName());
    }

}

```

Файл BookConvertor.java

```
@Service
public class BookConvertor {

    public String transform(String xml) throws TransformerException {
        TransformerFactory factory = TransformerFactory.newInstance();
        Source xslt = new StreamSource(new
File("src/main/resources/book.xslt"));
        Transformer transformer = factory.newTransformer(xslt);

        Source text = new StreamSource(new StringReader(xml));
        StringWriter writer = new StringWriter();
        transformer.transform(text, new StreamResult(writer));

        return writer.toString();
    }
}
```

Файл CustomUserDetailsService.java

```
@Service
public class CustomUserDetailsService implements UserDetailsService {
    @Autowired
    private final UserRepository userRepository;

    public CustomUserDetailsService(UserRepository userRepository){
        this.userRepository = userRepository;
    }

    @Override
    public UserDetails loadUserByUsername(String email) throws
UsernameNotFoundException {
        User user = userRepository.findUserByEmail(email);
        if (user == null) {
            throw new UsernameNotFoundException("User not found");
        }
        return new
org.springframework.security.core.userdetails.User(user.getUsername(),
user.getPassword(), Collections.emptyList());
    }
}
```

Файл HomeController.java

```
@Controller
@RequestMapping("/")
public class HomeController {

    @Autowired
    private final BookService bookService;

    public HomeController(BookService bookService){
        this.bookService = bookService;
    }
}
```

```

    @GetMapping
    public String home(Principal principal, Model model) {
        model.addAttribute("user",
bookService.getUserByPrincipal(principal));
        return "home";
    }
}

```

Файл SingUpController.java

```

@Controller
public class SingUpController {

    @Autowired
    private final UserService userService;

    public SingUpController(UserService userService){
        this.userService = userService;
    }

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

    @PostMapping("/signup")
    public String singUp(User user, Model model){
        if(!userService.addUser(user)){
            model.addAttribute("error", "Пользователь с логином: " +
user.getEmail() + " уже существует");
            return "signup";
        }
        userService.addUser(user);

        return "redirect:/login";
    }

    @GetMapping("/login")
    public String loginPage(@RequestParam(value = "error", required =
false) String error, Model model) {
        if (error != null) {
            model.addAttribute("errorMessage", "Неправильный логин или
пароль");
        }
        return "login";
    }
}

```

Файл UserController.java

```

@Controller
@RequestMapping("/user")
public class UserController {

    @Autowired

```

```

private final UserService userService;
@Autowired
private final BookService bookService;

public UserController(UserService userService,
                     BookService bookService){
    this.userService = userService;
    this.bookService = bookService;
}

@GetMapping
public String userInfo(Model model, Principal principal){
    model.addAttribute("user",
userService.getUserByPrincipal(principal));
    return "user";
}

@PostMapping("/edit")
public String editUser(@ModelAttribute("user") User user,
Principal principal){
    userService.updateUser(principal, user) ;
    return "redirect:/user";
}

@PostMapping("/delete")
public String deleteUser(Principal principal, Authentication
authentication, HttpServletRequest request, HttpServletResponse
response){
    User user = userService.getUserByPrincipal(principal);

    userService.deleteUser(user.getId());
    new SecurityContextLogoutHandler().logout(request, response,
authentication);
    return "redirect:/";
}
}

```

Файл MyLibController.java

```

@Controller
public class MyLibController {

    @Autowired
    private final BookService bookService;
    private final BookRepository bookRepository;
    private final BookConvertor convertor;

    public MyLibController(BookService bookService,
                          BookRepository bookRepository,
                          BookConvertor convertor){
        this.bookService = bookService;
        this.bookRepository = bookRepository;
        this.convertor = convertor;
    }

    @GetMapping("/mylib")

```

```

        public String myLib(Model model, Principal principal){
            model.addAttribute("mybooks",
bookService.findByUser(principal));
            model.addAttribute("user",
bookService.getUserByPrincipal(principal));
            return "mylib";
        }

        @GetMapping("/mylib/add")
        public String uploadBook(Model model, Principal principal){
            model.addAttribute("user",
bookService.getUserByPrincipal(principal));
            return "book-add";
        }

        @PostMapping("/mylib/add")
        @Transactional
        public String uploadBook(@RequestParam("file") MultipartFile file,
Book book, Principal principal) throws IOException {
            bookService.addBook(principal, book, file);
            return "redirect:/mylib";
        }

        @GetMapping("/download/{id}")
        @ResponseBody
        public ResponseEntity<byte[]> downloadBook(@PathVariable Long id)
{
            Book file = bookService.getBook(id);
            if (file == null) {
                return ResponseEntity.notFound().build();
            }
            return ResponseEntity.ok()
                .header(HttpHeaders.CONTENT_DISPOSITION, "attachment;
filename=\"\" + file.getOriginalNameFile() + \"\"")
                .body(file.getBytes());
        }

        @GetMapping("/mylib/{id}")
        public ResponseEntity<String> getBook(@PathVariable Long id)
throws TransformerException {
            Optional<Book> bookOptional = bookRepository.findById(id);

            if (bookOptional.isPresent()) {
                Book book = bookOptional.get();

                String xml = new String(book.getBytes(),
StandardCharsets.UTF_8);
                String output = convertor.transform(xml);

                HttpHeaders headers = new HttpHeaders();
                headers.setContentType(MediaType.TEXT_HTML);

                return new ResponseEntity<>(output, headers,
HttpStatus.OK);
            }
            else {
                return new ResponseEntity<>(HttpStatus.NOT_FOUND);
            }
        }
    }

```

```

    }
}

@PostMapping("/mylib/{id}/delete")
public String deleteBook(@PathVariable Long id){
    bookService.deleteBook(id);
    return "redirect:/mylib";
}
}

```

Файл LibraryController.java

```

@Controller
public class LibraryController {

    @Autowired
    private final BookService bookService;

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

    @GetMapping("/library")
    public String library(Model model, Principal principal){
        model.addAttribute("books", bookService.allBooks());
        model.addAttribute("user",
bookService.getUserByPrincipal(principal));
        return "library";
    }

    @PostMapping("find")
    public String find(@RequestParam String bookName, Model model,
Principal principal){
        if(bookName != null && !bookName.isEmpty()){
            model.addAttribute("books",
bookService.findByName(bookName));
        }else {
            model.addAttribute("books", bookService.allBooks());
        }
        model.addAttribute("user",
bookService.getUserByPrincipal(principal));
        return "library";
    }
}

```

Файл MvcConfig.java

```

@Configuration
public class MvcConfig implements WebMvcConfigurer {

    public void addViewControllers(ViewControllerRegistry registry) {
        registry.addViewController("/login").setViewName("login");
    }
}

```

Файл WebSecurityConfig.java

```
@Configuration
@EnableWebSecurity
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {

    @Autowired
    private final CustomUserDetailsService userDetailsService;

    public WebSecurityConfig(CustomUserDetailsService
userDetailsService){
        this.userDetailsService = userDetailsService;
    }

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http
            .authorizeRequests()
                .antMatchers("/", "/singup", "/images/**",
"/library/**", "/find").permitAll()
                .anyRequest().authenticated()
            .and()
                .formLogin()
                    .loginPage("/login")
                    .usernameParameter("email")
                    .passwordParameter("password")
                    .failureUrl("/login?error=true")
                    .permitAll()
            .and()
                .logout()
                    .logoutUrl("/logout")
                    .logoutSuccessUrl("/")
                    .invalidateHttpSession(true)
                    .deleteCookies("JSESSIONID")
                    .permitAll();
    }

    @Override
    protected void configure(AuthenticationManagerBuilder auth) throws
Exception {
        auth

            .userDetailsService(userDetailsService)
            .passwordEncoder(passwordEncoder());
    }

    @Bean
    public PasswordEncoder passwordEncoder(){
        return new BCryptPasswordEncoder(8);
    }
}
```

Файл library.html

```
<!DOCTYPE HTML>
<html xmlns:th="http://www.thymeleaf.org">
<head>
```

					ПГУ1.090301.21.001 ПЗ	Лист
						79
Изм.	Лист	№ докум.	Подпись	Дата		

```

<title>MyBooks</title>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-
8"/>
<link rel="stylesheet"
href="https://cdn.jsdelivr.net/npm/bootstrap@5.2.3/dist/css/bootstrap.
min.css">
<link rel="stylesheet" type="text/css" href="/images/style.css">
</head>
<body>
<header th:if="${user.email}"
th:insert="blocks/header :: headerLogin(${user})">
</header>
<header th:unless="${user.email}"
th:insert="blocks/header :: headerLogout">
</header>

<div class="main-container container mb-5 mt-5" style="width: 1200px">
  <div class="row py-3">

    <div class="d-flex justify-content-center" style="font-family:
'Ubuntu', sans-serif; font-size: 28px;">
      <p>Всегда найдется, что почитать!</p>
    </div>
    <div class="col d-flex justify-content-end" style="width:
450px">

      <form th:action="@{find}" method="post" class="mt-1"
role="search">
        <div class="input-group">
          <input type="search" class="form-control"
name="bookName" placeholder="Введите название книги" aria-
describedby="but-find">
          <input class="btn btn-outline-secondary"
type="submit" id="but-find" value="Найти">
        </div>
      </form>
    </div>

    </div>
    <div class="overflow-auto" id="element" style="max-height: 600px">
      <table class="table table-hover text-center">
        <thead>
          <tr>
            <th scope="col">ID</th>
            <th scope="col">Название</th>
            <th scope="col">Автор</th>
            <th scope="col">Владелец</th>
            <th scope="col">Размер</th>
            <th scope="col">Действие</th>
          </tr>
        </thead>
        <tbody>
          <tr th:each="el : ${books}">
            <td th:text="${el.id}"></td>
            <td th:text="${el.bookName}"></td>
            <td th:text="${el.author}"></td>
            <td th:text="${el.getUser().firstName} + ' ' +

```

```

    ${el.getUser().lastName}"></td>
        <td th:text="${el.fileSize}"></td>
        <td class="row g-0 d-flex align-items-center justify-
content-center" th:if="${user.email}">
            <form method="get" class="col-md-4"
th:action="@{/mylib/{id}(id=${el.id})}">
                <button type="submit" class="btn btn btn-
outline-light">
                    
                </button>
            </form>
            <form method="get" class="col-md-4"
th:action="@{/download/{id}(id=${el.id})}">
                <button type="submit" class="btn btn-outline-
light">
                    
                </button>
            </form>
        </td>
        <td class="row g-0 d-flex align-items-center justify-
content-center" th:unless="${user.email}">
            <form class="col-md-4" onclick="return
confirm('Для просмотра или скачивания книги необходимо войти')">
                <button type="submit" class="btn btn-outline-
light">
                    
                </button>
            </form>
            <form class="col-md-4" onclick="return
confirm('Для просмотра или скачивания книги необходимо войти')">
                <button type="submit" class="btn btn-outline-
light">
                    
                </button>
            </form>
        </td>
    </tr>
</tbody>
</table>
</div>
</div>

<footer th:insert="blocks/footer :: footer"></footer>
</body>
</html>

```

Файл mylib.html

```

<!DOCTYPE HTML>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <title>MyBooks</title>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-

```

```

8"/>
    <link rel="stylesheet"
href="https://cdn.jsdelivr.net/npm/bootstrap@5.2.3/dist/css/bootstrap.
min.css">
    <link rel="stylesheet" type="text/css" href="/images/style.css">
</head>
<body>
<header th:insert="blocks/header :: headerLogin(${user})"></header>

<div class="main-container container mb-5 mt-5" style="width: 1200px">
    <div class="row py-3">
        <div class="col px-5" style="font-family: 'Ubuntu', sans-
serif; font-size: 28px;">
            <h4 th:text="'Личная библиотека: ' +
${user.firstName}"></h4>
        </div>
        <div class="d-flex justify-content-end col">
            <form method="get" th:action="@{/mylib/add}">
                <button class="btn btn-outline-secondary"
type="submit">
                    
                </button>
            </form>
        </div>
    </div>
    <div class="overflow-auto" id="element" style="max-height: 600px">
        <table class="table table-hover text-center">
            <thead>
                <tr>
                    <th scope="col">ID</th>
                    <th scope="col">Название</th>
                    <th scope="col">Автор</th>
                    <th scope="col">Файл</th>
                    <th scope="col">Размер</th>
                    <th scope="col">Действие</th>
                </tr>
            </thead>
            <tbody>
                <tr th:each="el : ${mybooks}">
                    <td th:text="${el.id}"></td>
                    <td th:text="${el.bookName}"></td>
                    <td th:text="${el.author}"></td>
                    <td th:text="${el.contentType}"></td>
                    <td th:text="${el.fileSize}"></td>
                    <td class="row g-0 d-flex align-items-center justify-
content-center">
                        <form method="get" class="col-md-3"
th:action="@{/mylib/{id} (id=${el.id})}">
                            <button type="submit" class="btn btn btn-
outline-light">
                                
                            </button>
                        </form>
                        <form method="get" class="col-md-3"
th:action="@{/download/{id} (id=${el.id})}">

```

```

        <button type="submit" class="btn btn-outline-
light">
            
        </button>
    </form>
    <form method="post" class="col-md-3"
th:action="@{/mylib/{id}/delete(id=${el.id})}" onsubmit="return
confirm('Вы уверены, что хотите удалить книгу?')">
        <button type="submit" class="btn btn btn-
outline-light">
            
        </button>
    </form>
</td>
</tr>
</tbody>
</table>
</div>
</div>

<footer th:insert="blocks/footer :: footer"></footer>
</body>
</html>

```

Файл user.html

```

<!DOCTYPE HTML>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <title>MyBooks</title>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-
8"/>
    <link rel="stylesheet"
href="https://cdn.jsdelivr.net/npm/bootstrap@5.2.3/dist/css/bootstrap.
min.css">
    <link rel="stylesheet" type="text/css" href="/images/style.css">
</head>
<body>

<header th:insert="blocks/header :: headerLogin(${user})"></header>

<main class="main-container container px-3 mt-5 d-flex justify-
content-center" style="width: 1200px">
    <div style="width: 400px">
        <h1 class="h3 mb-3 mt-5 fw-normal">Редактирование</h1>

        <div class="row mt-2 mb-5 d-flex align-items-center justify-
content-center">
            <form method="post" th:action="@{/user/edit}">
                <input type="hidden" name ="id" th:field="${user.id}">

                <div class="form-floating mt-2">
                    <input type="text" class="form-control"

```

```

name="firstName" id="firstName" th:value="${user.firstName}">
    <label for="firstName">Имя</label>
</div>

<div class="form-floating mt-2">
    <input type="text" class="form-control"
name="lastName" id="lastName" th:value="${user.lastName}">
    <label for="lastName">Фамилия</label>
</div>

<div class="form-floating mt-2">
    <input type="email" class="form-control"
name="email" id="email" th:value="${user.email}">
    <label for="email">Логин</label>
</div>
<div class="form-floating mt-2">
    <input type="password" class="form-control"
name="password" id="password">
    <label for="password">Новый пароль</label>
</div>

<button class="w-100 btn btn-lg btn-outline-primary
mt-2" type="submit">Изменить</button>
</form>

<form class="col" method="post"
th:action="@{/user/delete}" onsubmit="return confirm('Вы уверены, что
хотите удалить аккаунт?')">
    <button type="submit" class="w-100 btn btn-lg btn-
outline-danger mt-2">Удалить аккаунт</button>
</form>
</div>

</div>

</main>

<footer th:insert="blocks/footer :: footer"></footer>
</body>
</html>

```

Файл singup.html

```

<!DOCTYPE HTML>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <title>MyBooks</title>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-
8"/>
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-
alpha3/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-
KK94CHFLLe+nY2dmCWGMq91rCGa5gtU4mk92HdvYe+M/SXH301p5ILy+dN9+nJ0Z"
crossorigin="anonymous">
</head>
<body class="text-center">

```

```

<main class="mx-auto" style="width: 350px">
  <form method="post" th:action="@{/signup}">
    
    <h1 class="h3 mb-3 fw-normal">Регистрация</h1>

    <div class="form-floating mt-2">
      <input type="text" class="form-control" name="firstName"
id="firstName" placeholder="Имя">
      <label for="firstName">Имя</label>
    </div>

    <div class="form-floating mt-2">
      <input type="text" class="form-control" name="lastName"
id="lastName" placeholder="Фамилия">
      <label for="lastName">Фамилия</label>
    </div>

    <div class="form-floating mt-2">
      <input type="email" class="form-control" name="email"
id="email" placeholder="name@example.com">
      <label for="email">Логин</label>
    </div>

    <div class="form-floating mt-2">
      <input type="password" class="form-control"
name="password" id="password" placeholder="Пароль">
      <label for="password">Пароль</label>
    </div>

    <button class="w-100 btn btn-lg btn-primary mt-2"
type="submit">Регистрация</button>
  </form>

  <div class="container" th:if="${error}">
    <div class="alert alert-info mt-2" th:text="${error}"></div>
  </div>
</main>

</body>
</html>

```

Файл login.html

```

<!DOCTYPE HTML>
<html xmlns:th="http://www.thymeleaf.org">
<head>
  <title>MyBooks</title>
  <meta http-equiv="Content-Type" content="text/html; charset=UTF-
8"/>
  <link rel="stylesheet"
href="https://cdn.jsdelivr.net/npm/bootstrap@5.2.3/dist/css/bootstrap.
min.css">
</head>
<body class="text-center">

```

```

<main class="mx-auto mt-3" style="width: 350px">
  <form th:action="@{/login}" method="post" class="mt-5">

    <h1 class="h3 mb-3 fw-normal">Вход</h1>

    <div class="form-floating mt-2">
      <input type="text" class="form-control" name="email"
id="email" placeholder="name@example.com">
      <label for="email">Логин</label>
    </div>
    <div class="form-floating mt-2">
      <input type="password" class="form-control"
name="password" id="password" placeholder="Пароль">
      <label for="password">Пароль</label>
    </div>

    <div th:if="${errorMessage}" class="alert alert-danger mt-2"
role="alert" th:text="${errorMessage}"></div>

    <button class="w-100 btn btn-lg btn-primary mt-2"
type="submit">Вход</button>
  </form>

  <form th:action="@{/signup}" method="get">
    <button class="w-100 btn btn-lg btn-primary mt-2"
type="submit">Регистрация</button>
  </form>

  <p class="mt-5 mb-3 text-body-secondary">© 2017-2023</p>

</main>

</body>
</html>

```

Приложение В

(Обязательное)

Клиент серверное приложение «Библиотека»

Презентация

					ПГУ1.090301.21.001 ПЗ	Лист
						87
Изм.	Лист	№ докум.	Подпись	Дата		

Клиент серверное приложение «Библиотека»

АВТОР РАБОТЫ:

СТУДЕНТ ГР. 19ВВП1

СЕРОВ ДАНИИЛ ВАЛЕРЬЕВИЧ

НАУЧНЫЙ РУКОВОДИТЕЛЬ:

К.Т.Н., ДОЦЕНТ КАФЕДРЫ ВТ ПГУ

ЮРОВА О.В.

Пенза 2023

АКТУАЛЬНОСТЬ ПРОБЛЕМЫ

- Цифровизация
- Предпочтение электронным ресурсам
- Доступность
- Комфорт использования
- Повышения уровня обслуживания

					ПГУ1.090301.21.001 ПЗ	Лист
						88
Изм.	Лист	№ докум.	Подпись	Дата		

ЦЕЛИ И ЗАДАЧИ

- Цель работы – разработать клиент серверное приложение, которое позволяет пользователям загружать книги на сервер и взаимодействовать с ними.
- Задачи:
 - Изучить предметную область;
 - Определить функционал;
 - Выбрать технологии и средства разработки приложения;
 - Разработать приложение.

ФУНКЦИОНАЛ

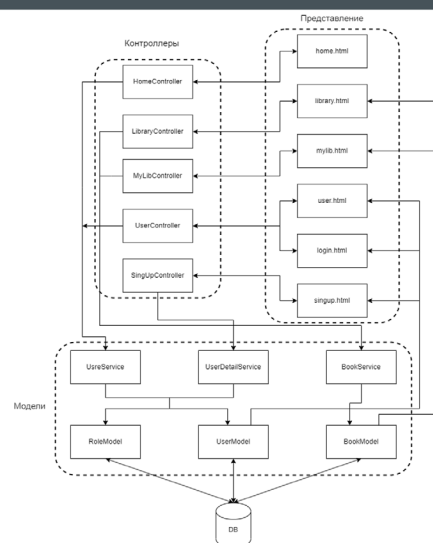
- Регистрация и авторизация;
- Определить права доступа к возможностям приложения;
- Загрузка экземпляра книги в базу данных;
- Просмотр книги в браузере;
- Возможность скачать экземпляр книги из базы данных;
- Удаление экземпляра книги;
- Редактирование и удаление аккаунта.

ИСПОЛЬЗУЕМЫЕ ТЕХНОЛОГИИ



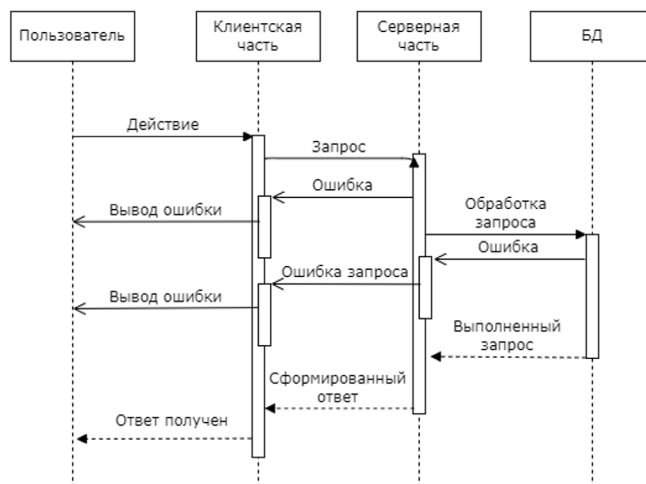
АРХИТЕКТУРА ПРИЛОЖЕНИЯ

Архитектура *MVC* (*Model-View-Controller*) - это популярный подход к разработке приложений, который разделяет приложение на три компонента: модель (*Model*), представление (*View*) и контроллер (*Controller*)

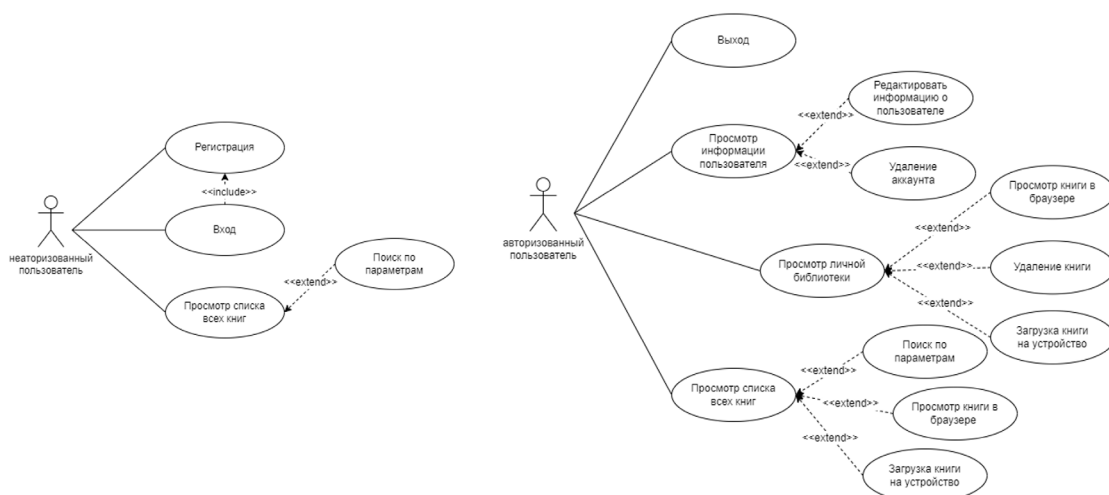


Изм.	Лист	№ докум.	Подпись	Дата

ПОСЛЕДОВАТЕЛЬНОСТЬ РАБОТЫ ПРИЛОЖЕНИЯ



UML ДИАГРАММА ВАРИАНТОВ ИСПОЛЬЗОВАНИЯ



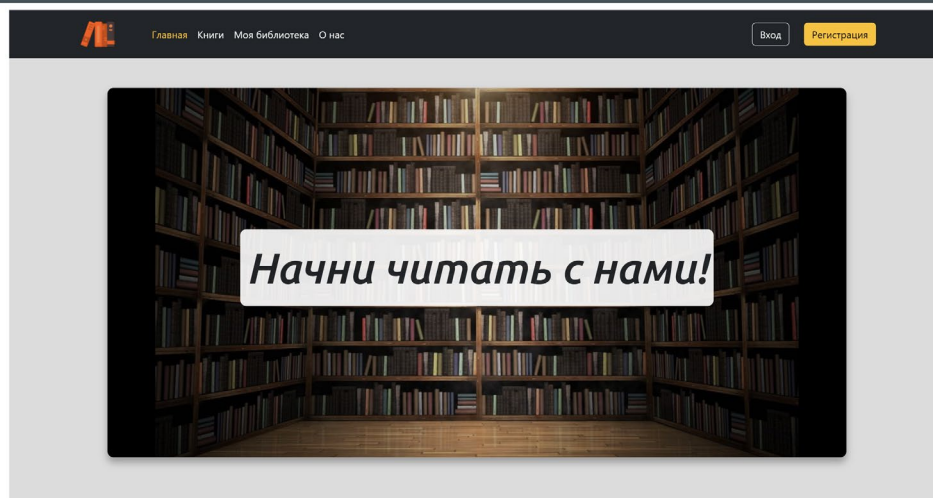
Изм.	Лист	№ докум.	Подпись	Дата

ПГУ1.090301.21.001 ПЗ

Лист

91

РАЗРАБОТКА ПРИЛОЖЕНИЯ



РЕГИСТРАЦИЯ И АВТОРИЗАЦИЯ



Регистрация

Имя
Фамилия
Логин
Пароль
Регистрация

Вход

Логин
Пароль
Вход
Регистрация

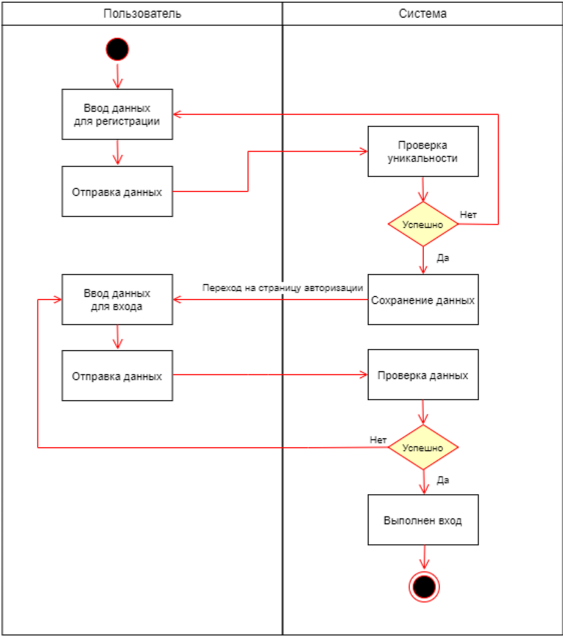
Изм.	Лист	№ докум.	Подпись	Дата

ПГУ1.090301.21.001 ПЗ

Лист

92

UML ДИАГРАММА ДЕЯТЕЛЬНОСТИ РЕГИСТРАЦИИ И АВТОРИЗАЦИИ



ОБЩАЯ БИБЛИОТЕКА



[Главная](#) [Книги](#) [Моя библиотека](#) [О нас](#)



Максим



Всегда найдется, что почитать!

Введите название книг

ID	Название	Автор	Владелец	Размер	Действие
6	Мастер и Маргарита	Булгаков	Егор Егоров	2124503	
7	Лекарь	Певухин	Максим Максимов	804012	
8	Преступление и наказание	Достоевский	Максим Максимов	2864284	
9	Маленький принц	Антуан де Сент-Экзюпери	Даниил Серов	997651	

ЛИЧНАЯ БИБЛИОТЕКА

 Главная Книги Моя библиотека О нас

Максим



Личная библиотека: Максим

Добавить

ID	Название	Автор	Файл	Размер	Действие
7	Лекарь	Певухин	application/octet-stream	804012	<div></div>
8	Преступление и наказание	Достоевский	application/octet-stream	2864284	<div></div>

ЗАГРУЗКА КНИГИ

 Главная Книги Моя библиотека О нас

Максим





Добавление книги

Название книги

Автор книги

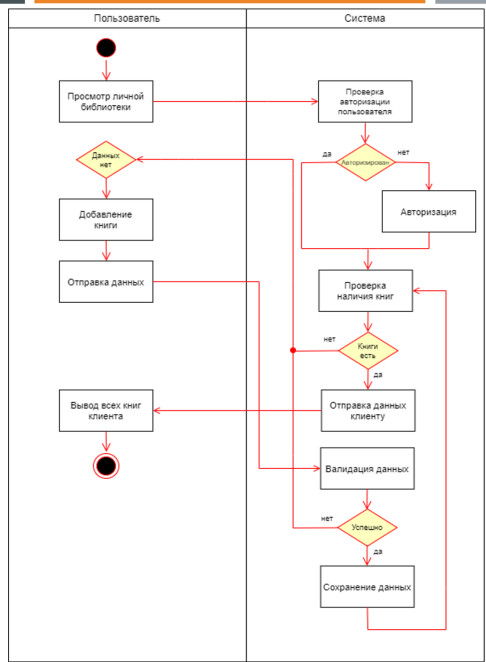
Загрузите книгу

Выберите файл

Файл не выбран

Добавить

UML ДИАГРАММА ДЕЯТЕЛЬНОСТИ ДОБАВЛЕНИЯ И ПРОСМОТРА



ПРОСМОТР КНИГИ

Преступление и наказание

Автор: Федор Достоевский

Часть первая

Часть первая

I

I

В начале июля, в чрезвычайно жаркое время, под вечер один молодой человек вышел из своей каморки, которую нанимал от жильцов в С-м переулке, на улицу и медленно, как бы в нерешимости, отправился к К-ну мосту.

Он благополучно избежал встречи с своею хозяйкой на лестнице. Каморка его приходилась под самую кровель высокого пятиэтажного дома и походила более на шкаф, чем на квартиру. Квартирная же хозяйка его, у которой он нанимал эту каморку с обедом и прислугой, помещалась одною лестницей ниже, в отдельной квартире, и каждый раз, при выходе на улицу, ему непременно надо было проходить мимо хозяйкиной кухни, почти всегда настежь открытой на лестницу. И каждый раз молодой человек, проходя мимо, чувствовал какое-то болезненное и трясливое ощущение, которого стыдился и от которого морщился. Он был должен крутом хозяйке и боялся с нею встретиться.

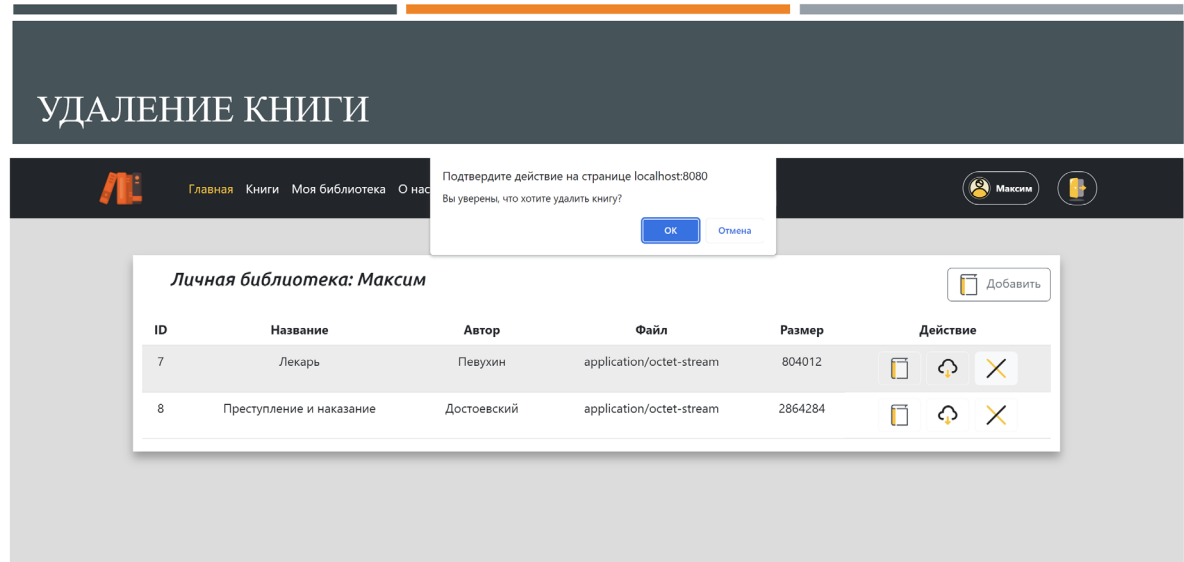
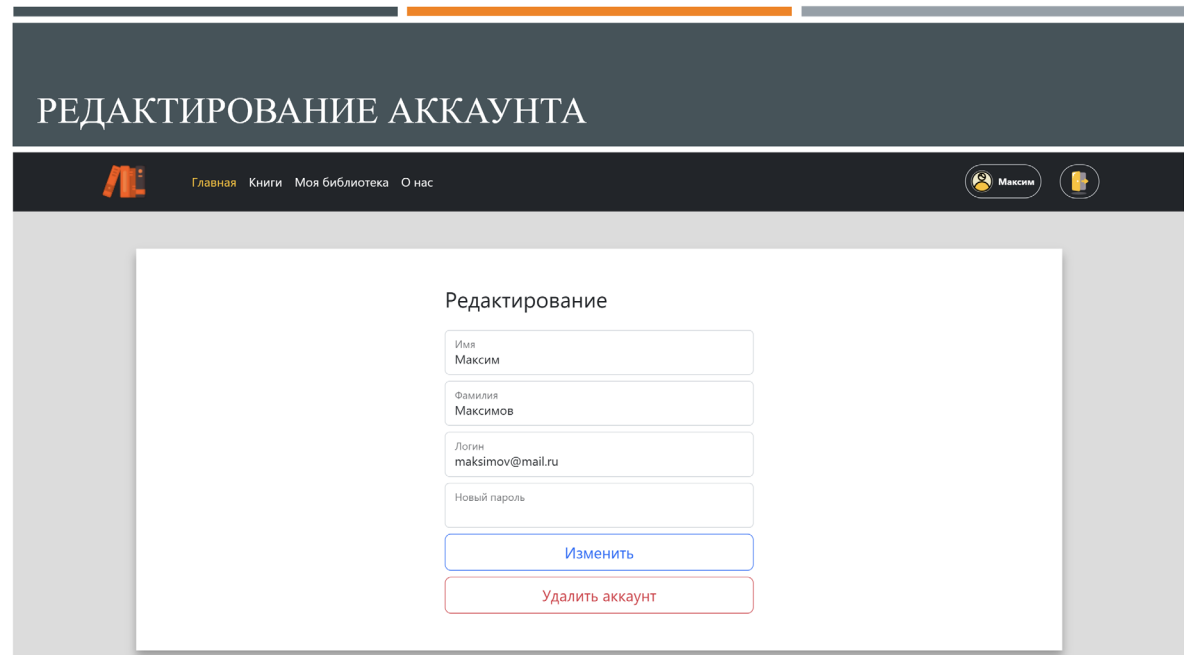
Не то чтоб он был так труслив и забит, совсем даже напротив; но с некоторого времени он был в раздражительном и напряженном состоянии, похожем на ипохондрию. Он до того углубился в себя и уединился от всех, что боялся даже всякой встречи, не только встречи с хозяйкой. Он был задавлен бедностью; но даже стесненное положение перестало в последнее время тяготить его. Насульными делами своими он совсем перестал и не хотел заниматься. Никакой хозяйки, в сущности, он не боялся, что бы та ни замыслила против него. Но останавливаясь на лестнице, слушать всякий вздор про всю эту обыденную дребедень, до которой ему нет никакого дела, все эти приставания о платеже, угрозы, жалобы, и при этом самому изворачиваться, извиняться, лгать, – нет уж, лучше проскользнуть как-нибудь кошкой по лестнице и улечься, чтобы никто не выдал.

Впрочем, на этот раз страх встречи с своею кредиторшей даже его самого поразил по выходе на улицу.

«На какое дело хочу покуситься и в то же время каких пустяков боюсь! – подумал он с странною улыбкой. – Гм... да... все в руках человека, и все-то он мимо носа пронесет единственно от одной трусости... это уж аксиома... Любопытно, чего люди больше всего боятся? Нового шага, нового собственного слова они всего больше боятся... А впрочем, я слышом много болтаю. Оттого и ничего не делаю, что болтаю. Пожалуй, впрочем, и так: оттого болтаю, что ничего не делаю. Это я в этот последний месяц выучился болтать, лежа по целым суткам в углу и думая... о царе Горохе. Ну зачем я теперь иду? Разве я способен на это? Разве это серьезно? Совсем не серьезно. Так, ради фантазии сам себя тешу; игрушки! Да, пожалуй, что и игрушки!»

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ1.090301.21.001 ПЗ



ЗАКЛЮЧЕНИЕ

В ходе написания данной выпускной квалификационной работы были выполнены следующие задачи: анализ актуальности темы, анализ и выбор среды разработки и инструментов, а также анализ архитектуры приложения, разработано клиент серверное приложение «Библиотека», разработана база данных, выполнено тестирование приложения.

Спасибо за внимание!

					ПГУ1.090301.21.001 ПЗ	Лист
						97
Изм.	Лист	№ докум.	Подпись	Дата		