

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет
вычислительной техники

Кафедра
«Вычислительная техника»

Направление подготовки: 09.03.01 «Информатика и вычислительная техника»

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

БАКАЛАВРСКАЯ РАБОТА

на тему

WEB-ПРИЛОЖЕНИЕ

«КНИЖНЫЙ ИНТЕРНЕТ-МАГАЗИН»

Студент Эрмолаев Артем Дмитриевич
(подпись, дата) 30.05.24

Руководитель Юрова Ольга Викторовна
(подпись, дата) 30.05.24

Нормоконтролёр Бычков Андрей Станиславович
(подпись, дата) 30.05.24

Работа допущена к защите (протокол зас. кафедры от 27.05.24 г. № 11)

Заведующий кафедрой М.А. Митрохин
(подпись) (фамилия, инициалы)

Работа защищена с отметкой отлично (протокол заседания ГЭК от
10.06.2024 № 13)

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

Пенза 2024

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

Факультет
вычислительной техники

Кафедра
«Вычислительная техника»

«Утверждаю»
Заведующий кафедрой
М.А. Митрохин
«19» 04 2024 г.

**ЗАДАНИЕ
НА БАКАЛАВРСКУЮ РАБОТУ**

1. Студент Ермолаев Артем Дмитриевич гр.20ВВП1 факультета вычислительной техники
2. Тема работы: *Web*-приложение «Книжный интернет-магазин»
- Тема утверждена приказом ПГУ № 405/о от " 19 " апреля 2024 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 *Web*-приложение «Книжный интернет-магазин» для возможности приобретения книг онлайн. Приложение должно позволять пользователям подбирать необходимую книгу по поиску, иметь возможность заказать книгу и просмотреть информацию о ранее созданных заказах. 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 подбирать, заказывать и иметь возможность просмотреть информацию о ранее заказанных книгах.

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 *SpringFramework*, *PostgreSQL*, *SpringSecurity* и т.д.

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1.	Изучение предметной области	50	19.04.2024- 30.04.2024	
2.	Анализ требований	10	01.05.2024- 02.05.2024	
3.	Выбор технологий, применяемых при проектировании и разработке	50	03.05.2024- 07.05.2024	
4.	Описание алгоритмов	40	08.05.2024- 11.05.2024	
5.	Разработка Web-приложения «Книжный интернет-магазин»	150	12.05.2024- 29.05.2024	
6.	Оформление пояснительной записки	24	30.05.2024- 31.05.2024	

Дата выдачи задания " 19 " апреля 2024 г.

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

Задание к исполнению принял студент  - Ермолаев А.Д.
(подпись, дата)

Работу к защите допустить
Декан факультета  31.05.2024 Фионова Л.Р.

Реферат

Бакалаврская работа содержит 100 листов, 33 рисунка, 5 таблиц, 6 источников и 3 приложения

WEB-ПРИЛОЖЕНИЕ «КНИЖНЫЙ ИНТЕРНЕТ-МАГАЗИН», SPRING FRAMEWORK, INTELLIJ IDEA, SPRING SECURITY, POSTGRESQL, PGADMIN4, HTML, SPRING MVC, JAVA PERSISTENCE API, CSS, JAVA

Цель работы – разработать клиент-серверное приложение, позволяющее выполнять продажу книг онлайн.

Разработанное приложение должно позволять пользователям подбирать необходимую книгу по поиску, иметь возможность заказать книгу и просмотреть информацию о ранее созданных заказах. 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 продажи книг пользователям. Разработка производилась с использованием масштабного стека технологий, к которым относятся: язык программирования Java, язык структурированных запросов SQL, Spring Framework, Spring Security, Spring MVC, спецификация для работы с базами данных в Java-приложениях Java Persistence API, язык гипертекстовой разметки HTML, язык каскадных таблиц стилей CSS, среда разработки IntelliJIDEA. Для хранения данных выбрана СУБД PostgreSQL, а для её администрирования применялся инструмент PGAdmin4.

					ПГУ 1.090301.06.001 ПЗ			
Изм.	Лист.	№ докум.	Подпись	Дата	<div>Web-приложение «Книжный интернет-магазин»</div>			
Разраб		Ермолаев А.Д.		30.05.24				
Провер.		Юрова О.В.		30.05.24				
Н. Контр.		Бычков А.С.		30.05.24				
Н. Контр.								
Утвер.		Митрохин М.А.		30.05.24				
					Лит.	Лист.	Листов	
						2	100	

Содержание

Перечень сокращений и обозначений	5
Введение	6
1 Анализ предметной области.....	7
1.1 Анализ актуальности темы.....	7
1.2 Анализ аналогов книжных интернет-магазинов.....	7
1.2.1 Книжный интернет-магазин «chitai-gorod.ru».....	8
1.2.2 Книжный интернет-магазин «book24.ru»	9
1.2.3 Книжный интернет-магазин «bookvoed.ru»	10
1.2.4 Вывод.....	13
1.3 Постановка задачи.....	13
2 Выбор технологий и средств разработки приложения	16
2.1 Выбор языка программирования	16
2.2 Выбор фреймворка	17
2.3 Spring JPA.....	19
2.4 Spring Security.....	19
2.5 Spring MVC	20
2.6 Выбор базы данных.....	21
2.7 pgAdmin4.....	22
2.8 Выбор среды разработки	23
3 Архитектура Web-приложения «Книжный интернет-магазин»	26
4 Реализация приложения.....	30
4.1 Описание работы приложения	30
4.2 Программная реализация приложения.....	30
4.2.1 Протокол общения пользовательского клиента и сервера	30
4.2.2 Протокол общения клиента-администратора системы и сервера	34
4.2.3 Протокол общения сервера и базы данных	34
4.3 Структура БД	36
4.4 Структура приложения	38
5 Тестирование приложения.....	41
5.1 Регистрация пользователя	41
5.2 Авторизация пользователя	43

5.3	Тестирование функционала взаимодействия с книгами	46
5.4	Редактирование данных пользователя	50
5.5	Функционал неавторизованного пользователя	52
6	Безопасность жизнедеятельности.....	54
6.1	Требования к помещению и рабочему месту	54
6.2	Требования к освещению	54
6.3	Требования к режиму работы и отдыха	55
6.4	Рекомендации по использованию программного обеспечения	55
	Заключение.....	57
	Список используемых источников	58
	Приложение А.....	59
	Приложение А.1-Диаграмма последовательностей.....	60
	Приложение А.2-Диаграмма развертывания	61
	Приложение А.3-Диаграмма классов	62
	Приложение А.4-Диаграммы вариантов использования.....	63
	Приложение Б	63
	Приложение В.....	86

Перечень сокращений и обозначений

В настоящей работе применяют следующие сокращения и обозначения

БД – база данных

СУБД – система управления базой данных

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

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

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

JSP – (от англ. Java Server Pages) – технология, разработанная Java для создания динамических веб-страниц в приложении

ORM - (от англ. Object-Relational Mapping) – используется для связи языка программирования с базой данных

CSRF (от англ. Cross-Site Request Forgery) – вид атак на сайт, который делает действия от своего имени в браузере

XSS (от англ. Cross-Site Scripting) — веб-атака для внесения вредоносного кода на страницу

JSON (от англ. JavaScript Object Notation) – используется для обмена данными и хранения их

MVC (от англ. Model-View-Controller) — шаблон, который разделяет программу на три модуля

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

Введение

На данный момент веб-приложения решают ключевые задачи и играют ключевую роль для автоматизации бизнес-процессов. Если затрагивать книжные разделы, то здесь с каждым днем использование таких приложений становится необходимым для пользователя.

В данной работе мы рассмотрим то, как можно разработать такое приложение, его аналоги, все возможные функции. Помимо этого, будет рассмотрена его архитектура, интерфейс и аналоги для дальнейшего использования.

Сделав небольшой итог, можно сказать, что эта работа является важной частью нашего общества, ведь улучшает подбор нужной литературы и упрощает возможность покупки.

					ПГУ1.090301.06.001 ПЗ	Лист
						6
Изм.	Лист	№ докум.	Подпись	Дата		

1 Анализ предметной области

Для начала нужно понять, кто будет конечным потребителем и что необходимо для этого реализовать. После этого уже понять где он будет применяться. Также рассмотрим теорию, которая служит для разработки клиент-серверного приложения. Без этих моментов приложение не будет работать как нужно и не даст конечному пользователю то, что требуется. Также это необходимо для решения проблем, которые могут появиться при создании приложения.

1.1 Анализ актуальности темы

На данный момент книги являются неотъемлемой частью нашей жизни и для покупки нужны какие-то сервисы. Возьмем, к примеру, сервис «chitai-gorod.ru», который обсудим немного ниже, там есть возможность простого заказа. Покупатель может сделать его в один клик. Соответственно наше приложение должно быть не хуже. Пользователю для удобного пользования нужно удобство, чтобы он мог быстро и точно заказать то, что нужно. Соответственно веб-приложение – это удобная вещь для реализации своих потребностей, где можно решить свои проблемы одним кликом из любого места на планете.

Обратимся к некоторым моментам, естественно для удобства нужно сделать удаленный доступ, по возможности реализовать быстроту выполнения заказа, информативность о сделанном заказе. Это все то, что нужно проанализировать при разработке приложения. Конечно, разработка требует время, силы и так далее, но для конечного пользователя это будет очень удобно, поэтому эти нюансы оправдывают потраченные средства.

1.2 Анализ аналогов книжных интернет-магазинов

Как мы и говорили ранее, начнем с книжного и, наверное, самого популярного магазина в интернете про книги «chitai-gorod.ru». Таких сайтов на самом деле очень много, но мы рассмотрим только три и самые популярные для

					ПГУ1.090301.06.001 ПЗ	Лист
						7
Изм.	Лист	№ докум.	Подпись	Дата		

определения нюансов при разработке в дальнейшем. У каждого магазина есть, как и плюсы, так и минусы, их мы также подробно рассмотрим. Хотелось бы отметить, что все приложения практически доведены до идеала, но про некоторые моменты все же стоит поговорить.

1.2.1 Книжный интернет-магазин «chitai-gorod.ru»

Читай-город, начнем с гиганта книжной индустрии, тут огромный ассортимент, можно найти все что угодно и даже больше. Упрощенный заказ, услуги доставки, красивый дизайн и это только малая часть этого сайта. Ниже приведены плюсы данной приложения:

- поиск книг, большой выбор, как и писалось ранее, большое количество фильтров, таких как авторы, категории, возрастные ограничения, жанры и т.д.;
- удобство, все вкладки интуитивно понятны и расположены в нужных местах;
- корзина книг, есть возможность добавить нужны литературу в корзину и позже ее приобрести.

Интерфейс сервиса «chitai-gorod.ru» представлен на рисунках (1, 2.).

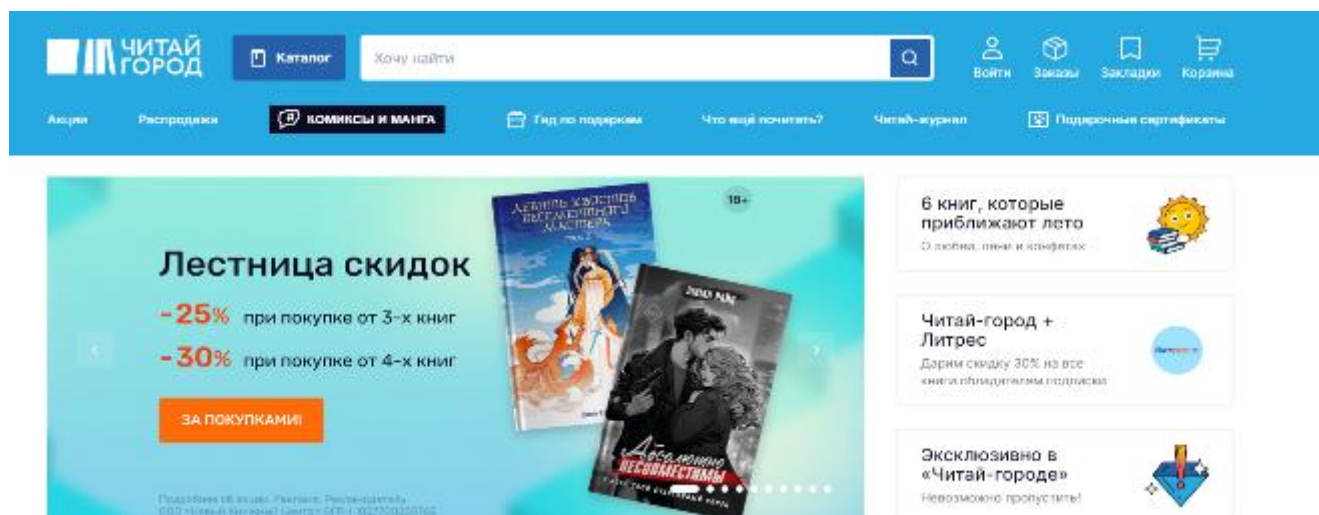


Рисунок 1 – Главная страница сервиса «chitai-gorod.ru»

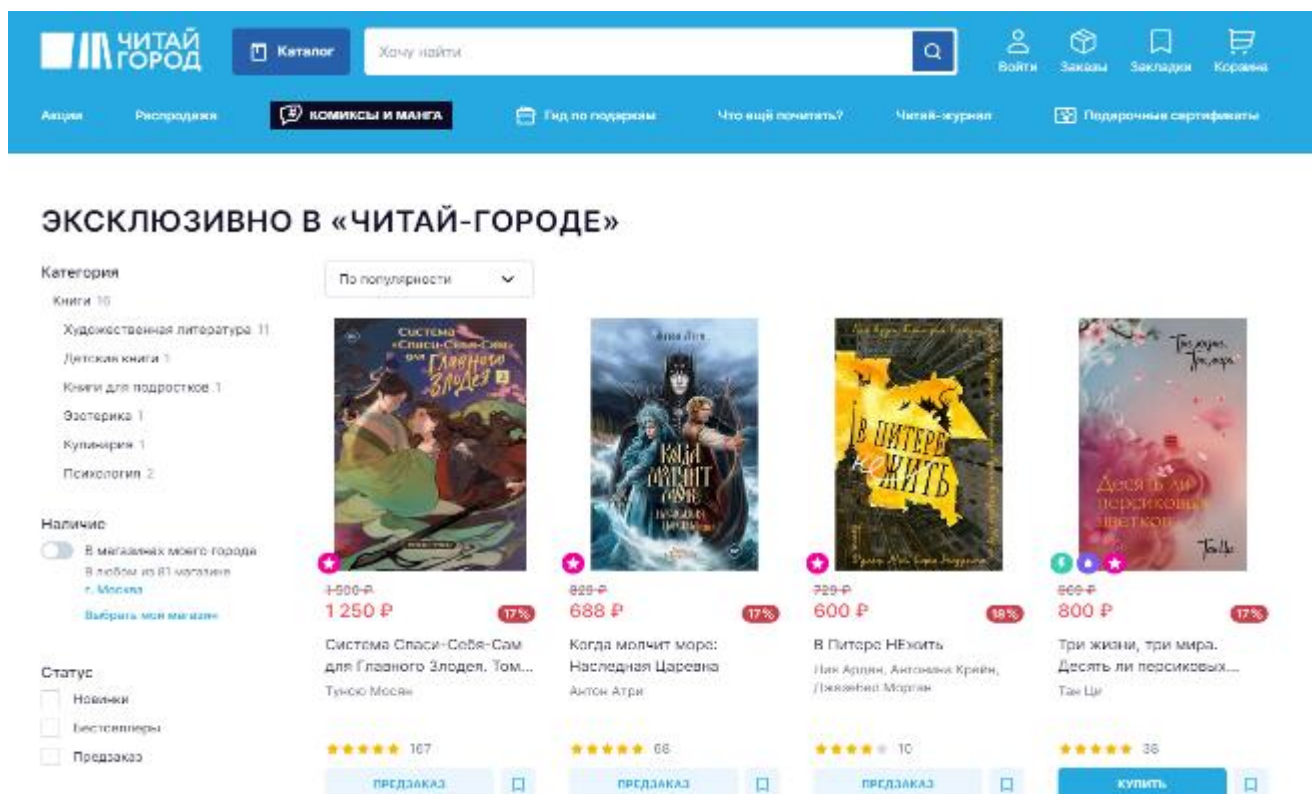


Рисунок 2 – Библиотека эксклюзивных книг

1.2.2 Книжный интернет-магазин «book24.ru»

Менее популярный, но тоже очень привлекательный сайт, который может заинтересовать любого любителя книг. Здесь обратим внимание на доставку, она может быть бесплатная от определенной суммы в корзине. При этом она действует на все регионы страны, что не может не радовать. Это также один из многих плюсов этого сайта. Давайте рассмотрим и другие плюсы:

- фильтры, благодаря им можно более точно подобрать нужную литературу. Есть фильтры по жанру, цене, возрасту, автору и так далее;
- изменение данных. Пользователь, который уже зарегистрировался на сайте, может в личном кабинете изменить данные, которые указывал при регистрации. Также у него есть возможность посмотреть историю покупок в нужной вкладке. Данный функционал очень полезен и необходим для такой категории сайтов.

Интерфейс сервиса «book24.ru» представлен на рисунках (3, 4.).



Рисунок 3 – Главная страница сервиса «book24.ru»

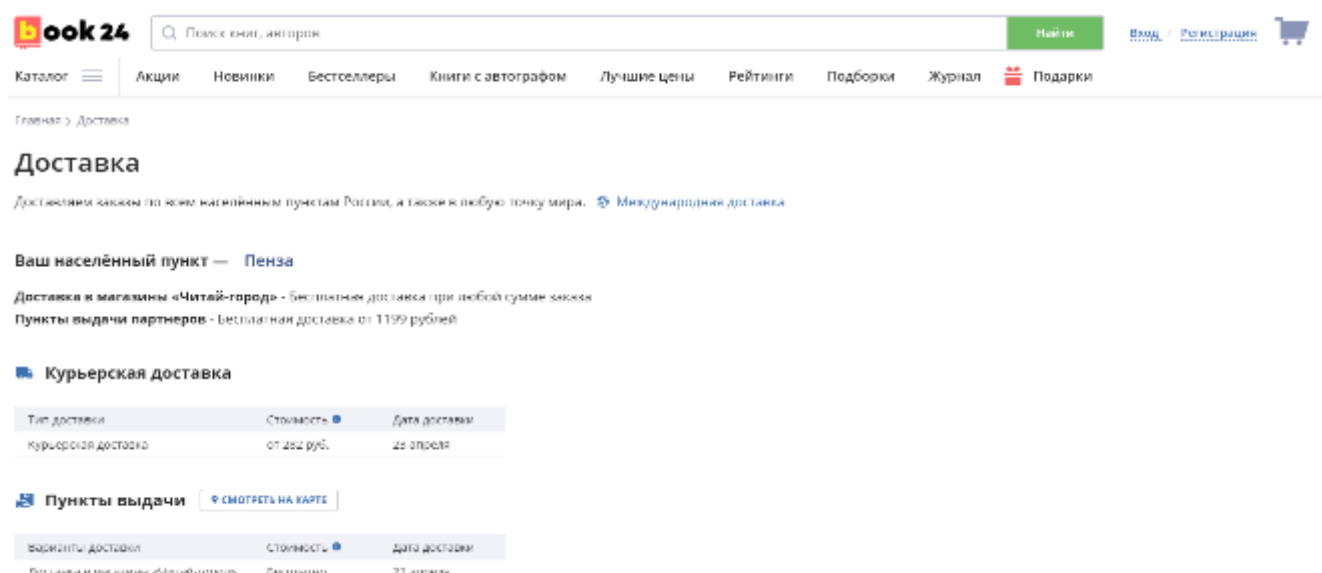


Рисунок 4 – Условия доставки для клиента

1.2.3 Книжный интернет-магазин «bookvoed.ru»

Он является третьим по значимости сайтом для любителей книг. Считается, что огромный плюсом этого сайта являются скидки и цены на книги. Отдельная вкладка была реализована для этого немаловажного нюанса. Пользователь сразу заинтересуется, ведь так покупка книги становится более выгодной. Также есть и другие плюсы, которые рассмотрим ниже:

– красивая картинка для каждой книги на сайте делает более привлекательным покупку книг именно на этом сайте;

– как и говорили ранее, скидки и акции являются главным преимуществом этого приложения;

– подбор книг на основе ранее сделанных заказов, это помогает не искать и не тратить время на подбор, ведь сайт подберет все необходимое за вас.

Интерфейс онлайн сервиса «bookvoed.ru» представлен на рисунке (5, 6.).

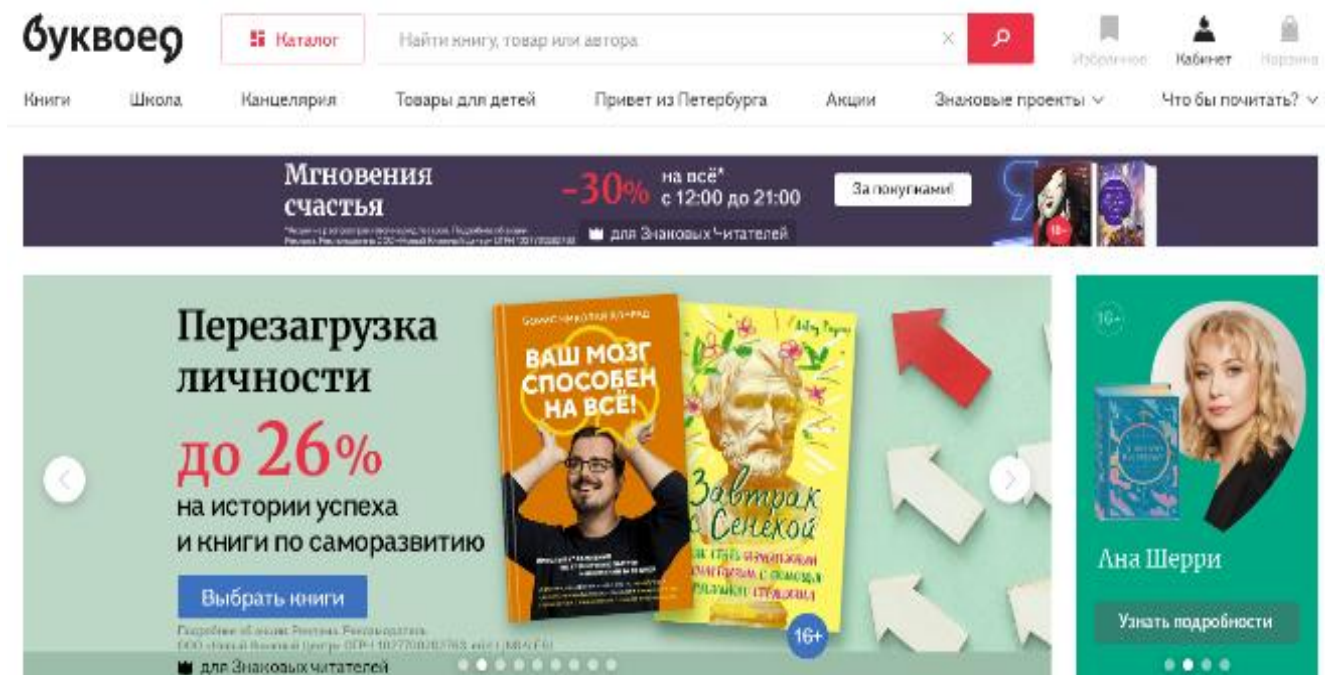


Рисунок 5 – Главная страница сервиса «bookvoed.ru»

Благодаря рассмотренным аналогам, можно сделать таблицу, в ней отражены все плюсы и минусы данных гигантов книжной индустрии. Ведь при разработке нашего приложения необходимо учесть это для успешной работы приложения.

Мгновения счастья

-30% на всё* с 12:00 до 21:00

За покупками!

*Некоторые товары не участвуют. Подробнее об акции. Рассылка. Рассылка с 30-дневным сроком возврата. 30-дневный возврат

👉 для Знаковых Читателей

Личный кабинет

Моя библиотека

Художественная литература

Книги для детей

Другие книжные жанры

Всё для школы

Подарки и сувениры

Товары для детей

Творчество и хобби

Канцелярия

Съедобное

Акции

Главная > Акции

Акции



Мгновения счастья

Скидка 30% на всё* с 12:00 до 21:00



Чемодан, Петербург, Буквоед!

Скидка 25% на путеводители и книги о здоровом образе жизни. Только для Знаковых Читателей



Бонусная программа стала ещё лучше



Перезагрузка личности

Скидки до 26% на истории успеха и книги по саморазвитию. Только для Знаковых Читателей

Рисунок 6 – Страница акций на клиенте

Таблица 1 – Достоинства и недостатки сайтов по продаже книг

Функционал	chitai-gorod.ru	book24.ru	bookvoed.ru
Поиск по критериям	+	+	+
Возможность покупки онлайн	+	+	+
Промо-коды на заказы книг	-	-	-
Акции и скидки	+	+	-
Возможность упрощенного заказа книг	-	+	-
Управление аккаунтом	+	-	+
Контактная информация	+	-	+
Понятный интерфейс	+	+	-

1.2.4 Вывод

Сделав анализ этих сайтов, было решено сделать сайт по продаже книг с названием "Книжный интернет-магазин". Здесь главным моментов являются интуитивно понятный интерфейс, удобный подбор, фильтры, приобретение и история заказов. Все это должно работать онлайн. В конечном итоге, пользователь должен быть доволен проделанной работой.

1.3 Постановка задачи

Web-приложение «Книжный интернет-магазин», нужно сделать следующий функционал:

На сервере:

- данные о книгах и пользователе;
- правильность вводимых данных;
- редактирование, добавление и удаление пользователя, а также книги в БД;

- разрешить приобретать книги только зарегистрированным пользователям.

На клиенте:

- интуитивно ясный интерфейс;
- регистрация и авторизация;
- работа фильтров, по которым пользователь может найти нужную книгу;
- добавление и редактирование данных у зарегистрированного пользователя;

- приобретение книг;

- история покупок.

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);

Веб-приложение делится на две части в разрезе пользователей:

- для неавторизованного пользователя;
- для авторизованного пользователя.

Незарегистрированный пользователь может только просматривать начальный интерфейс и книги, которые есть на сайте для заказа:

- просмотр книг;
- регистрация в приложении;
- вход в аккаунт.

Диаграмма вариантов использования приложения для неавторизованного пользователя представлена на рисунке 7.

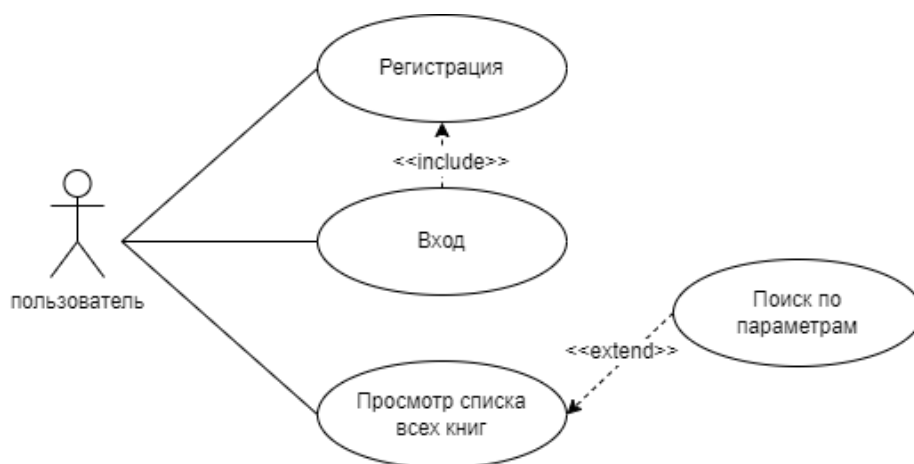


Рисунок 7 – UML диаграмма вариантов использования неавторизованного пользователя

Для зарегистрированных пользователей имеется больше возможностей. Они могут не только просматривать список книг, доступны на сайте, но и приобретать их книги прямо в браузере. Также они имеют возможность просматривать историю покупок, акции на определенные книги, и редактировать свою личную информацию:

- просмотр списка и приобретение, всех загруженных на сайт книг;
- просмотр истории приобретенных книг;
- просмотр книг по акции;
- редактирование своих данных;
- выход из аккаунта.

Диаграмма вариантов использования приложения для авторизованного пользователя представлена на рисунке 8.

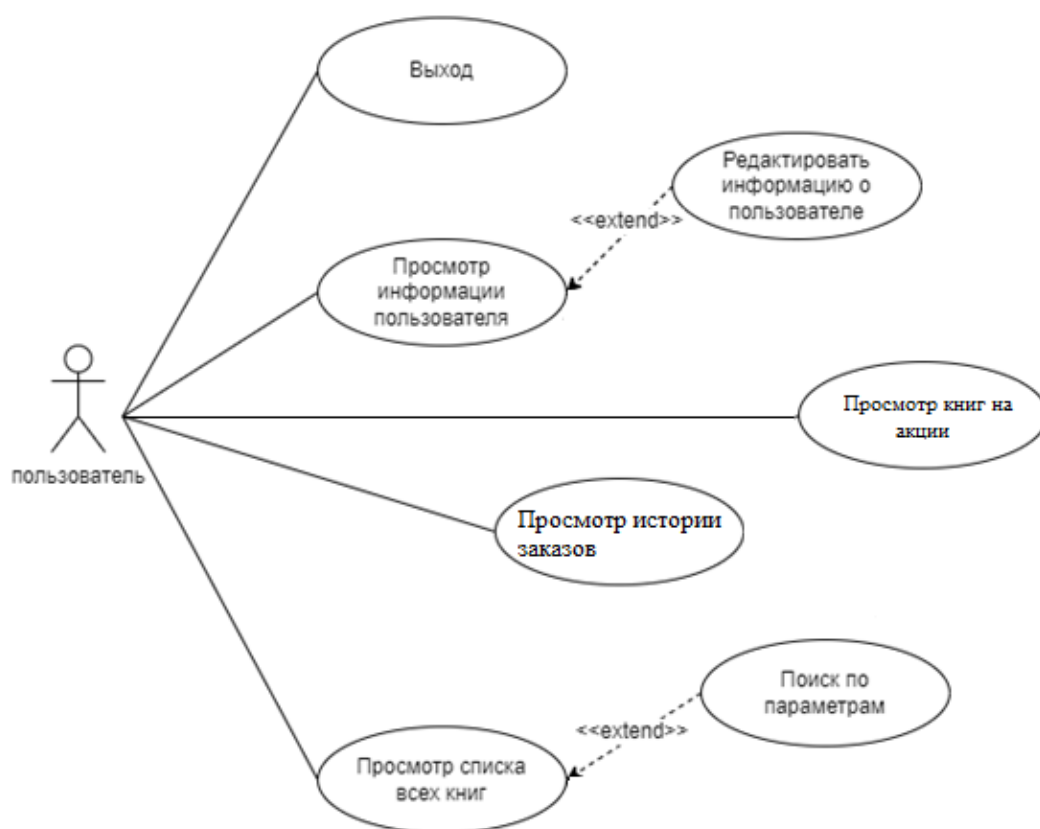


Рисунок 8 – UML диаграмма вариантов использования, авторизованного пользователя

2 Выбор технологий и средств разработки приложения

Здесь речь пойдет про технологии, которые необходимы для создания приложения "Книжный интернет-магазин". Далее сравним все популярные программы и сделаем выбор в пользу некоторых из них.

2.1 Выбор языка программирования

Соответственно для написания хорошего сайта, либо приложения нужен подходящий язык программирования. Здесь по всем критериям подходят Java, Python и C++. Конечно, для разработки нужна легкость в написании кода, быстрота действий, а также набор необходимых инструментов. Ниже будут представлены подробные примеры с описанием и набором средств:

– java. Считается лучшим вариантом написания сайтов на сегодняшний момент. Имеет в своем арсенале мощный вспомогательный инструмент Spring, который упрощает работу в разы. Но есть и нюанс, объемность кода. Помимо этого у Java есть очень много библиотек и инструментов, которые помогают и упрощают код, в плане создания веб-сайтов имеется очень хорошая безопасность и многопоточность, что не может не сказаться на работе приложения;

– python. Начнем сразу с минусов, и тут он есть. Очень важна табуляция, допустив где-то пробел, программа может не работать, но кому-то наоборот нравится. Также как и у Java имеется огромное кол-во библиотек, сервисов и т.д., помимо этого, несмотря на недостаток, легкость кода здесь есть и очень хорошая. Вернемся к минусам. Еще одним из них выступает производительность, все же она уступает Java;

– c++. Конечно, для создания сайтов не самый лучший вариант, но не рассмотреть мы его не можем. Начнем с того, что здесь нужно уметь управлять памятью, ведь неправильное распределение приведет к ошибкам. Да, производительность очень даже хорошая, но нужно правильно ее использовать. При всем этом, C++ тоже используют в написании сайтов, но для этого нужно хорошо владеть базовыми навыками.

Таблица 2 – Сравнение языков программирования для создания сайта

Язык	Плюсы	Минусы
Java	Платформенная независимость (работает на виртуальной машине Java) - Большая экосистема библиотек и фреймворков - Надежность и безопасность - Многопоточность и масштабируемость	Больше кода для реализации некоторых функций по сравнению с Python - Склонность к более высокому потреблению памяти по сравнению с Python
Python	Простота и читаемость кода - Большое количество библиотек для веб-разработки (например, Django, Flask) - Быстрая разработка благодаря динамической типизации	Интерпретируемость может снижать производительность по сравнению с компилируемыми языками - Ограничения при работе с многопоточностью
C++	Высокая производительность и эффективное управление ресурсами - Близость к аппаратным возможностям компьютера - Широкий спектр библиотек для работы с сетевыми приложениями	Сложность в сравнении с более высокоуровневыми языками (требует более тщательного управления памятью и более сложной структуры кода) - Меньшее количество готовых инструментов для веб-разработки

Несложно догадаться, но для создания сайта все будет использован Java язык программирования, ведь он имеет все оптимальные характеристики и нужные инструменты для создания сайта.

2.2 Выбор фреймворка

Так как был выбран язык Java, то самый лучший вариант фреймворка, конечно, будет Spring, но у него есть конкуренты, которые мы сейчас и рассмотрим:

– spring framework. Лучший выбор для создания сайтов по мнению многих разработчиков. В нем есть все, что необходимо для создания сайта, и инструменты, и безопасность, и большой выбор технологий, которые очень помогут при написании приложения. У него есть один огромный плюс – это Spring Boot, который позволяет легко и быстро создавать конфигурации буквально по нескольким строкам;

– hibernate. Второй по значимости фреймворк, который хорошо взаимодействует с Spring Framework, но тут нужно выбирать что-то одно. У него хорошо реализована работа с БД, настройка и взаимодействие;

– jsf. то фреймворк Java для создания веб-интерфейсов. JSF предоставляет компонентную модель разработки, которая упрощает создание веб-приложений с богатым пользовательским интерфейсом. JSF интегрируется с Java EE и поддерживает различные библиотеки компонентов.

Таблица 3 – Сравнение фреймворков

Фреймворк	Основные характеристики	Преимущества	Недостатки
Spring Framework	<ul style="list-style-type: none"> - Поддержка инверсии управления (IoC) и внедрения зависимостей (DI) - Мощный модуль Spring Boot для быстрого создания самостоятельных приложений - Поддержка аспектно-ориентированного программирования 	<ul style="list-style-type: none"> - Обширная экосистема с поддержкой множества инструментов и библиотек - Хорошая интеграция с другими технологиями, такими как Hibernate и Spring Security 	- Может показаться избыточно сложным для небольших проектов
Hibernate	<ul style="list-style-type: none"> - ORM (Object-Relational Mapping) для удобной работы с базами данных - Поддержка различных баз данных и управление персистентностью данных 	<ul style="list-style-type: none"> - Упрощение работы с реляционными базами данных, сокращение написания SQL-запросов 	- Может иметь небольшой накладной расход в производительности из-за генерации сложных SQL-запросов
JSF (JavaServer Faces)	- Компонентная модель разработки веб-интерфейсов с богатым набором	- Удобство создания интерактивных пользовательских интерфейсов,	- Некоторые разработчики могут считать JSF слишком сложным

	компонентов - Интеграция с другими технологиями Java EE и библиотеками	возможность повторного использования компонентов	и тяжеловесным для некоторых типов проектов
--	--	---	---

2.3 Spring JPA

Spring JPA - это модуль в составе Spring Framework, специально разработанный для упрощения взаимодействия приложений на Java с базами данных. Он предоставляет интуитивно понятные инструменты для работы с различными СУБД, делая процесс работы с данными более эффективным и удобным для разработчиков.

Основные возможности Spring JPA включают в себя использование аннотаций, которые помогают сопоставить объекты Java с таблицами в базе данных, устраняя необходимость в ручном написании SQL-запросов и уменьшая объем кода. Кроме того, механизмы репозитория предоставляют удобный интерфейс для работы с данными, позволяя разработчикам сосредоточиться на бизнес-логике приложения, а не на деталях взаимодействия с базой данных.

Одним из ключевых преимуществ Spring JPA является его встроенная поддержка управления транзакциями, что обеспечивает надежность и целостность данных в приложении. Это позволяет избежать проблем с параллельным доступом к данным и обеспечивает безопасность при выполнении транзакций.

2.4 Spring Security

Если поработать с Spring Security, то можно понять, что это очень удобная вещь для обеспечения безопасности сайта. Здесь хорошо настроена авторизация и регистрация пользователей, защита и предотвращение ошибок. Подробно рассмотрим их.

В авторизацию входят определенные правила и роли, в аутентификацию поддержка, форма и базовый метод, в защиту входит защита от XSS и CSRF атак, а также защита от атак на сесансы.

В нашей же программе используются следующие компоненты этого фреймворка:

- *authentication*. Используется для аутентификации;
- *authorization*. Используется для авторизации;
- *web security*. Используется для защиты.
- *config*. Используется для настройки Spring Security.

2.5 Spring MVC

Платформа Spring Web model-view-controller (MVC) разработана на основе DispatcherServlet, который отправляет запросы обработчикам с настраиваемыми сопоставлениями обработчиков, разрешением представления, языковым стандартом и разрешением темы, а также поддержкой загрузки файлов. Обработчик по умолчанию основан на аннотациях @Controller и @RequestMapping и предлагает широкий спектр гибких методов обработки. С появлением Spring 3.0 механизм @Controller также позволяет создавать веб-сайты и приложения RESTful с помощью аннотации @PathVariable и других функций.

В Spring Web MVC вы можете использовать любой объект в качестве объекта команды или поддержки формы; вам не нужно реализовывать интерфейс или базовый класс, специфичный для платформы. Привязка данных Spring очень гибка: например, он рассматривает несоответствия типов как ошибки проверки, которые могут быть оценены приложением, а не как системные ошибки. Таким образом, вам не нужно дублировать свойства ваших бизнес-объектов в виде простых нетипизированных строк в объектах формы просто для обработки недопустимых отправок или для правильного преобразования строк. Вместо этого часто предпочтительнее привязываться непосредственно к бизнес-объектам.

					ПГУ1.090301.06.001 ПЗ	Лист
						20
Изм.	Лист	№ докум.	Подпись	Дата		

Разрешение представления Spring чрезвычайно гибко. Контроллер обычно отвечает за подготовку карты модели с данными и выбор имени представления, но он также может записывать непосредственно в поток ответов и выполнять запрос. Разрешение имен представлений легко настраивается с помощью расширения файла или согласования типа содержимого заголовка Accept, имен компонентов, файла свойств или даже специальной реализации ViewResolver. Модель (М в MVC) представляет собой интерфейс карты, который позволяет полностью абстрагировать технологию представления. Вы можете напрямую интегрироваться с технологиями рендеринга на основе шаблонов, такими как JSP, Velocity и Freemarker, или напрямую генерировать XML, JSON, Atom и многие другие типы контента. Карта модели просто преобразуется в соответствующий формат, например атрибуты запроса JSP, модель шаблона Velocity.

2.6 Выбор базы данных

Все базы данных практически похожи, но нам нужно выбрать одну. Для примера приведем три самые популярные БД на данный момент: MySQL, PostgreSQL и Oracle Database. Естественно, у каждой из них есть плюсы и минусы, которые мы сейчас и рассмотрим.

Доступность:

- mysql. Имеет бесплатную лицензию, ее может использовать каждый пользователь без каких-либо проблем;
- postgresql. Также является бесплатной, но помимо этого у нее есть общество и форумов, где можно найти любые ответы;
- oracle database. Тут уже немного сложнее, ведь она имеет лицензию и, как правило, ее нужно дополнительно приобретать. Это как раз таки и отталкивается пользователей от использования именно этой БД.

Производительность:

					ПГУ1.090301.06.001 ПЗ	Лист
						21
Изм.	Лист	№ докум.	Подпись	Дата		

– mysql. Очень проста в своем использовании, подходит для простых запросов;

– postgresql. Тут уже немного сложнее, ведь она часто используется в больших компаниях для объемного хранения данных;

– oracle database. Так как тут нужно приобретать лицензию, то и БД сама по себе работает со сложными запросами и используется в больших компаниях.

Выше были приведены простые примеры сравнения, дальше будет составлена таблица 4 по всем остальным моментам, по которым уже более тщательно можно будет выбрать БД для нашего сайта.

Таблица 4 – Сравнение баз данных

Характеристика	MySQL	PostgreSQL	Oracle Database
Тип лицензии	Open Source (бесплатно)	Open Source (бесплатно)	Коммерческая
Модель данных	Реляционная	Реляционная	Реляционная
Поддержка транзакций	Да	Да	Да
Функциональные возможности	Базовые, подходит для веб-приложений	Расширенные, включая геоинформационные операции, JSONB и другие	Широкий набор, включая высокую доступность и анализ данных
Производительность	Хорошая для небольших до средних проектов	Очень хорошая, оптимизирована для сложных запросов	Высокая, особенно для больших предприятий
Стоимость лицензии	Бесплатно	Бесплатно	Платно

В ходе сравнения выбор пал на PostgreSQL потому, что он более оптимален для наших задач и более удобен в использовании.

2.7 pgAdmin4

По сути, PgAdmin4 является «дочкой» PostgreSQL, ведь без него работа в этой БД не представляется. Это, по сути, интерфейс для работы с БД, что делает

работу очень удобной и эффективной. Там можно управлять всеми процессами, объектами, изменять, добавлять и удалять их. Просматривать нужную информацию, делать запросы на определенные таблицы и т.д., просматривать состояние БД и мониторить соединение. При возникновении ошибок, это приложение также покажет в каком месте есть проблемы.

2.8 Выбор среды разработки

Вообще при написании на Java выбор пал на три самых популярных на сегодняшний день среды разработки: IntelliJ IDEA, Eclipse, NetBeans. У каждой из них есть как плюсы, так и минусы. Ниже рассмотрим более подробно их.

IntelliJ IDEA:

- самая популярная среда для разработки на Java на сегодняшний день;
- имеет авто дополнение кода, если это необходимо, отладку и прогон программы по шагам;
- все интуитивно понятно, удобный интерфейс и простота работы.

Eclipse:

- очень много расширений на маркетплейсах;
- можно развернуть в облаке;
- работает на всех платформах.

NetBeans:

- модульная архитектура для решения объемных задач;
- очень частое обновление;
- бесплатная программа, что не мало важно.

После рассмотрения всех программ, IntelliJ IDEA является более подходящей средой для создания сайта.

Таблица 5 – Сравнение сред разработки

Характеристика	IntelliJ IDEA	Eclipse	NetBeans
Тип лицензии	Коммерческая (есть бесплатная Community Edition)	Open Source	Open Source
Производительность	Высокая, отличная поддержка интеллектуального автозаполнения, анализа кода и отладки	Высокая, обширные возможности для настройки и расширений	Хорошая, удобная интеграция с платформой Java EE и JavaFX
Функциональность	Мощный инструмент с широким набором функций, интеграция с различными инструментами и технологиями	Обширный выбор плагинов и расширений, поддержка различных языков программирования	Поддержка разработки Java EE и JavaFX, интегрированные инструменты для быстрой разработки
Интеграция	Хорошая интеграция с системами управления версиями, сборщиками проектов, серверами приложений и другими инструментами	Обширная интеграция с различными инструментами и плагинами, поддержка платформы Eclipse Marketplace	Интеграция с серверами приложений, инструментами разработки Java EE
Настройка и расширяемость	Высокая настраиваемость, множество плагинов и расширений	Обширная настраиваемость, широкий выбор плагинов и расширений	Хорошая настраиваемость, поддержка плагинов и расширений
Сообщество и поддержка	Активное сообщество разработчиков, доступ к платной поддержке	Большое сообщество пользователей и разработчиков, бесплатная поддержка	Обширное сообщество, бесплатная поддержка

3 Архитектура Web-приложения «Книжный интернет-магазин»

Этот раздел описывает архитектуру клиент-серверного приложения и то, как работают выбранные нами ранее технологии между собой. MVC является в нашем проекте основным фундаментом для хорошей работы приложения.

Эта модель работает по принципу модели, контроллера и представления, как будет описано еще ниже. Все это разделяется на блоки и становится более удобно в пользовании как программисту, так и пользователю.

По сути модель - это определенное количество классов, которые хранят данные в БД и выдают права для пользователей. В нашем приложении там хранятся данные о книгах, пользователях и т.д.

За HTML шаблоны отвечает представление, само по себе название говорит за себя. Отражение и связь с пользователем – это и есть основная суть этого объекта.

Связующим же звеном является контроллер, по сути ключевой момент в приложении, именно он принимает, отдает, отправляет и выводит все сообщения для пользователя.

Контроллер выполняет роль промежуточного звена между моделью и представлением. Он принимает запросы от клиента, обрабатывает их, взаимодействует с моделью для получения необходимых данных и передает эти данные представлению для отображения пользователю.

Диаграмма компонентов представлена на рисунке 9.

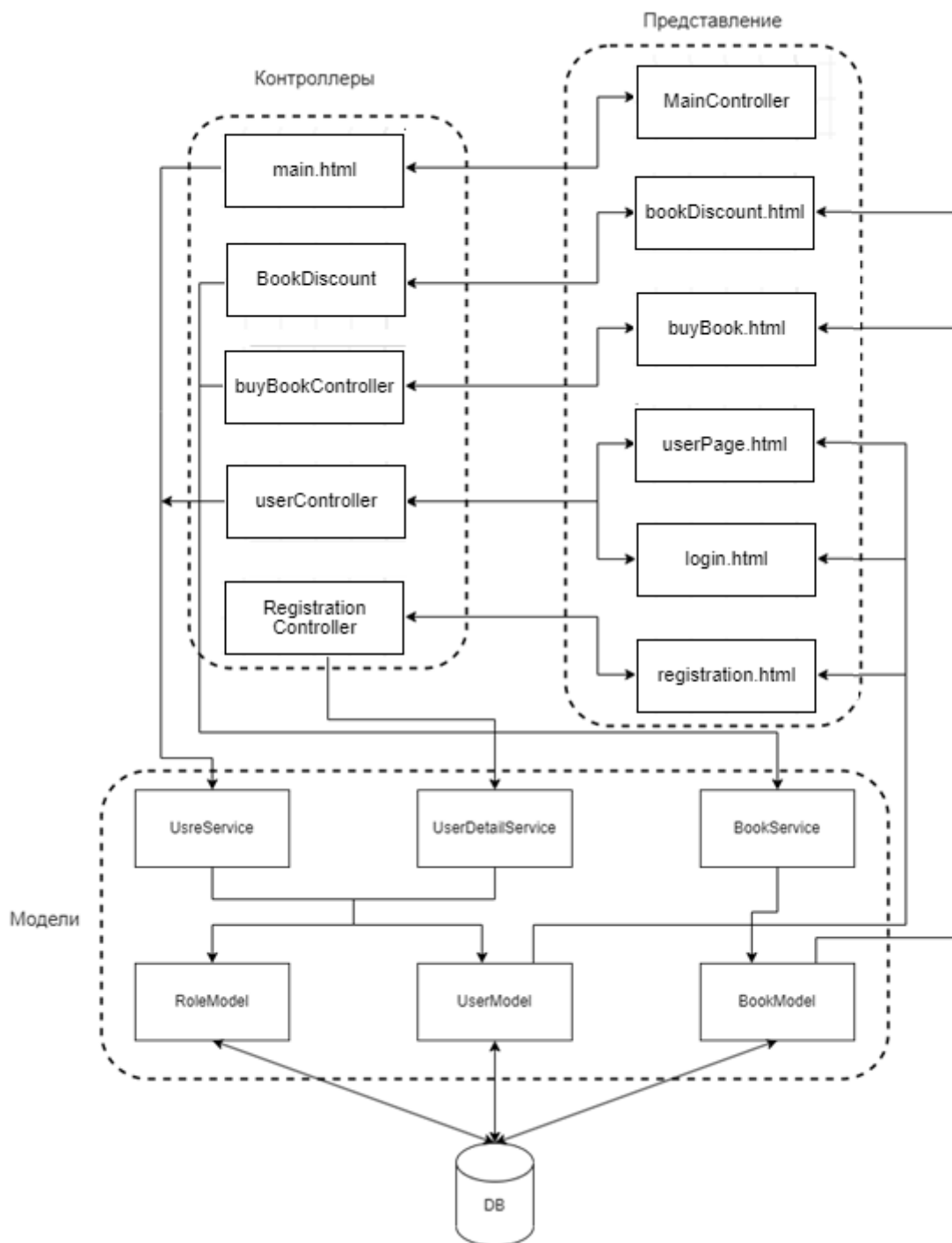


Рисунок 9 – UML диаграмма компонентов

Диаграмма классов представлена на рисунке 10.

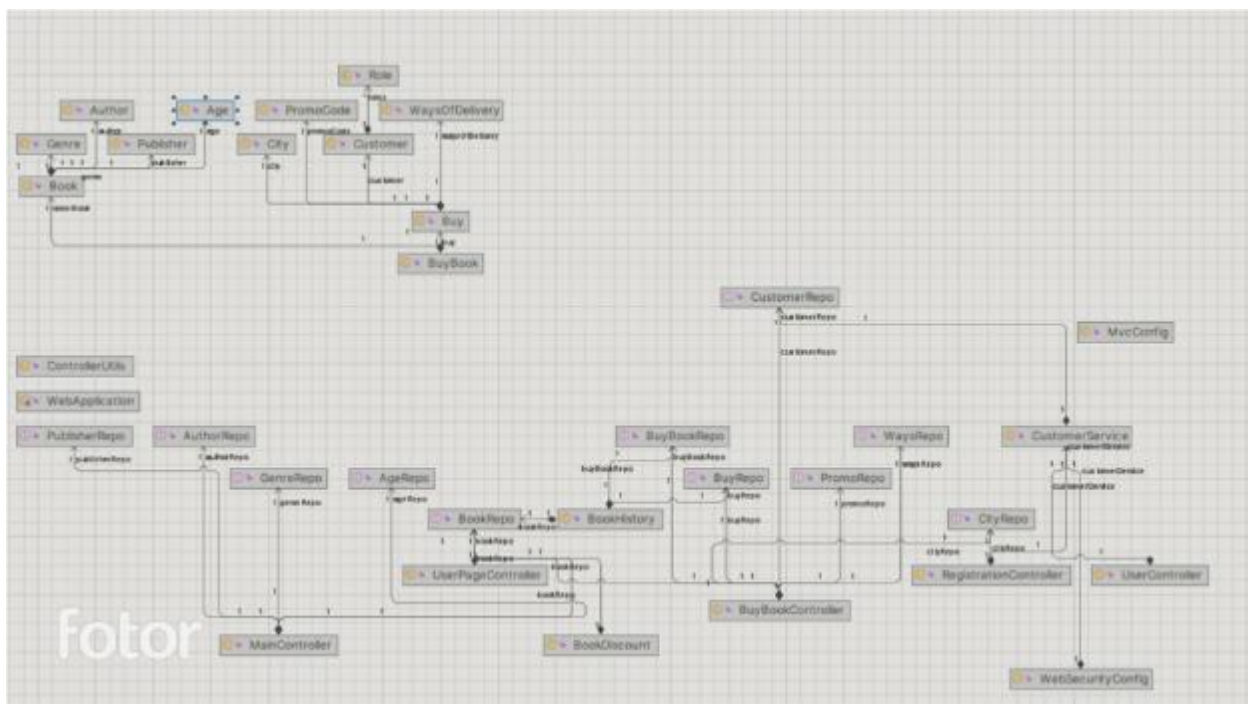


Рисунок 10 – UML диаграмма классов

Есть некоторые порядки для создания веб-приложения, приведем примеры ниже:

- на клиенте используется REST, HTML. Первый отвечает за архитектуру, второй за представление и удобство;
- на сервере естественно используется Java в совокупности с Spring для зависимостей. Помимо этого, Spring Boot, который сам настраивает конфигурацию и ускоряет работу приложения;
- связь БД обеспечивает JPA, которые настраивает удобство и легкое управление БД, в нашем случае PostgreSQL;
- Spring Security обеспечивает защиту данных пользователей и для предотвращения атак на них;
- сам сервер разворачивается Apache Tomcat, ведь это самый популярный и надежный сервис для без перебойной работы приложений.

Все технологии связаны и работают как единое целое, это и помогает работать нашему приложению правильно и без перебоев.

Последовательность выполняемых действий представлена на рисунке 11.

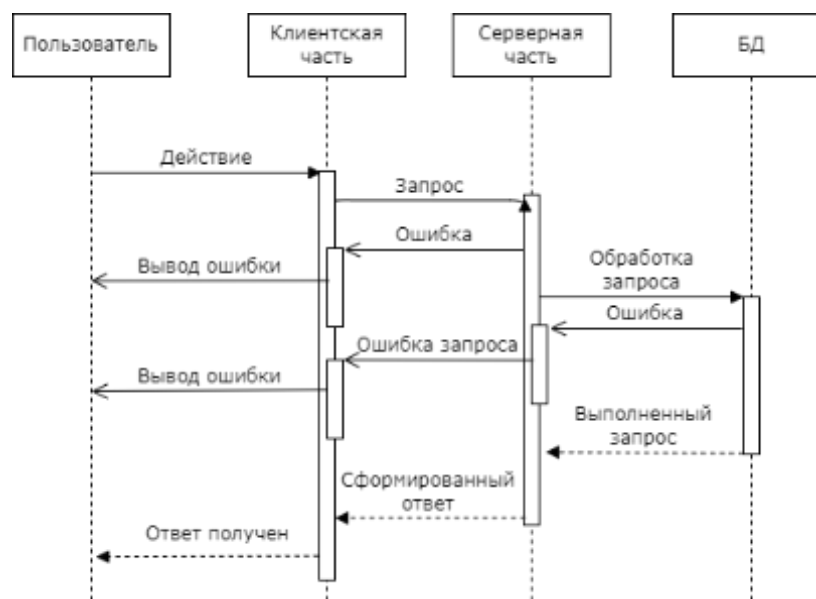


Рисунок 11 – UML диаграмма последовательности функционирования приложения

4 Реализация приложения

Реализация приложения самый главный момент в разработке. Суть заключается в презентации всех основных моментов приложения, подключения к БД, реализации, тестирования и показе всей логики. Как происходит покупка, применение промо-кодов, добавление, удаление и изменение данных. Таких моментов очень много и все это будет подробно описано в этом разделе.

4.1 Описание работы приложения

Начнем с отправки запросов на сервер. Допустим, пользователь нажимает на какую-то кнопку, после этого клиент создает запрос на сервер. Если в это время возникает проблемы, либо ошибки, то запрос обнуляется и выпадает в ошибку. Реализация сделана на HTML с CSS стилями и Thymeleaf. Это, что касается ошибки, но при успешном запросе связь создается с базой данных, где выполняет определенные операции. Пример таких операций – редактирование, удаление и добавление. Как обычно, если выполнение прошло, то идет возврат, и клиент получает ответ о завершении. При ошибки, кстати, также на клиент идет ответ об ошибке для последующего устранения.

Реализация и расположение всех компонентов представлены на диаграмме компонентов на рисунке 12.



Рисунок 12 – UML диаграмма развертывания системы

4.2 Программная реализация приложения

4.2.1 Протокол общения пользовательского клиента и сервера

Связь клиента с сервером была реализована с помощью MVC.

Это определенная схема данных, где все разделено на группы. Модель, контроллер, представление – это объекты этих данных. Если кратко расписать

все, то модель может менять состояние с помощью действий контроллера, контроллер делает уведомление модели о действиях пользователя, а представление выводит все при изменении двух других.

Далее поговорим про HTTP запрос, он общается с помощью `HandlerMapping`. Этот интерфейс как раз таки определяет логику работы между контроллерами. У контроллера есть свои GET и POST, это служебные методы, которые возвращают View.

Для упрощения понимания приведем пример конкретной ситуации общения клиента с сервером - приобретение книги. Когда клиент нажимает кнопку покупки книги, он переходит на страницу оформления заказа.

Контроллер для конкретного URL-адреса предоставляет методы GET и POST. С помощью метода GET в модель из базы данных заносятся все города и способы доставки, для дальнейшего отображения пользователю на странице оформления заказа. После заполнения пользователем всей информации и нажатия кнопки оплаты заказа происходит переход к методу POST. В нем все введенные пользователем данные заносятся в таблицу информации о покупке с применением созданных интерфейсов-репозиториев. Далее высчитывается сумма заказа, с применением возможной скидки и эти данные с текущей датой заказа заносятся в итоговую таблицу заказа. После этого пользователь перенаправляется на страницу о завершении покупки, содержащую номер заказа.

Все остальные действия, которые может выполнять пользователь (например редактирование профиля) происходят по подобному сценарию, описанному выше.

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

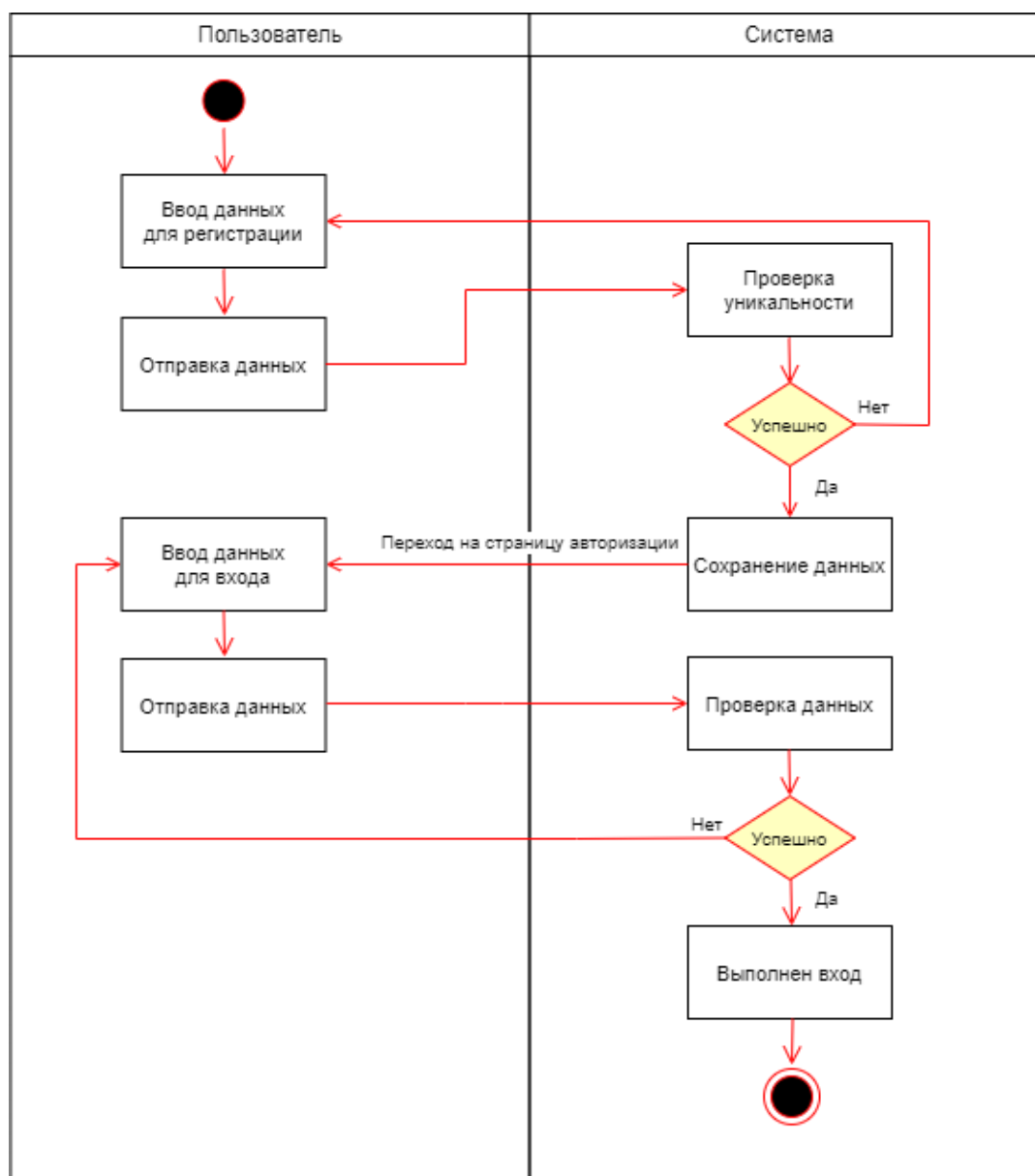


Рисунок 13 – UML диаграмма деятельности регистрации и авторизации

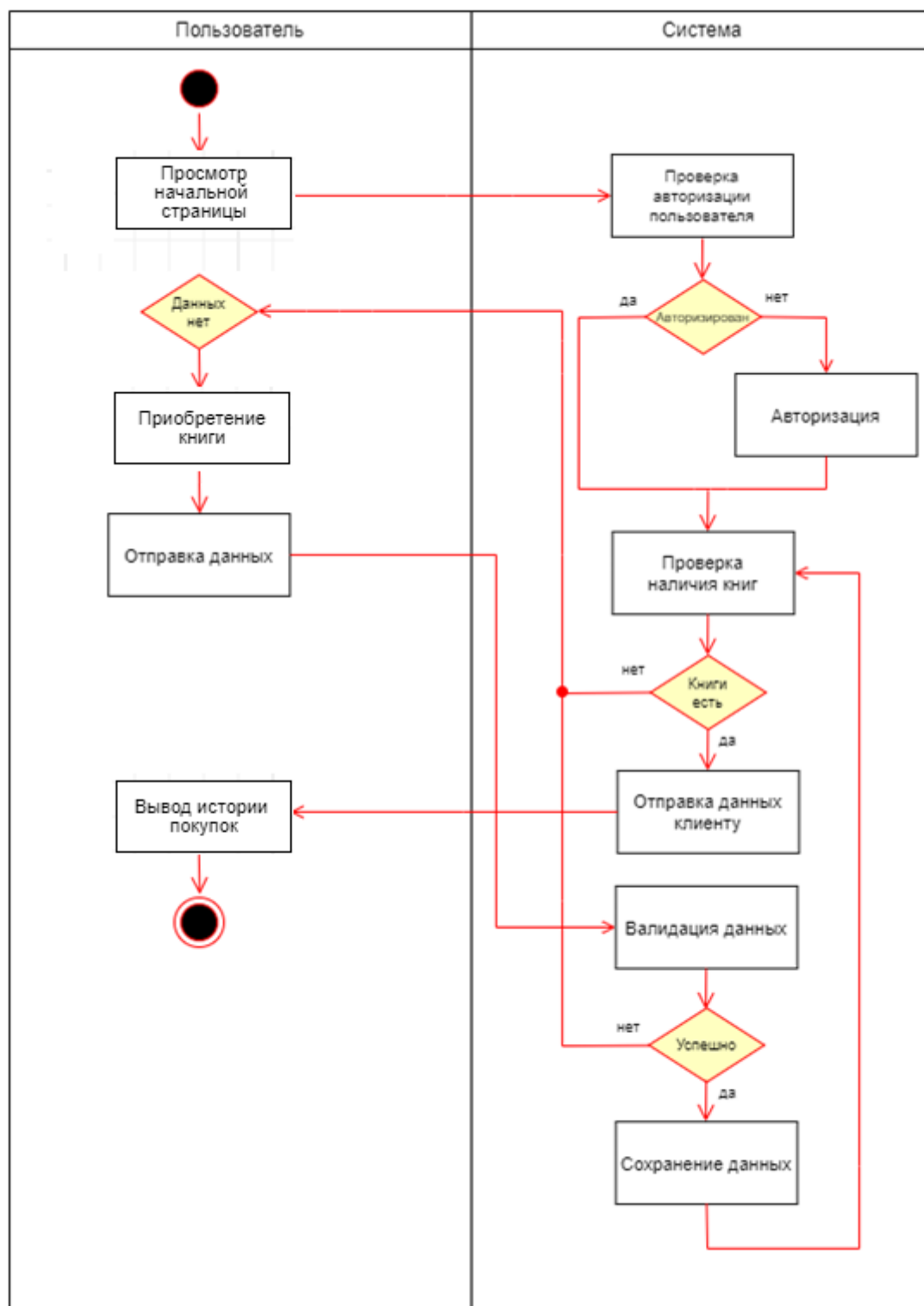


Рисунок 14 – UML диаграмма деятельности просмотра истории покупок и приобретения книг

4.2.2 Протокол общения клиента-администратора системы и сервера

Общение клиента-админа с сервером практически не отличается от общения обычного пользователя. Но у администратора есть собственная страница, к которой закрыт доступ обычным пользователям. Разграничение доступа происходит с помощью аннотации `@PreAuthorize("hasAuthority('ADMIN')")`. Таким образом указывается, что только пользователь с ролью администратора может получить доступ к данной странице. Если на такую страницу попытается зайти обычный пользователь, вернется ошибка прав доступа 403. Администратор может редактировать роли пользователей, то есть назначать новых клиентов-администраторов, а также редактировать количество книг в наличии. Если количество книг будет установлено в 0, книга удалится. Пример представлен на листинге 1 ниже.

```
@PreAuthorize("hasAuthority('ADMIN')")
@GetMapping("{customer}")
public String userEditForm(@PathVariable Customer customer,
Model model){
    model.addAttribute("user", customer);
    model.addAttribute("roles", Role.values());
    return "userEdit";
}
```

В методе в модель добавляются пользователи и их роли, далее они отображаются на странице редактирования.

4.2.3 Протокол общения сервера и базы данных

Каждая таблица базы данных представлена в виде класса. Аннотация `@Entity` позволяет представлять таблицу в виде Java-классов. Поля повторяют поля таблиц на стороне БД. Пример представления таблицы на листинге 2.

```
@Entity
@Table(name = "author")
@Data
public class Author{
    @Id
    @Column(name = "author_id")
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long author_id;
    @Column(name = "author_name")
    private String author_name;
    public Author() {
    }
    public Author(String author_name) {
```

```

        this.author_name = author_name;
    }
}

```

Построение запросов из сервера к базе реализовано через специальные интерфейсы-репозитории. Для каждой сущности создаются отдельные репозитории с методами без тела, а создает эти методы во время компиляции Spring Data.

Такие интерфейсы помечаются специальной аннотацией `@Repository`. Эта аннотация используется в классах Java, которые напрямую обращаются к базе данных. `@Repository` работает как маркер для любого класса, который выполняет роль хранилища или доступ к данным объекта. Эта аннотация имеет функцию автоматического перевода. Например, когда возникает исключение в `@Repository`, существует обработчик для этого исключения, и нет необходимости добавлять блок try-catch.

Все интерфейсы - репозитории наследуют интерфейс `JpaRepository`. `JpaRepository<сущность, тип_её_ID>` — включает все заготовленные запросы, чтобы обращаться к определённой сущности. Пример использования запросов, предоставленных `JpaRepository`, представлен на листинге 3 ниже.

```

@Repository
public interface PromoRepo extends JpaRepository<PromoCode,
Long> {
    PromoCode findPromoCodeByCode(String code);
}

```

На вход запроса приходит строка, по которой запрос ищет в базе данных соответствующий промо-код, если такой не найден возвращается null.

Также при помощи аннотации `@Query` можно писать собственные запросы на языке SQL. Подобный пример приведен на листинге 4 ниже.

```

@Repository
public interface AuthorRepo extends JpaRepository<Author, Long>
{
    @Query("select a from Author a where a.author_name=?1")
    Author findAuthorByAuthor_name(String name);
}

```

Написанный вручную запрос выполняет поиск автора по его ФИО.

4.3 Структура БД

Для создания клиент-серверного приложения, которое будет хранить информацию о пользователях и книгах, нужно сначала определить, какие данные будут храниться в базе данных и как они будут связаны между собой. Поскольку структура базы данных существенно влияет на ее эффективность и функциональность, в этом разделе рассматриваются основные таблицы, их атрибуты и связи, которые были разработаны специально для этого проекта.

После проведенного анализа было решено использовать следующие таблицы в приложении:

– таблица ages;

Таблица для хранения возрастных ограничений. Содержит уникальный идентификатор - первичный ключ, возрастное ограничение.

Все последующие таблицы хранят уникальный идентификатор - первичный ключ.

– таблица author;

Содержит ФИО автора книг.

– таблица publisher;

Содержит название издательства.

– таблица genre;

Содержит название жанра книг.

– таблица book;

Таблица хранит полную информацию о книге и внешними ключами связана с таблицей авторов, жанров, издательства и возрастных ограничений.

Также в ней содержится информация о названии, isbn (международный стандартный номер книги), годе публикации, цене и количестве книг в наличии. Все поля таблицы не могут быть нулевыми.

Если количество книг становится равным нулю, срабатывает триггер на удаление книги. Данная функция вызывается при срабатывании триггера amount_del.

– таблица roles;

Таблица ролей связана с таблицей пользователей внешним ключом. Содержит в себе уникальный идентификатор пользователя и его роль. Возможные роли: USER, ADMIN

– таблица customer;

Содержит информацию о пользователе - его ФИО, email и пароль

– таблица city;

Название городов, в которые осуществляется доставка.

– таблица promo_code;

Таблица с промо-кодами, дающими скидки на покупку. Содержит в себе название промо-кода и размер скидки, которую он предоставляет.

Эту таблицу наследует таблица personalized_promo_code, которая содержит в себе уникальных идентификатор пользователя, владеющего этим промо-кодом.

– таблица way_of_delivery;

Способы доставки книг и их стоимость.

– таблица buy;

Таблица, содержащая в себе информацию о покупке, имеет составной первичный ключ, в котором содержатся уникальный идентификатор заказа и дата его оформления. Внешними ключами связана с таблицей пользователей, городов, промо-кодов и способов доставки. В данной таблице происходит секционирование по дате оформления заказа. Создаются секции по месяцу заказа.

Также содержит комментарии к заказу и статус оплаты.

– таблица buy_book.

Внешними ключами связана с таблицей книг и информации о покупке. Фактически является итоговой таблицей заказа, объединяющей в себе информацию о покупателе и книге, которую он приобретает. Также содержит в себе дату оформления заказа и итоговую сумму покупки.

На рисунке 15 описана схема базы данных, разрабатываемого приложения, для наглядного рассмотрения взаимодействия таблиц.

					ПГУ1.090301.06.001 ПЗ	Лист
						37
Изм.	Лист	№ докум.	Подпись	Дата		

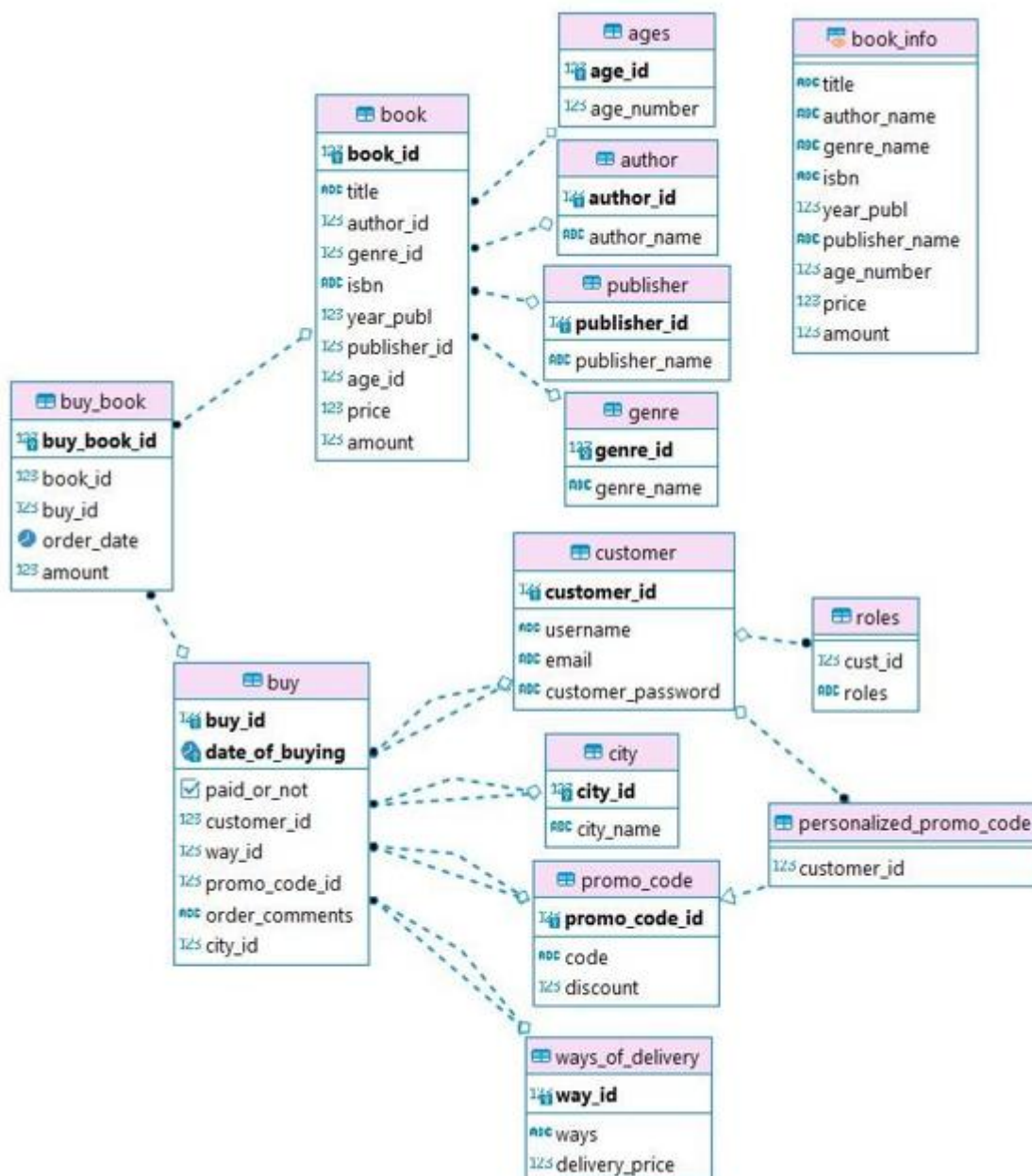


Рисунок 15 – Схема БД

4.4 Структура приложения

После создания базы данных происходит подключение к ней из сервера через localhost:5432. После запуска приложения происходит запуск сервера, и он начинает "слушать" порт 8075. Далее, введя в поисковую строку браузера localhost:8075, можно перейти на сайт книжного магазина. Если же сервер не был

запущен или произошла ошибка, обращение к данному порту не даст никакой информации.

Неавторизованный пользователь увидит лишь начальную страницу без возможности приобретения книги. Новый пользователь может пройти регистрацию и, если введенный e-mail не был найден в базе данных, новый клиент будет зарегистрирован. Необходимо пройти авторизацию, и если все прошло успешно, произойдет переход на страницу с возможностью покупки книги. При всех дальнейших действиях происходит отправка HTTP-запросов на сервер. При помощи контроллеров для конкретных URL-адресов происходит обработка данных запросов, и пользователь видит в браузере полученную информацию.

Всего используется 5 основных контроллеров:

- RegistrationController - обработка всех действий, связанных с регистрацией;
- BuyBookController - покупка книг (вывод пользователю информации на странице оформления покупки, добавление заказа в базу данных и др.);
- MainController - контроллер URL-адресов, по которым админ может добавлять книги и редактировать их количество;
- UserController - изменение администратором ролей пользователей, а также редактирование пользователями личных данных;
- UserPageController - вывод пользователю списка книг, а также поиск по заданным параметрам.

Основные алгоритмы работы приложения:

- регистрация и авторизация;

Происходит с использованием Spring Security. При регистрации нового пользователя, производится проверка на существование такого пользователя в базе данных по email. Если метод addCustomer возвращает false, выводится сообщение, что такой пользователь уже существует. Иначе пользователь добавляется в базу данных, ему назначается роль USER.

При авторизации посредством метода loadUserByUsername в базе данных ищется пользователь. Если найден - переход на главную страницу, иначе сообщение, что данный пользователь не найден.

– добавление новой книги;

Входными параметрами в метод add являются все поля книги (название, автор, цена и т.д.). Каждый параметр проверяется на корректность (например, не отрицательность цены и возрастного ограничения). Поля автор, издательство, возрастные ограничение и жанр сначала ищутся в своих таблицах при помощи репозитория, если не найдены, то они добавляются туда и возвращается объект каждой сущности. Далее книга создается, добавляется в базу данных и выводится на странице. В случае некорректности или не заполнения какого-либо поля выводится сообщение об ошибке.

– редактирование количества книг;

Входными параметрами в метод updateAmount является id выбранной для редактирования книги и новое количество экземпляров. Книга ищется в базе данных по id и ей устанавливается новое количество. В случае установки количества равным 0, книга сразу же удалится.

– изменение данных в личном кабинете;

На вход в метод updateProfile подается новый e-mail и пароль. Происходит проверка на равенство старого e-mail новому, и в случае их различия, новая почта пользователя сохраняется. Новый пароль сразу заносится зашифрованным. И обновленные данные о пользователе сохраняются в базу данных.

– покупка книги.

Входные параметры в метод buy: город доставки, комментарий к заказу, промо-код, способ доставки, а также id покупаемой книги и имя пользователя. Книга ищется по id в базе данных, также ищутся город, способ доставки и промо-код по строке. Создается объект информации о покупке.

Затем считается итоговая сумма заказа. В случае правильности введенного промо-кода, применяется скидка. Итоговые данные передаются в базу данных. Пользователю выводится номер заказа и его финальная стоимость.

					ПГУ1.090301.06.001 ПЗ	Лист
						40
Изм.	Лист	№ докум.	Подпись	Дата		

5 Тестирование приложения

Для проверки работоспособности клиент-серверного приложения «Книжный интернет-магазин» необходимо провести его тестирование.

Тестирование будет состоять из следующих действий:

- регистрация пользователя;
- авторизация пользователя;
- приобретение книг, просмотр истории покупок и вкладки «Акции»;
- редактирование аккаунта пользователя;
- проверка функционала неавторизованных пользователей.

5.1 Регистрация пользователя

Чтобы протестировать процесс регистрации пользователя, нужно перейти на страницу с формой регистрации и ввести запрошенную информацию: имя, фамилию, отчество, город проживания, логин и пароль. После отправки запроса на добавление нового пользователя, в базе данных должна появиться запись о нем.

Ввод данных в форму для регистрации приведен на рисунке 16.

Регистрация

Имя:

Фамилия:

Отчество:

E-mail:

Пароль:

Город:

Contact us

bookforeveryone@mail.ru +7(999) 555-555-8

Рисунок 16 – Регистрация пользователя

Если попытаться зарегистрировать пользователя с e-mail, который уже используется в базе данных, система выдаст сообщение об этой ошибке и предложит ввести другие данные. Этот процесс попытки регистрации с уже существующим логином и отображение соответствующего сообщения изображен на рисунках 17 и 18.

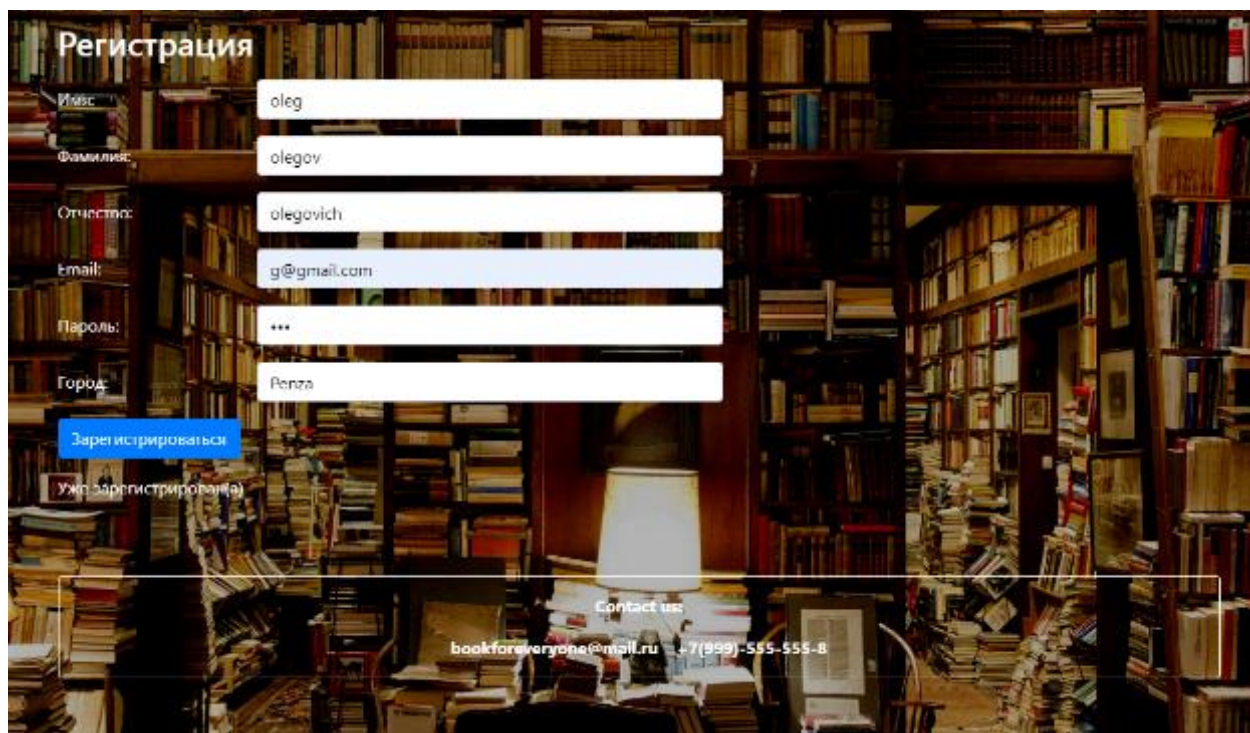


Рисунок 17 – Регистрация пользователя с существующим e-mail в БД

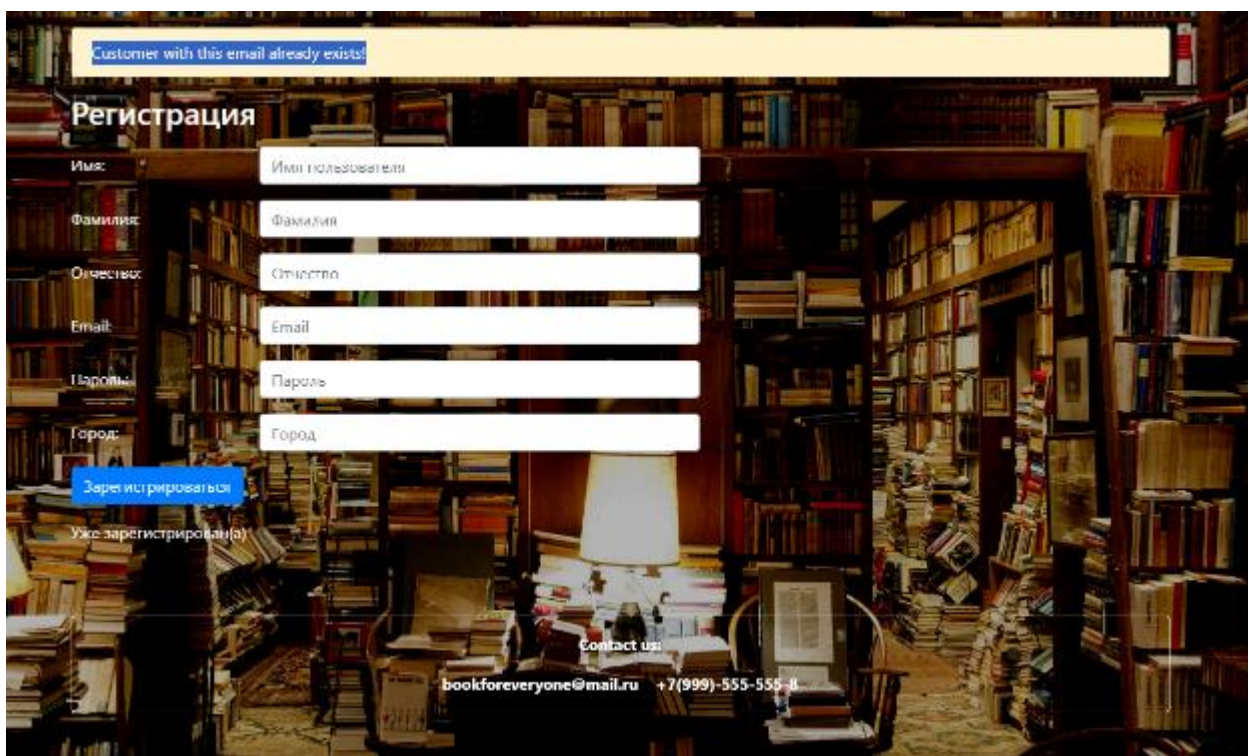


Рисунок 18 – Вывод сообщения

5.2 Авторизация пользователя

При попытке входа в систему пользователь вводит данные, которые ранее были добавлены в базу данных. Если эти данные совпадают с теми, что хранятся в базе данных, вход будет успешным. В случае несовпадения данных пользователю будет показано сообщение об ошибке.

На рисунке 19 изображен сценарий тестирования с ошибочными данными, а результат входа с неправильными данными показан на рисунке 20.



Рисунок 19 – Ввод неправильных данных

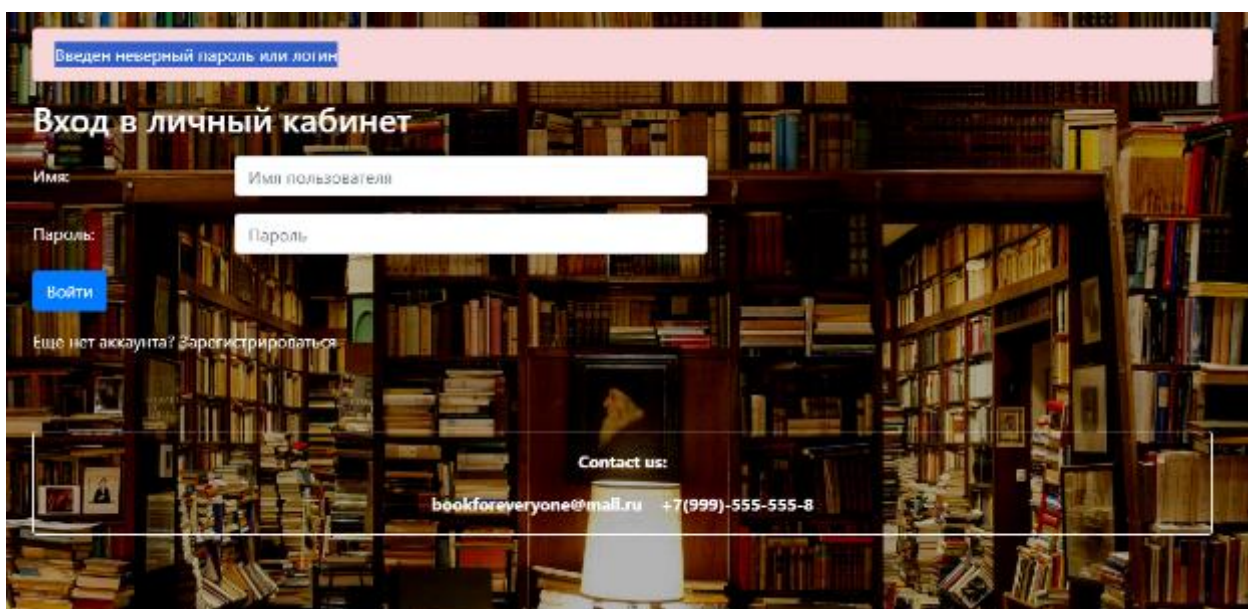


Рисунок 20 – Сообщение об ошибке

Теперь введем правильные данные. На рисунке 21 показан сценарий тестирования с верными данными, а результат успешного входа представлен на рисунке 20.



Рисунок 21 – Ввод верных данных

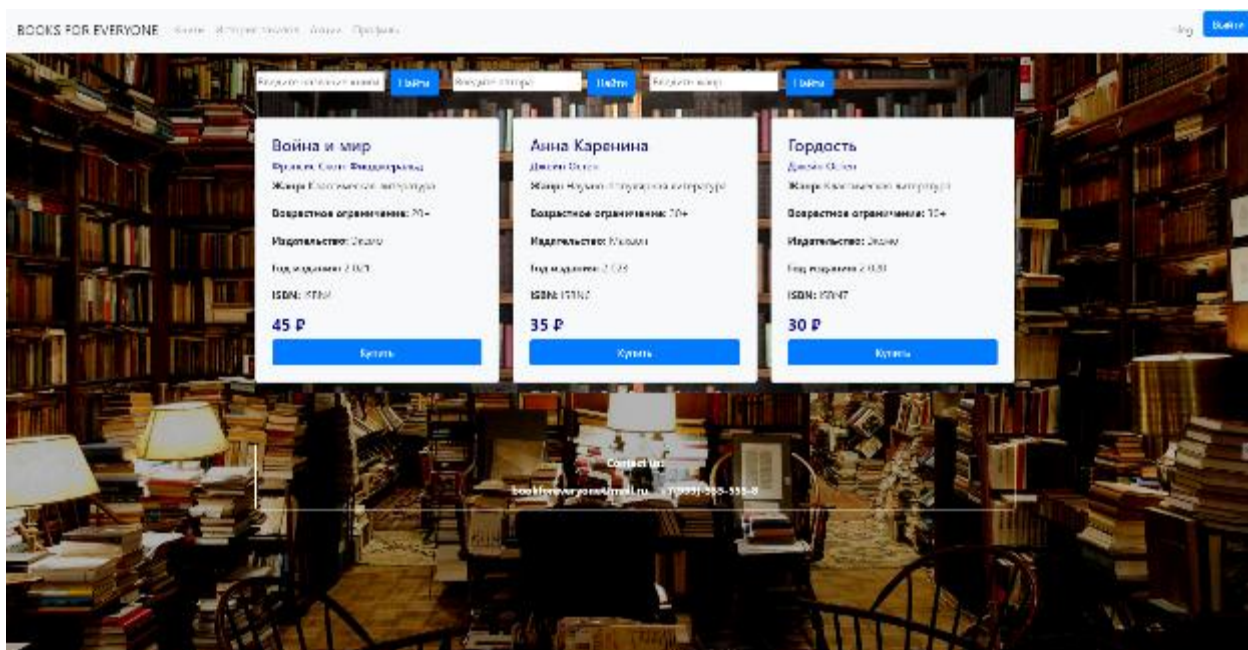


Рисунок 22 – Результат авторизации

В процессе тестирования все сценарии завершились успешно. При вводе неправильных данных пользователю выводилось сообщение об ошибке. При правильном вводе данных происходил переход на главную страницу, где отображалось имя пользователя, который вошел в систему.

5.3 Тестирование функционала взаимодействия с книгами

Авторизованным пользователям доступен функционал просмотра и приобретения книг. Тестовый сценарий состоит из перехода на страницу покупки, где можно будет ввести данные заказа, после ввода данных будет выведено сообщение о успешном заказе. Также есть функционал просмотра приобретенных книг. Помимо этого, авторизованный пользователь может приобрести книги по акции. Для выполнения данного действия нужно перейти на вкладку Акции, где будут отображены доступные книги.

На рисунке 23 отображен результат после нажатия кнопки «Купить»



Выберите город

Адрес доставки

Комментарии к заказу

Промокод

Выберите способ доставки

Оплатить

Contact us:
bookforeveryone@mail.ru +7(999) 555 555-8

Рисунок 23 – Страница ввода данных для заказа

Также имеется выбор способа доставки, который представлен на рисунке

24

Рисунок 24 – Выбор способа доставки

Если у пользователя имеется промо-код, он может добавить его и применится автоматическая скидка на заказ

Рисунок 25 – Примененный промо-код

После ввода всех данных будет выведено сообщение, показывающее информацию о доставке, успешно примененный промо-код, благодарность за заказ и возможность перехода на главную страницу.



Рисунок 26 – Информация о сделанном заказе

После перехода пользователь может перейти на вкладку «История заказов», где будет информация о ранее созданных заказах.

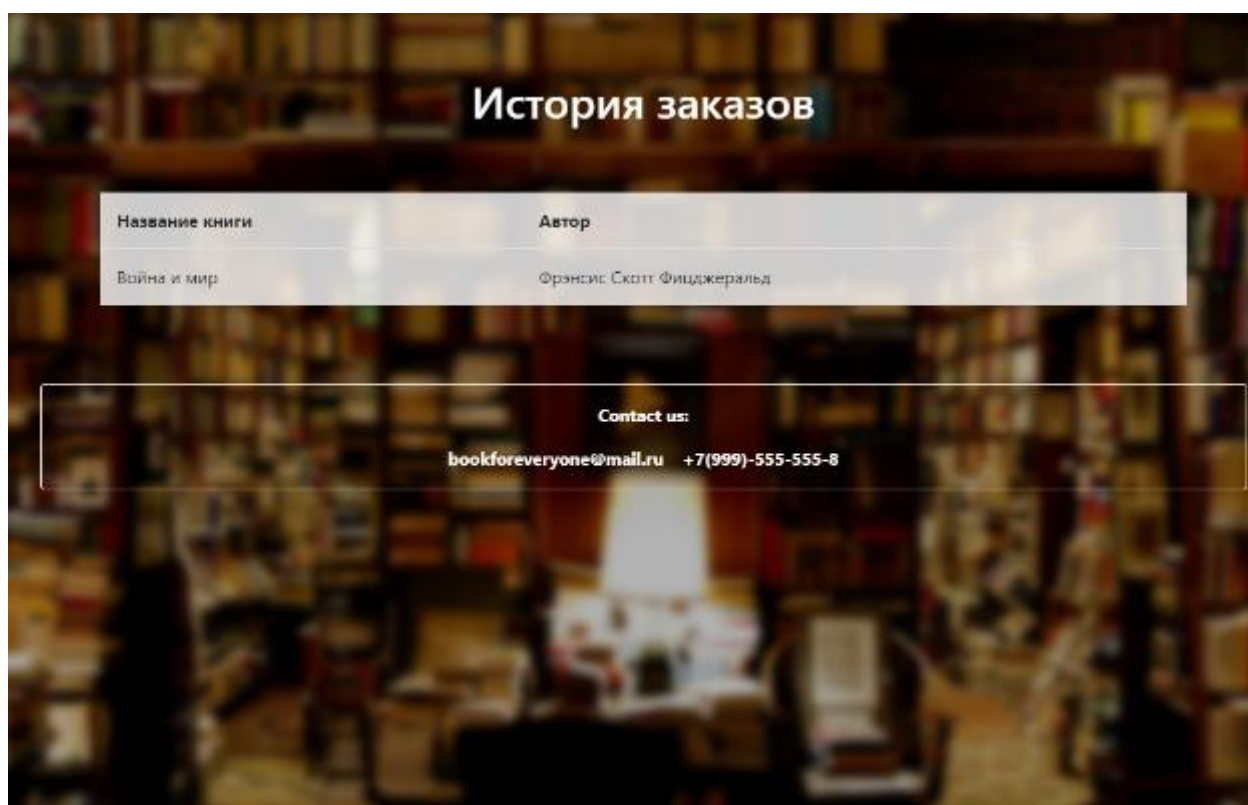


Рисунок 27 – Информация о сделанных заказах

Также авторизированный пользователь может подобрать необходимую книгу по определенным критериям. Для этого нужно ввести в строку критериев ключевые слова для поиска этих книг.

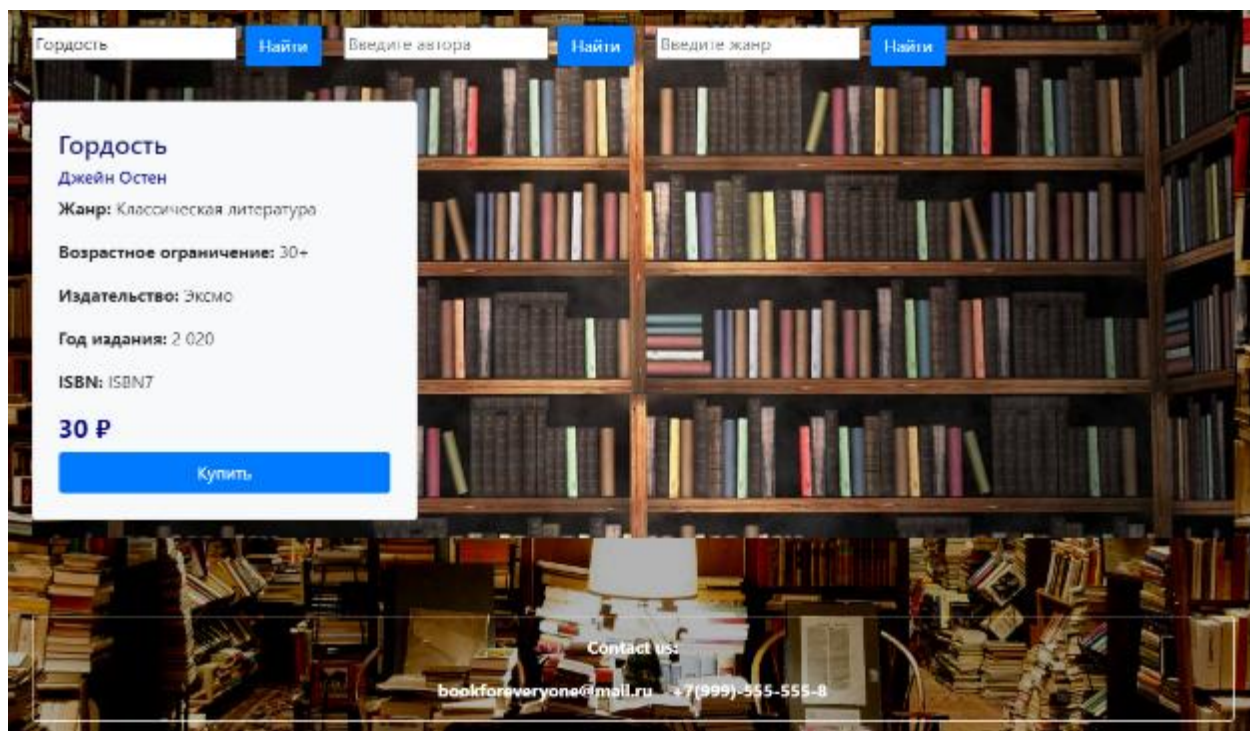


Рисунок 28 – Подбор книг по определенным критериям

На вкладке акции у авторизованного пользователя будет список книг, доступный на данный момент по скидке. Пользователь точно также может выбрать и приобрести любую понравившуюся книгу. Данный функционал показан на рисунке 29.

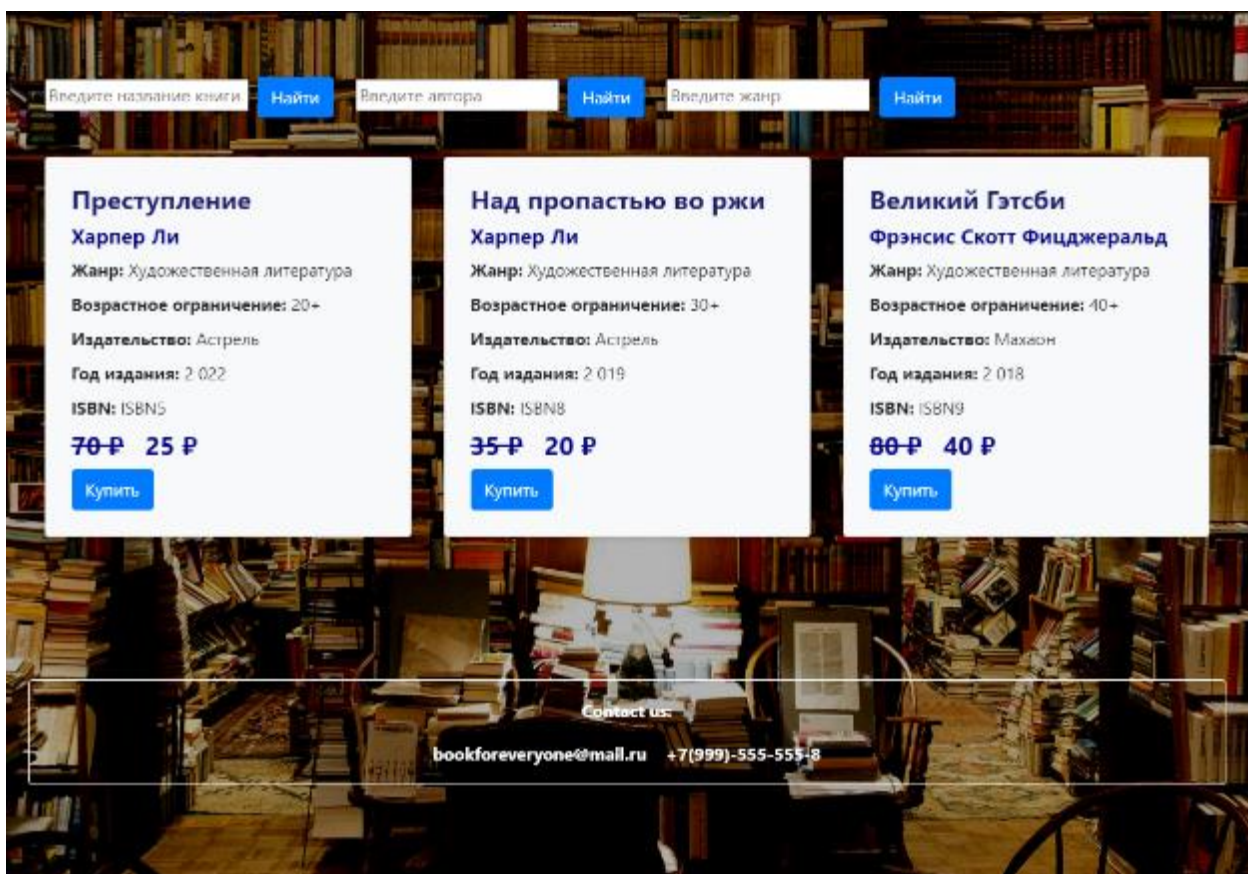


Рисунок 29 – Вкладка «Акции»

После выполнения всех шагов ожидаемый результат совпал с реальным, что подтверждает успешное прохождение тестирования этих функций.

5.4 Редактирование данных пользователя

При переходе на вкладку «Профиль», пользователь может изменить собственную информацию о своем профиле и нажать кнопку «Сохранить» для применения их.

oleg
 Имя: oleg
 Фамилия: olegov
 Отчество: olegovich
 Email: g3@gmail.com
 Пароль: Пароль
 Город: Penza
 Сохранить

Contact us:
 bookforeveryone@mail.ru +7(999)-555-555-8

Рисунок 30 – Данные пользователя

oleg
 Имя: artem
 Фамилия: Ermolacv
 Отчество: Dmitrievich
 Email: g3@gmail.com
 Пароль: 111
 Город: Penza
 Сохранить

Contact us:
 bookforeveryone@mail.ru +7(999)-555-555-8

Рисунок 31 – Новые данные пользователя

После нажатия кнопки «Сохранить» на рисунке 31 можно увидеть, что имя справа сверху изменилось на актуальное.

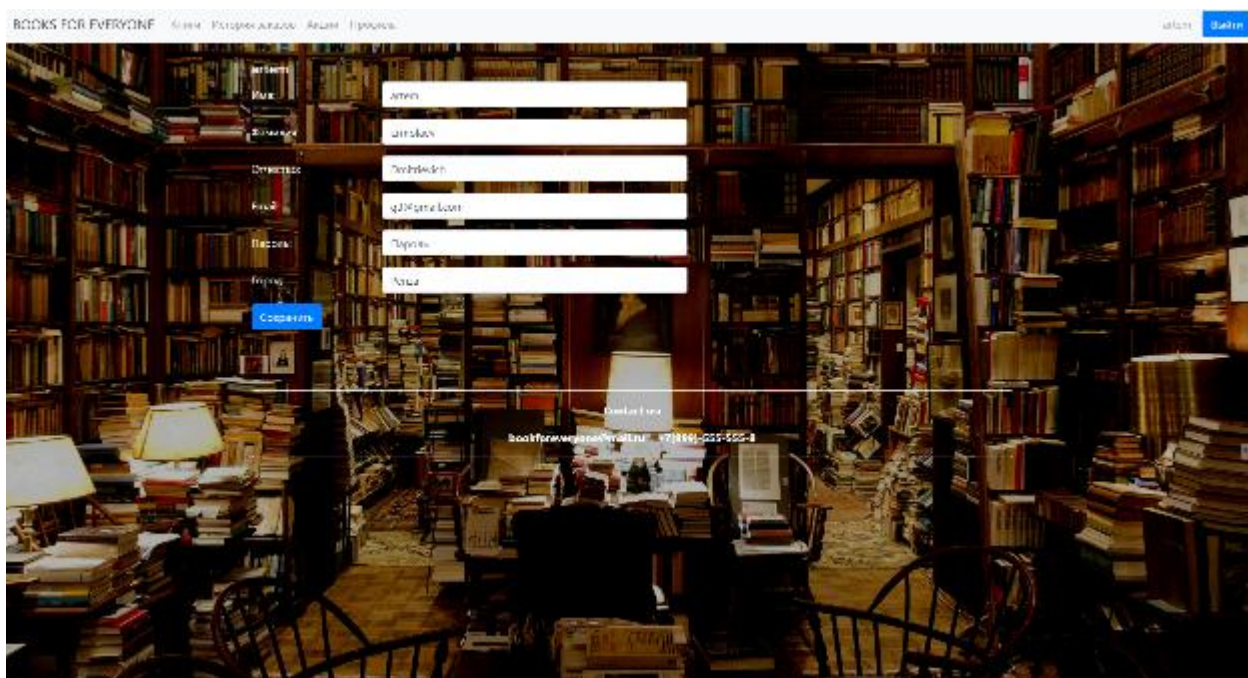


Рисунок 32 – Новые данные пользователя применились

5.5 Функционал неавторизированного пользователя

Пользователь, который не вошел в систему, имеет доступ только к страницам «Главная», «Книги» и информации «Contact us:». На странице «Книги» у него ограниченный доступ: он может просматривать только список книг, доступный для покупки, но не может приобретать их. Данный функционал показан на рисунке 32.

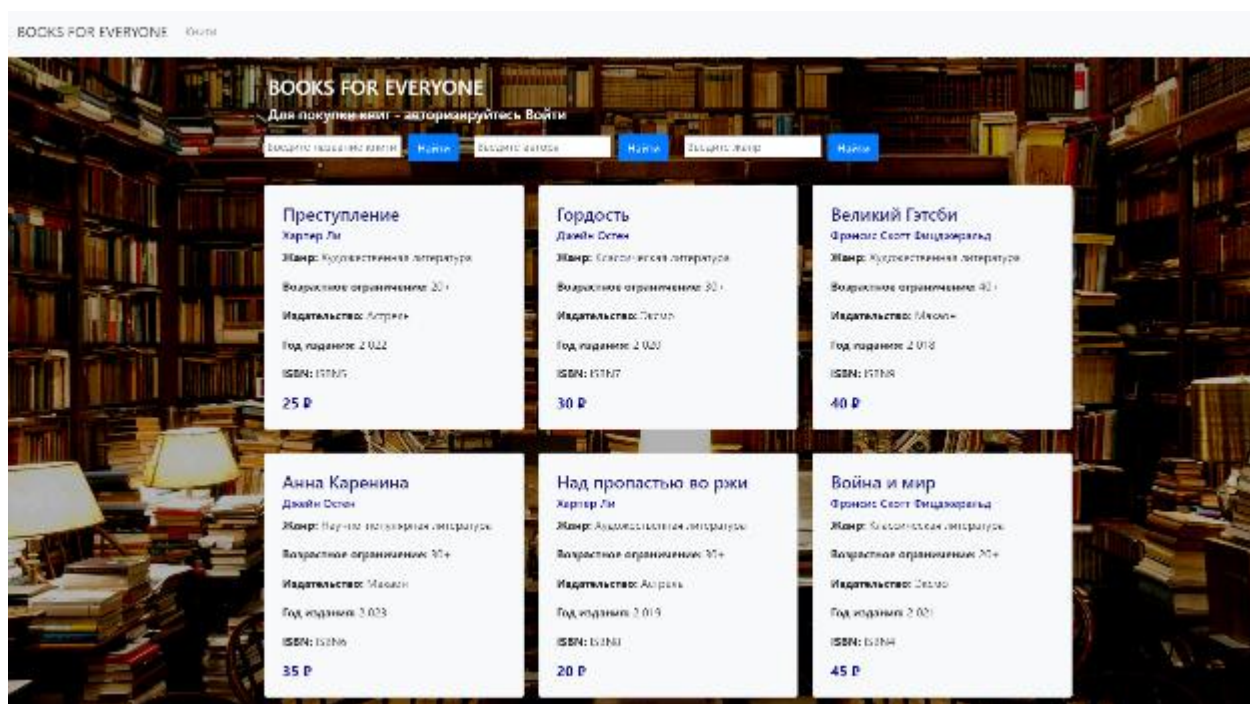


Рисунок 33 – функционал неавторизованного пользователя

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ1.090301.06.001 ПЗ

Лист

53

6 Безопасность жизнедеятельности

Безопасность при работе за компьютером играет важную роль для разработчиков и других профессионалов, которые проводят много времени за монитором. Невыполнение правил безопасности может привести к различным проблемам со здоровьем, таким как напряжение глаз, нарушения осанки, повышенный риск развития карпального туннельного синдрома и других заболеваний.

В этом разделе мы рассмотрим основные аспекты безопасности при работе за компьютером и предоставим рекомендации для разработчиков.

6.1 Требования к помещению и рабочему месту

При выборе места для рабочего места с компьютером рекомендуется отдавать предпочтение помещениям, ориентированным на север или северо-восток. Площадь одного рабочего места с компьютером и жидкокристаллическим монитором должна быть не менее 4,5 квадратных метров. Помещения, предназначенные для работы с компьютером, должны иметь как искусственное, так и естественное освещение, а также должны быть обеспечены достаточной вентиляцией для поддержания свежего воздуха и снижения концентрации вредных веществ, таких как формальдегид, который может выделяться из компьютера. Концентрация вредных веществ в воздухе таких помещений не должна превышать установленных гигиенических норм. Рекомендуемый уровень шума в помещении не должен превышать 50 децибел, чтобы избежать отвлекающих звуков и избыточного стресса.

6.2 Требования к освещению

Если не придерживаться простых правил по освещению, то можно получить немалые проблемы со здоровьем. Возьмем, к примеру, глаза, ведь неправильный свет очень сильно изнашивает их и дальше будут очень серьезные проблемы. Для предотвращения нужен правильный монитор, включенный свет,

					ПГУ1.090301.06.001 ПЗ	Лист
						54
Изм.	Лист	№ докум.	Подпись	Дата		

возможно даже очки, которые продаются в специализированных магазинах. Есть в документах норма, которой нужно придерживаться. Вот примеры этих норм: рабочее место должно хорошо освещаться, температура должна быть в рамках 3900 до 6100 градусов по Кельвину, монитор желательно иметь антибликовый и со средним уровнем яркости как утверждает производитель в инструкции. Требования к режиму труда и отдыха

6.3 Требования к режиму работы и отдыха

Очень важная часть, ведь переработки очень плохо сказываются на здоровье и постоянный труд может привести к негативным последствиям. Есть разделение на требования, среди них физические, психологические и организационные.

Физические требования включают:

- прямая осанка и положение;
- время от времени нужно делать разминку;
- правильное расположение монитора;
- правильный свет.

Психологические требования включают:

- паузы между работой;
- помимо работы разгружаться, лучше всего прогулки.

Организационные требования включают:

- нормальная температура;
- график и план для четкой работы;
- в дни отдыха не следует работать и напрягать себя.

6.4 Рекомендации по использованию программного обеспечения

Есть разные варианты решения, к примеру, программы, которые следят за временем, глазами и нарушениями.

					ПГУ1.090301.06.001 ПЗ	Лист
						55
Изм.	Лист	№ докум.	Подпись	Дата		

Вот примеры таких программ:

- управление шрифтом, для снижения чувствительности глаз и более плавной картинке;
- разные фильтры, для глаз, они помогают снизить напряжение;
- для оптимизации графика, есть приложения для распределения рабочего времени и повышения трудоспособности.

					<i>ПГУ1.090301.06.001 ПЗ</i>	Лист
						56
Изм.	Лист	№ докум.	Подпись	Дата		

Заключение

В процессе написания всей этой работы были сделан анализ актуальной темы, по определенным причинам выбрана среда для разработки и сделана архитектура. Благодаря этим действиям проект был выполнен, приложение "Книжный интернет-магазин" успешно создано. Для проверки использовалось тестирование разных вариантов использования, при этом и проблемные тоже. В ходе диагностики некоторые моменты дорабатывались по ходу.

Благодаря этой работе, были получены полезные и, что самое главное, важные навыки в работе с Java языком. После доработки, все нюансы были исправлены и приложение полностью работает.

Дальнейшее улучшение приложения может включать добавление функционала администратора для обеспечения большей безопасности пользователей, а также улучшение интерфейса клиентской части для повышения комфорта использования.

Опыт, полученный в процессе выполнения задач и разработки клиент-серверного приложения, наверняка пригодится в будущем при трудоустройстве и личном развитии.

					ПГУ1.090301.06.001 ПЗ	Лист
						57
Изм.	Лист	№ докум.	Подпись	Дата		

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 20.04.2024).
2. Марк Массе. Свод правил проектирования REST API. - 1-е изд. О'Рейли Медиа, 2011. - 5с.
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 15.04.2024).
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 27.04.2024).

Приложение А

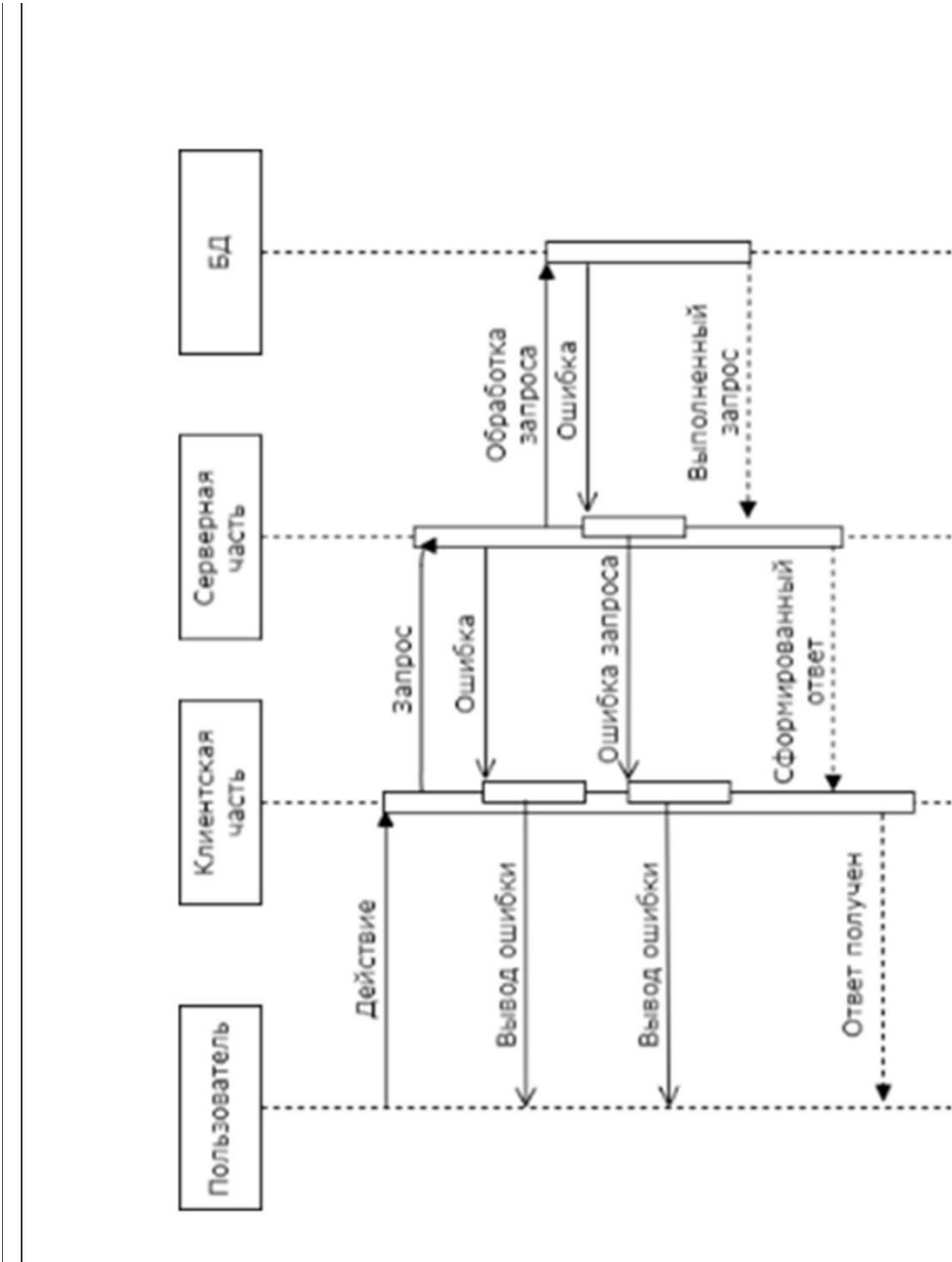
(Обязательное)

Клиент серверное приложение «Книжный интернет-магазин»

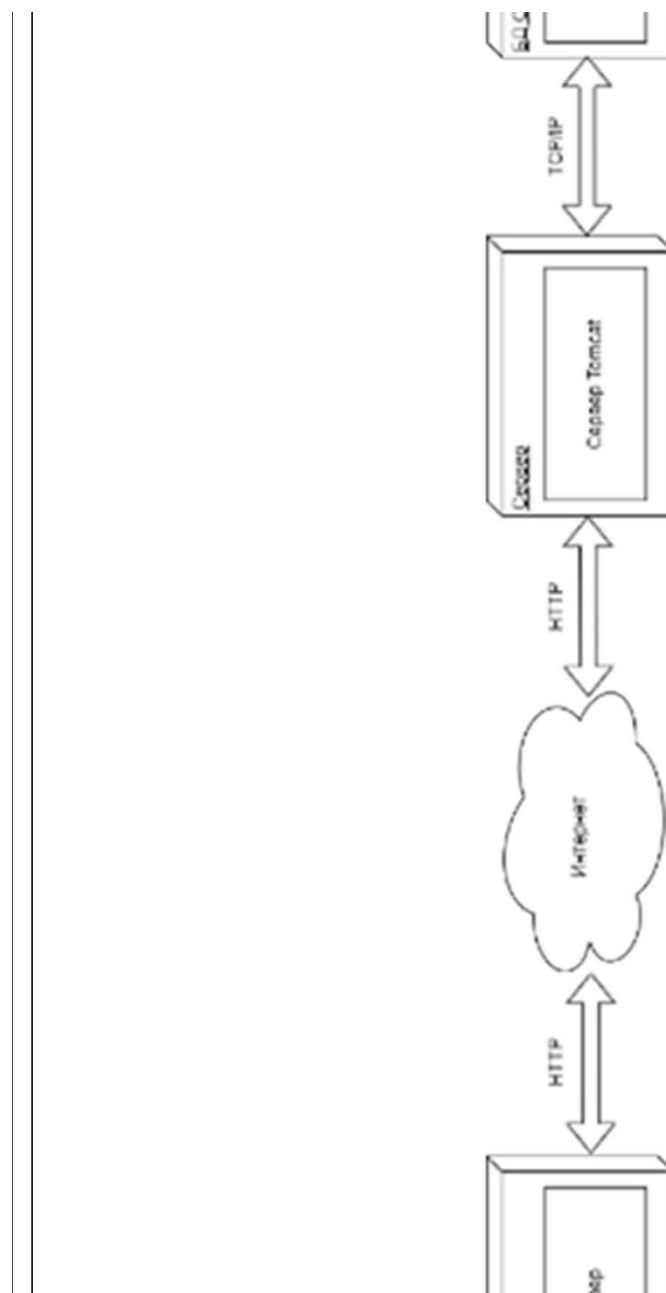
UML-диаграммы

					ПГУ1.090301.06.001 ПЗ	Лист
						59
Изм.	Лист	№ докум.	Подпись	Дата		

Приложение А.1-Диаграмма последовательностей



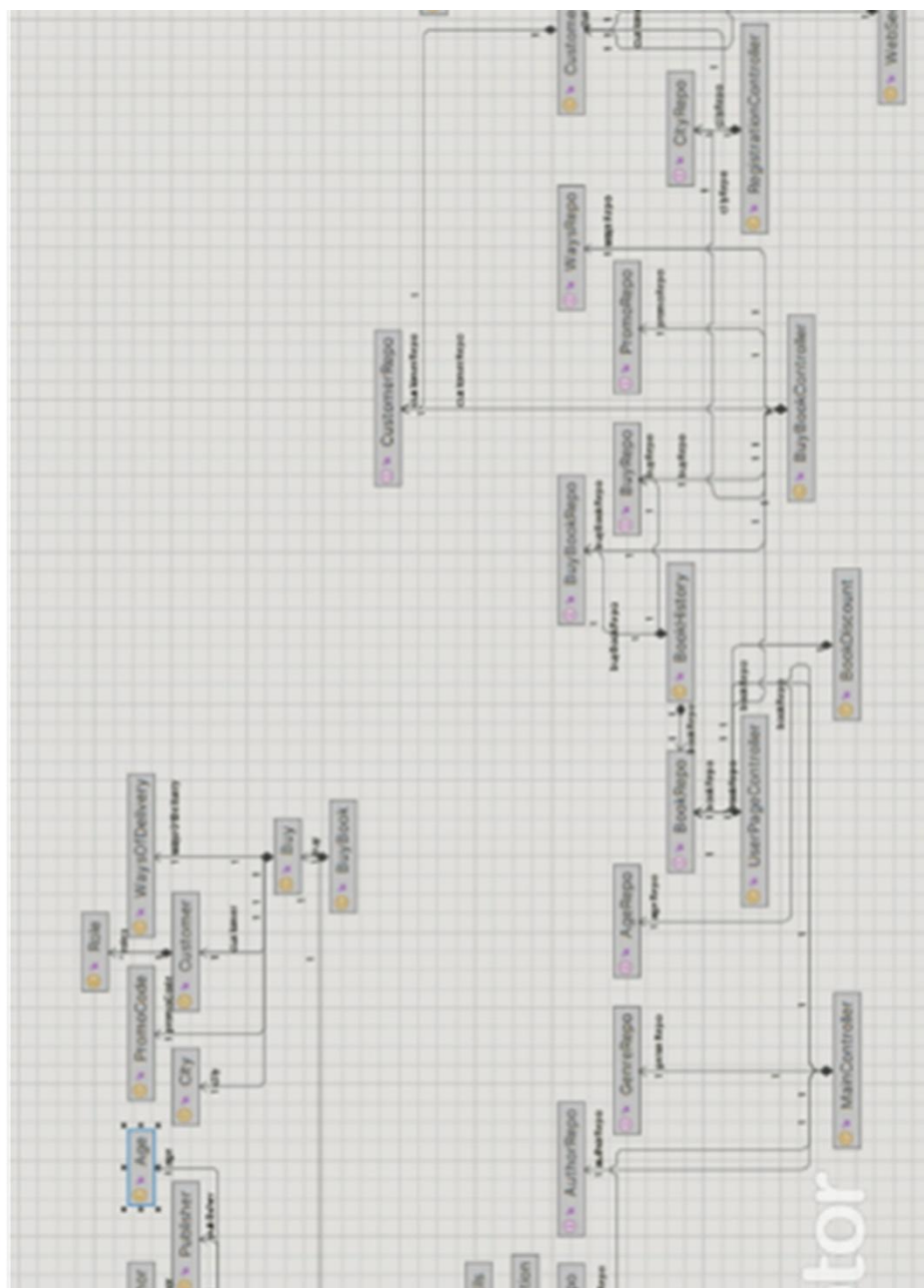
Приложение А.2-Диаграмма развертывания



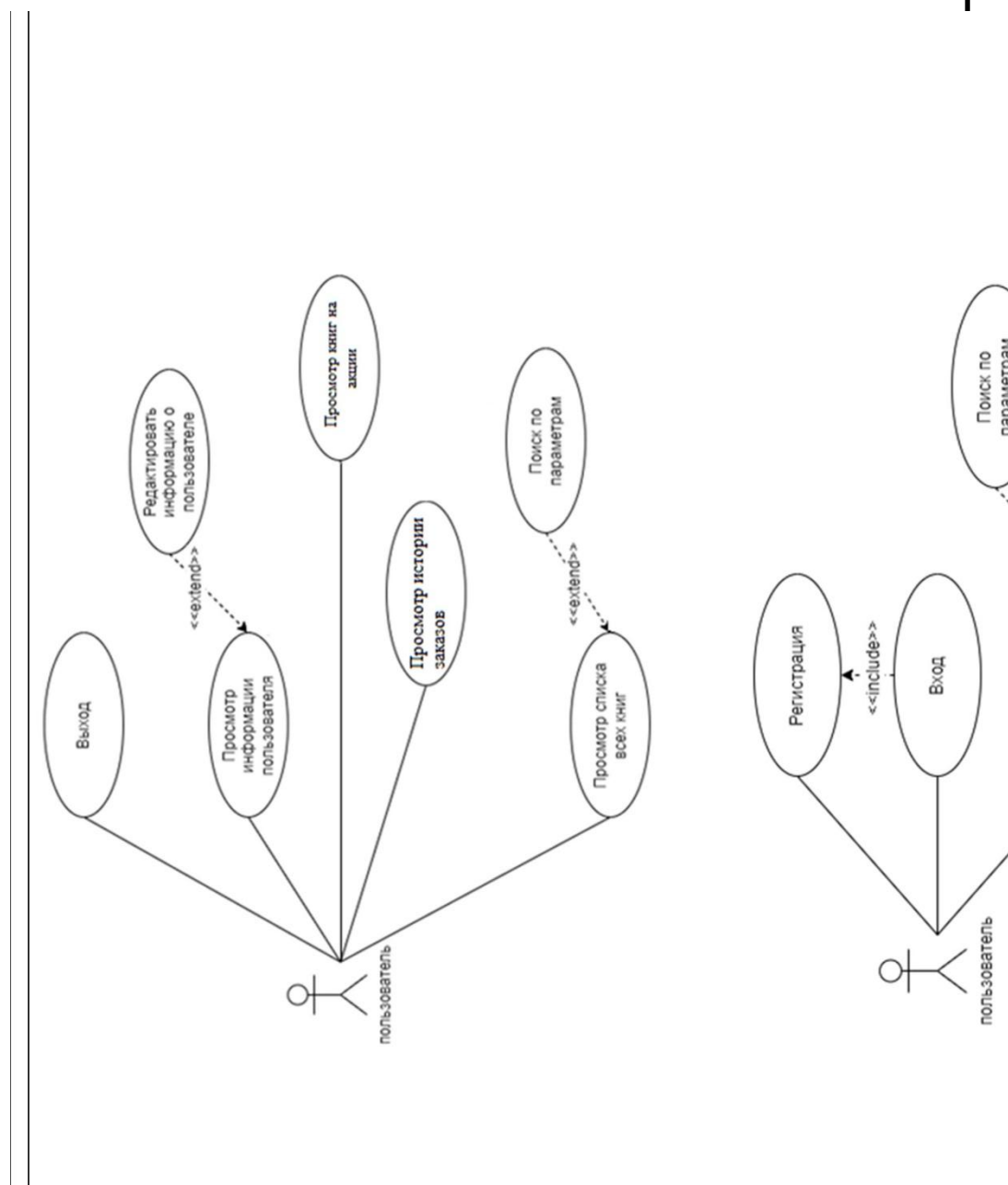
Изм.	Лист	№ докум.	Подпись	Дата

ПГУ1.090301.06.001 ПЗ

Приложение А.3-Диаграмма классов



Приложение А.4-Диаграммы вариантов использования



Приложение Б

(Обязательное)

Клиент серверное приложение «Книжный интернет-магазин»

Листинг программы

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ1.090301.06.001 ПЗ

Лист
63

Файл MainController.java

```
@Controller
@PreAuthorize("hasAuthority('ADMIN')")
public class MainController {
    private final BookRepo bookRepo;
    private final AuthorRepo authorRepo;
    private final GenreRepo genreRepo;
    private final PublisherRepo publisherRepo;
    private final AgeRepo ageRepo;
    public MainController(BookRepo bookRepo, AuthorRepo
authorRepo, GenreRepo genreRepo, PublisherRepo publisherRepo,
AgeRepo ageRepo) {
        this.bookRepo = bookRepo;
        this.authorRepo = authorRepo;
        this.genreRepo = genreRepo;
        this.publisherRepo = publisherRepo;
    }
}
```

					ПГУ1.090301.06.001 ПЗ	Лист
						64
Изм.	Лист	№ докум.	Подпись	Дата		

```

        this.ageRepo = ageRepo;
    }
    @GetMapping("/main")
    public String main(@RequestParam(required = false,
defaultValue = "") String filter, @RequestParam(required =
false, defaultValue = "") String filterAuthor, Model model) {
        Iterable<Book> books = bookRepo.findAll();
        books = bookRepo.findAll();
        model.addAttribute("books", books);
        model.addAttribute("filter", filter);
        model.addAttribute("filterAuthor", filterAuthor);
        return "main";
    }
    /**/
    @PostMapping("/main")
    public String add(
        @RequestParam String title,
        @RequestParam String isbn,
        @RequestParam Integer year_publ,
        @RequestParam Integer price,
        @RequestParam String author,
        @RequestParam String genre,
        @RequestParam String publisher,
        @RequestParam Integer age,
        @RequestParam Integer amount,
        Model model
    ) throws IOException {
        Publisher publisherFromDB =
publisherRepo.findPublisherByPublisher_name(publisher);
        Genre genreFromDB =
genreRepo.findGenreByGenre_name(genre);
        Age ageFromDB = ageRepo.findAgeByAge_number(age);
        Author authorFromDB =
authorRepo.findAuthorByAuthor_name(author);
        if (publisherFromDB == null)
        {
            publisherFromDB = new Publisher(publisher);
            publisherRepo.save(publisherFromDB);
        }
        if (authorFromDB == null)
        {
            authorFromDB = new Author(author);
            authorRepo.save(authorFromDB);
        }
        if (ageFromDB == null)
        {
            ageFromDB = new Age(age);
            ageRepo.save(ageFromDB);
        }
        if (genreFromDB == null)
        {
            genreFromDB = new Genre(genre);
            genreRepo.save(genreFromDB);
        }
    }

```

					ПГУ1.090301.06.001 ПЗ	Лист
						65
Изм.	Лист	№ докум.	Подпись	Дата		

```

    }
    Book book = new Book(title, isbn, year_publ,
        price, amount, authorFromDB,
        genreFromDB, publisherFromDB, ageFromDB);
    bookRepo.save(book);
    Iterable<Book> books = bookRepo.findAll();
    model.addAttribute("books", books);
    return "main";
}
@GetMapping("/edit/{book_id}")
public String bookEdit(
    Model model,
    @RequestParam(required = false) Book book
) {
    /*Iterable<Book> books =
bookRepo.findBookByBook_id(curr.getBook_id());
    model.addAttribute("books", books);*/
    return "bookEdit";
}
@PostMapping("/edit/{book_id}")
public String updateAmount(
    @ModelAttribute("book_id") long id,
    @RequestParam("amount") Integer amount
) throws IOException {
    Book book = bookRepo.findBookByBook_id(id);
    if (!StringUtils.isEmpty(amount)) {
        book.setAmount(amount);
    }
    bookRepo.save(book);
    return "redirect:/main";
}
}

```

Файл RegistrationController.java

```

@Controller
public class RegistrationController {
    private final CustomerService customerService;
    private final CityRepo cityRepo;
    public RegistrationController(CustomerService
customerService, CityRepo cityRepo) {
        this.customerService = customerService;
        this.cityRepo = cityRepo;
    }
    @GetMapping("/registration")
    public String registration(){
        return "registration";
    }
    @PostMapping("/registration")
    public String addCustomer(@Valid Customer customer,
BindingResult bindingResult, Model model){
        String city = customer.getCity();
        City city1 = new City();
        city1.setCity_name(city);
    }
}

```

```

        cityRepo.save(city1);
        if (bindingResult.hasErrors()) {
            Map<String, String> errors =
ControllerUtils.getErrors(bindingResult);
            model.mergeAttributes(errors);
            return "registration";
        }
        if (!customerService.addCustomer(customer))
        {
            model.addAttribute("message", "Customer with this
email already exists!");
            return "registration";
        }
        return "redirect:/login";
    }
}

```

Файл UserController.java

```

package com.serveeer.controllers;
import com.serveeer.domain.Customer;
import com.serveeer.domain.Role;
import com.serveeer.service.CustomerService;
import org.springframework.security.access.prepost.PreAuthorize;
import
org.springframework.security.core.annotation.AuthenticationPrinc
ipal;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.*;
import java.util.Map;
@Controller
@RequestMapping("/user")
public class UserController {
    private final CustomerService customerService;
    public UserController(CustomerService customerService) {
        this.customerService = customerService;
    }
    @PreAuthorize("hasAuthority('ADMIN')")
    @GetMapping
    public String userList(Model model) {
        model.addAttribute("users", customerService.findAll());

        return "userList";
    }
    @PreAuthorize("hasAuthority('ADMIN')")
    @GetMapping("{customer}")
    public String userEditForm(@PathVariable Customer customer,
Model model){
        model.addAttribute("user", customer);
        model.addAttribute("roles", Role.values());
        return "userEdit";
    }
    @PreAuthorize("hasAuthority('ADMIN')")

```

					ПГУ1.090301.06.001 ПЗ	Лист
						67
Изм.	Лист	№ докум.	Подпись	Дата		

```

        @PostMapping
        public String userSave(@RequestParam Map<String, String>
form,
                                @RequestParam("userId") Customer customer){
            customerService.saveCustomer(customer, form);
            return "redirect:/user";
        }
        @GetMapping("profile")
        public String getProfile(Model model,
@AuthenticationPrincipal Customer customer) {
            model.addAttribute("username", customer.getUsername());
            model.addAttribute("email", customer.getEmail());
            model.addAttribute("surname",customer.getSurName());

model.addAttribute("lastsurname",customer.getLastSurname());
            model.addAttribute("city",customer.getCity());
            return "profile";
        }
        @PostMapping("profile")
        public String updateProfile(
            @AuthenticationPrincipal Customer customer,
            @RequestParam String email,
            @RequestParam String customer_password,
            @RequestParam String username,
            @RequestParam String surname,
            @RequestParam String lastsurname,
            @RequestParam String city
        ) {
            customerService.updateProfile(customer,
customer_password, email, username,surname,lastsurname,city);
            return "redirect: /user/profile";
        }
    }
}

```

Файл UserPageController.java

```

package com.serveeer.controllers;
import com.serveeer.domain.Book;
import com.serveeer.repos.BookRepo;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestParam;
import java.util.ArrayList;
import java.util.List;
import java.util.stream.Collectors;
import java.util.stream.StreamSupport;
@Controller
public class UserPageController {
    private final BookRepo bookRepo;

    public UserPageController(BookRepo bookRepo) {
        this.bookRepo = bookRepo;
    }
}

```

					ПГУ1.090301.06.001 ПЗ	Лист
						68
Изм.	Лист	№ докум.	Подпись	Дата		

```

    @GetMapping("/")
    public String greeting(@RequestParam(required = false,
defaultValue = "") String filter,
                           @RequestParam(required = false,
defaultValue = "") String filterAuthor,
                           @RequestParam(required = false,
defaultValue = "") String filterGenre,
                           Model model) {
        Iterable<Book> books = bookRepo.findAll();
        if(filter != null && !filter.isEmpty()) {
            books = bookRepo.findByTitle(filter);
        } else if(filterAuthor != null &&
!filterAuthor.isEmpty()) {
            books = bookRepo.findByAuthor(filterAuthor);
        } else if(filterGenre != null && !filterGenre.isEmpty())
{
            books = bookRepo.findByGenre(filterGenre);
        }else
            books = bookRepo.findAll();
        model.addAttribute("books", books);
        model.addAttribute("filter", filter);
        model.addAttribute("filterAuthor", filterAuthor);
        return "greeting";
    }
    @GetMapping("/userPage")
    public String main(@RequestParam(required = false,
defaultValue = "") String filter,
                       @RequestParam(required = false,
defaultValue = "") String filterAuthor,
                       @RequestParam(required = false,
defaultValue = "") String filterGenre,
                       Model model) {
        Iterable<Book> allBooks = bookRepo.findAll();
        List<Book> books = new ArrayList<>();

        for (Book book : allBooks) {
            if (filter != null && !filter.isEmpty() &&
!book.getTitle().contains(filter)) {
                continue;
            }
            if (filterAuthor != null && !filterAuthor.isEmpty()
&& !book.getAuthor().equals(filterAuthor)) {
                continue;
            }
            if (filterGenre != null && !filterGenre.isEmpty() &&
!book.getGenre().equals(filterGenre)) {
                continue;
            }
            if (book.getDiscount() == null || book.getDiscount()
== 0) {
                books.add(book);
            }
        }
    }

```

```

        model.addAttribute("books", books);
        model.addAttribute("filter", filter);
        model.addAttribute("filterAuthor", filterAuthor);
        model.addAttribute("filterGenre", filterGenre);
        return "userPage";
    }
}

```

Файл ControllerUtils.java

```

package com.serveeer.controllers;
import org.springframework.validation.BindingResult;
import org.springframework.validation.FieldError;
import java.util.Map;
import java.util.stream.Collectors;
import java.util.stream.Collectors;
public class ControllerUtils {
    static Map<String, String> getErrors(BindingResult
bindingResult) {
        Collector<FieldError, ?, Map<String, String>> collector
= Collectors.toMap(
            fieldError -> fieldError.getField() + "Error",
            FieldError::getDefaultMessage
        );
        return
bindingResult.getFieldErrors().stream().collect(collector);
    }
}

```

Файл BuyBookController.java

```

package com.serveeer.controllers;
import com.serveeer.domain.*;
import com.serveeer.repos.*;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestParam;
import java.io.IOException;
import java.util.Date;
@Controller
public class BuyBookController {
    private final WaysRepo waysRepo;
    private final BookRepo bookRepo;
    private final CityRepo cityRepo;
    private final CustomerRepo customerRepo;
    private final BuyRepo buyRepo;
    private final BuyBookRepo buyBookRepo;
    private final PromoRepo promoRepo;
    public BuyBookController(WaysRepo waysRepo, BookRepo
bookRepo,
                            CityRepo cityRepo, CustomerRepo
customerRepo,

```

					ПГУ1.090301.06.001 ПЗ	Лист
						70
Изм.	Лист	№ докум.	Подпись	Дата		

```

        BuyRepo buyRepo, BuyBookRepo
buyBookRepo, PromoRepo promoRepo) {
    this.waysRepo = waysRepo;
    this.bookRepo = bookRepo;
    this.cityRepo = cityRepo;
    this.customerRepo = customerRepo;
    this.buyRepo = buyRepo;
    this.buyBookRepo = buyBookRepo;
    this.promoRepo = promoRepo;
}
@GetMapping("/buybook/{name}/{book_id}")
public String orderInfo(
    Model model,
    @RequestParam(required = false) Book book
) {
    Iterable<City> city = cityRepo.findAll();
    model.addAttribute("cities", city);
    Iterable<WaysOfDelivery> waysOfDeliveries =
waysRepo.findAll();
    model.addAttribute("waysOfDeliveries",
waysOfDeliveries);
    return "buyBook";
}
@PostMapping("/buybook/{name}/{book_id}")
public String buy(@RequestParam("citySelect") String
citySelect,
                  @RequestParam(required=false,
name="order_comments") String order_comments,
                  @RequestParam(required=false,
name="promo_code") String promo_code,
                  @RequestParam("waysSelect") String
waysSelect,
                  @ModelAttribute("book_id") long id,
                  @ModelAttribute("name") String name,
                  Model model
) throws IOException {
    Book book = bookRepo.findBookByBook_id(id);
    City city = cityRepo.findCityByCity_name(citySelect);
    WaysOfDelivery way =
waysRepo.findWaysOfDeliveriesByWays(waysSelect);
    Customer customer = customerRepo.findByUsername(name);
    PromoCode promoCode =
promoRepo.findPromoCodeByCode(promo_code);
    Buy buy = new Buy(new Date(),true, promoCode, customer,
way, order_comments, city);
    buyRepo.save(buy);

    int sum = book.getPrice() + way.getDelivery_price();
    if(promoCode != null)
    {
        double discount = (100 - promoCode.getDiscount()) /
100.0;
        sum *= discount;
    }
}

```

					ПГУ1.090301.06.001 ПЗ	Лист
						71
Изм.	Лист	№ докум.	Подпись	Дата		

```

        model.addAttribute("ifDiscount", promo_code);
    }
    BuyBook buyBook = new BuyBook(new Date(), sum, book,
buy);
    buyBookRepo.save(buyBook);
    model.addAttribute("orderNumber",
buyBook.getBuy_book_id());
    model.addAttribute("sum", sum);
    Integer amount = book.getAmount();
    book.setAmount(amount - 1);
    bookRepo.save(book);
    return "afterBuy";
}
}

```

Файл BookHistory.java

```

package com.serveeer.controllers;
import com.serveeer.domain.Book;
import com.serveeer.domain.Buy;
import com.serveeer.domain.BuyBook;
import com.serveeer.domain.Customer;
import com.serveeer.repos.BookRepo;
import com.serveeer.repos.BuyBookRepo;
import com.serveeer.repos.BuyRepo;
import org.springframework.security.core.Authentication;
import
org.springframework.security.core.context.SecurityContextHolder;
import
org.springframework.security.core.userdetails.UserDetails;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;
import javax.servlet.http.HttpServletRequest;
import java.lang.reflect.Array;
import java.util.ArrayList;
import java.util.List;
import java.util.Optional;
@Controller
public class BookHistory {
    private final BuyBookRepo buyBookRepo;
    private final BookRepo bookRepo;
    private final BuyRepo buyRepo;
    public BookHistory(BuyBookRepo buyBookRepo, BookRepo
bookRepo, BuyRepo buyRepo) {
        this.buyBookRepo = buyBookRepo;
        this.bookRepo = bookRepo;
        this.buyRepo = buyRepo;
    }
    @GetMapping("/book-history")
    public String bookHistory(Model model) {
        Authentication authentication =
SecurityContextHolder.getContext().getAuthentication();

```

					ПГУ1.090301.06.001 ПЗ	Лист
						72
Изм.	Лист	№ докум.	Подпись	Дата		

```

        Customer customer = (Customer)
authentication.getPrincipal();
        List<BuyBook> buyBooks = buyBookRepo.findAll();
        List<Buy> listBuy = buyRepo.findAll();
        ArrayList<Buy> buyArrayList = new ArrayList<>();
        for(int i=0; i<listBuy.size();i++){

if(listBuy.get(i).getCustomer().getCustomer_id().equals(customer
.getCustomer_id())){
            buyArrayList.add(listBuy.get(i));
        }
    }
    List<Book> books = new ArrayList<>();
    for (int i = 0; i < buyArrayList.size(); i++) {
        Optional<Book> bookOptional =
bookRepo.findById(buyArrayList.get(i).getBuy_book_id());
        if (bookOptional.isPresent()) {
            books.add(bookOptional.get());
        }
    }
    // Добавляем коллекцию книг в модель
    model.addAttribute("books", books);
    return "book-history";
}
}

```

Файл BookDiscount.java

```

package com.serveeer.controllers;
import com.serveeer.domain.Book;
import com.serveeer.repos.BookRepo;
import org.junit.platform.commons.util.StringUtils;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestParam;
import java.util.ArrayList;
import java.util.List;
import java.util.stream.Collectors;
import java.util.stream.StreamSupport;
@Controller
public class BookDiscount {
    private final BookRepo bookRepo;
    public BookDiscount(BookRepo bookRepo) {
        this.bookRepo = bookRepo;
    }
    @GetMapping("/bookDiscount")
    public String main(@RequestParam(required = false,
defaultValue = "") String filter,
        @RequestParam(required = false,
defaultValue = "") String filterAuthor,
        @RequestParam(required = false,
defaultValue = "") String filterGenre,
        Model model) {

```

```

        Iterable<Book> allBooks = bookRepo.findAll();
        List<Book> books = new ArrayList<>();
        for (Book book : allBooks) {
            if (book.getDiscount() != 0) {
                books.add(book);
            }
        }
        model.addAttribute("books", books);
        model.addAttribute("filter", filter);
        model.addAttribute("filterAuthor", filterAuthor);
        return "bookDiscount";
    }
}

```

Файл MvcConfig.java

```

package com.serveer.config;
import org.springframework.context.annotation.Configuration;
import
org.springframework.web.servlet.config.annotation.ResourceHandle
rRegistry;
import
org.springframework.web.servlet.config.annotation.ViewController
Registry;
import
org.springframework.web.servlet.config.annotation.WebMvcConfigur
er;
@Configuration
public class MvcConfig implements WebMvcConfigurer {
    public void addViewControllers(ViewControllerRegistry
registry) {

registry.addViewController("/login").setViewName("login");
    }
    @Override
    public void addResourceHandlers(ResourceHandlerRegistry
registry) {
        registry.addResourceHandler("/static/**")
            .addResourceLocations("classpath:/static/");
    }
}

```

Файл WebSecurityConfig.java

```

package com.serveer.config;
import com.serveer.service.CustomerService;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import
org.springframework.security.config.annotation.authentication.bu
ilders.AuthenticationManagerBuilder;
import
org.springframework.security.config.annotation.method.configurat
ion.EnableGlobalMethodSecurity;

```

```

import
org.springframework.security.config.annotation.web.builders.Http
Security;
import
org.springframework.security.config.annotation.web.configuration
.EnableWebSecurity;
import
org.springframework.security.config.annotation.web.configuration
.WebSecurityConfigurerAdapter;
import
org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder
;
import
org.springframework.security.crypto.password.PasswordEncoder;
@Configuration
@EnableWebSecurity
@EnableGlobalMethodSecurity(prePostEnabled = true)
public class WebSecurityConfig extends
WebSecurityConfigurerAdapter {
    private final CustomerService customerService;
    public WebSecurityConfig(CustomerService customerService,
PasswordEncoder passwordEncoder) {
        this.customerService = customerService;
        this.passwordEncoder = passwordEncoder;
    }
    private final PasswordEncoder passwordEncoder;
    @Bean
    public static PasswordEncoder getPasswordEncoder() {
        return new BCryptPasswordEncoder(8);
    }
    @Override
    protected void configure(HttpSecurity http) throws Exception
    {
        http
            .authorizeRequests()
            .antMatchers("/", "/registration", "/static/**",
"/userPage").permitAll()
            .anyRequest().authenticated()
            .and()
            .formLogin()
            .loginPage("/login").passwordParameter("customer_password").defa
ultSuccessUrl("/userPage").permitAll()
            .and()
            .logout()
            .permitAll()
            ;
    }
    @Override
    protected void configure(AuthenticationManagerBuilder auth)
throws Exception {
        auth.userDetailsService(customerService)
            .passwordEncoder(passwordEncoder);
    }
}

```

					ПГУ1.090301.06.001 ПЗ	Лист
						75
Изм.	Лист	№ докум.	Подпись	Дата		

```
}
```

Файл Age.java

```
package com.serveeer.domain;
import lombok.Data;
import javax.persistence.*;
@Entity
@Table(name = "ages")
@Data
public class Age {
    @Id
    @Column(name = "age_id")
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long age_id;
    @Column(name = "age_number")
    private int age_number;
    public Age(int age_number) {
        this.age_number = age_number;
    }
    public Age() {}
}
```

Файл Author.java

```
package com.serveeer.domain;

import lombok.Data;
import javax.persistence.*;
@Entity
@Table(name = "author")
@Data
public class Author{
    @Id
    @Column(name = "author_id")
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long author_id;
    @Column(name = "author_name")
    private String author_name;
    public Author() {
    }
    public Author(String author_name) {
        this.author_name = author_name;
    }
}
```

Файл Book.java

```
package com.serveeer.domain;
import lombok.Data;
import javax.persistence.*;
@Entity
@Table(name = "book", schema = "public")
@Data
public class Book {
    public Book() {}
    @Id
```

					ПГУ1.090301.06.001 ПЗ	Лист
						76
Изм.	Лист	№ докум.	Подпись	Дата		

```

@Column(name = "book_id")
@GeneratedValue(strategy = GenerationType.AUTO)
private Long book_id;
@Column(name = "title")
private String title;
@Column(name = "isbn")
private String isbn;
@Column(name = "year_publ")
private Integer year_publ;
@Column(name = "price")
private Integer price;
@Column(name = "amount")
private Integer amount;
@ManyToOne
@JoinColumn(name = "author_id")
private Author author;
@ManyToOne
@JoinColumn(name = "genre_id")
private Genre genre;
@ManyToOne
@JoinColumn(name = "publisher_id")
private Publisher publisher;
@ManyToOne
@JoinColumn(name = "age_id")
private Age age;
private Long discount;
public Book(String title, String isbn, Integer year_publ,
            Integer price, Integer amount, Author author,
            Genre genre, Publisher publisher, Age age) {
    this.title = title;
    this.isbn = isbn;
    this.year_publ = year_publ;
    this.price = price;
    this.amount = amount;
    this.author = author;
    this.genre = genre;
    this.publisher = publisher;
    this.age = age;
}
}

```

Файл Buy.java

```

package com.serveer.domain;
import lombok.Data;
import javax.persistence.*;
import java.util.Date;
@Entity
@Table(name = "buy")
@Data
public class Buy {
    @Id
    @Column(name = "buy_id")
    @GeneratedValue(strategy = GenerationType.IDENTITY)

```

					ПГУ1.090301.06.001 ПЗ	Лист
						77
Изм.	Лист	№ докум.	Подпись	Дата		

```

private Long buy_book_id;
@Column(name = "date_of_buying")
private Date date_of_buying;
@Column(name = "paid_or_not")
private boolean paid_or_not;
@ManyToOne
@JoinColumn(name = "customer_id")
private Customer customer;
@ManyToOne
@JoinColumn(name = "way_id")
private WaysOfDelivery waysOfDelivery;
@Column(name = "order_comments")
private String order_comments;
@ManyToOne
@JoinColumn(name = "city_id")
private City city;
@ManyToOne
@JoinColumn(name = "promo_code_id")
private PromoCode promoCode;
public Buy(Date date_of_buying, boolean paid_or_not,
PromoCode promoCode, Customer customer, WaysOfDelivery
waysOfDelivery, String order_comments, City city) {
    this.date_of_buying = date_of_buying;
    this.paid_or_not = paid_or_not;
    this.promoCode = promoCode;
    this.customer = customer;
    this.waysOfDelivery = waysOfDelivery;
    this.order_comments = order_comments;
    this.city = city;
}

public Buy() { }
}

```

Файл BuyBook.java

```

package com.serveeer.domain;
import lombok.Data;
import javax.persistence.*;
import java.util.Date;
@Entity
@Table(name = "buy_book")
@Data
public class BuyBook {
    @Id
    @Column(name = "buy_book_id")
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long buy_book_id;
    @Column(name = "order_date")
    private Date order_date;
    @Column(name = "amount")
    private Integer orderPrice;
    @OneToOne
    @JoinColumn(name = "book_id")

```

```

        private Book orderBook;
        @OneToOne
        @JoinColumn(name = "buy_id")
        private Buy buy;
        public BuyBook(Date order_date, Integer orderPrice, Book
orderBook, Buy buy) {
            this.order_date = order_date;
            this.orderPrice = orderPrice;
            this.orderBook = orderBook;
            this.buy = buy;
        }
        public BuyBook() {}
    }
}

```

Файл City.java

```

package com.serveeer.domain;
import lombok.Data;
import javax.persistence.*;
@Entity
@Table(name = "city")
@Data
public class City {
    @Id
    @Column(name = "city_id")
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long city_id;
    @Column(name = "city_name")
    private String city_name;
}

```

Файл Customer.java

```

package com.serveeer.domain;
import lombok.Data;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.userdetails.UserDetails;
import javax.persistence.*;
import javax.validation.constraints.NotBlank;
import javax.validation.constraints.Email;
import java.util.Collection;
import java.util.Set;
@Entity
@Table(name = "customer")
@Data
public class Customer implements UserDetails {
    @Id
    @Column(name = "customer_id")
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long customer_id;
    @NotBlank(message = "Username cannot be empty")
    @Column(name = "username")
    private String username;
    @NotBlank(message = "Surname cannot be empty")

```

```

@Column(name = "surname")
private String surName;
@NotBlank(message = "Lastusername cannot be empty")
@Column(name = "lastusername")
private String lastSurname;
@NotBlank(message = "City cannot be empty")
@Column(name = "city")
private String city;
@NotBlank(message = "Password cannot be empty")
@Column(name = "customer_password")
private String customer_password;
>Email(message = "Email is not correct")
@NotBlank(message = "Email cannot be empty")
@Column(name = "email")
private String email;
@ElementCollection(targetClass = Role.class, fetch =
FetchType.EAGER)
@CollectionTable(name = "roles", joinColumns =
@JoinColumn(name = "cust_id"))
@Enumerated(EnumType.STRING)
private Set<Role> roles;
public boolean isAdmin(){
    return roles.contains(Role.ADMIN);
}
public boolean isUser(){
    return roles.contains(Role.USER);
}
@Override
public Collection<? extends GrantedAuthority>
getAuthorities() {
    return getRoles();
}
@Override
public String getPassword() {
    return customer_password;
}
@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 Genre.java

```
package com.serveeer.domain;
import lombok.Data;
import javax.persistence.*;
@Entity
@Table(name = "genre")
@Data
public class Genre {
    @Id
    @Column(name = "genre_id")
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long genre_id;
    //@NotBlank(message = "Please fill the genre")
    @Column(name = "genre_name")
    private String genre_name;
    public Genre(String genre_name) {
        this.genre_name = genre_name;
    }
    public Genre() {
    }
}
```

Файл PromoCode.java

```
package com.serveeer.domain;
import lombok.Data;
import javax.persistence.*;
@Entity
@Table(name = "promo_code")
@Data
public class PromoCode {
    @Id
    @Column(name = "promo_code_id")
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long promo_code_id;
    @Column(name = "code")
    private String code;
    @Column(name = "discount")
    private Integer discount;
}
```

Файл Publisher.java

```
package com.serveeer.domain;
import lombok.Data;
import javax.persistence.*;

@Entity
@Table(name = "publisher")
@Data
public class Publisher {
    @Id
    @Column(name = "publisher_id")
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long publisher_id;
```

					ПГУ1.090301.06.001 ПЗ	Лист
						81
Изм.	Лист	№ докум.	Подпись	Дата		

```

    //@NotBlank(message = "Please fill the publisher")
    @Column(name = "publisher_name")
    private String publisher_name;
    public Publisher(String publisher_name) {
        this.publisher_name = publisher_name;
    }
    public Publisher() {
    }
}

```

Файл Role.java

```

package com.serveeer.domain;
import org.springframework.security.core.GrantedAuthority;
public enum Role implements GrantedAuthority {
    USER, ADMIN;
    @Override
    public String getAuthority() {
        return name();
    }
}

```

Файл WaysOfDelivery.java

```

package com.serveeer.domain;

import lombok.Data;
import javax.persistence.*;

@Entity
@Table(name = "ways_of_delivery")
@Data
public class WaysOfDelivery {
    @Id
    @Column(name = "way_id")
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long way_id;
    @Column(name = "ways")
    private String ways;
    @Column(name = "delivery_price")
    private Integer delivery_price;
}

```

Файл AgeRepo.java

```

package com.serveeer.repos;
import com.serveeer.domain.Age;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;
import org.springframework.stereotype.Repository;
@Repository
public interface AgeRepo extends JpaRepository<Age, Long> {
    @Query("select a from Age a where a.age_number=?1")
    Age findAgeByAge_number(int name);
}

```

Файл AuthorRepo.java

					ПГУ1.090301.06.001 ПЗ	Лист
						82
Изм.	Лист	№ докум.	Подпись	Дата		

```

package com.serveeer.repos;
import com.serveeer.domain.Author;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;
import org.springframework.stereotype.Repository;
@Repository
public interface AuthorRepo extends JpaRepository<Author, Long>
{
    @Query("select a from Author a where a.author_name=?1")
    Author findAuthorByAuthor_name(String name);
}

```

Файл BookRepo.java

```

package com.serveeer.repos;
import com.serveeer.domain.Book;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.CrudRepository;
import org.springframework.stereotype.Repository;
import java.util.List;
@Repository
public interface BookRepo extends CrudRepository<Book, Long> {
    List<Book> findByTitle(String title);
    @Query("select b from Book b where b.book_id=?1")
    Book findBookByBook_id(Long id);
    @Query("select b from Book b join Author a on
b.author.author_id = a.author_id where a.author_name=?1")
    List<Book> findByAuthor(String author);
    @Query("select b from Book b join Genre a on
b.genre.genre_id = a.genre_id where a.genre_name=?1")
    List<Book> findByGenre(String genre);
}

```

Файл BuyBookRepo.java

```

package com.serveeer.repos;
import com.serveeer.domain.BuyBook;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

@Repository
public interface BuyBookRepo extends JpaRepository<BuyBook,
Long> {
}

```

Файл BuyRepo.java

```

package com.serveeer.repos;
import com.serveeer.domain.Buy;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;
import java.util.List;
@Repository
public interface BuyRepo extends JpaRepository<Buy, Long> {
}

```

Файл CityRepo.java

					ПГУ1.090301.06.001 ПЗ	Лист
						83
Изм.	Лист	№ докум.	Подпись	Дата		

```

package com.serveeer.repos;
import com.serveeer.domain.City;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;
import org.springframework.stereotype.Repository;
@Repository
public interface CityRepo extends JpaRepository<City, Long> {
    @Query("select c from City c where c.city_name=?1")
    City findCityByCity_name(String city);
}

```

Файл CustomerRepo.java

```

package com.serveeer.repos;
import com.serveeer.domain.Customer;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;
import org.springframework.stereotype.Repository;

@Repository
public interface CustomerRepo extends JpaRepository<Customer, Long> {
    @Query("select c from Customer c where c.username=?1")
    Customer findByUsername(String username);
    Customer findCustomerByEmail(String email);
}

```

Файл GenreRepo.java

```

package com.serveeer.repos;
import com.serveeer.domain.Genre;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;
import org.springframework.stereotype.Repository;
@Repository
public interface GenreRepo extends JpaRepository<Genre, Long> {
    @Query("select g from Genre g where g.genre_name=?1")
    Genre findGenreByGenre_name(String name);
}

```

Файл PromoRepo.java

```

package com.serveeer.repos;
import com.serveeer.domain.PromoCode;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;
@Repository
public interface PromoRepo extends JpaRepository<PromoCode, Long> {
    PromoCode findPromoCodeByCode(String code);
}

```

Файл PublisherRepo.java

```

package com.serveeer.repos;
import com.serveeer.domain.Publisher;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;

```

					ПГУ1.090301.06.001 ПЗ	Лист
						84
Изм.	Лист	№ докум.	Подпись	Дата		

```

import org.springframework.stereotype.Repository;
@Repository
public interface PublisherRepo extends JpaRepository<Publisher,
Long> {
    @Query("select p from Publisher p where
p.publisher_name=?1")
    Publisher findPublisherByPublisher_name(String name);
}

```

Файл WaysRepo.java

```

package com.serveeer.repos;
import com.serveeer.domain.WaysOfDelivery;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;
@Repository
public interface WaysRepo extends JpaRepository<WaysOfDelivery,
Long> {
    WaysOfDelivery findWaysOfDeliveriesByWays(String ways);
}

```

					ПГУ1.090301.06.001 ПЗ	Лист
						85
Изм.	Лист	№ докум.	Подпись	Дата		

Приложение В

(Обязательное)

Клиент серверное приложение «Книжный интернет-магазин»

Презентация

					ПГУ1.090301.06.001 ПЗ	Лист
						86
Изм.	Лист	№ докум.	Подпись	Дата		



Книжный интернет-магазин



Актуальность темы

- В современном мире онлайн-книжные магазины становятся все более важными источниками литературы и информации для многих читателей. С развитием информационных технологий и доступности интернета пользователи все чаще обращаются к электронным ресурсам для поиска и приобретения книг.
- Веб-приложения для книжных магазинов могут значительно улучшить доступность и удобство использования книжных ресурсов. Они позволяют клиентам быстро и легко найти интересующие их книги, делать заказы, а также получать доступ к электронным версиям произведений.



Изм.	Лист	№ докум.	Подпись	Дата

ПГУ1.090301.06.001 ПЗ

Лист

87



Преимущества

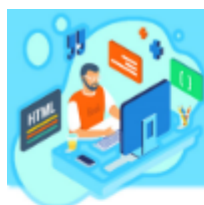
- 1 Удаленный доступ
- 2 Повышение эффективности
- 3 Улучшение обслуживания
- 4 Быстрый доступ к книгам
- 5 Удобство использования



Анализ аналогов

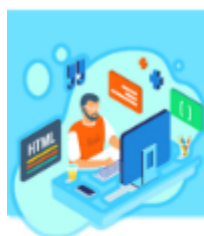
- Каждый год на рынке появляются новые сервисы и приложения, расширяя горизонты возможностей для пользователей. Они приносят с собой не только удобство, но и инновации, делая использование продуктов еще более комфортным и интересным. Одним из таких инновационных приложений являются библиотечные платформы, позволяющие легко и удобно искать, заказывать и анализировать книги в электронном формате.
- Разработка подобного приложения требует тщательного анализа рынка и аналогов. Изучение существующих решений позволяет выявить их сильные и слабые стороны, а также определить ключевые требования к будущему продукту. В данном примере будут рассмотрены три известных сервиса для покупки книг и их особенности, чтобы понять, что можно усовершенствовать или изменить в нашем приложении.





Названия аналогов

- книжный интернет-магазин «*chitai-gorod.ru*»;
- книжный интернет-магазин «*book24.ru*»;
- книжный интернет-магазин «*bookvoed.ru*»;



Плюсы и минусы

Функционал	chitai-gorod.ru	book24.ru	bookvoed.ru
Поиск по критериям	+	+	+
Возможность покупки онлайн	+	+	+
Промо-коды на заказы книг	-	-	-
Акции и скидки	+	+	-
Возможность упрощенного заказа книг	-	+	-
Управление аккаунтом	+	-	+
Контактная информация	+	-	+
Понятный интерфейс	+	+	-





Архитектура приложения

Данный раздел представляет обзор архитектуры клиент-серверного приложения и взаимодействия выбранных технологий внутри продукта. Основным архитектурным паттерном для данного приложения была выбрана MVC архитектура (Model-View-Controller).

MVC (Model-View-Controller) – это распространенный подход к разработке приложений, который структурирует приложение на три основных компонента: модель (Model), представление (View) и контроллер (Controller). Этот подход помогает разделить бизнес-логику, пользовательский интерфейс и управление потоком данных, делая код более организованным и легким для поддержки и развития.

Модель – это набор классов, отвечающих за бизнес-логику приложения. Здесь реализуется работа с данными, их хранение, чтение и запись в базу данных, а также проверка прав доступа пользователей. В контексте данного приложения модель отвечает за хранение информации о единицах, авторах, пользователях и других сущностях, а также за управление доступом к ним.

Представление – это часть приложения, отвечающая за отображение данных пользователю. В данном случае оно включает в себя HTML-шаблоны, формирование пользовательского интерфейса и взаимодействие с пользователями.

Контроллер – это промежуточное звено между моделью и представлением. Он получает запросы от клиента, обрабатывает их, взаимодействует с моделью для получения необходимых данных и передает их представлению для отображения пользователю.



Диаграмма компонентов

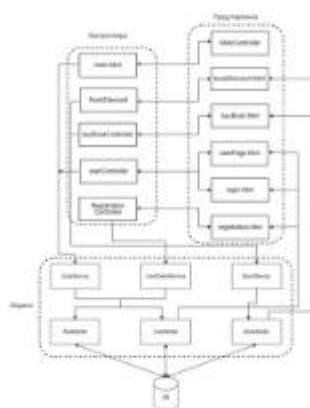
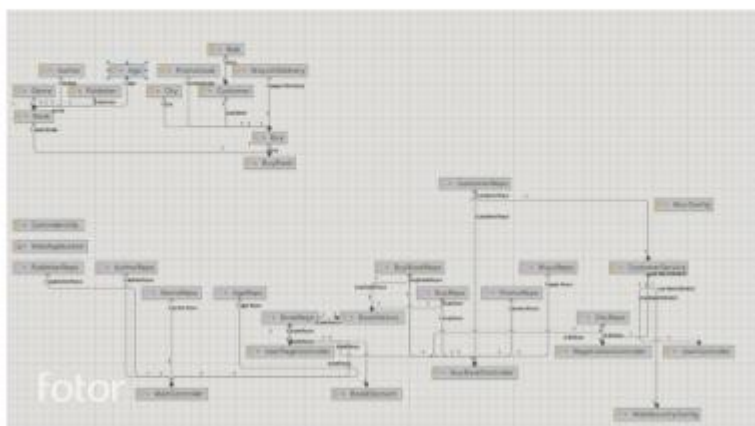




Диаграмма классов



Архитектура приложения

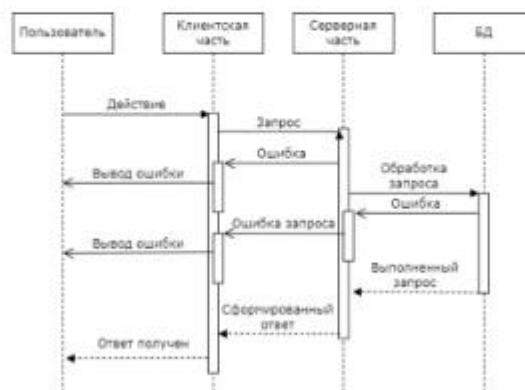
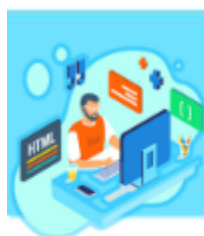
Понимая архитектурные принципы, следует рассмотреть выбранные технологии для реализации клиент-серверного взаимодействия:

- клиентская часть приложения разработана с использованием HTML создания интерфейса пользователя. Для обеспечения взаимодействия с сервером используется архитектурный стиль REST.
- серверная часть приложения реализована на языке программирования Java с использованием Spring Framework и Spring Boot. Spring Framework применяется для управления зависимостями и аннотации зависимостей, а Spring Boot обеспечивает быструю настройку приложения и автоматическую конфигурацию.
- для взаимодействия с базой данных используется Spring JPA, который обеспечивает удобный доступ к базе данных PostgreSQL. Для управления базой данных используется инструмент DBeaver.
- аутентификация и авторизация пользователей осуществляется через Spring Security, что гарантирует безопасность приложения и контроль доступа к его ресурсам.
- для развертывания приложения используется веб-сервер Apache Tomcat, который является контейнером сервлетов Java и обеспечивает быстрый и надежный запуск приложения.
- Взаимодействие этих технологий и архитектурных принципов позволяет надежно, масштабируемо и безопасно работать клиент-серверному приложению, которое соответствует современным требованиям разработки программного обеспечения.

Изм.	Лист	№ докум.	Подпись	Дата



Диаграмма последовательности



Тестирование приложения

Для проверки работоспособности клиент-серверного приложения «Книжный интернет-магазин» необходимо провести его тестирование.

Тестирование будет состоять из следующих действий:

- регистрация пользователя;
- авторизация пользователя;
- приобретение книг, просмотр истории покупок и вкладки «Акции»;
- редактирование аккаунта пользователя;
- проверка функционала неавторизованных пользователей.





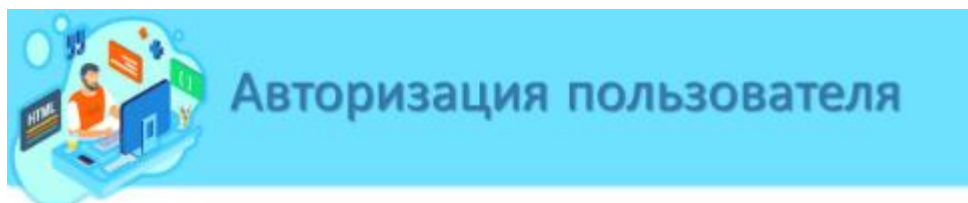
Регистрация пользователя

- Чтобы протестировать процесс регистрации пользователя, нужно перейти на страницу с формой регистрации и ввести запрошенную информацию: имя, фамилию, отчество, город проживания, логин и пароль. После отправки запроса на добавление нового пользователя, в базе данных должна появиться запись о нем.

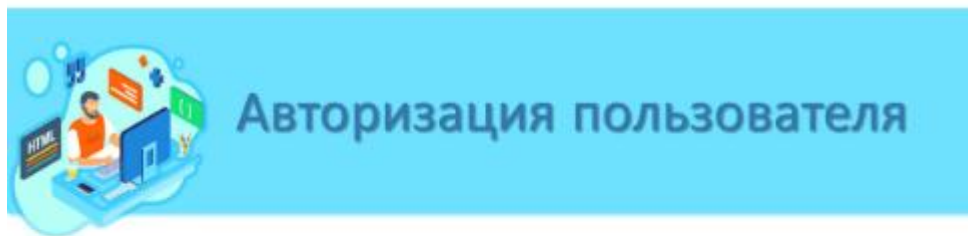


Регистрация пользователя

- Если попытаться зарегистрировать пользователя с e-mail, который уже используется в базе данных, система выдаст сообщение об этой ошибке и предложит ввести другие данные. Этот процесс попытки регистрации с уже существующим логином и отображение соответствующего сообщения изображен на рисунках 17 и 18.

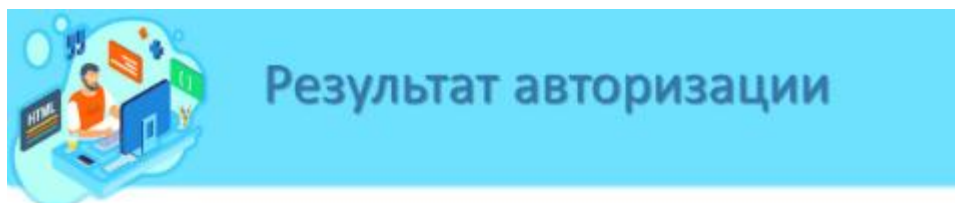
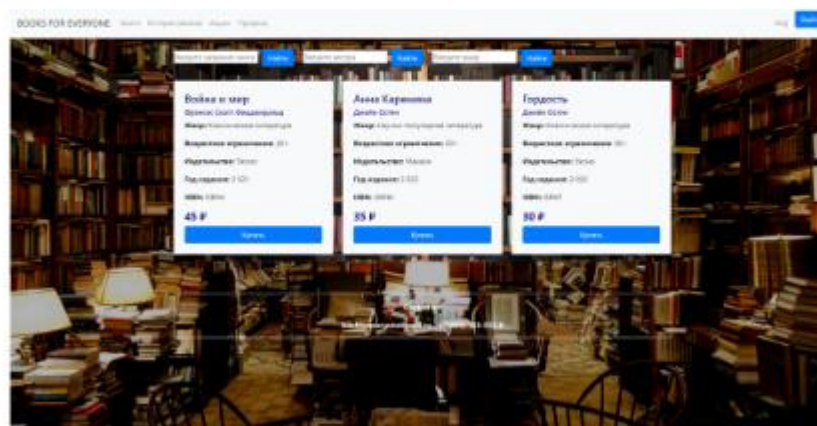


При попытке входа в систему пользователь вводит данные, которые ранее были добавлены в базу данных. Если эти данные совпадают с теми, что хранятся в базе данных, вход будет успешным. В случае несовпадения данных пользователю будет показано сообщение об ошибке.



Изм.	Лист	№ докум.	Подпись	Дата

ПГУ1.090301.06.001 ПЗ



- Авторизованным пользователям доступен функционал просмотра и приобретения книг. Тестовый сценарий состоит из перехода на страницу покупки, где можно будет ввести данные заказа, после ввода данных будет выведено сообщение о успешном заказе. Также есть функционал просмотра приобретенных книг. Помимо этого, авторизованный пользователь может приобрести книги по акции. Для выполнения данного действия нужно перейти на выкладку Акции, где будут отображены доступные книги.



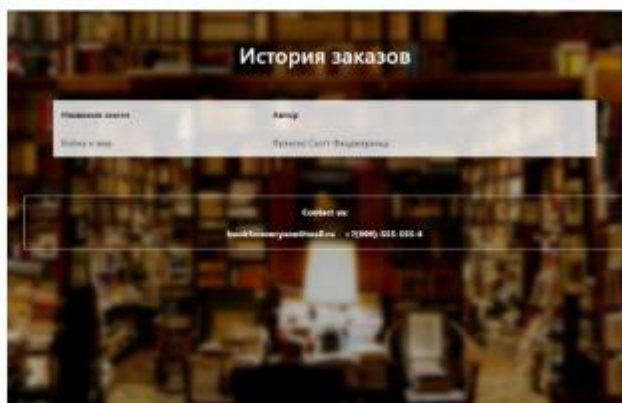
Приобретение книги

- После ввода всех данных будет выведено сообщение, показывающее информацию о доставке, успешно примененный промо-код, благодарность за заказ и возможность перехода на главную страницу.



Просмотр приобретенных книг

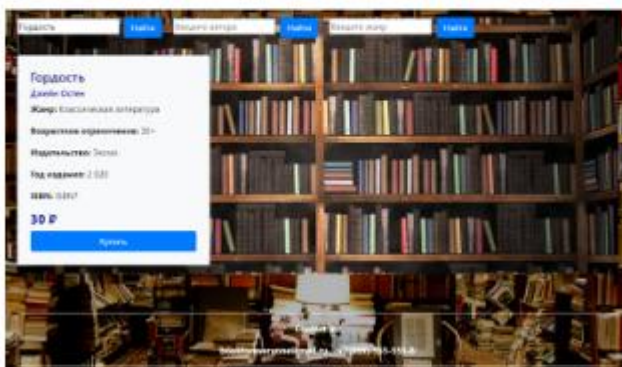
- После перехода пользователь может перейти на вкладку «История заказов», где будет информация о ранее созданных заказах.





Подбор по критериям

- Также авторизованный пользователь может подобрать необходимую книгу по определенным критериям. Для этого нужно ввести в строку критериев ключевые слова для поиска этих книг.



Данные пользователя

- При переходе на вкладку «Профиль», пользователь может изменить собственную информацию о своем профиле и нажать кнопку «Сохранить» для применения их.



Новые данные пользователя

- После нажатия кнопки «Сохранить» на рисунке 31 можно увидеть, что имя справа сверху изменилось на актуальное.



Функционал неавторизированного пользователя

- Пользователь, который не вошел в систему, имеет доступ только к страницам «Главная», «Книги» и информации «Contact us». На странице «Книги» у него ограниченный доступ: он может просматривать только список книг, доступный для покупки, но не может приобретать их.





ПГУ ФВТ 2024

Спасибо за внимание!



					ПГУ1.090301.06.001 ПЗ	Лист
						99
Изм.	Лист	№ докум.	Подпись	Дата		