

СОДЕРЖАНИЕ

ВВЕДЕНИЕ.....	6
1. Описание предметной области.....	7
2. Постановка задач и обзор методов ее решения	8
3. Функциональное моделирование на основе стандарта IDEF0.....	11
3.1. Методология IDEF0.....	11
3.2. Описание разработанной функциональной модели	12
4. Информационная модель системы и ее описание	15
5. Описание алгоритмов, реализующих бизнес-логику серверной части проектируемой системы	18
6. Руководство пользователя	19
6.1. Алгоритм работы регистрации и авторизации	19
6.2. Алгоритм работы покупателя.....	19
6.3. Алгоритм работы продавца.....	21
6.4. Алгоритм работы администратора.....	22
7. Архитектурный шаблон проектирования.....	23
7.1. Диаграмма вариантов использования.....	23
7.2. Диаграмма классов.....	23
1.1 7.3 Диаграмма последовательностей	24
1.2 7.4 Диаграмма состояний	26
8. Результаты тестирования разработанной системы	27
ЗАКЛЮЧЕНИЕ	28
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	29
ПРИЛОЖЕНИЕ А (обязательное) Листинг программного кода	30
ПРИЛОЖЕНИЕ Б (обязательное) Антиплагиат	45
ПРИЛОЖЕНИЕ В (обязательное) Ведомость документов	46

ВВЕДЕНИЕ

Создание интернет магазина, становится все более популярной услугой. По последним данным аудитория в интернете стремительно растет, а продажи через интернет в крупных городах, достигают до 25%, при этом специалисты подчеркивают тенденцию к росту продаж именно через интернет. Сайт интернет-магазин - является современным торговым каналом. С помощью интернет магазина, Вы имеете возможность продавать Ваши товары или услуги огромной аудитории, использующей доступ в Интернет. Интернет магазин для покупателя это: экономия времени, денег и сил. Именно поэтому, по статистике, все больше и больше людей в России совершает свои покупки через интернет магазин.

Ежегодно количество интернет-магазинов увеличивается, так как это действительно прибыльно для Вас и удобно для покупателя, так же интернет - магазин экономит Ваш бюджет и время! Интернет-магазин работает круглые сутки и может продавать определенные товары в автоматическом режиме без участия продавца. Так же не надо закупать товар заранее, а это существенная экономия, на складских помещениях. Вам достаточно договориться с поставщиками, и в нужный момент, просто выкупить товар, который у вас закажут. По сравнению с обычным магазином, территория продаж которого ограничивается населением города или района, территория охвата интернет-магазина увеличивается на весь мир и русскоязычную аудиторию в других странах, ведь товар можно доставлять не только курьерской службой, но и почтой!

Создание интернет магазина, поможет Вашей компании укрепить свои позиции на традиционных рынках и выходить на новые, с помощью интернет магазина Вы сможете увеличить свои продажи, а соответственно оборот и прибыль. Особенно создание интернет магазина, полезно, если вы начинающий предприниматель, вы можете с минимальными затратами организовать свой собственный бизнес, а интернет - магазин поможет увеличить продажи и открыть новые рынки сбыта товаров.

Целью проекта является создать интуитивно понятный интернет-магазин для пользователей, облегченной системой покупок и добавления книг. Верстка и программирование сайта интернет-магазина с понятным графическим пользовательским интерфейсом, с облегченной системой покупки для пользователей и облегченной страницей добавления книг для продавцов.

1. ОПИСАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ

Интернету присущ имидж «среды безграничных возможностей». Однако в нем совсем немного направлений, где можно по-настоящему зарабатывать деньги. Одно из них — интернет-магазины. Они позволяют без больших затрат увеличить число покупателей, предложив им возможности дистанционного заказа и доставки товаров по месту требования.

Интернет-магазины быстрее прижились в тех странах, где уже давно существует устоявшаяся система торговли по каталогам. Сегодня трудности «переходного периода» уже позади, и открытие новых отечественных Интернет-магазинов — заурядное явление. Нужды администраторов интернет-магазина в складском, торговом, бухгалтерском и налоговом учете должны поддерживаться невидимой посетителям частью интернет-магазина — бэк-офисом. Экономически эффективной практикой создания интернет-магазинов является применение специализированных систем учета.

Есть две разновидности интернет-магазинов, в зависимости от вида торговли:

1) Магазины, которые продают товар со своего склада. Такой магазин — прекрасный вариант дополнительного сбыта товара, обычно дают более низкую цену, чем даже в своем реальном магазине;

2) Магазины, которые продают товар других магазинов/людей. Это может быть торговля внутри страны, либо международная торговля. В этом случае интернет-магазин зарабатывает на комиссии, которую платят продавцы за выставление товара. Здесь интернет-магазин выступает гарантом сделки между продавцом и покупателем. Такие магазины используют систему "репутация" продавца. Кроме того, покупатель может пожаловаться администрации сайта на продавца и получить необходимую помощь по возвращению денег, в случае обмана.

Также магазины могут отличаться по способу продажи:

1) Фиксированная цена товара - с доставкой, включенной в стоимость, либо с доставкой, которая считается отдельно, после оформления заказа (очень часто эффект низкой цены бывает испорчен из-за высокой стоимости доставки, продавцы нарочно могут ставить низкую стоимость на товар, а на доставку наоборот - высокую, на чем и зарабатывают).

2) Система аукциона - на товар объявляется аукцион. Кроме начальной цены, продавец может объявлять так называемую блиц-цену — стоимость, за которую продавец готов отдать товар без торга. Есть такой нюанс, как скрытая цена - продавец ставит очень низкую цену на товар (чтобы при поиске товара, клиент заметил именно его лот), но включает опцию "минимальная ставка" — это минимальная цена, которая скрыта от глаз покупателя, и он должен повышать ставки, пока не достигнет ее, иначе ставка не будет принята.

2. ПОСТАНОВКА ЗАДАЧ И ОБЗОР МЕТОДОВ ЕЕ РЕШЕНИЯ

Необходимо выполнить автоматизацию продаж, работ и анализ добавленных книг в курсовом проекте.

Для реализации данного проекта необходимо выполнить следующие задачи:

- разработать собственную иерархию классов;
- изучить предметную область;
- реализовать клиент-серверное приложение;
- связать базу данных с контроллерами;
- создать удобный интерфейс для продавца и покупателя;
- распределить роли (администратор, продавец и пользователь);
- реализовать не менее двух паттернов проектирования;
- использовать сокрытие данных, перегрузку методов, переопределение методов;
- предусмотреть обработку исключительных ситуаций;
- предоставить продавцам и администраторам аналитическую информацию в виде статистики.
- разработать базу данных MySQL;

Общие возможности всех ролей: регистрация, просмотр каталога, пользоваться поисковиком, добавление комментария к книгам.

Со стороны администратора программа должна предусматривать следующие возможности:

- добавление, редактирование удаление всех книг;
- просмотр статистики продаж своих книг.
- редактирование и удаление пользователей;

Продавец должен иметь возможность реализовать следующие задачи:

- просмотр статистики продаж своих книг.
- работа с корзиной;
- покупка книг;
- скачивание купленных книг;
- добавление, редактирование удаление только своих книг;

Пользователь должен иметь следующие возможности:

- покупка книг;
- работа с корзиной;
- скачивание купленных книг.

Для решения поставленных задач используются MySQL-сервер, IntelliJ IDEA, AllFusion ERwin Data Modeler, Spring Boot, Spring MVC, Spring Security, Thymeleaf, Maven, язык программирования Java.

MySQL. Свободная реляционная система управления базами данных. Разработку и поддержку MySQL осуществляет корпорация Oracle, получившая права на торговую марку вместе с поглощённой Sun Microsystems, которая ранее приобрела шведскую компанию MySQL AB. Продукт распростра-

няется как под GNU General Public License, так и под собственной коммерческой лицензией. Помимо этого, разработчики создают функциональность по заказу лицензионных пользователей. Именно благодаря такому заказу почти в самых ранних версиях появился механизм репликации.

Преимущества:

- предоставляет большой функционал;
- хорошие функции безопасности;
- легко использовать;
- легко масштабируется и подходит для больших баз данных;
- обеспечивает хорошую скорость и производительность;
- обеспечивает хорошее управление пользователями и множественный

контроль доступа.

IntelliJ IDEA. Интегрированная среда разработки программного обеспечения для многих языков программирования, в частности Java, JavaScript, Python, разработанная компанией JetBrains. Первая версия появилась в январе 2001 года и быстро приобрела популярность как первая среда для Java с широким набором

интегрированных инструментов для рефакторинга, которые позволяли программистам быстро реорганизовывать исходные тексты программ.

Преимущества:

- автодополнение кода и качественная отладка;
- удобная навигация;
- безопасный рефакторинг – применить изменения во всем проекте можно за пару кликов;
- функция Live Edit позволяет мгновенно посмотреть все изменения в браузере;
- интерфейс будет понятен даже новичкам.

AllFusion ERwin Data Modeler. Это компьютерная программа для проектирования и документирования баз данных. Модели данных помогают визуализировать структуру данных, обеспечивая эффективный процесс организации, управления и администрирования таких аспектов деятельности предприятия, как уровень сложности данных, технологий баз данных и среды развертывания. Изначально разработанный компанией Logic Works, erwin был приобретен рядом компаний, прежде чем был выделен частной инвестиционной фирмой Parallax Capital Partners, которая приобрела и объединила его в качестве отдельной организации, erwin, Inc., с генеральным директором Адамом Famularo.

Преимущества:

- поддержка стандартной нотации IDEF1x для ER-диаграмм моделей данных, нотации IE и специальной нотации, предназначенной для проектирования хранилищ данных – Dimensional;
- поддержка проектирования информационных хранилищ (на основе Red Brick и Teradata);
- поддержка совместного проектирования (версия для ModelMart);

- поддержка триггеров, хранимых процедур и шаблонов;
- развитые средства проверки корректности моделей данных Reverse Engineering (генерация модели данных на основе анализа существующей базы данных), включая восстановление связей по индексам;
- автоматическая генерация SQL DDL для создания баз данных;
- полная совместимость и поддержка 20-ти типов СУБД на основе прямого доступа к системному каталогу баз данных (отпадает потребность в использовании ODBC).

Spring Framework. Универсальный фреймворк с открытым исходным кодом для Java-платформы. Также существует форк для платформы .NET Framework, названный Spring.NET. Первая версия была написана Родом Джонсоном, который впервые опубликовал её вместе с изданием своей книги «Expert One-on-One Java EE Design and Development».

Основное преимущество Spring – возможность разработки приложения как набора слабосвязанных (loose-coupled) компонентов. Чем меньше компоненты приложения знают друг о друге, тем проще разрабатывать новый и поддерживать существующий функционал приложения. Классический пример – управление транзакциями. Spring позволяет вам управлять транзакциями совершенно независимо от основной логики взаимодействия с БД. Изменение этой логики не порушит транзакционность, равно как изменение логики управления транзакциями не сломает логику программы. Spring поощряет модульность. Компоненты можно добавлять и удалять (почти) независимо друг от друга. В принципе, приложение можно разработать таким образом, что оно даже не будет знать, что управляется Spring.

Паттерны проектирования (шаблоны проектирования) – это готовые к использованию решения часто возникающих в программировании задач. Это не класс и не библиотека, которую можно подключить к проекту, это нечто большее. Паттерны проектирования, подходящий под задачу, реализуется в каждом конкретном случае. Следует, помнить, что такой паттерн, будучи примененным неправильно или к неподходящей задаче, может принести немало проблем. Тем не менее, правильно примененный паттерн поможет решить задачу легко и просто.

Типы паттернов:

- порождающие паттерны;
- структурные паттерны;
- шаблонные паттерны.

Разумное использование паттернов проектирования приводит к повышению надежности обслуживания кода, поскольку в дополнение к тому, чтобы быть хорошим решением общей проблемы, паттерны проектирования могут быть распознаны другими разработчиками, что уменьшает время при работе с определенным кодом.

3. ФУНКЦИОНАЛЬНОЕ МОДЕЛИРОВАНИЕ НА ОСНОВЕ СТАНДАРТА IDEF0

3.1. Методология IDEF0

IDEF0 - нотация графического моделирования, используемая для создания функциональной модели, отображающей структуру и функции системы, а также потоки информации и материальных объектов, связывающих эти функции. Стандарт IDEF0 (Integration Definition for Function Modeling) утвержден в США в 1993 как Федеральный стандарт обработки информации.

Функциональная модель компании. Функциональная модель IDEF0 представляет собой набор блоков, каждый из которых представляет собой «черный ящик» со входами и выходами, управлением и механизмами, которые детализируются (декомпозируются) до необходимого уровня. Наиболее важная функция расположена в верхнем левом углу. А соединяются функции между собой при помощи стрелок и описаний функциональных блоков. При этом каждый вид стрелки или активности имеет собственное значение. Данная модель позволяет описать все основные виды процессов, как административные, так и организационные.

Стрелки могут быть:

Входящие – вводные, которые ставят определенную задачу.

Исходящие – выводящие результат деятельности.

Управляющие (сверху вниз) – механизмы управления (положения, инструкции и пр.).

Механизмы (снизу вверх) – что используется для того, чтобы произвести необходимую работу.

Входящие и исходящие стрелки точнее было бы называть вводными и выводными, так как по-английски они называются *Input* и *Output* соответственно. Но особенности перевода и привычные названия выглядят уже так, как сложилось. И все же для правильного понимания терминов важно помнить их значение в данном случае. Это подтверждается еще и тем, что данная нотация создана прежде всего для разработки ПО, и термины переводить правильнее в этой точки зрения.

У IDEF0 есть еще одно неоспоримое преимущество. Эта методика была разработана сравнительно давно, и за три десятилетия она прошла тщательную шлифовку и корректировку. Поэтому создать функциональную модель компании можно быстро и минимальной вероятностью ошибки.

Естественно, есть и другие методологии, так почему мы рекомендуем именно IDEF0? Отпилить кусок металлической трубы можно и ножовкой, но, согласитесь, сделать это гораздо проще с помощью болгарки. Так и с IDEF0: нет более функционального инструмента для моделирования, с ним получить вы легко и быстро получите нужный вам результат.

Стандарт IDEF0 и его требования.

О базовых требованиях IDEF0 мы говорили чуть выше.

– главный элемент – в верхнем левом углу;

- каждый элемент должен иметь входящие и исходящие стрелки. Причем входящие стрелки находятся слева, справа – исходящие;
- сверху располагаются управляющие элементы, снизу – механизмы;
- при расположении нескольких блоков на одном листе или экране последующие размещаются справа внизу от предыдущего;
- схемы следует создавать так, чтобы стрелки пересекались минимальное количество раз.

Естественно, в стандарте IDEF0 есть общепринятые нормы, требования и обозначения. Подробно на них останавливаться не будем, при желании эту информацию несложно найти.

3.2. Описание разработанной функциональной модели

На диаграмме IDEF0 представлено описание процесса управления финансами предприятия. На рисунке 3.2.1 представлена контекстная диаграмма системы.

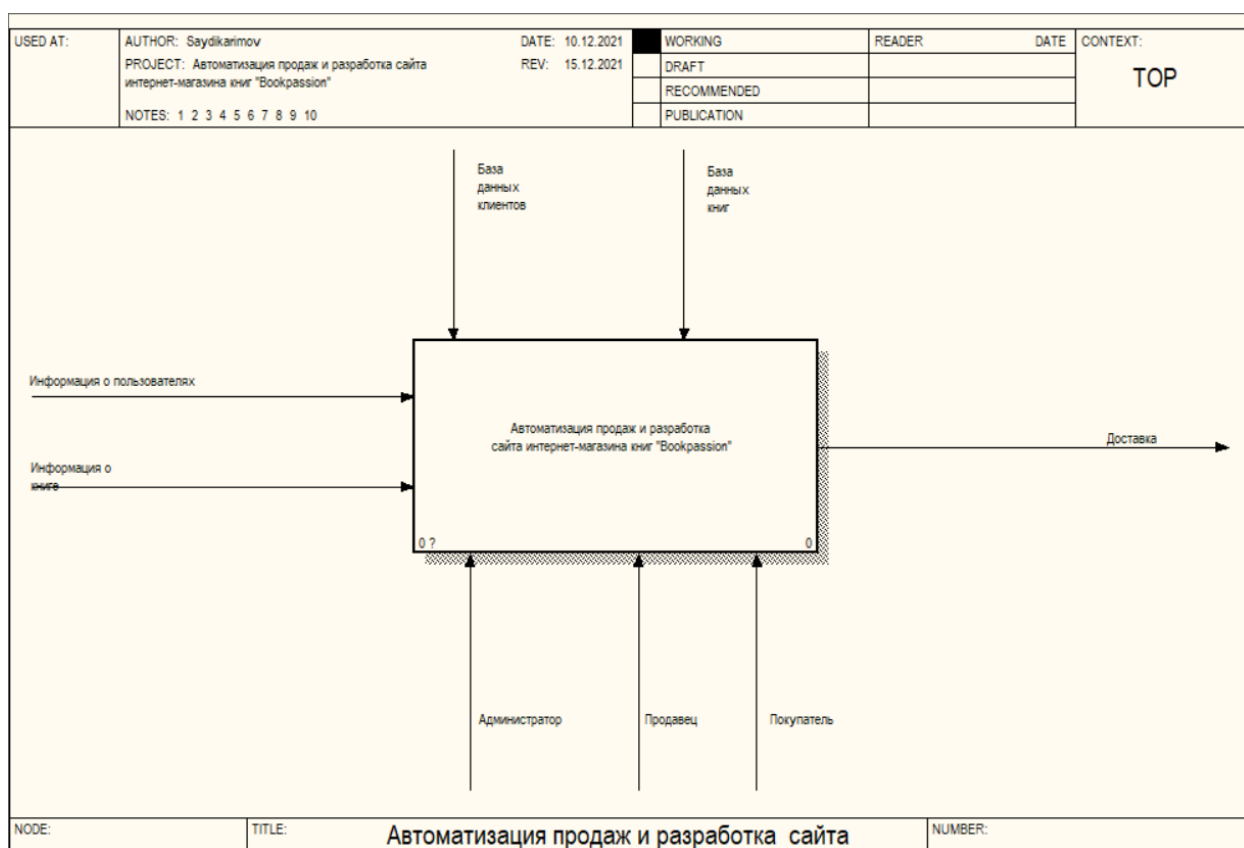


Рисунок 3.2.1 – Контекстная диаграмма системы

Входные данные: информация о пользователях и информация о книге. Механизмами являются: администратор, продавец и покупатель. Управлени-ями будут база данных клиентов, база данных книг.

На следующем рисунке 3.2.2 представлена декомпозиция работы «Раз-работка интернет-магазина».

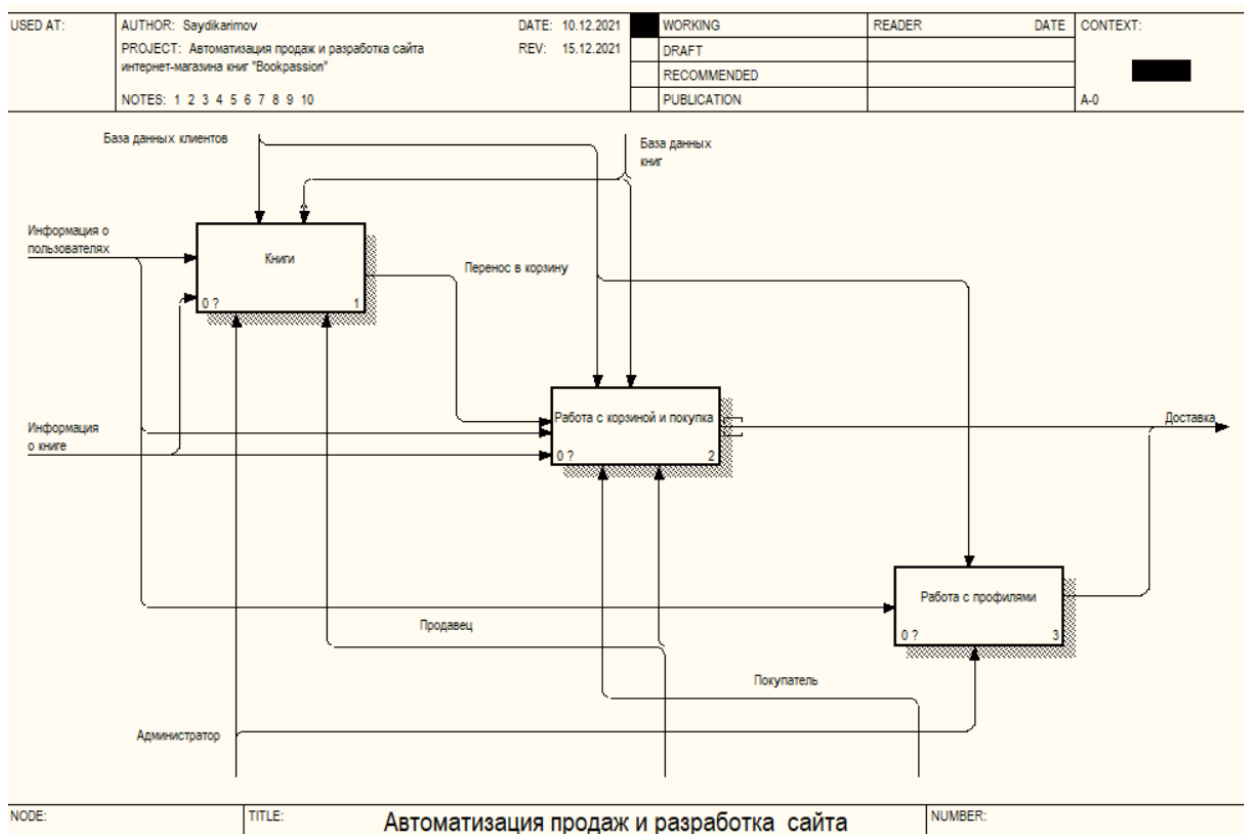


Рисунок 3.2.2 – Декомпозиция работы «Разработка интернет-магазина»

Данный уровень разбит на 3 функциональных блока:

- книги;
- работа с корзиной и покупки;
- работа с профилями.

Стрелки управления:

- «база данных клиентов» входит во все блоки;
- «база данных книг» входит в первые два блока;

Стрелки ввода:

- «информация о пользователях» входит во все блоки;
- «информация о книге» входит первые два блока.

Стрелки механизма:

- «администратор» входит в первый и последний блок;
- «продавец» входит в первые два блока;
- «покупатель» входит в блок «работа с корзиной и покупки».

В блоке «книги» происходит работа с книгами со стороны администратора и продавца: добавление, редактирование и удаление книг.

В блоке «работа с корзиной и покупки» идет процесс покупки книги и основная бизнес-логика текущего курсового проекта.

В блоке «работа с профилями» идет процесс редактирования профилей со стороны администратора.

На следующем рисунке 3.2.3 показана декомпозиция работы «Книги».

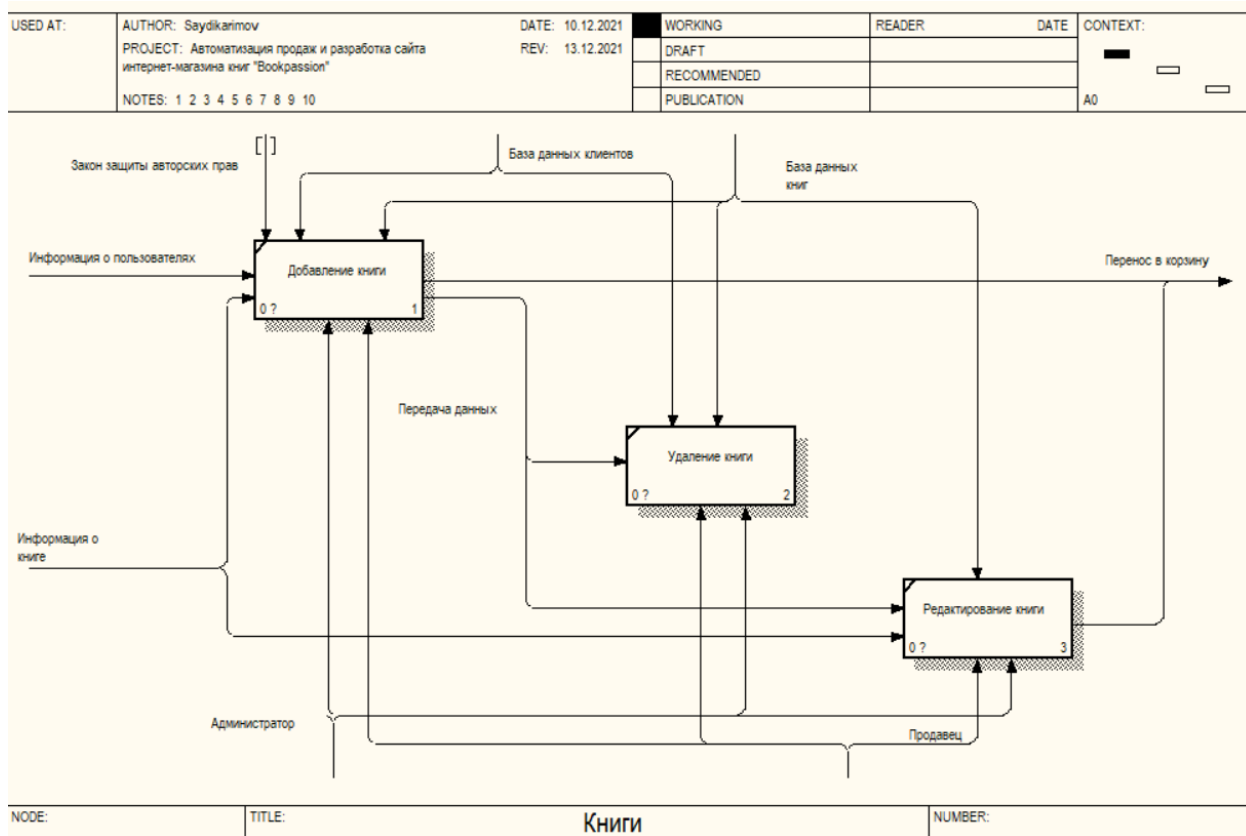


Рисунок 3.2.3 – Декомпозиция работы «Книги»

4. ИНФОРМАЦИОННАЯ МОДЕЛЬ СИСТЕМЫ И ЕЕ ОПИСАНИЕ

Информационная модель — модель объекта, представленная в виде информации, описывающей существенные для данного рассмотрения параметры и переменные величины объекта, связи между ними, входы и выходы объекта и позволяющая путём подачи на модель информации об изменениях входных величин моделировать возможные состояния объекта

Формами представления информационной модели могут быть: любое словесное описание (в том числе описание алгоритма), таблица, рисунок, схема, чертеж, формула, компьютерная программа и т. д.

Компьютерная информационная система (ИС) – система, предназначенная для хранения, поиска и обработки информации, и соответствующие организационные ресурсы (человеческие, технические, финансовые и т. д.), которые обеспечивают и распространяют информацию (ISO/IEC 2382:2015). Предназначена для своевременного обеспечения надлежащих людей надлежащей информацией, то есть для удовлетворения конкретных информационных потребностей в рамках определённой предметной области, при этом результатом функционирования компьютерных информационных систем является информационная продукция – документы, информационные массивы, базы данных и информационные услуги.

Современное понимание информационной системы предполагает использование в качестве основного технического средства переработки информации персонального компьютера (сервера, периферийного оборудования и т.д.).

Необходимо понимать разницу между компьютерами и информационными системами. Компьютеры, оснащенные специализированными программными средствами, являются технической базой и инструментом для информационных систем. Информационная система немыслима без персонала, взаимодействующего с компьютерами и телекоммуникациями.

Говоря об информационной системе, следует рассмотреть следующие вопросы: структура ИС, классификации ИС.

Структура ИС обычно рассматривается как совокупность различных подсистем. Все подсистемы можно рассматривать как по отдельности, так и во взаимосвязи друг с другом.

Классифицировать информационные системы можно по различным признакам. В отечественной литературе по информационным системам управления ИС классифицируют обычно по следующим признакам:

- по типу объекта управления (ИС управления технологическим процессом, ИС организационного управления);
- по степени интеграции (локальные, интегрированные);
- по уровню автоматизации управления (информационно-справочные системы, системы обработки данных, информационно-советующие системы, системы принятия решений, экспертные системы);

- по уровню управления (информационные системы управления предприятием, корпорацией, отраслью);
- по характеру протекания технологических процессов на объекте управления (автоматизированная система управления дискретным производством, автоматизированная система управления непрерывным производством)
- по признаку структурированности задачи;
- и другие.

Для обеспечения минимальной избыточности и физического объёма данных, а также для более быстрого доступа, модель приведена к 3 нормальной форме.

Для моделирования системы существует 3 этапа проектирования:

- концептуальное проектирование;
- логическое проектирование;
- физическое проектирование.

Концептуальное проектирование представляет собой начальный этап разработки программного обеспечения (ПО), когда определяется базовая структура информационной системы (ИС), ее компоненты, их назначение и взаимосвязь.

Следующим этапом является логическое проектирование, оно представляет собой описание логической структуры данных посредством системы управления базами данных (СУБД), для которой проектируется БД (см. рис. 4.1).

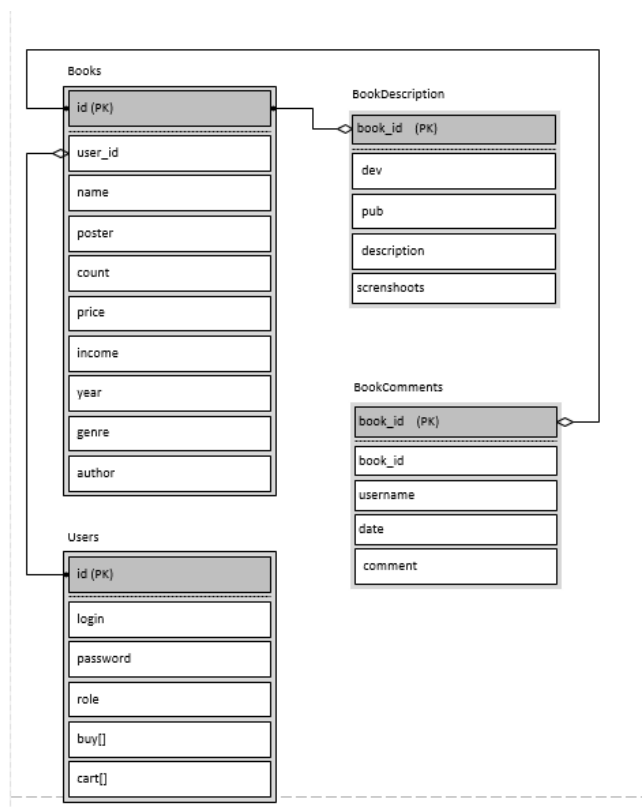


Рисунок 4.1 – Логическая модель системы

Логическая модель системы разработана с помощью редактора Microsoft Visio. Система содержит 4 сущности: пользователи (Users), книги (Books), описание книги (BookDescription), Комментарии (BookComments).

Сущность – это любой различимый объект, о котором необходимо хранить информацию в базе данных. Каждая сущность содержит атрибуты. Они отображают качества и свойства объекта.

Сущность «пользователи» содержит 6 атрибутов: логин, пароль, роль, два массива cart (корзина) и buy (купленные) в которых находятся id книг.

Сущность «книги» содержит 9 атрибутов: id – уникальный идентификационный номер (отдельно от пользователей), название книги, картину постера, суммарный заработок, количество проданных копий, год выпуска, жанр, цена и автор.

Сущность «комментарии» содержит 4 атрибута: id книги, имя пользователь, дата и комментарий. Сущность связана с сущностью «книга».

Сущность «описание» содержит 6 атрибутов: id книги, издатель, описание, вес книги, скриншоты книги и штрих-коды.

Этап физического проектирования. Это описание физической структуры базы данных. Физическая модель системы создана при помощи программы Microsoft Visio (см. рис. 4.2).

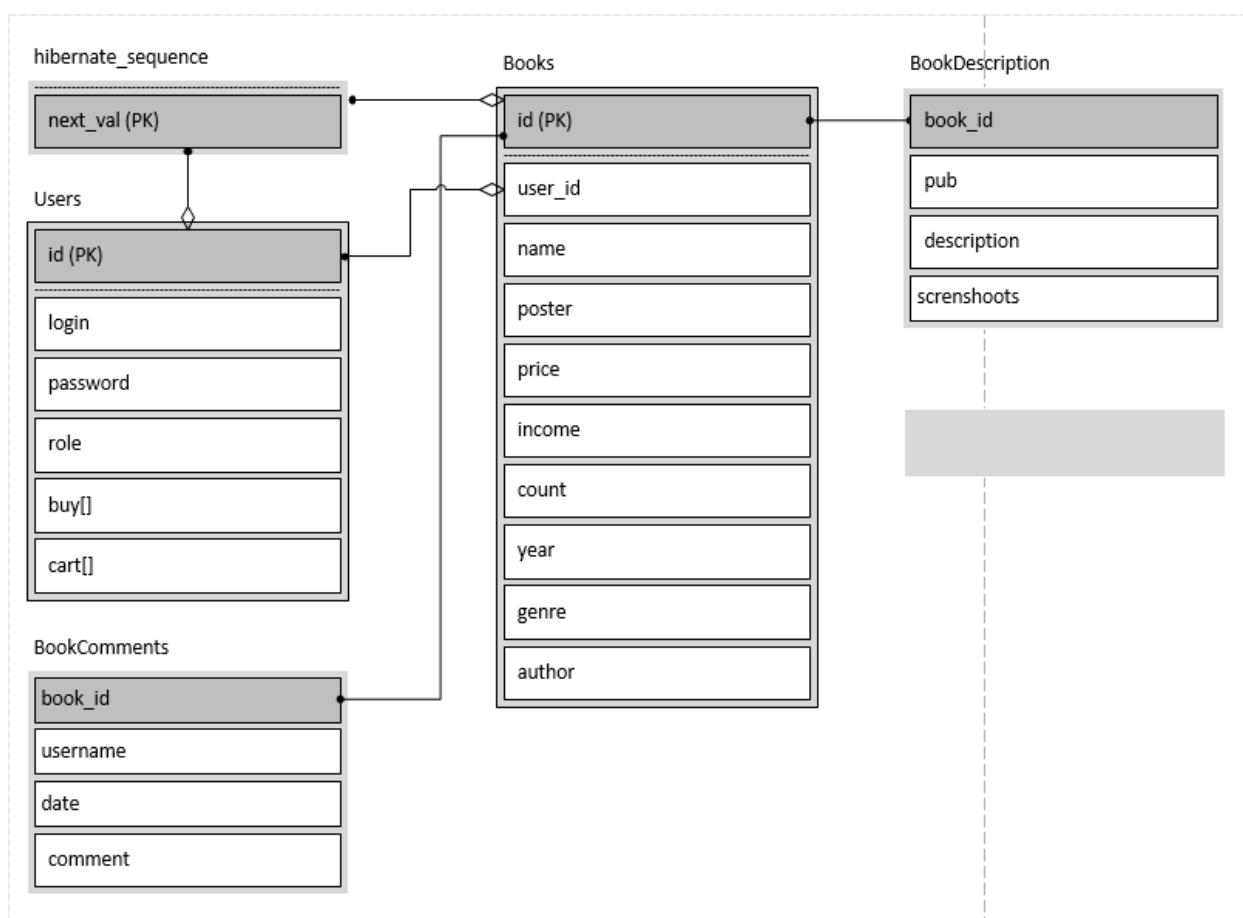


Рисунок 4.2 – Физическая модель системы

5. ОПИСАНИЕ АЛГОРИТМОВ, РЕАЛИЗУЮЩИХ БИЗНЕС-ЛОГИКУ СЕРВЕРНОЙ ЧАСТИ ПРОЕКТИРУЕМОЙ СИСТЕМЫ

Алгоритм - это точно определённая инструкция, последовательно применяя которую к исходным данным, можно получить решение задачи.

Рассмотрим алгоритм работы продажи книг (см. рисунок 5.1).



Рисунок 5.1 – Алгоритм продажи книг

Для начала, администратор или продавец добавляет книгу для возможности приобрести их клиентам.

Далее покупатель или продавец (не владелец) покупает книгу или добавляет ее в корзину.

Основным этапом является алгоритм добавления ИД книг в корзину или в купленные книги Пользователя.

В случае удачной покупки из корзины, ИД из корзины Пользователя переходит в купленные книги.

6. РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

Данный сайт интернет-магазина представляет собой электронную коммерческую деятельность и разработку интернет-магазина.

6.1. Алгоритм работы регистрации и авторизации

При переходе на страницу «Входа», пользователь должен ввести логин и пароль (см. рис. 6.1.1), или перейти на регистрацию. При добавление нового пользователя ему присваивается роль покупателя, только пользователи с ролью админа могут изменять роли другим пользователям.

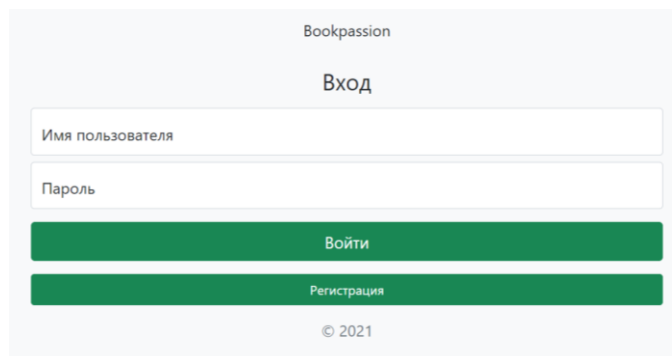


Рисунок 6.1.1 – Страница «Входа»

При регистрации нового пользователя, длина логина должна быть не больше 20 символов и длина пароля не меньше 8 символов (см. рис. 6.1.2).

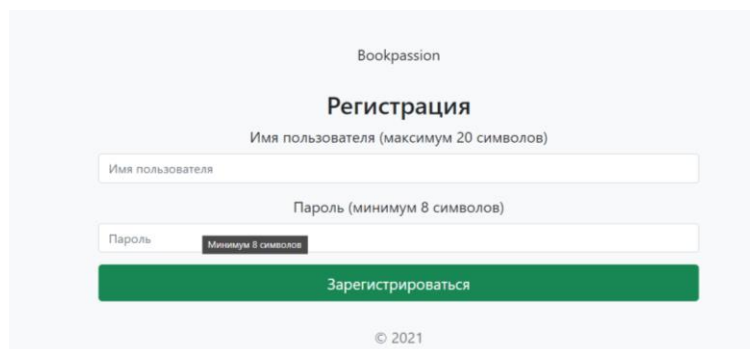


Рисунок 6.1.2 – Страница «Регистрации»

В случае успешной регистрации логин не должен совпадать с уже зарегистрированными пользователями, пользователь будет перенаправлен в страницу входа в случае успешной регистрации.

6.2. Алгоритм работы покупателя

Покупатель имеет возможность работать с собственной корзиной (см. рис. 6.2.1), просматривать каталог (см. рис. 6.2.2), искать книги, покупать книги (см. рис. 6.2.3).

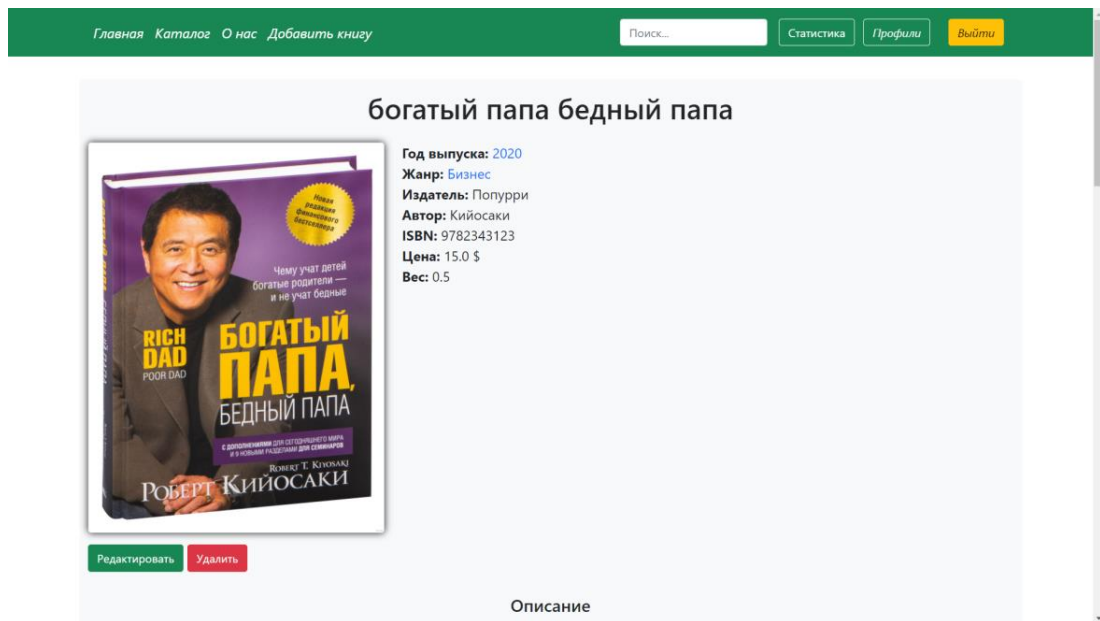


Рисунок 6.2.1 – Работа с корзиной

Рисунок 6.2.2 – Каталог

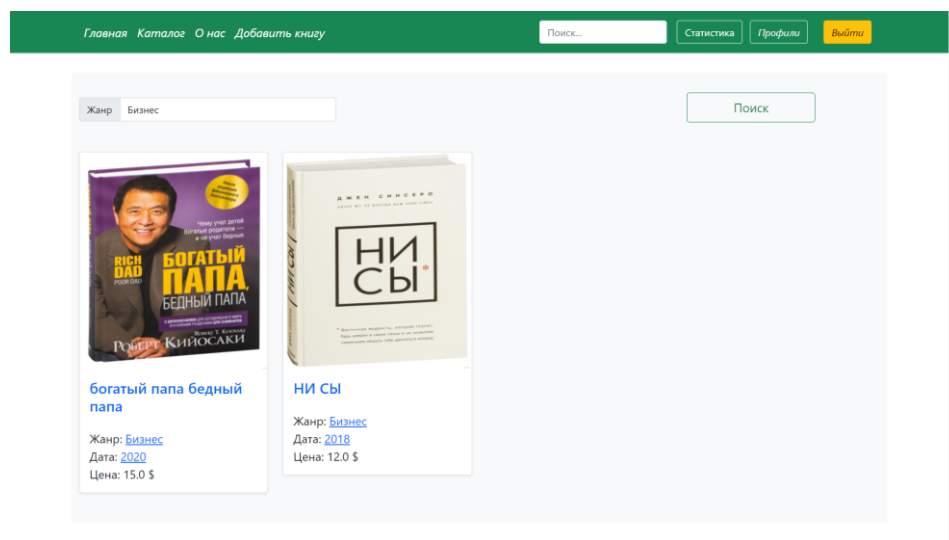


Рисунок 6.2.2 – Кнопки добавления в корзину и прямой покупки

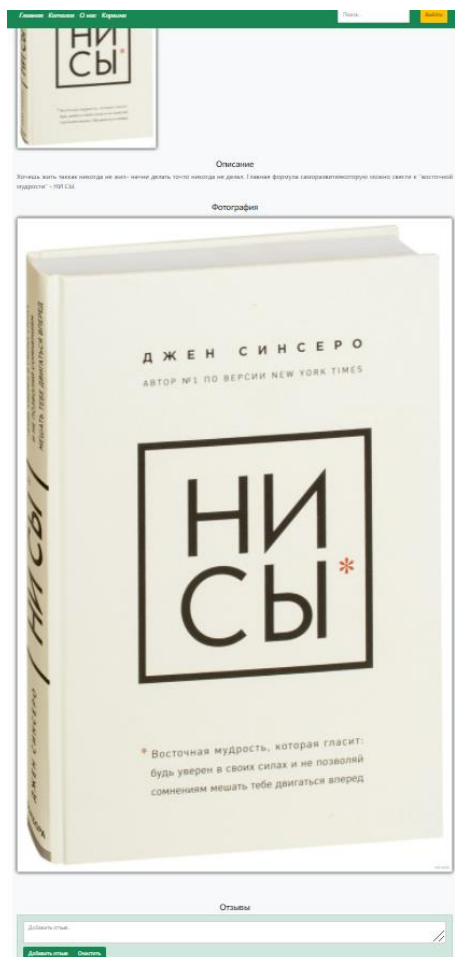


Рисунок 6.2.2 – Результат после покупки

6.3. Алгоритм работы продавца

Продавец имеет такие возможности что и покупатель, но имеет еще возможности добавлять книгу (см. рис. 6.3.1), редактировать добавленную книгу (см. рис. 6.3.2) и смотреть свою статистику продаж книги (см. рис. 6.3.3).

Рисунок 6.3.1 – Страница добавления книги

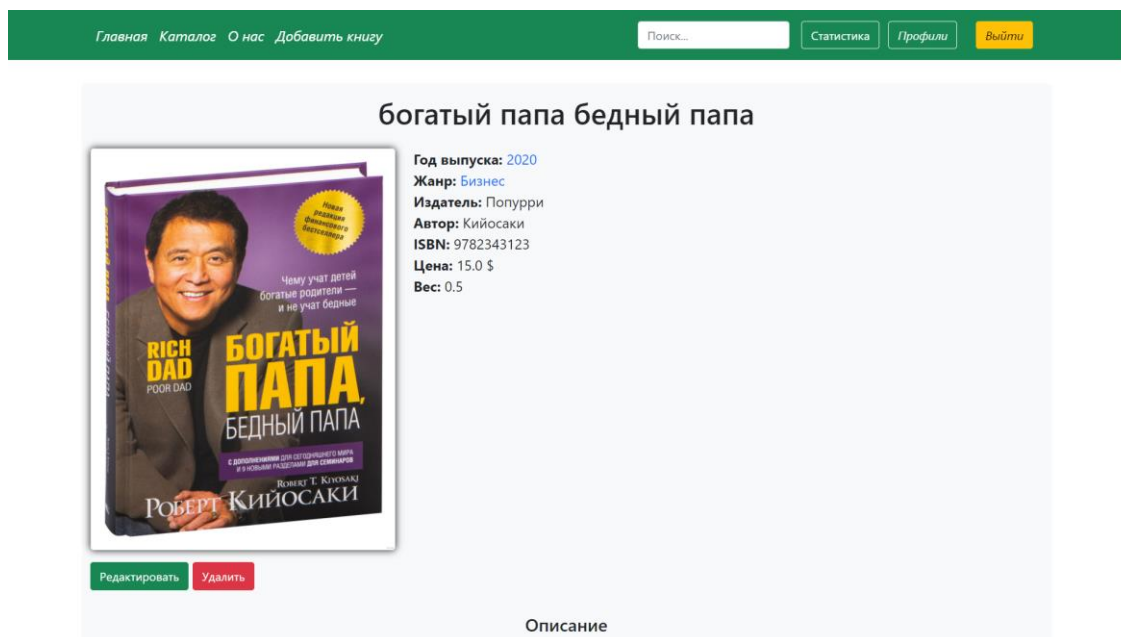


Рисунок 6.3.2 – Страница книги с ролью продавца и владельца

Bookpassion			
Название книги	Количество проданных книги	Стоимость одной копии	Выручка
богатый папа бедный папа	0	15.0	0.0
НИ СЫ	1	12.0	12.0
Выручка со всех книг			12.0

Рисунок 6.3.3 – Статистика продаж

6.4. Алгоритм работы администратора

Администратор имеет те же возможности что и продавец, за исключением работы с корзиной, но имеет возможности редактирования и удаления всех книг, редактировать профили (см. рис. 6.4.1).

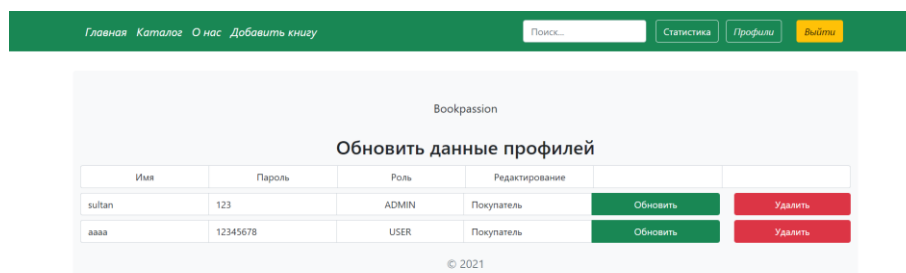


Рисунок 6.4.1 – Страница редактирования профилей

7. АРХИТЕКТУРНЫЙ ШАБЛОН ПРОЕКТИРОВАНИЯ

7.1. Диаграмма вариантов использования

Диаграмма вариантов использования (см. рис. 7.1.1) строится во время изучения технического задания, она состоит из графической диаграммы, описывающей действующие лица и прецеденты (варианты использования), а также спецификации, представляющего собой текстовое описание конкретных последовательностей действий (потока событий), которые выполняет пользователь при работе с системой.



Рисунок 7.1.1 – Диаграмма вариантов использования

Гость может только просматривать каталог и регистрироваться.

Покупатель имеет такие же права как у покупателя, еще добавочно может покупать книгу, добавлять комментарии и работать с корзиной.

Продавец имеет те же права что покупатель, еще добавочно может добавлять книгу в каталог, просматривать статистику, редактировать свои книги и удалять их.

Администратор имеет все права доступа и все возможности других ролей, также имеет возможности редактировать профили, редактировать все книги и удалять их.

7.2. Диаграмма классов

Диаграмма классов (см. рис. 7.2.1) – это набор статических, декларативных элементов модели. Диаграммы классов могут применяться и при прямом проектировании, то есть в процессе разработки новой системы, и при обратном проектировании - описании существующих и используемых систем.

Рисунок 7.2.1 – Диаграмма классов

7.3. Диаграмма последовательностей

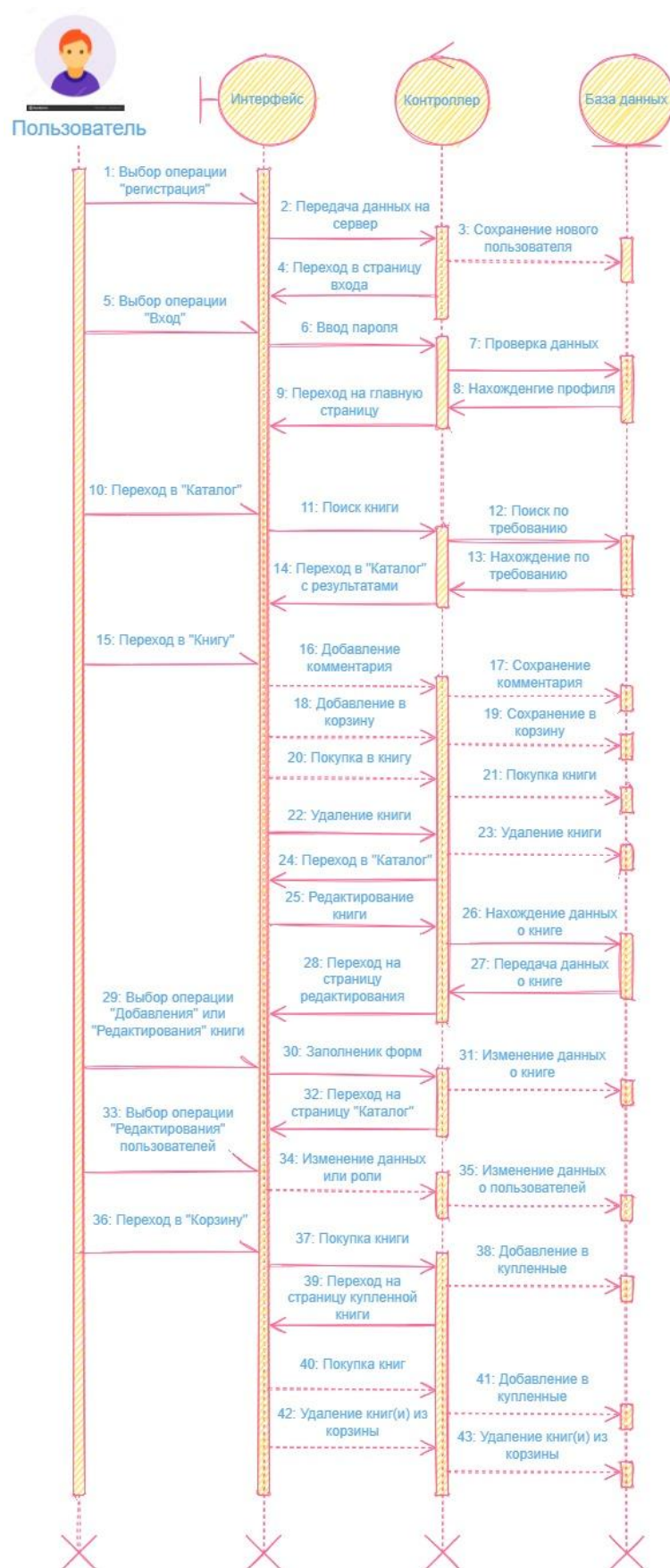


Рисунок 7.3.1 – Диаграмма последовательностей

На диаграмме показан алгоритм работы пользователей в страницах регистрации, вход, каталог, книга, редактирование книги, редактирование пользователей и корзина. При работе через интерфейс происходит запуск логики контроллера и взаимодействие с базой данных книг и пользователей.

7.4. Диаграмма состояний

Диаграмма состояний (см. рис. 7.4.1) – это, по существу, диаграмма состояний из теории автоматов со стандартизированными условными обозначениями, которая может определять множество систем от компьютерных программ до бизнес-процессов.

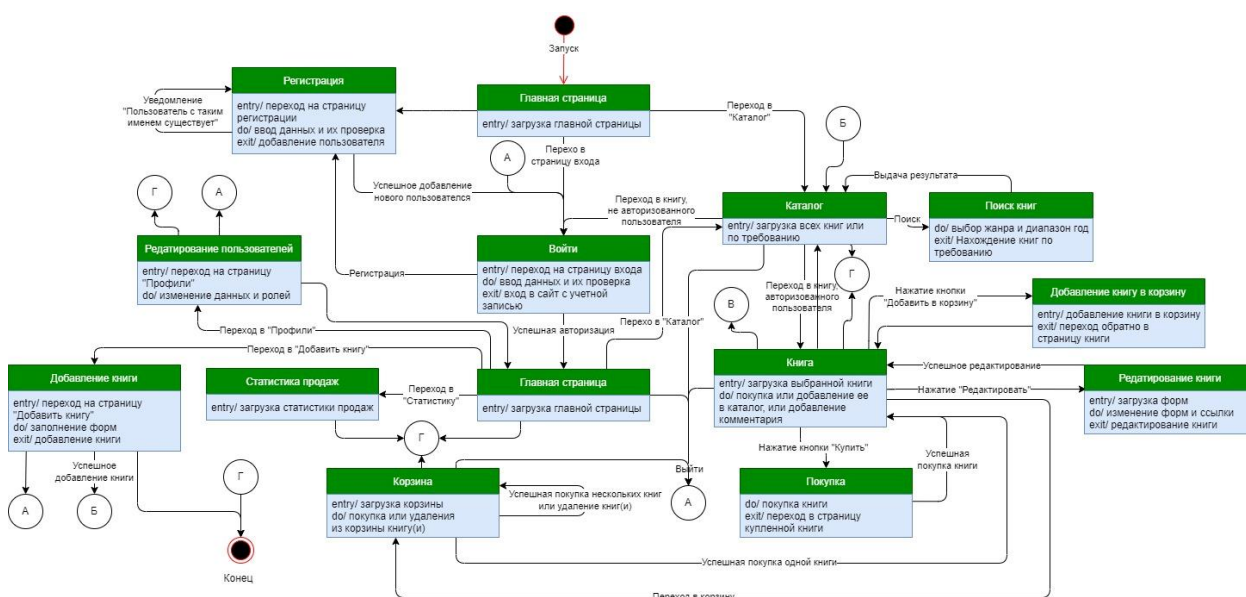


Рисунок 7.4.1 – Диаграмма состояний

В диаграмме показывает, как объект переходит из одного состояния в другое. Например, для входа в систему нужно авторизоваться, если нету, то новый пользователь переходит в страницу регистрации и создает новый профиль, при создании проверяет уникальность логина, после успешной регистрации нового пользователя, контроллер перенаправляет на страницу вход, после входа пользователь перенаправляется в главную страницу, или, администратор захотел изменить роль продавца у пользователя на покупателя, после изменения все добавленные книги этим пользователем до этого момента будут удалены, или администратор захотел сам изменить свою роль на покупателя, то так все книги добавленные до этого момента будут удалены и пользователя контроллер перенаправит в главную страницу.

8. РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ РАЗРАБОТАННОЙ СИСТЕМЫ

В ходе процесса выполнения курсового проекта были вынесены следующие классификации видов тестирования (критерий — степень изолированности кода). Тестирование бывает:

Блочное (UNIT-TESTING) — тестирование одного модуля в изоляции.

Интеграционное (INTEGRATION TESTING) — тестирование группы взаимодействующих модулей.

Системное (SYSTEM TESTING) — тестирование системы в целом.

Классификация хорошая и понятная. Однако на практике выясняется, что у каждого вида тестирования есть свои особенности. И если их не учитывать, тестирование становится обременительным и им не занимаются в должной мере. Здесь собраны подходы к реальному применению различных видов тестирования.

В процессе разработки были обработаны исключения со стороны контроллера и интерфейса при регистрации, добавлении книги, удаление.

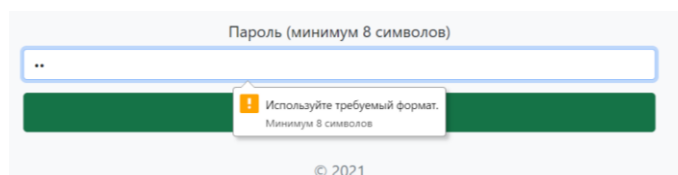


Рисунок 8.1 – Уведомление при регистрации нового пользователя

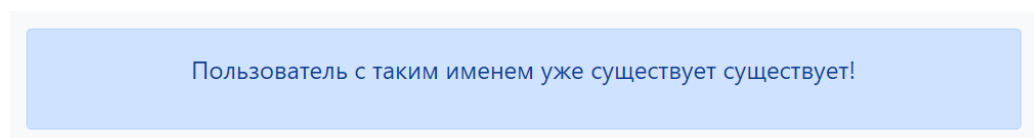


Рисунок 8.2 – Уведомление контроллера при регистрации нового пользователя с уже существующим именем

ЗАКЛЮЧЕНИЕ

В ходе выполнения курсового проекта был произведен анализ предметной области, реализовано программное и информационное обеспечение, подготовлены контрольные примеры и произведены тестирования.

Система имеет простой для пользования интерфейс для покупателя и продавца, программа автоматизирует работу для удобной реализации продажи книг.

Все поставленные задачи были выполнены после завершения программы. Обновления и усовершенствования могут быть сделаны при необходимости в будущем для более масштабного проекта, такие как: внутри каталог книг; поиск книг по дате выпуска; поиск книг по издателям; добавление книг, не смотря на то, что данная программа является законченной.

В процессе написания курсового проекта были изучены Spring Framework, и усовершенствованы навыки работы с базами данных, используя систему управления базами данных MySQL, также были изучена работа с гипертекстовой разметкой страниц сайта и их визуальное обработаны стили (HTML/CSS), был использован для облегченной и корректной разметки страниц проекта с помощью разделений на сетки благодаря платформе Bootstrap, дополнительно были использованы сторонние платформы Spring Framework, такие как паттерн проектирования Spring MVC, работа с пользователями Spring Security и набор настроенных модулей Spring Boot упрощающих конфигурацию приложений, написанного на платформе Spring Framework.

При тестировании данного приложения не было выявлено фатальных ошибок, а все выявленные незначительные ошибки были устранены.

Все поставленные задачи решены. В будущем приложение будет совершенствоваться посредством улучшения и расширения информационной модели системы, оптимизации имеющихся функций и добавления новых. Также будет улучшен интерфейс приложения.

Выполнены следующие поставленные задачи при выполнении данного проекта. Создан интуитивно понятный визуально минимальным интерфейсом сайта для пользователей, облегченной системой покупок и добавления книги. Добавлена система облегченной покупки книги для пользователей, облегченной страницей добавления книги для продавцов и администраторов. Было реализовано адаптивность веб-дизайн для разных устройств, такие как: широкоформатные экраны; телефоны; ноутбуки.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

- [1] Academic [Электронный ресурс]. – Режим доступа:
<https://dic.academic.ru/dic.nsf/ruwiki/309283>
- [2] chemport [Электронный ресурс]. – Режим доступа:
http://www.chemport.ru/data/chemipedia/article_1048.html
- [3] Хабр [Электронный ресурс]. – Режим доступа:
<https://habr.com/ru/post/350864/>
- [4] Skillbox [Электронный ресурс]. – Режим доступа:
https://skillbox.ru/media/code/chto_takoe_git_obyasnyаем_na_skhemakh/
- [5] Ruseller [Электронный ресурс]. – Режим доступа:
<https://ruseller.com/lessons.php?id=666>
- [9] Vebrost [Электронный ресурс]. – Режим доступа:
<https://vebrost.ru/blog/chto-takoe-internet-magazin/>
- [10] spring [Электронный ресурс]. – Режим доступа:
<https://spring.io/guides/gs/securing-web/>
- [11] spring [Электронный ресурс]. – Режим доступа:
<https://spring.io/guides/gs/accessing-data-mysql/>
- [12] javarush [Электронный ресурс]. – Режим доступа:
<https://javarush.ru/groups/posts/spring-framework-java-1>
- [13] it-brain [Электронный ресурс]. – Режим доступа:
https://ru.it-brain.online/tutorial/spring/spring_quick_guide/
- [14] proglib [Электронный ресурс]. – Режим доступа:
<https://proglib.io/p/java-spring/>
- [15] Bussines studio [Электронный ресурс]. – Режим доступа:
<https://www.businessstudio.ru/wiki/docs/current/doku.php/ru/csdesign/bpmodeling/idef0>
- [16] itproger [Электронный ресурс]. – Режим доступа:
<https://itproger.com/course/java-spring>
- [17] Calltouch [Электронный ресурс]. – Режим доступа:
<https://www.calltouch.ru/glossary/elektronnaya-kommertsiya/>
- [18] habr [Электронный ресурс]. – Режим доступа:
<https://habr.com/ru/company/audiomania/blog/375533/>

ПРИЛОЖЕНИЕ А

(обязательное)

Листинг программного кода

```
BookCont
@GetMapping("/book/{id}")
public String book(@PathVariable(value = "id") Long id, Model model) {
    if (!repoBooks.existsById(id)) return "redirect:/catalog";
    long userid = 0, userIdFromBD, bookid = 0, cart = 1, buy = 1;
    Users userFromDB = new Users();

    Optional<Books> temp = repoBooks.findById(id);
    List<Books> books = new ArrayList<>();
    temp.ifPresent(books::add);

    for (Books g : books) {
        userid = g.getUserid();
        bookid = g.getId();
        break;
    }

    Authentication auth = SecurityContextHolder.getContext().getAuthentication();
    if (!(auth instanceof AnonymousAuthenticationToken) && auth != null) {
        UserDetails userDetails = (UserDetails) auth.getPrincipal();
        if (userDetails != null) userFromDB = repoUsers.findByUsername(userDetails.getUsername());
    }

    userIdFromBD = userFromDB.getId();
    if (userIdFromBD == userid) model.addAttribute("userid", userid);

    List<Comments> comments = repoComments.findAllByBookid(bookid);
    if (userFromDB.getCart() != null) {
        long[] carts = userFromDB.getCart();
        for (long c : carts)
            if (c == id) {
                model.addAttribute("cart", cart);
                break;
            }
    }

    if (userFromDB.getBuy() != null) {
        long[] buys = userFromDB.getBuy();
        for (long b : buys)
            if (b == id) {
                model.addAttribute("buy", buy);
                break;
            }
    }

    model.addAttribute("books", books);
    model.addAttribute("comments", comments);
    model.addAttribute("role", checkUserRole());
    return "book";
}
```

```

@PostMapping("/book/{id}/comment_add")
public String comment_add(
    @PathVariable(value = "id") Long id,
    @RequestParam String date, @RequestParam String[] comment
){
    StringBuilder com = new StringBuilder();
    for (String s : comment) com.append(s);

    String username = "";

    Authentication auth = SecurityContextHolder.getContext().getAuthentication();
    if (!(auth instanceof AnonymousAuthenticationToken) && auth != null) {
        UserDetails userDetails = (UserDetails) auth.getPrincipal();
        if (userDetails != null) {
            Users userFromDB = repoUsers.findByUsername(userDetails.getUsername());
            username = userFromDB.getUsername();
        }
    }

    Comments c = new Comments(id, username, date, com.toString());

    repoComments.save(c);
    return "redirect:/book/{id}";
}

```

CartCont

```

@GetMapping("/cart")
public String cart(Model model) {
    Users userFromDB = new Users();
    Authentication auth = SecurityContextHolder.getContext().getAuthentication();

    if (!(auth instanceof AnonymousAuthenticationToken) && auth != null) {
        UserDetails userDetails = (UserDetails) auth.getPrincipal();
        if (userDetails != null) {
            userFromDB = repoUsers.findByUsername(userDetails.getUsername());
        }
    }

    if (userFromDB.getCart() != null) {
        long[] cart = userFromDB.getCart();
        float summary = 0;
        List<Books> books = new ArrayList<>(), temp = repoBooks.findAll();

        for (Books g : temp) for (long c : cart) if (g.getId() == c) books.add(g);
        for (Books g : books) summary += g.getPrice();

        model.addAttribute("summary", summary);
        model.addAttribute("books", books);
        int i = 0;
        for (Books g : books) {
            i++;
            if (i == 2) {
                model.addAttribute("more", i);
                break;
            }
        }
    }
}

```

```

        model.addAttribute("role", checkUserRole());
        return "cart";
    }

    @PostMapping("/book/{id}/add_cart")
    public String add_cart(@PathVariable(value = "id") Long id) {
        Users userFromDB = new Users();

        Authentication auth = SecurityContextHolder.getContext().getAuthentication();
        if (!(auth instanceof AnonymousAuthenticationToken) && auth != null) {
            UserDetails userDetails = (UserDetails) auth.getPrincipal();
            if (userDetails != null) userFromDB = repoUsers.findByUsername(userDetails.getUsername());
        }

        long[] cart;
        if (userFromDB.getCart() == null) cart = new long[]{id};
        else {
            cart = new long[userFromDB.getCart().length + 1];
            for (int i = 0; i < userFromDB.getCart().length; i++) cart[i] = userFromDB.getCart()[i];
            cart[userFromDB.getCart().length] = id;
        }

        userFromDB.setCart(cart);

        repoUsers.save(userFromDB);
        return "redirect:/book/{id}";
    }

    @PostMapping("/book/{id}/remove_cart")
    public String remove_cart(@PathVariable(value = "id") Long id) {
        Users userFromDB = new Users();

        Authentication auth = SecurityContextHolder.getContext().getAuthentication();
        if (!(auth instanceof AnonymousAuthenticationToken) && auth != null) {
            UserDetails userDetails = (UserDetails) auth.getPrincipal();
            if (userDetails != null) userFromDB = repoUsers.findByUsername(userDetails.getUsername());
        }

        if (userFromDB.getCart().length == 1) userFromDB.setCart(null);
        else {
            long[] cart = new long[userFromDB.getCart().length - 1];
            int i = 0;
            for (long c : userFromDB.getCart()) {
                if (id == c) continue;
                cart[i] = c;
                i++;
            }
            userFromDB.setCart(cart);
        }

        repoUsers.save(userFromDB);
        return "redirect:/cart";
    }

    @PostMapping("/book/remove_cart_all")
    public String remove_cart_all(Model model) {
        Users userFromDB = new Users();

```

```

Authentication auth = SecurityContextHolder.getContext().getAuthentication();
if (!(auth instanceof AnonymousAuthenticationToken) && auth != null) {
    UserDetails userDetail = (UserDetails) auth.getPrincipal();
    if (userDetail != null) userFromDB = repoUsers.findByUsername(userDetail.getUsername());
}

userFromDB.setCart(null);

repoUsers.save(userFromDB);
return "redirect:/cart";
}

@PostMapping("/book/{id}/buy")
public String buy(@PathVariable(value = "id") Long id) {
    Users userFromDB = new Users();

    Authentication auth = SecurityContextHolder.getContext().getAuthentication();
    if (!(auth instanceof AnonymousAuthenticationToken) && auth != null) {
        UserDetails userDetail = (UserDetails) auth.getPrincipal();
        if (userDetail != null) userFromDB = repoUsers.findByUsername(userDetail.getUsername());
    }

    if (userFromDB.getCart() != null) {
        if (userFromDB.getCart().length == 1) userFromDB.setCart(null);
        else {
            long[] cart = new long[userFromDB.getCart().length - 1];
            int i = 0;
            for (long c : userFromDB.getCart()) {
                if (id == c) continue;
                cart[i] = c;
                i++;
            }
            userFromDB.setCart(cart);
        }
    }

    long[] buy;
    if (userFromDB.getBuy() == null) buy = new long[]{id};
    else {
        buy = new long[userFromDB.getBuy().length + 1];
        for (int i = 0; i < userFromDB.getBuy().length; i++) buy[i] = userFromDB.getBuy()[i];
        buy[userFromDB.getBuy().length] = id;
    }

    Optional<Books> temp = repoBooks.findById(id);
    List<Books> books = new ArrayList<>();
    temp.ifPresent(books::add);

    for (Books g : books) {
        g.setCount(g.getCount() + 1);
        g.setIncome(g.getIncome() + g.getPrice());
        repoBooks.save(g);
        break;
    }

    userFromDB.setBuy(buy);
}

```

```

        repoUsers.save(userFromDB);
        return "redirect:/book/{id}";
    }

    @PostMapping("/book/buy_cart_all")
    public String buy_cart_all(Model model) {
        Users userFromDB = new Users();

        Authentication auth = SecurityContextHolder.getContext().getAuthentication();
        if (!(auth instanceof AnonymousAuthenticationToken) && auth != null) {
            UserDetails userDetails = (UserDetails) auth.getPrincipal();
            if (userDetails != null) userFromDB = repoUsers.findByUsername(userDetails.getUsername());
        }

        long[] buy;
        if (userFromDB.getBuy() == null) {
            buy = new long[userFromDB.getCart().length];
            for (int i = 0; i < userFromDB.getCart().length; i++) buy[i] = userFromDB.getCart()[i];
        } else {
            buy = new long[userFromDB.getBuy().length + userFromDB.getCart().length];
            for (int i = 0; i < buy.length; i++) {
                for (int j = 0; j < userFromDB.getBuy().length; j++) {
                    buy[i] = userFromDB.getBuy()[j];
                    i++;
                }
                for (int j = 0; j < userFromDB.getCart().length; j++) {
                    buy[i] = userFromDB.getCart()[j];
                    Optional<Books> temp = repoBooks.findById(userFromDB.getCart()[j]);
                    List<Books> books = new ArrayList<>();
                    temp.ifPresent(books::add);
                    for (Books g : books) {
                        g.setCount(g.getCount() + 1);
                        g.setIncome(g.getIncome() + g.getPrice());
                        repoBooks.save(g);
                        break;
                    }
                    i++;
                }
            }
        }

        userFromDB.setBuy(buy);
        userFromDB.setCart(null);

        repoUsers.save(userFromDB);

        return "redirect:/cart";
    }

    BookAddEditCont
    @GetMapping("/book/add")
    public String book_add(Model model) {
        model.addAttribute("role", checkUserRole());
        return "book_add";
    }

    @PostMapping("book/add")

```

```

public String add(
    @RequestParam String name, @RequestParam("poster") MultipartFile poster,
    @RequestParam("screenshots") MultipartFile[] screenshots, @RequestParam String pub,
    @RequestParam String author, @RequestParam String isbn,
    @RequestParam int year, @RequestParam float price, @RequestParam float weight, @RequestParam Gen-
re genre,
    @RequestParam String[] description
) throws IOException {
    Books newBooks = new Books(name, author, pub, isbn, year, price, weight, genre);

    StringBuilder des = new StringBuilder();
    for (String s : description) des.append(s);
    newBooks.setDescription(des.toString());
    String uuidFile = UUID.randomUUID().toString();

    if (poster != null && !poster.getOriginalFilename().isEmpty()) {
        File uploadDir = new File(uploadPath);
        if (!uploadDir.exists()) uploadDir.mkdir();
        String result_poster = uuidFile + "_" + poster.getOriginalFilename();
        poster.transferTo(new File(uploadPath + "/" + result_poster));
        newBooks.setPoster(result_poster);
    }

    if (screenshots != null && !screenshots[0].getOriginalFilename().isEmpty()) {
        uuidFile = UUID.randomUUID().toString();
        String result_screenshot;
        String[] result_screenshots = new String[screenshots.length];
        for (int i = 0; i < result_screenshots.length; i++) {
            result_screenshot = uuidFile + "_" + screenshots[i].getOriginalFilename();
            screenshots[i].transferTo(new File(uploadPath + "/" + result_screenshot));
            result_screenshots[i] = result_screenshot;
        }
        newBooks.setScreenshots(result_screenshots);
    }

    Authentication auth = SecurityContextHolder.getContext().getAuthentication();
    if (!(auth instanceof AnonymousAuthenticationToken) && auth != null) {
        UserDetails userDetail = (UserDetails) auth.getPrincipal();
        if (userDetail != null) {
            Users userFromDB = repoUsers.findByUsername(userDetail.getUsername());
            newBooks.setUserid(userFromDB.getId());
        }
    }

    repoBooks.save(newBooks);
    return "redirect:/catalog/all";
}

@GetMapping("/book/{id}/edit")
public String book_edit(@PathVariable(value = "id") Long id, Model model) {
    if (!repoBooks.existsById(id)) return "redirect:/catalog";
    Optional<Books> temp = repoBooks.findById(id);
    ArrayList<Books> books = new ArrayList<>();
    temp.ifPresent(books::add);
    model.addAttribute("books", books);
    model.addAttribute("role", checkUserRole());
    return "book_edit";
}

```

```

@PostMapping("/book/{id}/edit")
public String edit(
    @PathVariable(value = "id") Long id,
    @RequestParam String name, @RequestParam("poster") MultipartFile poster,
    @RequestParam("screenshots") MultipartFile[] screenshots, @RequestParam String pub,
    @RequestParam String author, @RequestParam String isbn,
    @RequestParam int year, @RequestParam float price, @RequestParam float weight, @RequestParam Gen-
re genre,
    @RequestParam String[] description
) throws IOException {
    Books g = repoBooks.findById(id).orElseThrow();
    StringBuilder des = new StringBuilder();
    for (String s : description) des.append(s);

    g.setDescription(des.toString());
    g.setName(name);
    g.setPub(pub);
    g.setAuthor(author);
    g.setIsbn(isbn);
    g.setYear(year);
    g.setPrice(price);
    g.setWeight(weight);
    g.setGenre(genre);
    String uuidFile = UUID.randomUUID().toString();

    if (poster != null && !poster.getOriginalFilename().isEmpty()) {
        File uploadDir = new File(uploadPath);
        if (!uploadDir.exists()) uploadDir.mkdir();
        String result_poster = uuidFile + "_" + poster.getOriginalFilename();
        poster.transferTo(new File(uploadPath + "/" + result_poster));
        g.setPoster(result_poster);
    }

    if (screenshots != null && !screenshots[0].getOriginalFilename().isEmpty()) {
        uuidFile = UUID.randomUUID().toString();
        String result_screenshot;
        String[] result_screenshots = new String[screenshots.length];
        for (int i = 0; i < result_screenshots.length; i++) {
            result_screenshot = uuidFile + "_" + screenshots[i].getOriginalFilename();
            screenshots[i].transferTo(new File(uploadPath + "/" + result_screenshot));
            result_screenshots[i] = result_screenshot;
        }
        g.setScreenshots(result_screenshots);
    }

    repoBooks.save(g);
    return "redirect:/book/{id}/";
}

@GetMapping("/book/{id}/delete")
public String book_delete(@PathVariable(value = "id") Long id) {
    repoBooks.deleteById(id);
    List<Users> users = repoUsers.findAll();

    for (Users user : users)
        if (user.getCart() != null)
            for (long carts : user.getCart())

```



```

        if (id == carts) {
            if (user.getCart().length == 1) user.setCart(null);
            else {
                long[] cart = new long[user.getCart().length - 1];
                int i = 0;
                for (long c : user.getCart()) {
                    if (id == c) continue;
                    cart[i] = c;
                    i++;
                }
                user.setCart(cart);
            }
        }

        for (Users user : users) repoUsers.save(user);
        return "redirect:/catalog/all";
    }
}

```

CatalogCont

```

@GetMapping("/catalog/all")
public String catalog_main(Model model) {
    List<Books> books = repoBooks.findAll();
    model.addAttribute("books", books);
    model.addAttribute("role", checkUserRole());
    return "catalog";
}

@PostMapping("/catalog/book_search")
public String catalog_search(@RequestParam Genre genre, Model model) {
    List<Books> books = repoBooks.findAllByGenre(genre);

    List<Books> res = new ArrayList<>();
    for (Books g : books) {
        System.out.println("1 " + g.getYear());
        res.add(g);
        System.out.println("2 " + g.getYear());
    }

    model.addAttribute("books", res);
    model.addAttribute("role", checkUserRole());
    return "catalog";
}

@GetMapping("/catalog/genre/{genre}")
public String catalog_genre_search(@PathVariable(value = "genre") Genre genre, Model model) {
    List<Books> res = repoBooks.findAllByGenre(genre);

    model.addAttribute("books", res);
    model.addAttribute("role", checkUserRole());
    return "catalog";
}

@GetMapping("/catalog/year/{year}")
public String catalog_year_search(@PathVariable(value = "year") int year, Model model) {
    List<Books> books = repoBooks.findAllByYear(year);
    model.addAttribute("books", books);
}

```

```

        model.addAttribute("role", checkUserRole());
        return "catalog";
    }

    @PostMapping("/catalog/search")
    public String catalogSearch(@RequestParam String search, Model model) {
        List<Books> temp = repoBooks.findAll(), books = new ArrayList<>();
        for (Books g : temp) if (g.getName().contains(search)) books.add(g);
        model.addAttribute("books", books);
        model.addAttribute("role", checkUserRole());
        return "catalog";
    }
}

LoginRegCont

@GetMapping("/login")
public String login(Model model) {
    model.addAttribute("role", checkUserRole());
    return "login";
}

/* reg */

@GetMapping("/reg")
public String reg(Model model) {
    model.addAttribute("role", checkUserRole());
    return "reg";
}

@PostMapping("/reg")
public String addUser(Users user, Model model) {
    model.addAttribute("role", checkUserRole());
    Users userFromDB = repoUsers.findByUsername(user.getUsername());
    if (userFromDB != null) {
        model.addAttribute("message", "Пользователь с таким именем уже существует существует!");
        return "reg";
    }
    user.setRole(Role.USER);
    repoUsers.save(user);

    return "redirect:/login";
}
}

Main
String checkUserRole() {
    Authentication auth = SecurityContextHolder.getContext().getAuthentication();
    if (!(auth instanceof AnonymousAuthenticationToken) && auth != null) {
        UserDetails userDetails = (UserDetails) auth.getPrincipal();
        if (userDetails != null) {
            Users userFromDB = repoUsers.findByUsername(userDetails.getUsername());
            return String.valueOf(userFromDB.getRole());
        }
        return "NOT";
    }
    return "NOT";
}
}
}

```

MainCont

```
@GetMapping("/about")
public String about(Model model) {
    model.addAttribute("role", checkUserRole());
    return "about";
}
```

```
@GetMapping
public String index1(Model model) {
    model.addAttribute("role", checkUserRole());
    return "index";
}
```

```
@GetMapping("/index")
public String index2(Model model) {
    model.addAttribute("role", checkUserRole());
    return "index";
}
}
```

SalesCont

```
@GetMapping("/sales")
public String sales(Model model) {
    Users userFromDB = new Users();

    Authentication auth = SecurityContextHolder.getContext().getAuthentication();
    if (!(auth instanceof AnonymousAuthenticationToken) && auth != null) {
        UserDetails userDetail = (UserDetails) auth.getPrincipal();
        if (userDetail != null) userFromDB = repoUsers.findByUsername(userDetail.getUsername());
    }

    List<Books> books = repoBooks.findAllByUserId(userFromDB.getId());
    float income = 0;
    for (Books g : books) income += g.getIncome();

    model.addAttribute("income", income);
    model.addAttribute("books", books);
    model.addAttribute("role", checkUserRole());
    return "sales";
}
}
```

UserListCont

```
@GetMapping("/userList")
public String userList(Model model) {
    Users userFromDB = new Users();

    Authentication auth = SecurityContextHolder.getContext().getAuthentication();
    if (!(auth instanceof AnonymousAuthenticationToken) && auth != null) {
        UserDetails userDetail = (UserDetails) auth.getPrincipal();
        if (userDetail != null) userFromDB = repoUsers.findByUsername(userDetail.getUsername());
    }

    List<Books> books = repoBooks.findAllByUserId(userFromDB.getId());
    float income = 0;
    for (Books g : books) income += g.getIncome();

    List<Users> users = repoUsers.findAll();
}
```

```

        model.addAttribute("income", income);
        model.addAttribute("books", books);
        model.addAttribute("users", users);
        model.addAttribute("role", checkUserRole());
        return "userList";
    }

    @PostMapping("/userList/{id}/edit")
    public String userUpdate(
        @PathVariable(value = "id") Long id, @RequestParam String username,
        @RequestParam String password, @RequestParam Role role
    ) {
        Users temp = repoUsers.findById(id).orElseThrow();
        temp.setUsername(username);
        temp.setPassword(password);
        temp.setRole(role);
        repoUsers.save(temp);
        return "redirect:/userList";
    }

    @PostMapping("/userList/{id}/delete")
    public String userDelete(@PathVariable(value = "id") Long id) {
        repoUsers.deleteById(id);
        return "redirect:/userList";
    }
}

SecurityConfig
@Autowired
private UserService userService;

@Override
protected void configure(HttpSecurity http) throws Exception {
    http
        .csrf().disable()
        .authorizeRequests()
        .antMatchers("/", "/reg", "/catalog/all", "/static/**", "/img/**").permitAll()
        .anyRequest()
        .authenticated()
        .and()
        .formLogin()
        .loginPage("/login").permitAll()
        .defaultSuccessUrl("/index")
        .and()
        .logout()
        .logoutRequestMatcher(new AntPathRequestMatcher("/logout", "POST"))
        .invalidateHttpSession(true)
        .clearAuthentication(true)
        .deleteCookies("JSESSIONID")
        .logoutSuccessUrl("/login");
}

@Override
protected void configure(AuthenticationManagerBuilder auth) throws Exception {
    auth
        .userDetailsService(userService)
        .passwordEncoder(NoOpPasswordEncoder.getInstance());
}
}

```

```

MvcConfig
@Value("${upload.path}")
private String uploadPath;

@Override
public void addResourceHandlers(ResourceHandlerRegistry registry) {
    registry.addResourceHandler("/img/**")
        .addResourceLocations("file:" + uploadPath + "/");
    registry.addResourceHandler("/static/**")
        .addResourceLocations("classpath:/static/");
}
}

RepoBooks
List<Books> findAllByGenre(Genre genre);
List<Books> findAllByYear(int year);
List<Books> findAllByUserid(long userid);
}

RepoComments
public interface RepoComments extends JpaRepository<Comments, Long> {
    List<Comments> findAllByBookid(long id);
}

RepoUsers
public interface RepoUsers extends JpaRepository<Users, Long> {
    Users findByUsername(String username);
}

UserService
@Autowired
private RepoUsers repoUsers;

@Override
public UserDetails loadUserByUsername(String s) throws UsernameNotFoundException {
    return repoUsers.findByUsername(s);
}
}

Role
USER,
ADMIN,
PUB;

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

Books
@Id
@GeneratedValue(strategy = GenerationType.AUTO)
private long id;

private long userid;

private String
    name,
    author,
    pub,
    description,
    isbn;

```

```

private String poster;
private String[] screenshots;

private int year;

private float price, weight;

private Genre genre;

private int count;
private float income;

public Books() {
}

public Books(
    String name, String author, String pub, String isbn,
    int year, float price, float weight, Genre genre
) {
    this.name = name;
    this.author = author;
    this.pub = pub;
    this.isbn = isbn;
    this.year = year;
    this.price = price;
    this.weight = weight;
    this.genre = genre;
}

public long getId() {
    return id;
}

public long getUserId() {
    return userid;
}

public void setUserId(long userId) {
    this.userid = userId;
}

public String getName() {
    return name;
}

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

public String getPub() {
    return pub;
}

public void setPub(String pub) {
    this.pub = pub;
}

public String getDescription() {

```

```

        return description;
    }

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

    public String getPoster() {
        return poster;
    }

    public void setPoster(String poster) {
        this.poster = poster;
    }

    public String[] getScreenshots() {
        return screenshots;
    }

    public void setScreenshots(String[] screenshots) {
        this.screenshots = screenshots;
    }

    public int getYear() {
        return year;
    }

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

    public float getPrice() {
        return price;
    }

    public void setPrice(float price) {
        this.price = price;
    }

    public Genre getGenre() {
        return genre;
    }

    public void setGenre(Genre genre) {
        this.genre = genre;
    }

    public int getCount() {
        return count;
    }

    public void setCount(int count) {
        this.count = count;
    }

    public float getIncome() {
        return income;
    }

```

```
public void setIncome(float income) {  
    this.income = income;  
}  
  
public String getAuthor() {  
    return author;  
}  
  
public void setAuthor(String author) {  
    this.author = author;  
}  
  
public String getIsbn() {  
    return isbn;  
}  
  
public void setIsbn(String isbn) {  
    this.isbn = isbn;  
}  
  
public float getWeight() {  
    return weight;  
}  
  
public void setWeight(float weight) {  
    this.weight = weight;  
}  
}
```


ПРИЛОЖЕНИЕ Б

(обязательное)

Антиплагиат

Оригинальность 95,28%

Заимствования 4,72%

СЕМАНТИЧЕСКИЕ ХАРАКТЕРИСТИКИ

Рубрикатор ВАК документа

01.01.09 Дискретная математика и математическая кибернетика: 29.65%
05.13.18 Математическое моделирование, численные методы и комплексы программ: 22.53%
05.13.13 Телекоммуникационные системы и компьютерные сети: 16.55%
05.13.06 Автоматизация и управление технологическими процессами и производствами (по отраслям): 16.30%
08.00.05 Экономика и управление народным хозяйством (по отраслям и сферам деятельности): 14.98%

Рубрикатор ГРНТИ документа

50 Автоматика. Вычислительная техника: 51.22%
27 Математика: 29.93%
28 Кибернетика: 18.85%

Рубрикатор УДК документа

00 Наука в целом: 60.64%
51 Математика: 29.39%
08 Языкознание. Филология. Художественная литература. Литературоведение: 9.97%

Оценка связности текста: 84.2105263157895

Наличие описания результатов исследования: да

Наличие описания метода исследования: да

Наличие введения: да

Наличие выводов: нет

Наличие библиографии: да

Наличие аннотации: нет

Доля общей лексики: 0

Доля научной лексики: 100

ПРИЛОЖЕНИЕ В
(обязательное)
Ведомость документов