

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет
вычислительной техники

Кафедра
МОиПЭВМ

Направление подготовки 09.03.02 «Информационные системы и технологии»

Направленность Информационные системы и технологии

БАКАЛАВРСКАЯ РАБОТА

на тему

Проектирование и разработка
автоматизированной информационной
системы книжного магазина

Студент _____ Кудайбергенова Жанылмырза Доолосбековна
(подпись, дата) (ФИО полностью)

Руководитель _____ Макарычев П.П.
(подпись, дата) (фамилия, инициалы)

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

Работа допущена к защите (протокол заседания кафедры от _____ № _____)

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

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

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

Пенза, 2024

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

Факультет

Вычислительной техники

Кафедра

МО и ПЭВМ

«Утверждаю»
Заведующий кафедрой
_____ А.Ю. Козлов
«25» декабря 2023 г

**ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ
БАКАЛАВРА**

1. Студент Кудайбергенова Жанылмырза Доолосбековна гр. 20ВИ1 факультета ВТ
<Фамилия, имя, отчество полностью>

2. Тема работы Проектирование и разработка автоматизированной информационной системы книжного магазина

Тема утверждена приказом ПГУ № _____ от " ____ " _____ 20 ____

2. Руководитель работы д.т.н., профессор Макарычев Петр Петрович

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

Назначение разработки: автоматизация работы книжного магазина.

Основные функции системы:

- регистрация пользователей в системе;
 - добавление, редактирование, удаление товаров со стороны продавца;
 - возможность продажи книг через интернет, управление электронными заказами и доставкой;
 - управление клиентской базой: система может хранить информацию о клиентах, истории покупок и отправлять уведомления о новых поступлениях или акциях.
- Технология разработки: ООП, ASP.NET Core 6.0.
- Языки программирования: JavaScript, C#.
- Инструментальные средства: VS Code, Argo UML, Ramus, Visual Paradigm.

4. Перечень подлежащих разработке вопросов

- анализ предметной области и постановка задачи;
- анализ функциональных требований;
- планирование разработки и оценка бюджета;
- проектирование информационной системы;
- реализация информационной системы;
- контроль качества;
- оформление пояснительной записки.

5. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Анализ предметной области и требований к системе	10%	08.04.2024 – 13.04.2024	
2	Проектирование программных средств	20%	15.04.2024 – 25.04.2024	
3	Реализация программных средств	30%	26.04.2024 – 12.05.2024	
4	Тестирование программных средств	20%	13.05.2024 – 22.05.2024	
5	Оформление пояснительной записки	20%	23.05.2024 – 03.06.2024	

Дата выдачи задания «21» декабря 2023г.

Руководитель бакалаврской работы _____
(подпись, дата)

Задание к исполнению принял студент _____
(подпись, дата)

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

Реферат

Пояснительная записка содержит 77 листов, 25 рисунков, 10 таблиц, 16 листингов, 15 использованных источников и 3 приложения.

АВТОМАТИЗИРОВАННАЯ ИНФОРМАЦИОННАЯ СИСТЕМА,
JAVASCRIPT, C#, MONGO, КЛИЕНТ, СЕРВЕР, АНАЛИЗ

Предмет разработки: проектирование и разработка автоматизированной информационной системы книжного магазина.

Объект разработки: оптимизация процессов работы книжного магазина, специализирующегося на продаже книг.

Цель работы: разработка автоматизированной информационной системы книжного магазина.

Разработка серверной части приложения осуществлялась на языке программирования C#. Для реализации серверной части использовалась среда разработки Visual Studio Code.

Разработка клиентской части приложения осуществлялась на языке программирования JavaScript с использованием библиотеки для создания пользовательских интерфейсов React. Для реализации клиентской части приложения использовалась среда разработки VS Code.

Результат работы: спроектирована и разработана информационная система книжного магазина с веб-интерфейсом.

					ПГУ 09.03.02 – 08ВИ201.10 ПЗ			
Изм.	Лист	№ докум.	Подпись	Дата	Проектирование и разработка автоматизированной информационной системы книжного магазина Пояснительная записка	Лит.	Лист	Листов
Разраб.		Кудайбергенова Ж.Д.						
Провер.		Макарычев П.П.					4	77
Реценз.						группа 20ВИ1		
Н. Контр.		Такташкин Д.В.						
Утв.		Козлов А.Ю.						

Содержание

Введение	6
1 Постановка задачи и анализ аналогов разрабатываемой системы	8
1.1 Постановка задачи	8
1.2 Анализ информационных систем аналогичного назначения	8
1.3 Выбор технологий проектирования и разработки	14
2 Планирование и расчет бюджета разрабатываемой системы	18
3 Проектирование информационной системы	23
3.1 Анализ бизнес-процессов	23
3.2 Проектирование логической модели данных	26
3.3 Анализ прецедентов	29
3.4 Диаграмма состояний	32
3.5 Диаграмма кооперации	33
4 Реализация информационной системы	35
4.1 Реализация серверной части приложения	35
4.2 Реализация клиентской части приложения	36
4.3 Реализация пользовательского интерфейса	40
5 Тестирование информационной системы	45
5.1 Выбор методологии тестирования	45
5.2 Тестирование чёрным ящиком	47
5.3 Метрика Холстеда	49
5.4 Автоматизированное тестирование	52
Заключение	53
Список использованных источников	54
Приложение А (справочное) Глоссарий	56
Приложение Б (рекомендуемое) Исходный код серверной части	58
Приложение В (рекомендуемое) Исходный код клиентской части	65

Введение

Современный мир невозможен без широкого использования интернет-технологий в различных сферах, что способствует развитию глобальной информационной среды. Страны активно стремятся к информатизации и популяризации интернет-технологий, что играет ключевую роль в формировании цифровой экономики. Значительное влияние на этот процесс оказывают интернет-магазины, которые стимулируют развитие экономики в целом и цифровой экономики в частности.

Интернет-торговля стремится создать собственные бизнес-процессы и модели, которые являются основой ее деятельности. Интернет-магазины играют важную роль в современной экономике, обеспечивая доступ к товарам и услугам в любое время суток. Развитие интернет-торговли тесно связано с тенденциями экономики, где информация и знания преобразуются в товары и услуги. В этом контексте создание интернет-магазина для продажи книг имеет больше значение, учитывая значительный спрос на литературу. Интернет-торговля книгами предоставляет возможность быстрого доступа к широкому ассортименту литературы, а также удобные способы поиска, заказа и оплаты. Кроме того, интернет-магазины книг могут быстро реагировать на изменения в литературном рынке и предлагать пользователям новинки и специализированные издания.

Актуальность работы заключается в том, что разработка автоматизированной информационной системы книжного магазина позволяет пользователям легко находить и покупать книги различных жанров и авторов. А также покупка книг в онлайн-магазине занимает минимум времени и не требует походов в физические магазины, что особенно удобно для занятых людей.

Объектом выпускной квалификационной работы является автоматизация деятельности книжного магазина.

Предметом является информационная система автоматизации деятельности книжного магазина.

Целью выпускной квалификационной работы является проектирование и разработка автоматизированной информационной системы книжного магазина, которая предоставит пользователям возможность быстрого и удобного доступа к книжным произведениям.

Для достижения поставленной цели были определены следующие задачи:

- проанализировать существующие автоматизированные информационные системы книжных магазинов аналогичного назначения;
- определить требования к автоматизированной информационной системе книжного магазина;
- спланировать и рассчитать бюджет разработки информационной системы книжного магазина;
- выбрать технологии проектирования и разработки;
- спроектировать информационную систему книжного магазина;
- реализовать автоматизированную информационную систему книжного магазина;
- протестировать разработанное приложение.

Таким образом, разработка приложения для книжного магазина будет способствовать улучшению доступности каталога книг, увеличению удобства продажи и покупок.

В результате была спроектирована и разработана автоматизированная информационная система для книжного магазина, включая создание веб-интерфейса, позволяющего автоматизировать основные процессы работы магазина.

1 Постановка задачи и анализ аналогов разрабатываемой системы

1.1 Постановка задачи

Книжный магазин является веб-платформой, представляющая услуги по продаже книг различных жанров и авторов. Покупатели могут посетить веб-приложение магазина, выбрать и заказать книги, после чего им будет предоставлена возможность удобной доставки и оплаты. Основные функции автоматизированной информационной системы книжного магазина следующие.

Учет клиентов и заказов. Необходимо разработать функционал для учета клиентов и заказов на покупку книг. Система должна позволять создать новые заказы, просматривать и редактировать существующие заказы, а также фиксировать статус оплаты и доставки.

Учет книг. Система должна вести учет книг, представленных в магазине. Для каждой книги должна возможность указать название, автора, жанра и цену.

Веб-интерфейс. Для удобства пользователей система должна иметь веб-интерфейс, который обеспечит доступ к магазину через интернет. Клиенты смогут просматривать ассортименты, делать заказы и отслеживать статус доставки. Для заказа важна информация о клиенте, адрес доставки и оплата. Оплата может производиться несколькими способами: при получении, наличными, или банковской картой.

Сотрудники магазина смогут управлять ассортиментом, заказами и отчетами через веб-интерфейс.

Разработка информационной системы управления онлайн книжным магазином с веб-интерфейсом позволит оптимизировать работу магазина, улучшить обслуживание клиентов и повысить эффективность продаж литературных произведений.

1.2 Анализ информационных систем аналогичного назначения

Перед началом разработки автоматизированной информационной системы для книжного магазина необходимо провести анализ существующих систем,

чтобы учесть их преимущества и недостатки. На рынке существует множество автоматизированных систем для книжных магазинов, отличающихся по функциональности, технической реализации, стоимости и другим параметрам. Как показал анализ, необходимой для разработки интернет-магазина помимо программной части является:

- архитектура создаваемого интернет-магазина;
- принцип ее работы;
- информация о каталогах и услуг;
- расположение, отображение;
- размещение новостей, связанных с товарами компании;
- определенной тематики;
- ссылки на контактную информацию, информация о компании;
- методах ее работы.

Рассмотрим несколько существующих аналогов разрабатываемой информационной системы в России: Буквоед, Лабиринт, ReadRate.

Буквоед – сеть книжных магазинов с онлайн-присутствием, предлагающая как бумажные, так и электронные книги. Можно проанализировать работу сайта на различных устройствах, качество обслуживания клиентов через онлайн-консультантов, а также наличие специальных предложений и акций для покупателей [1].

Сайт Буквоед может выделяться своим уникальным дизайном, который может быть более ярким или адаптированным под определенную аудиторию.

Буквоед также предоставляет следующие возможности:

- общения с онлайн консультантами для решения вопросов клиентов и помощи в выборе книг;
- специальные предложения и акции для покупателей, которые могут отличаться от тех, что предлагают другие магазины;
- обеспечивает поддержку клубной карты с бонусами и скидками;
- предоставление информации о наличии товаров в магазинах.

На рисунке 1 представлена торговая площадка Буквоед.

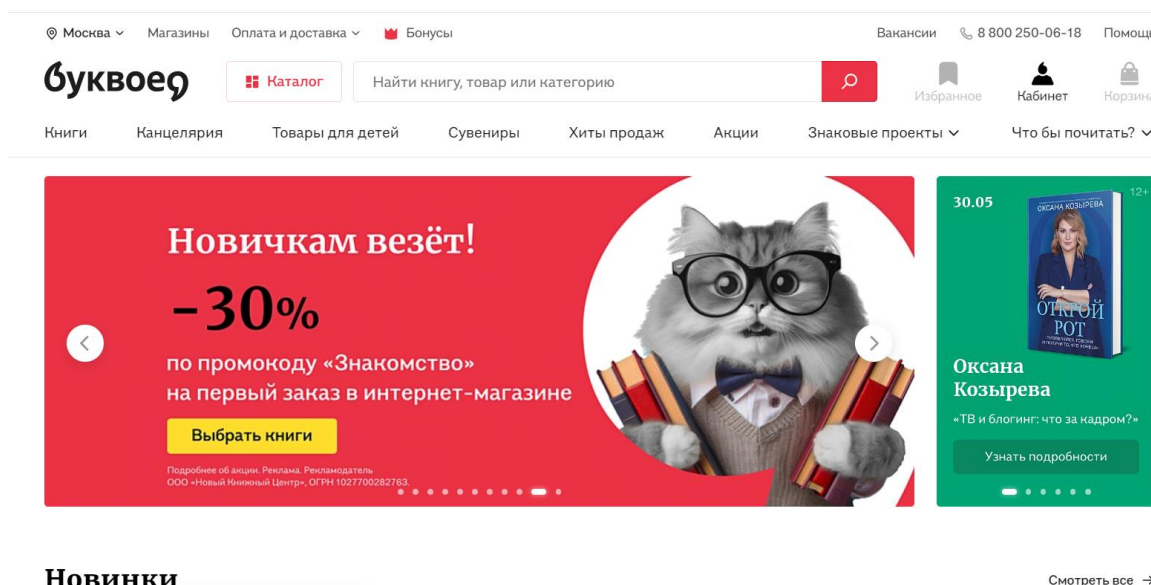


Рисунок 1 – Интерфейс книжного магазина Буквоед

Лабиринт – один из популярных онлайн магазинов книг в России. Здесь можно обратить внимание на дизайн сайта, функциональность поиска книг, возможность подписки на новости и акции, а наличие интерактивных рекомендации книг [2].

Лабиринт предоставляет следующие возможности:

- предлагает широкий выбор книг и удобную систему поиска с различными фильтрами, что позволяет пользователям быстро находить интересующих книг;
- имеет современный и интуитивно понятный дизайн, который обеспечить удобство использование как на компьютерах, так и на мобильных устройствах;
- специальные предложения и акции: регулярно проводить акции, распродажи предлагает скидки на книги, что делает покупки более привлекательными для клиентов;
- мобильное приложение для удобства покупок;
- акции, скидки и специальные предложения удобства покупателей;
- возможность предзаказа книжных новинок;
- широкий ассортимент книг и подарков.

Кроме того, на сайте имеется информация о расположении пунктов выдачи товаров, контактные данные, способы оплаты и вся необходимая дополнительная информация, нужная покупателю.

Интерфейс онлайн-магазина Лабиринт представлен на рисунке 2.

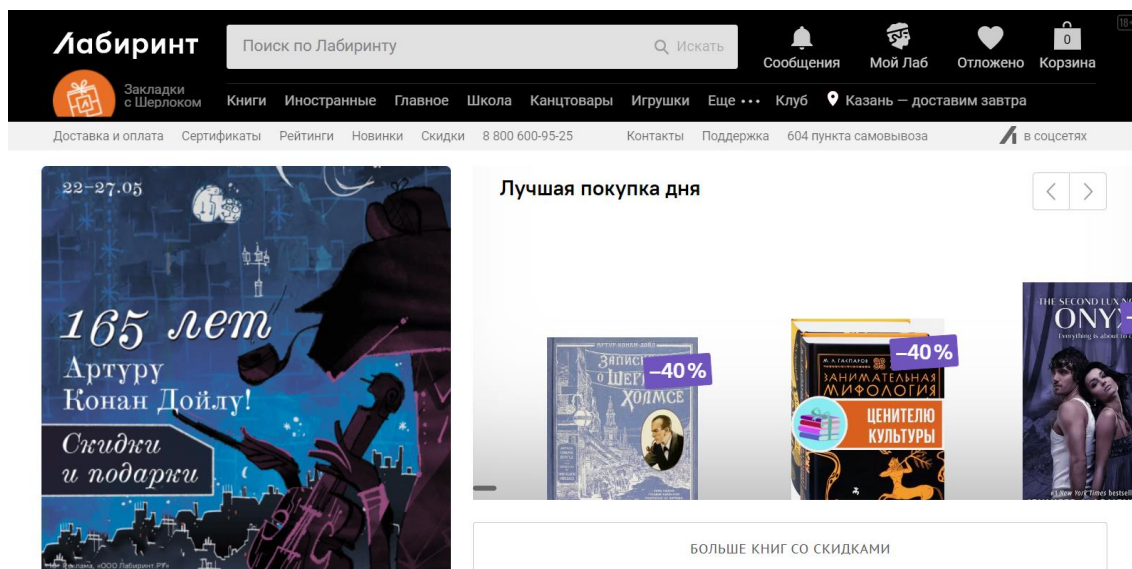


Рисунок 2 – Интерфейс книжного магазина Лабиринт

ReadRate – этот интернет-магазин специализируется на книгах и аудиокнигах. Имеется удобство поиска по категориям, наличие отзывов и рейтингов, возможность сравнения товаров, а также программа лояльности для покупателей [3].

Основные особенности ReadRate:

- ReadRate предлагает удобную систему поиска книг и аудиокниг по различным категориям, жанрам и авторам, что позволяет быстро найти интересующие товары;
- пользователи могут оставлять отзывы и ставить рейтинг товарам, что помогает другим покупателям принимать более обоснованные решения при выборе книг или аудиокниг;
- предоставляет функцию сравнения различных товаров, что позволяет пользователям быстро сопоставить характеристики и выбрать наиболее подходящий вариант;
- программа лояльности для покупателей;

- ReadRate может использовать специфические технические решения для обеспечения удобства пользователей, такие как интуитивный интерфейс, быструю загрузку страниц и оптимизация для мобильных устройств;
- ReadRate может предлагать конкурентоспособные цены на книги и аудиокниги, а также различные скидки и акции для покупателей;
- уникальные условия других магазинов, предлагая свои уникальные бонусы и вознаграждения;
- ReadRate может иметь особенности в доставке товаров, вариантах оплаты, возможности предзаказа книг и аудиокниг, а также специальные предложения для подписчиков новостной рассылки.

Интерфейс онлайн-магазина ReadRate представлен на рисунке 3.

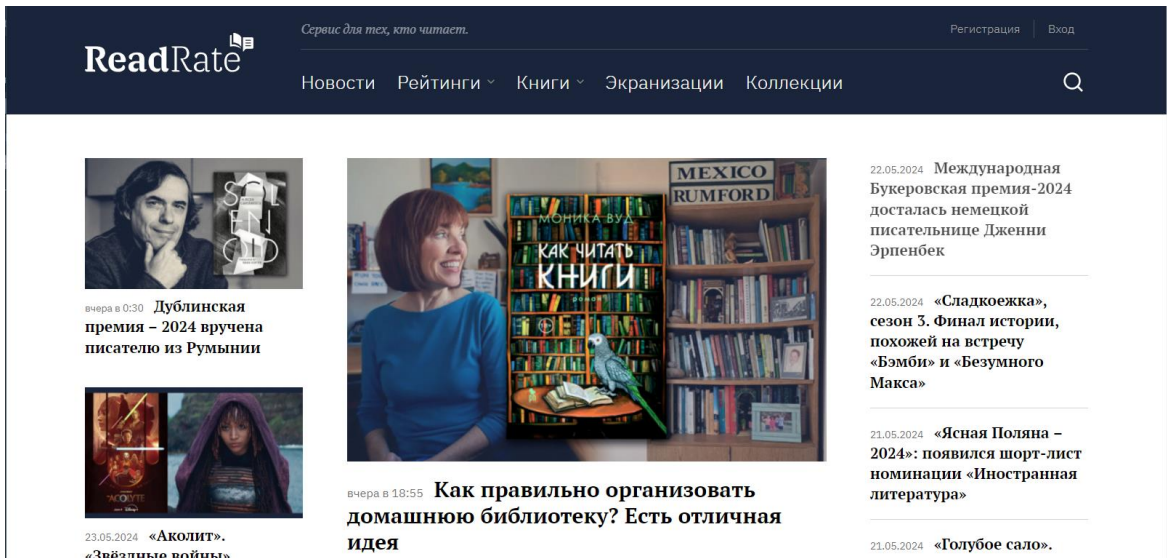


Рисунок 3 – Интерфейс книжного магазина ReadRate

Рассмотрев функциональные характеристики каждого аналога, был приведен сравнительный анализ, результаты которого приведены в таблице 1.

Таблица 1 – Сравнительный анализ систем-аналогов

Функциональные возможности	Буквоед	Лабиринт	ReadRate	Разрабатываемая система
Работа с документами	Да	Да	Да	Да
Email-уведомления клиентов	Нет	Да	Да	Да

Продолжение таблицы 1

Функциональные возможности	Буквоед	Лабиринт	ReadRate	Разрабатываемая система
Личный кабинет клиента	Да	Да	Нет	Да
Личный кабинет продавца	Нет	Нет	Нет	Да
Платформы	Веб-приложение	Веб-приложение	Веб-приложение	Веб-приложение
Отчетность	Да	Нет	Да	Нет

В ходе анализа было принято решение разработать систему с поддержкой веб-интерфейса, чтобы у пользователя был доступ в систему с компьютера, имеющий доступ в интернет. В системах аналогичного назначения нет возможности создания личного кабинета для продавцов, поэтому в разрабатываемой системе было принято решение добавить доступ к системе для продавцов организации, которые смогут следить за статусом заказов на товары.

В соответствии с характеристиками, приведенными в описании аналогичных систем, проведено сравнение по следующим критериям:

- функциональная сложность системы;
- качество пользовательского интерфейса;
- производительность;
- стоимость разработки.

Результаты представлены в таблице 2.

Таблица 2 – Оценка систем, основанная на критериях аналогов

Системы	Функциональная сложность	Качество пользовательского интерфейса	Производительность	Стоимость
Буквоед	Высокая	Среднее	Высокая	Средняя
Лабиринт	Средняя	Среднее	Высокая	Высокая
ReadRate	Высокая	Низкое	Средняя	Высокая

Из анализа, учитывая заданные критерии, можно заключить, что существующие информационные системы часто имеют высокую стоимость и содержат излишний функционал, что делает их неэффективными для небольших компаний. В результате возникает потребность в разработке системы, которая

будет включать только основные функции, необходимые для работы малых книжных магазинов. Такой подход окажется оптимальным, поскольку позволит сократить расходы на приобретение программного обеспечения и время.

1.3 Выбор технологий проектирования и разработки

Для проектирования автоматизированной информационной системы книжного магазина требуется построение UML-диаграмм, функциональной диаграммы в нотации IDEF0, а также модели данных в нотации IDEF1X. Для построения перечисленных диаграмм были выбраны такие CASE-средства, как Visual Paradigm и Erwin Data Modeler.

Erwin Data Modeler – инструмент моделирования данных, объединяющий в себе возможности проектирования баз данных: от логических и физических моделей до архитектурных планов и углубленных схем интеграции данных, позволяя эффективно решать задачи управления данными.

Visual Paradigm – это универсальная платформа для моделирования и проектирования, которая охватывает широкий спектр диаграмм и моделей, включая:

- диаграммы базы данных: ERD, логические и физические модели данных;
- UML-диаграммы: классы, варианты использования, последовательность, деятельность и т. д.;
- бизнес-процессы: BPMN, диаграммы потоков данных и организационные диаграммы;
- архитектурные диаграммы: архитектура предприятия, программное обеспечение и системная архитектура.

При выборе инструментов учитывались такие факторы, как наличие необходимого функционала, удобство использования и пользовательский интерфейс. Эти факторы имеют важное значение для успешной разработки информационной системы.

Для разработки информационной системы книжного магазина были

выбраны следующие технологии и средства разработки: среда разработки Visual Studio Code, языки программирования JavaScript и C#, библиотека для создания пользовательских интерфейсов React, платформа разработки ASP.NET, веб-сервер Apache, MongoDB.

C# — это объектно-ориентированный язык программирования, разработанный Microsoft. Он широко используется для создания различных приложений, включая настольные, веб-приложения, мобильные приложения и облачные сервисы.

JavaScript — универсальный язык программирования, который широко применяется в веб-разработке. Он отвечает за все действия на веб-странице и взаимодействие пользователя с ней. Фактически все современные браузеры понимают JavaScript. Этот язык определяет функциональность страницы и способы ее взаимодействия с пользователем и сервером.

Веб-страница — это не только JavaScript. Она создается с помощью трех основных технологий: HTML, CSS и JavaScript. HTML определяет то, что отображается на странице: ее содержимое хранится в нем, состоит из элементов, которые называются тегами. Они описывают конкретный компонент страницы.

CSS — это шаблон веб-страницы. Размер шрифта, семейство шрифтов и расположение текста на странице задаются с его помощью. JavaScript, HTML и CSS комбинируются вместе, чтобы создавать удобный пользовательский интерфейс, который удовлетворяет потребности конечного пользователя.

Одним из основных преимуществ данного подхода является то, что клиентская часть приложения выполняется на стороне пользователя, что позволяет ускорить его работу и снизить нагрузку на сервер. Кроме того, использование JavaScript, HTML и CSS позволяет разработчиком создавать универсальные интерфейсы, которые показываются на всех типах устройств, без необходимости создавать отдельных версий для каждого типа устройства.

React — это JavaScript библиотека для создания пользовательских интерфейсов веб-приложений. Она позволяет разрабатывать сложные и интерактивные пользовательские интерфейсы из множества изолированных

компонентов.

React оптимизирует время разработки и повышает производительность приложений за счет использования DOM (Document Object Model). DOM позволяет обновлять только те части страницы, которые претерпели изменения, в результате чего уменьшается количество запросов к серверу. Это приводит к более быстрому выполнению приложения и снижает нагрузку на сервер.

Платформа ASP.NET (Active Server Pages .NET) – это веб-фреймворк с открытым исходным кодом, разработанный Microsoft для создания динамических веб-приложений. Он основан на платформе .NET и использует язык программирования C#.

ASP.NET предлагает ряд преимуществ:

- упрощенная разработка: ASP.NET предоставляет множество готовых компонентов, таких как элементы управления формой, средства проверки ввода и доступ к данным, что упрощает разработку веб-приложений;
- безопасность: ASP.NET включает встроенные функции безопасности, такие как проверка подлинности, авторизация и защита от межсайтового скриптинга (XSS), что помогает защитить веб-приложения от атак;
- производительность: ASP.NET использует компиляцию в машинный код, что обеспечивает высокую производительность веб-приложений;
- масштабируемость: ASP.NET можно масштабировать для поддержки больших объемов трафика и пользователей благодаря использованию серверных ферм и балансировщиков нагрузки;
- интеграция с другими технологиями Microsoft: ASP.NET тесно интегрирован с другими технологиями Microsoft, такими как .NET Framework, SQL Server и Visual Studio, что упрощает разработку и развертывание веб-приложений.

Хранение данных находится на стороне баз данных MongoDB Atlas. MongoDB Atlas – это передовая облачная база данных, которая представляет надежное и масштабируемое хранилище для данных. Ее гибкость и простота в использовании делают ее идеальным выбором для широкого спектра проектов.

Для установки и настройки приложения на сервере был выбран инструмент Apache. Apache – это веб-сервер, предназначенный для доставки веб-страниц и приложений. Он обрабатывает запросы и передаёт контент по протоколу HTTP. Надёжность, масштабируемость и модульная архитектура делают Apache одним из самых популярных и широко используемых веб-серверов в мире.

Выбранные технологии разработки и проектирования информационных систем позволяет с успехом выполнить необходимую работу, одновременно являясь при этом удобными и эффективными в использовании.

2 Планирование и расчет бюджета разрабатываемой системы

Функционально-стоимостный анализ веб-приложения книжного магазина позволит:

- определить функциональные требования приложения;
- оценить стоимость разработки в сравнении с ожидаемой ценностью приложения.

Для успешной реализации проекта необходимо:

- установить чёткие цели и разработать план их достижения;
- выделить необходимые ресурсы (время, бюджет, персонал) и оценить возможные риски;
- тщательно продумать план проекта, чтобы обеспечить эффективное управление и своевременную реализацию.

Для планирования и расчета бюджета разработки веб-сайта книжного магазина необходимо пройти следующие этапы:

- определение целей и требований веб-приложения, целевой аудитории и функциональных возможностей, которые необходимо реализовать;
- исследование рынка: провести исследование рынка, чтобы понять конкурентов, тенденции отрасли и потребности целевой аудитории;
- определение технических спецификаций: нужно определить технические характеристики веб-приложения, такие как используемые технологии, языки программирования и требования к серверу;
- оценка трудозатрат, необходимые для разработки и развертывания веб-приложения, а также связанные с этим затраты;
- составление подробного бюджета, который включает стоимость разработки.

Для планирования и управления проектами используются различные методы, одним из основных является диаграмма Ганта.

Диаграмма Ганта – визуальный инструмент для планирования и

управления проектами, который позволяет:

- отслеживать задачи и их временные рамки;
- определять общую продолжительность проекта;
- выявлять зависимости между задачами;
- оценивать риски и потенциальные проблемы.

В ходе планирования проекта веб-приложения книжного магазина может быть выделено несколько задач, таких как разработка дизайна, создание базы данных, настройка системы оплаты и доставки, разработка интерфейса пользователя [4].

Для планирования и расчета бюджета разрабатываемой системы использовалось программное обеспечения Microsoft Office Project. В процессе планирования было выделено 25 задач, каждая из которых описана в таблице 3.

Таблица 3 – Задачи проекта

№ п/п	Название задачи	Длительность	Начало	Окончание	Предшественники
1	Начало проекта	0 дней	12.04.2024	12.04.2024	
2	Анализ предметной области и требований к системе	4 дня	12.04.2024	16.04.2024	
3	Анализ бизнес-процессов предприятия	1 день	12.04.2024	12.04.2024	
4	Поиск и анализ информационных систем аналогичного назначения	1 день	13.04.2024	13.04.2024	3
5	Анализ требований и постановка задач	1 день	14.04.2024	14.04.2024	4
6	Выбор технологий проектирования и разработки	1 день	15.04.2024	15.04.2024	5
7	Завершение анализа предметной области и требований к системе	0 дней	16.04.2024	16.04.2024	2
8	Проектирование информационной системы	9 дней	17.04.2024	26.04.2024	7
9	Проектирование логической модели базы данных	4 дня	17.04.2024	20.04.2024	
10	Проектирование структуры данных	2 дня	21.04.2024	22.04.2024	9
11	Проектирование пользовательского интерфейса	3 дня	23.04.2024	26.04.2024	10
12	Завершение проектирования информационной системы	0 дней	26.04.2024	26.04.2024	8
13	Реализация информационной системы	11 дней	27.04.2024	11.05.2024	12
14	Реализация конфигурации	4 дня	27.04.2024	01.05.2024	

Продолжение таблицы 3

№ п/п	Название задачи	Длительность	Начало	Окончание	Предшественники
15	Реализация веб-интерфейса системы	7 дней	03.05.2024	11.05.2024	14
16	Завершение реализации информационной системы	0 дней	11.05.2024	11.05.2024	13
17	Тестирование программных средств	9 дней	12.05.2024	21.05.2024	16
18	Тестирование метрикой Холстеда	5 дней	12.05.2024	17.05.2024	
19	Функциональное тестирование	4 дня	18.05.2024	21.05.2024	
20	Завершение тестирования программных средств	0 дней	21.05.2024	21.05.2024	17
21	Оформление документации	10 дней	22.05.2024	01.06.2024	20
22	Написание документации	7 дней	22.05.2024	28.05.2024	
23	Оформление и печать документации	3 дня	29.05.2024	31.06.2024	22
24	Завершение оформления документации	0 дней	01.06.2024	01.06.2024	21
25	Конец проекта	0 дней	01.06.2024	01.06.2024	24

Также были определены необходимые ресурсы. Список ресурсов представлен в таблице 4.

Таблица 4 – Ресурсы и затраты проекта

№ п/п	Название ресурса	Тип ресурса	Стандартная ставка	Начисление
1	Аналитик	Трудовой	200,00 Р/ч	Пропорциональное
2	Проектировщик	Трудовой	200,00 Р/ч	Пропорциональное
3	Программист	Трудовой	250,00 Р/ч	Пропорциональное
4	Тестировщик	Трудовой	200,00 Р/ч	Пропорциональное
5	Компьютер	Затраты		В начале
6	Принтер	Затраты		В начале
7	Бумага	Материальный	300 Р	В начале
8	Интернет	Затраты		В начале
9	Электроэнергия	Затраты		По окончании

Выделенные задачи и ресурсы были занесены в проект в Microsoft Office Project. Результаты представлены на рисунке 4.

Начало проекта	1 день
Анализ предметной области и требований к системе	5 дней
Анализ бизнес-процессов предприятия	1 день
Поиск и анализ информационных систем аналогичного назначения	1 день
Анализ требований и постановка задач	1 день
Выбор технологий проектирования и разработки	1 день
Завершение анализа предметной области и требований к системе	0 дней
Проектирование информационной системы	9 дней
Проектирование логической модели базы данных	4 дней
Проектирование структуры данных	2 дней
Проектирование пользовательского интерфейса	3 дней
Завершение проектирования информационной системы	0 дней
Реализация информационной системы	11 дней
Реализация конфигурации	4 дней
Реализация веб-интерфейса системы	8 дней
Завершение реализации информационной системы	0 дней
Тестирование программных средств	8 дней
Тестирование метрикой Холстеда	5 дней
Функциональное тестирование	4 дней
Завершение тестирования программных средств	0 дней
Оформление документации	9 дней
Написание документации	7 дней
Оформление и печать документации	3 дней
Завершение оформления документации	0 дней
Конец проекта	0 дней

Рисунок 4 – Задачи проекта с распределенными по ним ресурсами в Microsoft Office Project

В соответствии с заданными задачами и ресурсами была построена диаграмма Ганта, представленная на рисунке 5.

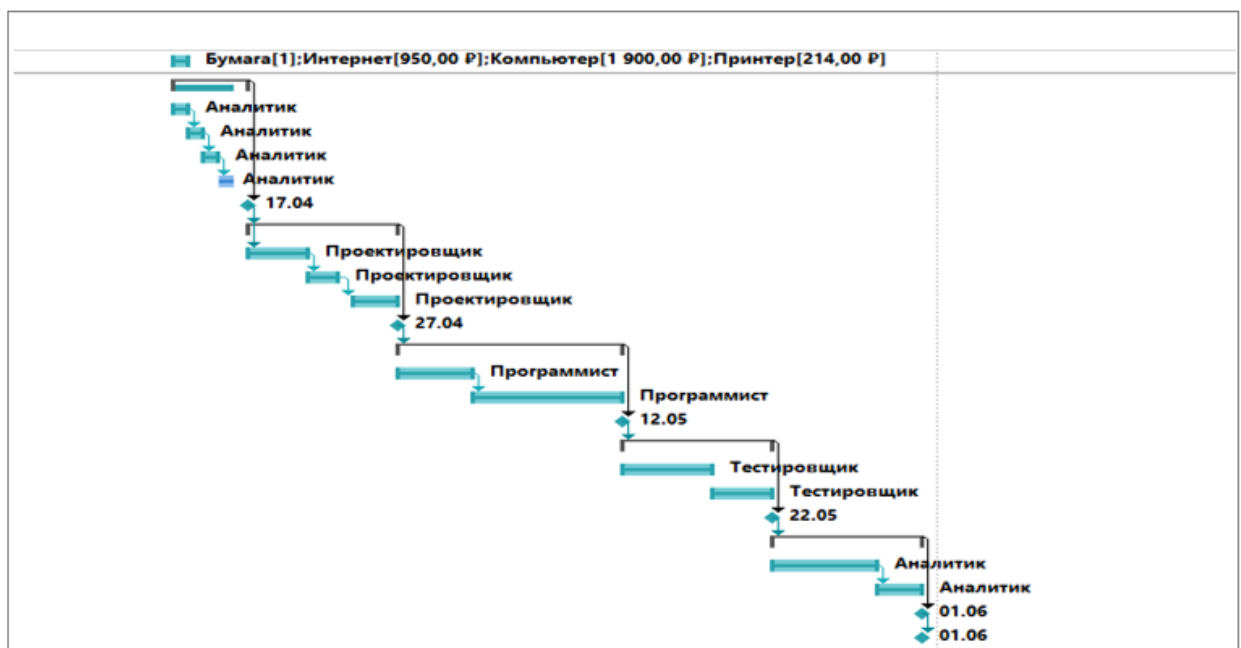


Рисунок 5 – Диаграмма Ганта

В соответствии с построенной диаграммой был сформирован отчет по затратам на разработку, который представлен на рисунке 6.

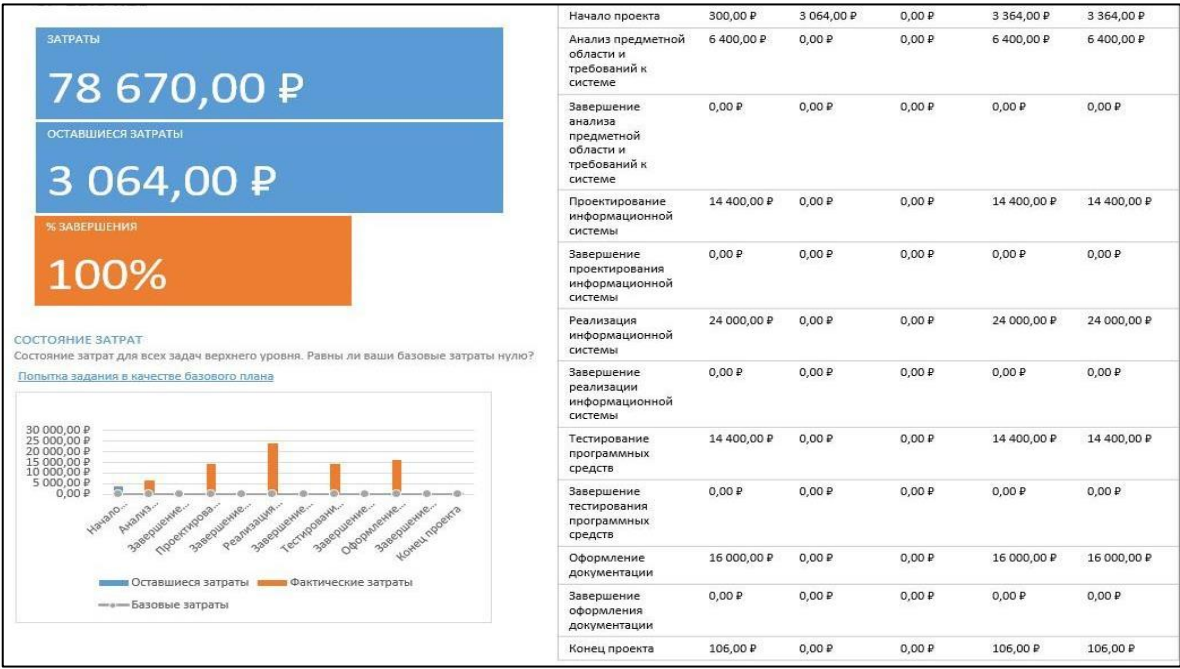


Рисунок 6 – Отчет по затратам проекта

После проведения анализа затрат и функциональности было установлено, что для завершения проекта потребуется 51 день, и это включает в себя выходные и праздничные дни. Общая стоимость выполнения проекта, учитывая все расходы на ресурсы, составит 78 670 рублей. Оценка стоимости играет важную роль в процессе планирования проекта, позволяя определить бюджет, эффективно распределять ресурсы, контролировать расходы и принимать обоснованные решения на всех этапах работы. Правильная оценка стоимости также способствует снижению рисков и обеспечивает успешное выполнение проекта.

3 Проектирование информационной системы

3.1 Анализ бизнес-процессов

Перед разработкой информационной системы для автоматизированной информационной системы книжного магазина необходимо провести анализ бизнес-процессов. Для этого можно построить функциональную модель в нотации IDEF0, которая поможет формализовать и описать бизнес-процессы.

Для построения диаграммы бизнес-процессов был выбран Ramus.

Контекстная диаграмма бизнес-процессов в нотации IDEF0 для предметной области книжного магазина представлен на рисунке 7.

Контекстная диаграмма состоит из единственного функционального блока, который служит общим описанием системы и ее взаимодействие с внешней средой [5].

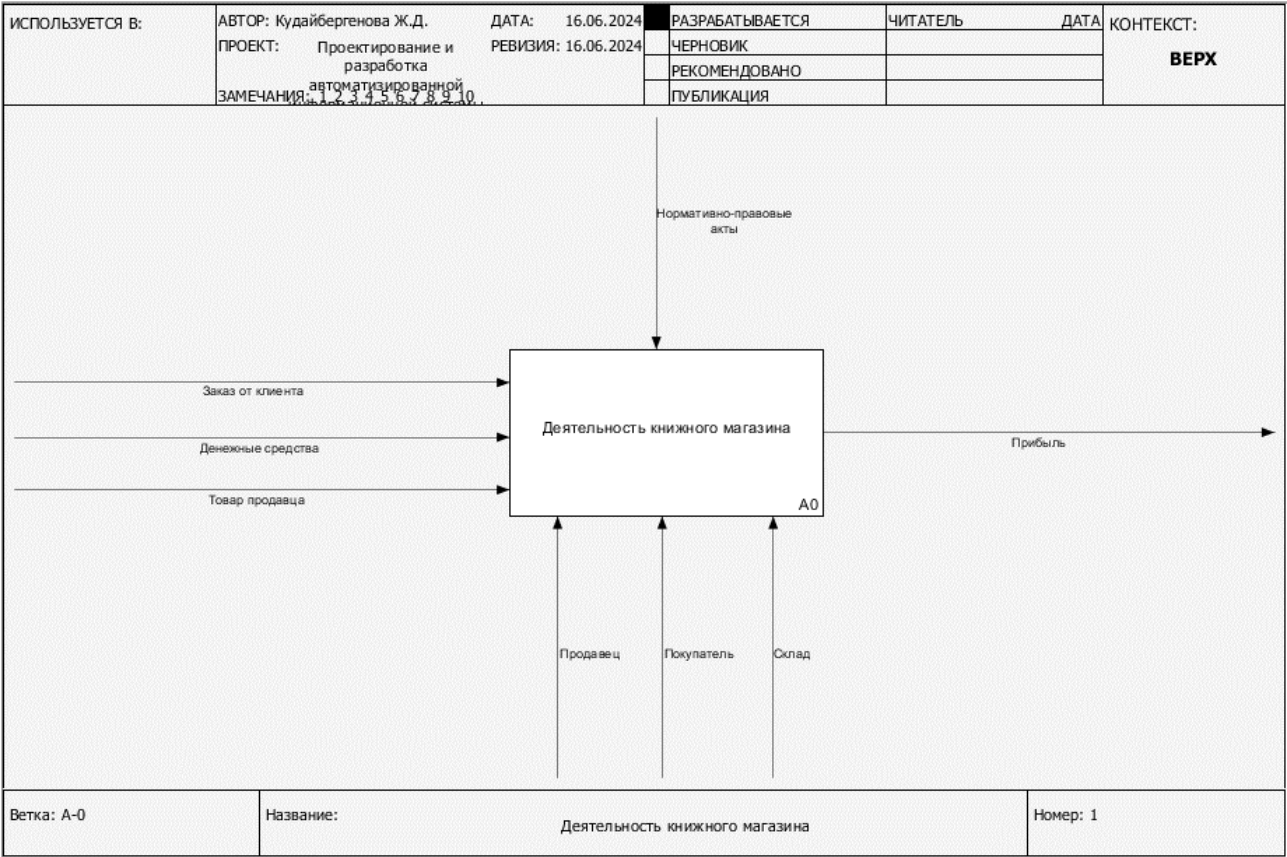


Рисунок 7 – Контекстная диаграмма бизнес-процессов книжного магазина в нотации IDEF0

Основная функция «Деятельность книжного магазина», представленная на контекстной диаграмме, была декомпозирована на составные части путем создания низкоуровневой дочерней диаграммы. В результате декомпозиции основного процесса «Деятельность книжного магазина» было выделено 4 дочерних процесса: «Закупка товаров и их размещение в приложении», «Оформление заказа», «Оплата покупки» и «Отправка товаров покупателю». Диаграмма декомпозиции представлена на рисунке 8.

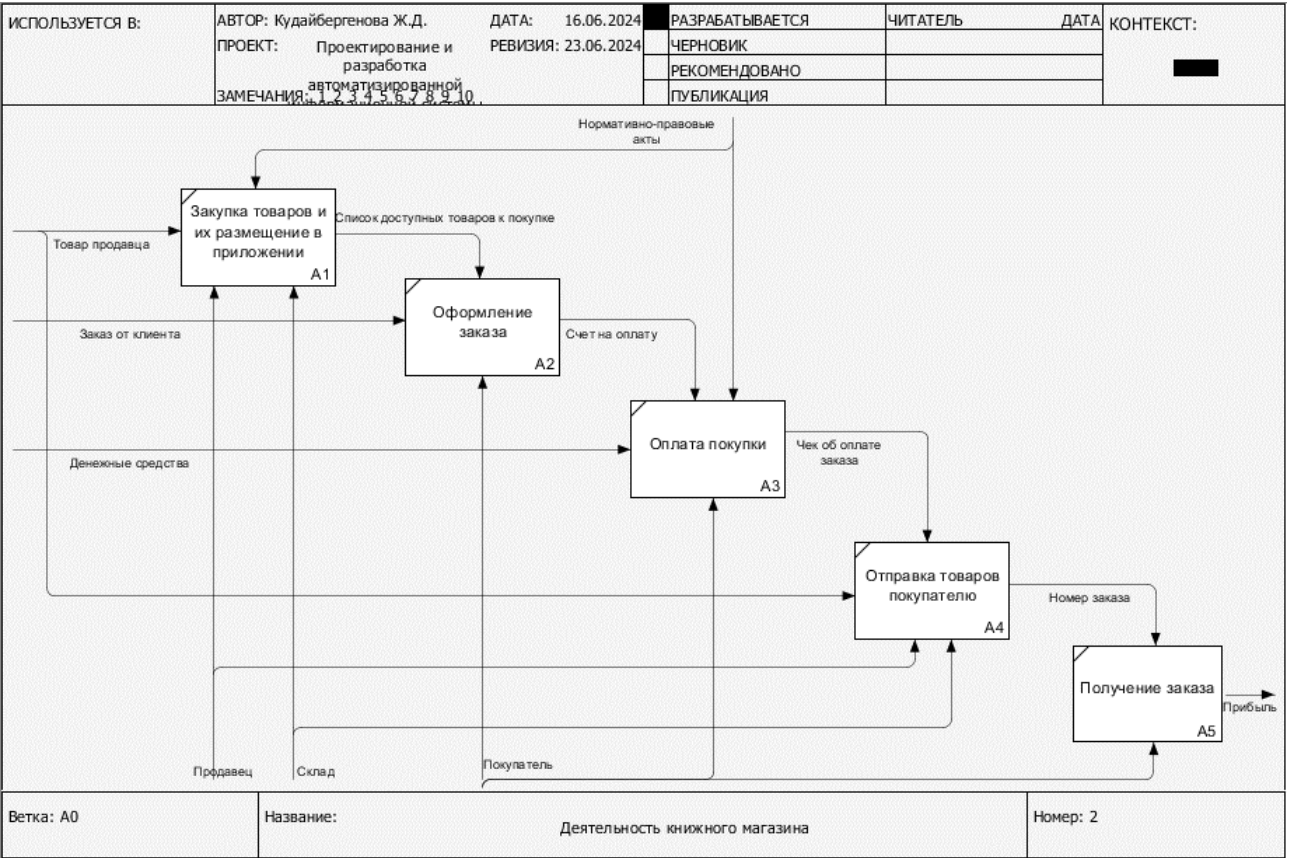


Рисунок 8 – Детализирующая диаграмма процесса «Деятельность книжного магазина» в нотации IDEF0

Процесс «Закупка товаров и их размещение на сайте» предполагает несколько ключевых этапов, которые необходимо тщательно продумать и организовать для эффективной работы интернет-магазина. В первую очередь, это включает в себя анализ рынка и потребностей клиентов для определения ассортимента товаров, которые будут приобретаться. Далее необходимо выбрать надежных поставщиков и провести переговоры о ценах, условиях поставки и других важных деталях.

После заключения договоров с поставщиками следует осуществить процесс закупки товаров, контролируя качество и соответствие товара установленным стандартам. Затем товары необходимо принять на складе, правильно отразить в учете и подготовить к размещению на сайте. Размещение товаров на сайте должно быть информативным, привлекательным для покупателей и удобным для навигации.

Кроме того, важно проводить мониторинг спроса на товары, анализировать продажи и реакцию клиентов, чтобы оперативно реагировать на изменения рыночной ситуации. Весь процесс «Закупка товаров и их размещение на сайте» требует хорошей координации между различными отделами компании, а также постоянного совершенствования стратегий для достижения успеха в электронной торговле.

Процесс «Оформление заказа» предполагает несколько этапов, начиная с поиска желаемой книги на сайте. После выбора книги клиент должен иметь возможность добавить её в корзину и перейти к оформлению заказа. Важно, чтобы процесс оформления заказа был простым и интуитивно понятным для клиента.

После добавления товара в корзину клиент должен иметь возможность проверить содержимое корзины, внести изменения, если это необходимо, и выбрать удобный способ доставки. Далее необходимо заполнить информацию о доставке и оплате заказа. Клиент должен иметь возможность выбрать удобный способ оплаты, такой как банковская карта или наличные при получении.

После завершения этого этапа клиент должен получить подтверждение заказа на указанный им адрес электронной почты или через SMS. После этого интернет-магазин должен обеспечить оперативную обработку заказа, упаковку книг и передачу их в службу доставки.

Важным этапом также является обеспечение возможности отслеживания статуса доставки заказа клиентом. После получения книги клиент должен иметь возможность оставить отзыв о процессе покупки и качестве обслуживания.

Эффективное оформление заказа в интернет-магазине включает в себя

удобство для клиента, оперативную обработку заказов и прозрачную коммуникацию на всех этапах сделки.

3.2 Проектирование логической модели данных

При проектировании базы данных (БД) необходимо разработать модель предметной области, которая независима от конкретной системы управления базами данных. Для этой цели часто используются модели «сущность-связь», которые представляют собой абстрактные описания данных и их взаимосвязей. Существует несколько методологий построения моделей «сущность-связь», однако наиболее широкое распространение получила методология IDEF1X [6].

Методология IDEF1X подходит для создания дизайна баз данных различного масштаба, начиная от небольших приложений и заканчивая крупными корпоративными системами.

Логическая модель «сущность-связь», которая представлена на рисунке 9, была построена с помощью средств Erwin Data Modeler. Модель содержит 13 сущностей, связанных между собой логическими связями.

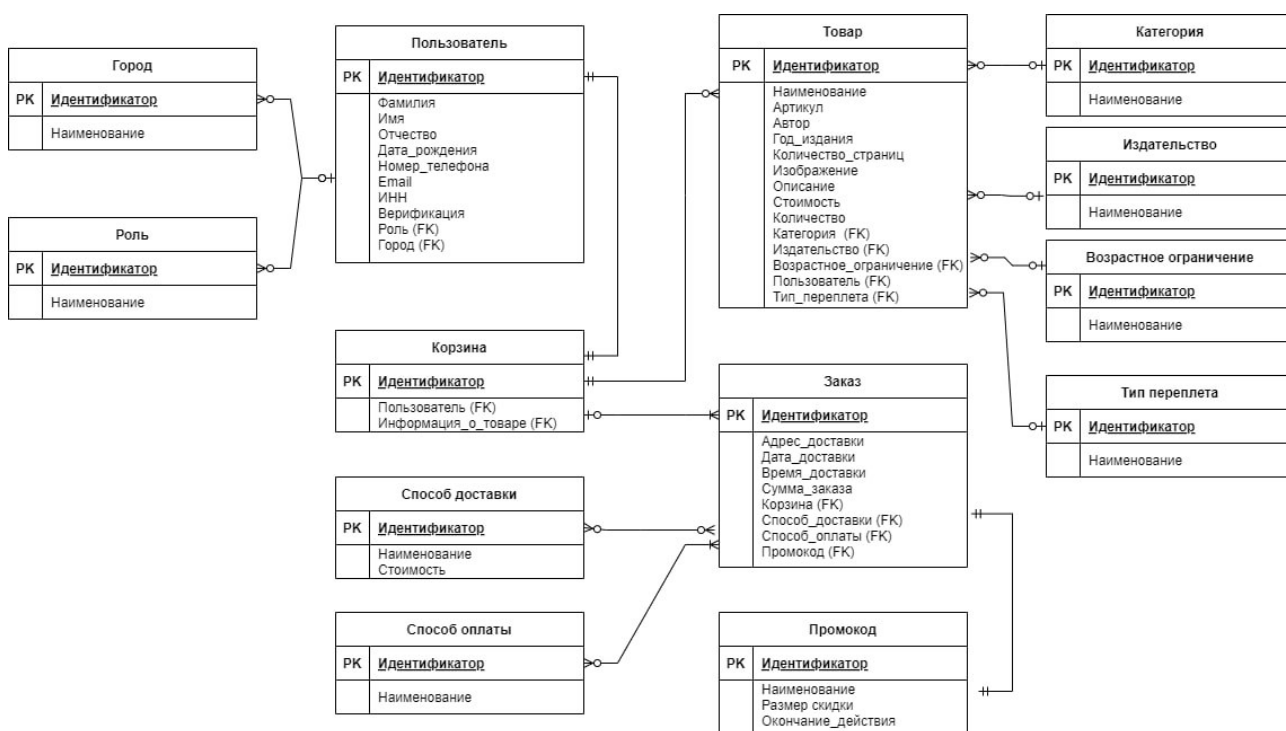


Рисунок 9 – Логическая модель данных

Описание сущностей и их атрибутов логической модели представлены в

таблице 5.

Таблица 5 – Описание сущностей и атрибутов логической модели

Сущность	Атрибут	Роль атрибута	Описание атрибута
Пользователь	Идентификатор	РК	Уникальный код пользователя
	ФИО		Фамилия, имя и отчество пользователя
	Дата рождения		Дата рождения пользователя
	Номер телефона		Номер телефона пользователя
	Email		Электронная почта пользователя
	ИНН		Идентификационный номер налогоплательщика
	Верификация		Проверка страницы на подлинность
	Роль	FK	Роль
	Город	FK	Город
Город	Идентификатор	РК	Уникальный код города
	Наименование		Название города
Товар	Идентификатор	РК	Уникальный код товара
	Наименование		Название товара
	Артикул		Артикул товара
	Автор		Автор книги
	Год издания		Год издания книги
	Категория	FK	Категория, к которой относится книга
	Издательство	FK	Издательство, которое выпустило книгу
	Тип переплета	FK	Тип переплета книги
	Возрастное ограничение	FK	Возрастное ограничение книги
	Количество страниц		Количество страниц в книге
	Изображение		Наименование изображения
	Описание		Описание книги
	Стоимость		Стоимость книги
	Пользователь	FK	Покупатель
	Количество		Количество книг
Категория	Идентификатор	РК	Уникальный код категории
	Наименование		Наименование категории
Издательство	Идентификатор	РК	Уникальный код издательства
	Наименование		Наименование издательства
Тип переплета	Идентификатор	РК	Уникальный код типа переплета книги
	Наименование		Наименование типа переплета книги
Возрастное ограничение	Идентификатор	РК	Уникальный код возрастного ограничения
	Наименование		Наименование возрастного ограничения

Продолжение таблицы 5

Сущность	Атрибут	Роль атрибута	Описание атрибута
Корзина	Идентификатор	PK	Уникальный код корзины
	Пользователь	FK	Идентификатор покупателя
	Информация о товаре		Дополнительная информация о товаре
	Товар	FK	Идентификатор товара
	Количество		Количество книг
	Сумма		Сумма
Заказ	Идентификатор	PK	Идентификатор заказа
	Корзина	FK	Идентификатор корзины
	Адрес доставки		Адрес доставки товаров
	Дата доставки		Дата доставки товаров
	Время доставки		Время доставки товаров
	Способ доставки	FK	Способ доставки товаров
	Способ оплаты	FK	Способ оплаты товаров
	Промокод	FK	Промокод
	Сумма заказа		Сумма заказа
Способ оплаты	Идентификатор	PK	Идентификатор способа оплаты
	Наименование		Наименование способа оплаты
Способ доставки	Идентификатор	PK	Уникальный код способа доставки
	Наименование		Наименование способа доставки
	Стоимость		Стоимость доставки
Промокод	Идентификатор	PK	Идентификатор промокода
	Наименование		Наименование промокода
	Размер скидки		Размер скидки
	Окончание действия		Дата окончания действия промокода

После завершения логического моделирования данных, требуется выполнить процесс нормализации. Это означает организацию данных таким образом, чтобы обеспечить их целостность и сделать базу данных более гибкой, лишенной избыточности и противоречий. Цель нормализации заключается в устранении повторяющейся информации в таблицах, обеспечении возможности изменения структуры таблиц и снижении воздействия структурных изменений базы данных на функционирование программного обеспечения.

Поскольку каждый не ключевой атрибут зависит только от первичного ключа и не зависит от других атрибутов, сущности в логической модели базы данных содержат только простые атрибуты. Исходя из этого, можно утверждать, что данная модель базы данных находится в третьей нормальной форме и

Описание вариантов использования приведено в таблице 6.

Таблица 6 – Описание вариантов использования

Действующее лицо	Вариант использования	Описание
Покупатель	Войти в личный кабинет	Покупатель может использовать свои учетные данные чтобы войти в свой личный кабинет
	Зарегистрироваться	Пользователь имеет возможность зарегистрироваться в системе
	Посмотреть корзину	Покупатель может просмотреть содержимое своей корзины, чтобы увидеть список товаров
	Добавить товар в корзину	Покупатель может выбрать интересующий его товар и добавить его в корзину
	Просмотреть каталог	Покупатель может просматривать каталог товаров в системе
	Оформить заказ	Покупатель имеет возможность оформить заказ
	Найти товар по наименованию или артикул	Покупатель использует функцию поиска на сайте, чтобы найти конкретный товар, который интересует
	Удалить товар из корзины	Если покупатель случайно добавил неправильный товар в корзину или решили отказаться от покупки, может удалить его из корзины
	Изменить количество товара	Пользователь имеет возможность изменить количество определенного товара в корзине
	Выбрать дату и время доставки	Пользователю при оформлении заказа будет предоставлена возможность выбрать удобную дату и время доставки
	Выбрать способ оплаты	Пользователь может выбрать предпочтительный способ оплаты
	Ввести логин и пароль	Пользователь вводит свой логин и пароль для доступа к личному кабинету покупателя
	Указать адрес доставки	Пользователь указывает адрес доставки
Модератор	Выбрать способ доставки	Пользователь может выбрать способ доставки
	Просмотреть данные продавца	Модератор может просматривать информацию о продавце, это позволяет модератору быть в курсе актуальной информации о продавце
Модератор	Войти в личный кабинет	Модератор имеет возможности использовать свои учетные данные для входа в систему

Продолжение таблицы 6

Действующее лицо	Вариант использования	Описание
Модератор	Зарегистрироваться	Если модератор еще не зарегистрирован в системе, он может пройти процесс регистрации
	Просмотреть каталог	Модератор может просматривать каталог товаров и описание, предоставленные продавцами
	Ввести логин и пароль	Модератор вводит свой логин и пароль для доступа к личному кабинету продавца. Это позволяет ему получить доступ к необходимым функциям и данным
	Найти товар по наименованию или артикулу	Модератор осуществляет доступ поиска для нахождения товара по его наименованию или артикулу. Это помогает быстро найти нужный товар среди множества доступных книг
	Зарегистрировать модератора	Модератор имеет возможность зарегистрировать пользователя
Продавец	Просмотреть каталог	Продавец имеет возможность просмотреть каталог книг
	Войти в личный кабинет	Зарегистрированный продавец может войти в личный кабинет
	Удалить товар	Продавец может удалить свой товар, если книга больше не доступна для продажи или есть необходимость удалить её из каталога
	Добавить данные о товаре	Если продавец получает новый товар для продажи, он может добавить его в каталог
	Изменить данные о товаре	Продавец имеет возможность изменить данные о товаре, если у товара изменяются характеристики, цена, количество или другая информация
	Отправить товар по заказу	Продавец может отправить товар по заказу, после получения заказа от покупателя, продавец должен подготовить товар для отправки
	Найти товар по наименованию или артикулу	Продавец может найти товар его наименование или артикулу с помощью функции поиска
	Заполнить поля	Продавец при создании нового товара и редактировании существующего может заполнить различные поля, такие как наименование товара, описание, цена, количество и другие параметры

Окончание таблицы 6

Действующее лицо	Вариант использования	Описание
	Ввести трек-номер посылки	Продавец после отправки товара может ввести трек-номер посылки в соответствующее поле в личном кабинете. Такая информация позволяет клиентам отслеживать статус доставки своего заказа
	Зарегистрироваться	Продавец имеет возможность зарегистрироваться в системе
	Ввести логин и пароль	Прежде, чем войти в систему, продавец должен ввести логин и пароль от своего личного кабинета

В результате проведенного анализа были определены и описаны действующие лица системы и прецеденты, а также построена диаграмма вариантов использования.

3.4 Диаграмма состояний

Диаграмма состояний является визуальным инструментом для моделирования поведения системы, отображающим возможные состояния объектов и переходы между ними. Она включает в себе состояния, переходы и события, помогая определить требования системы и показать, как объекты должны вести себя в различных ситуациях. Диаграмма состояний полезна для понимания взаимосвязей между объектами и их поведением в системе. Для отображения изменения состояний объекта.

Диаграммы состояний широко применяются в разработке программного обеспечения для моделирования сложных системных процессов, автоматизации и управления. Они позволяют лучше понять поведение системы, выделить различные состояния и переходы между ними, а также предсказать возможные проблемы или ошибки в работе системы [8].

Для отображения смены состояний объекта «Добавить товар», была разработана диаграмма состояний, представленная на рисунке 11.



Рисунок 11 – Диаграмма состояний

Таким образом, была разработана диаграмма состояний, которая показала возможные состояния объекта «Добавить товар».

3.5 Диаграмма кооперации

Диаграмма кооперации в UML отображает взаимодействия между объектами в процессе выполнения определенной задачи или действия. Она фокусируется на способе взаимодействия объектов для достижения конкретной цели в системе [9].

Для разработки диаграммы кооперации был выбран вариант использования «Оформить заказ». При проектировании были выделены следующие объекты: компоненты приложения React, хранилище состояний MobX, web-сервер Apache и Mongo Atlas. Построенная диаграмма кооперации приведена на рисунке 12.

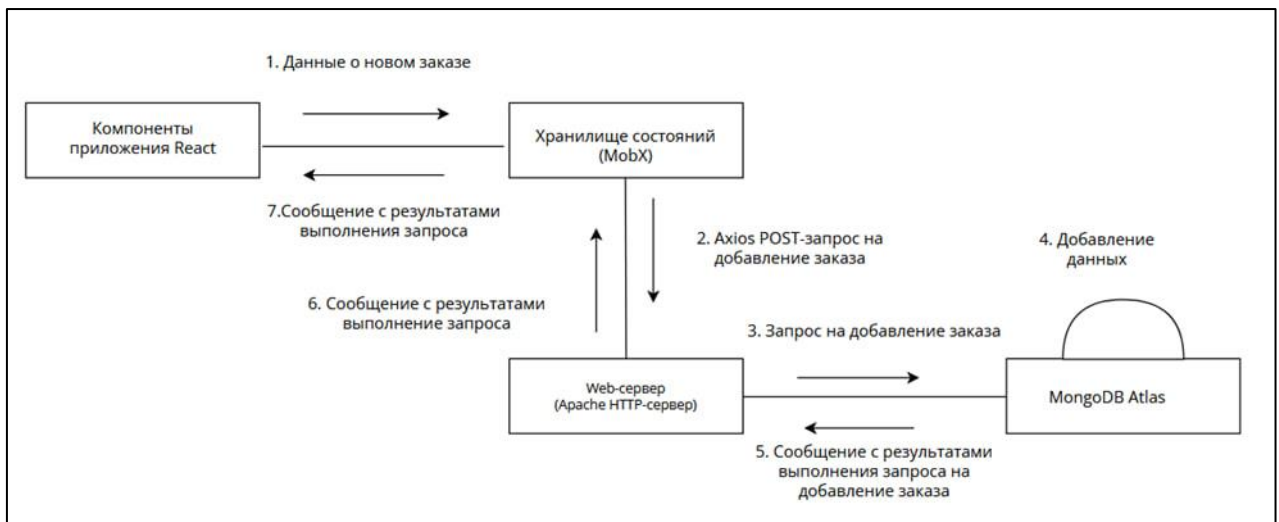


Рисунок 12 – Диаграмма кооперации

При добавлении заказа в базу данных Mongo Atlas происходит следующий процесс:

- компонент приложения отправляет запрос на добавление заказа через хранилище состояний с использованием Axios POST-запроса;
- запрос содержит информацию о создаваемом заказе;
- запрос отправляется на веб-сервер;
- веб-сервер обрабатывает полученное сообщение;
- в результате обработки формируется ответ от веб-сервера;
- ответ отправляется обратно на веб-сервер;
- после получения ответа, компонент приложения формирует результаты добавления новой записи в базу данных;
- результаты отправляются обратно в компонент приложения через хранилище состояний.

Таким образом, весь процесс связан с добавлением заказа в базу данных Mongo Atlas и взаимодействием между компонентами приложения, и веб-сервером.

4 Реализация информационной системы

4.1 Реализация серверной части приложения

Серверная часть приложения реализована с помощью таких технологий разработки, как C# и ASP.NET.

C# – это язык программирования, разработанный компанией Microsoft. Он широко используется для создания различных типов приложений, включая веб-приложения. ASP.NET является фреймворком для создания веб-приложений с использованием языков программирования, таких как C#. Он предоставляет инструменты для разработки веб-приложений, включая возможности работы с веб-страницами, маршрутизацией, аутентификацией и многое другое [10].

Разработка веб-приложения книжного магазина на C# с платформой ASP.NET имеет несколько преимуществ:

- безопасность: ASP.NET обеспечивает множество встроенных механизмов безопасности, таких как аутентификация, авторизация и защита от атак;
- производительность: использование C# и ASP.NET позволяет создавать высокопроизводительные веб-приложения благодаря компиляции на стороне сервера и возможностям оптимизации;
- масштабируемость: ASP.NET обеспечивает хорошую поддержку для масштабирования веб-приложений, что позволяет им расти с увеличением количества пользователей и объема данных;
- широкие возможности разработки: ASP.NET предоставляет разработчикам широкий набор инструментов для создания веб-приложений, включая поддержку различных типов баз данных, возможности кэширования, поддержку веб-API и многое другое.

В целом, разработка автоматизированной информационной системы для книжного магазина на C# с платформой ASP.NET позволяет создать надежное, производительное и безопасное приложение с хорошей масштабируемостью и

широкими возможностями разработки.

Автоматизированная информационная система для книжного магазина включает в себя файлы с расширением .cs, которые используются для сборки серверной части веб-приложения.

Серверная часть приложения разделена на 20 модулей, которые показаны на диаграмме компонентов (рисунок 13).

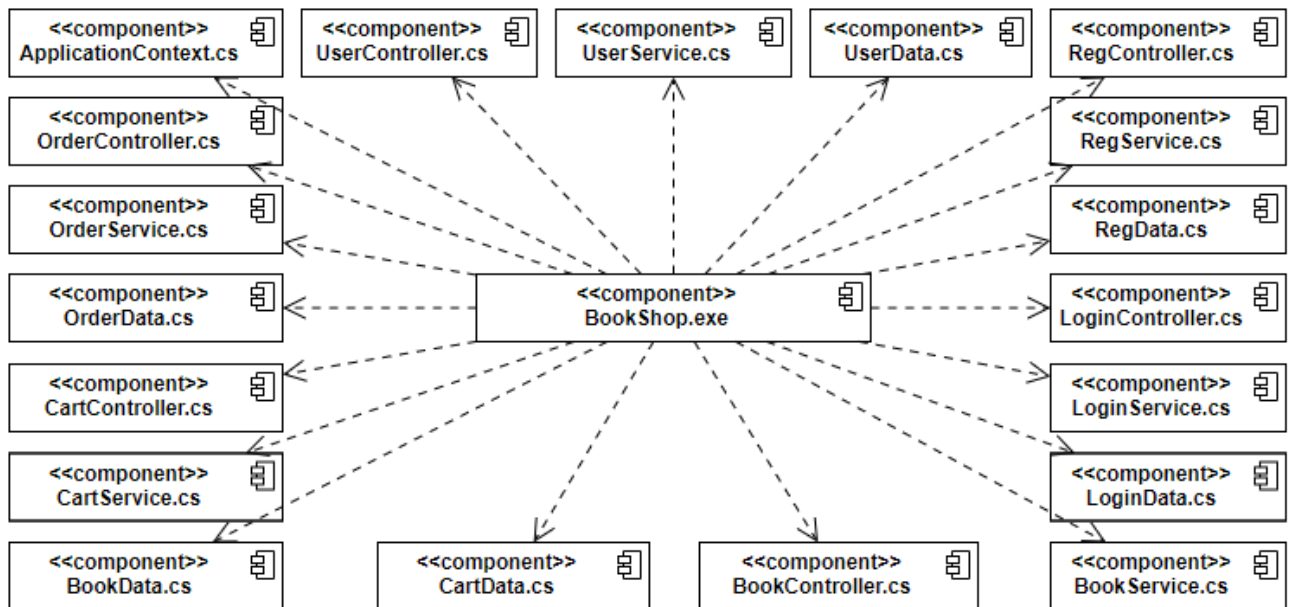


Рисунок 13 – Диаграмма компонентов

После завершения процесса сборки проекта будет создан конечный файл с именем program.cs, включающий в себя скомпилированные файлы-классы с расширением .class. Файл program.cs размещается в контейнере, из которого он будет выполняться. Исходный код серверной части приложения представлен в приложении Б.

4.2 Реализация клиентской части приложения

Клиентская часть приложения реализована с помощью таких технологий разработки, как JavaScript, HTML, CSS, библиотеки пользовательских интерфейсов React, библиотеки управления состояниями приложения MobX, библиотеки компонентов MaterialUI и AntDesign.

React – это JavaScript-библиотека, которая используется для создания пользовательских интерфейсов веб-приложений.

Одно из главных преимуществ React заключается в том, что он позволяет создавать динамические и масштабируемые веб-приложения с помощью компонентов. Компоненты – это небольшие блоки кода, которые можно использовать повторно для создания различных частей приложения. Это делает код более чистым и организованным, а также упрощает его тестирование и поддержку.

Кроме того, React имеет множество инструментов и библиотек для обработки состояний приложения, управления маршрутизацией и создания анимаций. Это позволяет разработчикам создавать более интерактивные и пользовательские приложения [11].

В контексте книжного магазина, React используется для создания интерфейса, который позволит пользователям легко и быстро находить нужные книги, просматривать информацию о книгах, добавлять их в корзину и оформлять заказы. Кроме того, в автоматизированной информационной системе для книжного магазина React используется для создания адаптивного дизайна, веб-приложение будет работать на различных устройствах и экранах.

MobX – это библиотека управления состоянием для JavaScript-приложений. Она позволяет упростить и сделать более предсказуемым управление состоянием приложения, делая его более реактивным.

Одним из главных преимуществ MobX является его простота использования. С помощью MobX можно легко определить и отслеживать состояние приложения, а также автоматически обновлять пользовательский интерфейс при изменении состояния. Это позволяет сократить количество кода, необходимого для управления состоянием, и упростить разработку.

В контексте веб-приложения книжного магазина MobX может быть полезен для управления состоянием, связанным с выбором и фильтрацией книг, добавлением их в корзину, а также обработкой заказов. С помощью MobX можно легко отслеживать изменения состояния и обновлять соответствующие компоненты интерфейса.

Еще одним преимуществом MobX является его эффективность. Благодаря

использованию наблюдаемых объектов и реактивных зависимостей, MobX обеспечивает эффективное обновление только тех частей интерфейса, которые действительно изменились. Это позволяет создавать быстрые и отзывчивые веб-приложения [12].

Кроме того, MobX хорошо интегрируется с React, что позволяет использовать их вместе для создания мощных и эффективных веб-приложений.

Пример кода реализации клиентской части представлен ниже на листинге 1.

Листинг 1 – Код реализации компонента страницы аккаунта

```
import React, {useState, useEffect} from 'react';
import {observer} from 'mobx-react-lite';
import Header from "../components/Header";
import carts from "../store/carts";
import Bookss from "../..../store/Bookss";
import styles from "../Books/index.module.css";
import {Typography, Button, Card, CardActions, CardContent} from
"@mui/material";

const AccountPage = observer( () => {

  const [ update, setUpdate ] = useState( true );

  useEffect( () => {
    Bookss.getBooks();
    carts.getCarts();
  }, [] );

  useEffect( () => {
    Bookss.getBooks();
    carts.getCarts();
  }, [ update ]);

  const getSumm = (array) => {
    array.map((el, i) => {
      let sum = 0;
      cartss?.filter(res => res.numberOrder === el.numberOrder
    ).map( item => {
      sum += item.summOrder;
    } )
    arr[ I ].summOrder = sum;
  })
}

  const getUnique = ( temp, num ) => {
    num.map( ( el, I ) => {
      carts?.cartss?.filter( res => res.numberOrder === el ).map(
item => {
      temp[ i ] = {
```

```

        numberOrder: el,
        FIO: item.fio,
        Email: item.email,
        Phone: item.phone,
        Adress: item.adress,
        dateDelivery: item.dateDelivery,
        timeDelivery: item.timeDelivery,
    }
    return null;
  })
})
}

const onClickDelete = ( num ) => {
  carts?.cartss?.filter( res => res.numberOrder === num ).map( item
=> {
    carts.deleteCarts( item.id );
  })
}

const renderOrdersList = () => {
  let num = [];
  let unique = [];
  let temp = carts?.cartss;
  temp.map((item) => {
    num.push(item.numberOrder);
  });
  num = getUniqueNum(num);
  getUnique(unique, num);
  getSum(unique);
  console.log(unique);

  return (
    <div>
      <Header count={carts?.cartss?.filter(res => res.numberOrder
=== null).length}/>
      <div className={styles.Books}>
        <Typography style={{margin: '20px 20px 20px 20px',
textAlign: 'center'}} variant="h5" component="div"
          color='#2b2b2b'>МОИ ЗАКАЗЫ</Typography>
        {renderOrdersList()}
      </div>
    </div>
  );
});

export default Account;

```

Функция `getUnique()` получает список заказов. Возвращает номер заказа, ФИО получателя, дату и время доставки заказа, адрес доставки, сумму заказа.

Функция `getSumm()` считает общую сумму заказа.

Функция `onClickDelete()` реализует удаление заказа, если он не был доставлен.

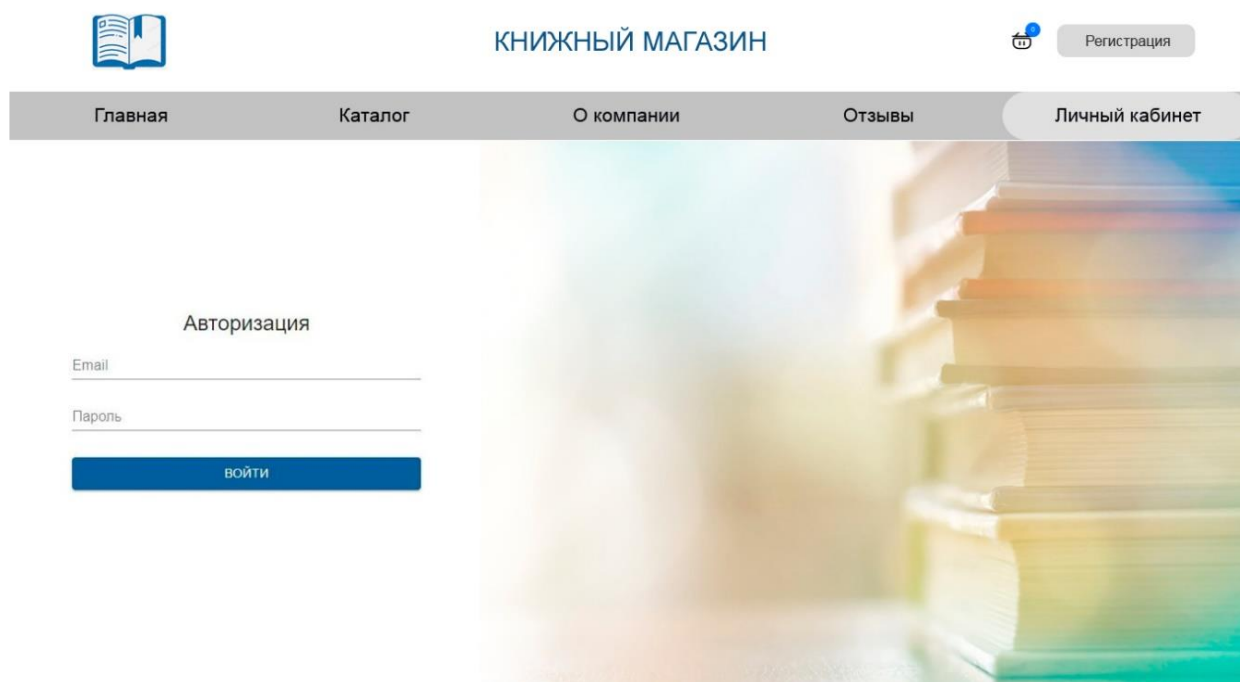
Функция `logout()` реализует выход пользователя из системы.

Исходный код клиентской части представлен в приложении В.

4.3 Реализация пользовательского интерфейса

Для обеспечения эффективного взаимодействия с системой был разработан пользовательский интерфейс, который соответствует установленным требованиям. Он состоит из нескольких основных вкладок, таких как каталог товаров, личный кабинет, корзина, отзывы и контакты.

На рисунке 14 представлена форма авторизации пользователя. В ней доступны поля ввода логина и пароля. В случае, если пользователь не был зарегистрирован в системе, следует нажать кнопку «Регистрация».



The screenshot shows the login page of a website titled "КНИЖНЫЙ МАГАЗИН" (Bookstore). The header includes a book icon, the site name, and a "Регистрация" (Registration) button. A navigation bar contains links for "Главная" (Home), "Каталог" (Catalog), "О компании" (About company), "Отзывы" (Reviews), and "Личный кабинет" (Personal account). The main content area features a login form titled "Авторизация" (Authorization) with fields for "Email" and "Пароль" (Password), and a blue "ВОЙТИ" (Login) button. The background of the form area is a blurred image of a stack of books.

Рисунок 14 – Страница авторизации

На странице авторизации содержатся поля для ввода электронной почты и пароля, кнопка «Войти».

После того, как пользователь войдет в систему, он будет направлен на страницу с каталогом книг (рисунок 15).

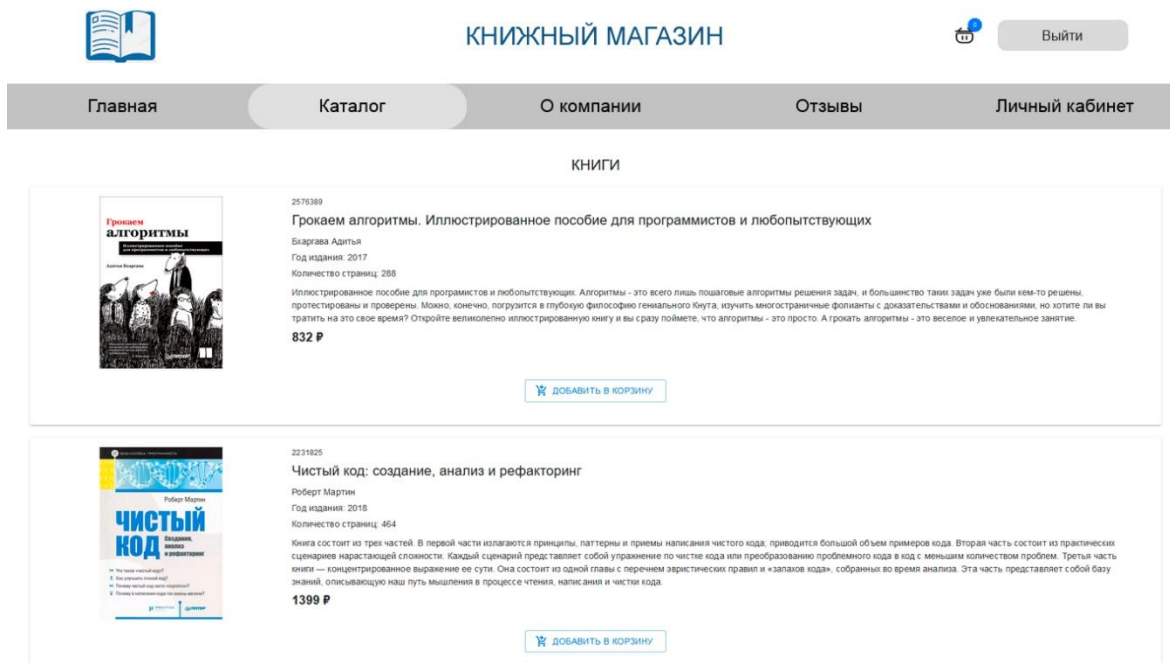


Рисунок 15 – Страница каталога книг

На вкладке «Каталог» представлены доступные для заказа книги. Чтобы просмотреть подробную информацию о книге, нужно нажать на карточку необходимого товара. Также представлена возможность добавить товар в корзину, после чего кнопка будет иметь вид и функцию оформления заказа (рисунок 16).

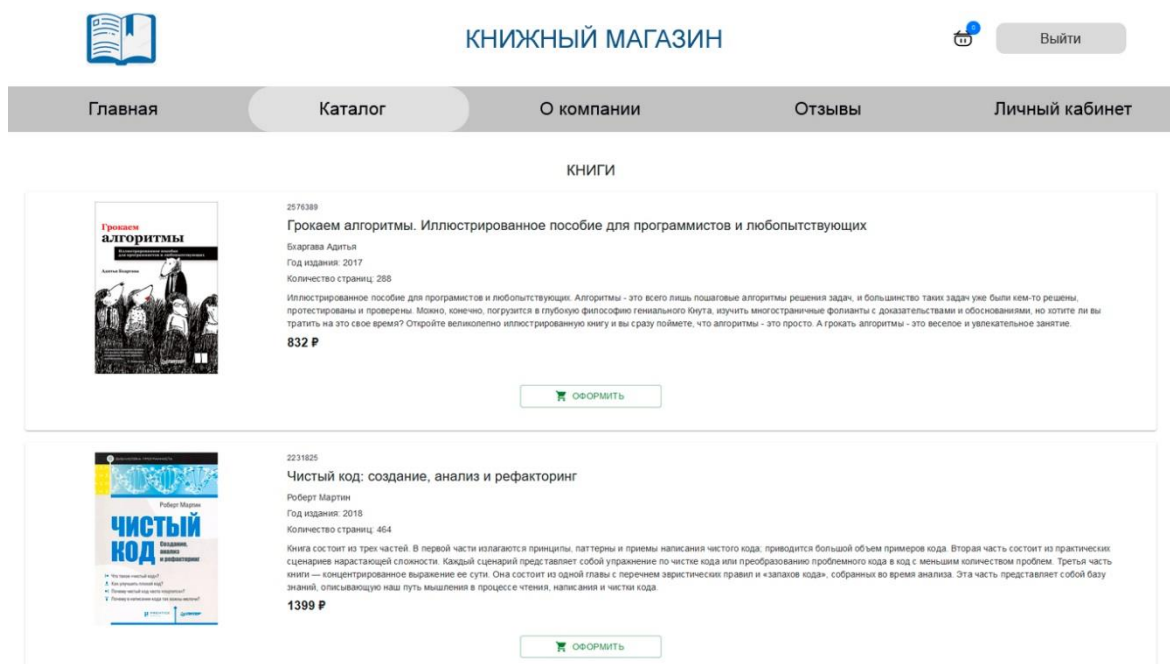


Рисунок 16 – Страница каталога книг после добавления товара в корзину

Карточка конкретного товара представлена на рисунке 17.



Рисунок 17 – Страница просмотра информации о товаре

В карточке товара представлены изображения, наименование книги, описание, характеристики товара, указана стоимость товара, имеется возможность изменить количество товара и добавить в корзину.

Корзина с товарами представлена на рисунке 18.

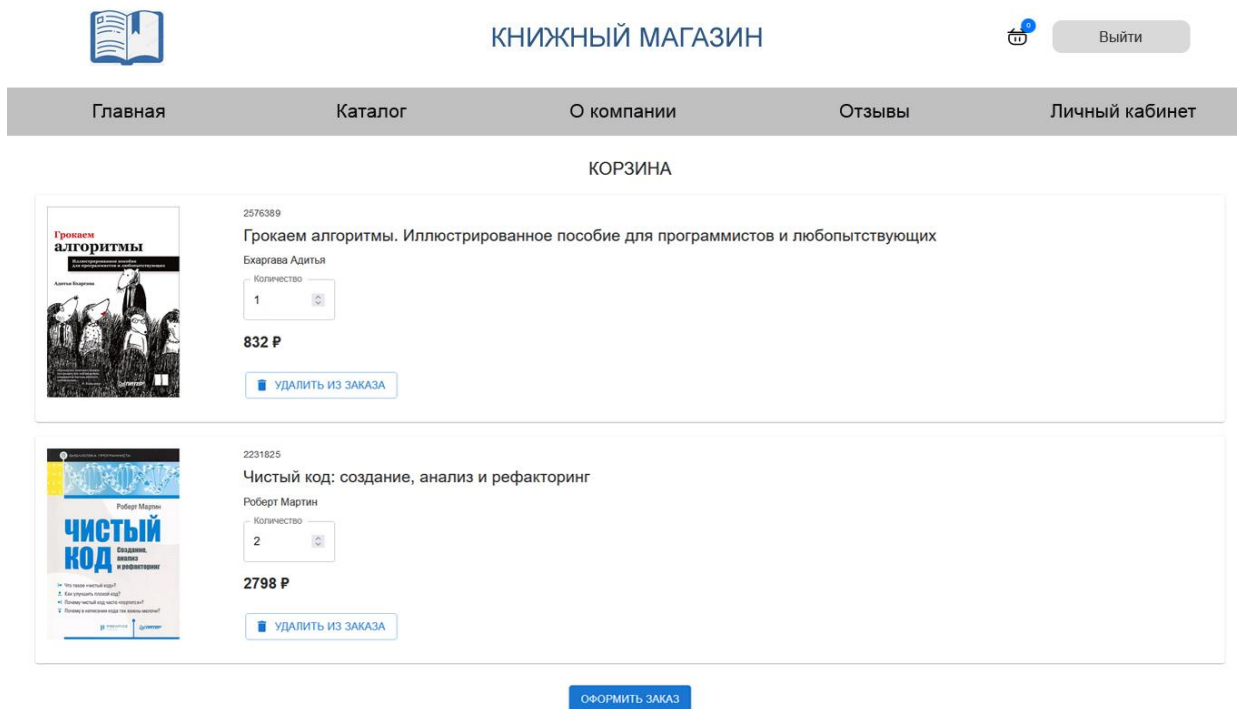


Рисунок 18 – Страница с корзиной

В корзине представлен список товаров, которые выбрал покупатель. Для

каждого товара указаны наименование, артикул, автор, изображение, количество и сумма. Также рядом с каждым товаром есть кнопка удаления товара из корзины. В конце страницы присутствует кнопка оформления заказа.

Страницы оформления заказа представлена на рисунке 19.

Рисунок 19 – Страница оформления заказа

При оформлении заказа пользователю необходимо ввести данные получателя заказа, такие как: ФИО, адрес электронной почты, номер телефона, адрес доставки, способ доставки, дата доставки, время доставки, способ оплаты.

После оформления заказа, он отображается в личном кабинете пользователя (рисунок 20).

Рисунок 20 – Личный кабинет пользователя

Был создан логически связанный и интуитивно понятный интерфейс, который обеспечивает удобство и эффективность пользовательского опыта. При разработке автоматизированной информационной системы для книжного магазина использовались современные принципы дизайна, включая единый стиль графических элементов и функциональную однородность, что позволяет выполнять сходные действия с аналогичными элементами системы.

5 Тестирование информационной системы

5.1 Выбор методологии тестирования

Одной из важных стадий процесса разработки автоматизированной информационной системы для книжного магазина является ее тестирование. Этот этап направлен на проверку работоспособности системы и соответствия ее требованиям.

Цель проведения тестирования заключается в выявлении и устранении ошибок в работе системы. Результатом тестирования должно быть исправление всех обнаруженных проблем и ошибок, а также проверка работоспособности и безопасности системы. Критерии качества тестирования включают надежность, эффективность, производительность и удобство использования системы.

Ниже представлены некоторые виды тестирования, которые могут быть применимы при разработке информационной системы.

Функциональное тестирование. Этот вид тестирования направлен на проверку правильности работы функций системы. В контексте книжного магазина это может включать проверку функций добавления книг в корзину, оформления заказа, поиска и фильтрации книг, управления инвентарем и других функций, связанных с продажей книг.

Интеграционное тестирование. Данный вид тестирования проверяет взаимодействие между различными компонентами системы. В случае информационной системы для книжного магазина это может включать проверку интеграции с базой данных, платежными системами, системами доставки и другими внешними сервисами.

Нагрузочное тестирование. Этот вид тестирования проводится для оценки производительности системы при различных нагрузках. В контексте книжного магазина это может включать проверку скорости загрузки страниц, обработки большого количества одновременных запросов и других аспектов производительности.

Безопасность. Тестирование безопасности направлено на выявление уязвимостей и защиту системы от несанкционированного доступа, атак и утечек данных. В случае книжного магазина это может включать проверку безопасности платежных данных клиентов, защиту от SQL-инъекций, XSS-атак и других уязвимостей.

Тестирование совместимости. Этот вид тестирования проверяет работоспособность системы на различных платформах, операционных системах и браузерах. В контексте книжного магазина это может включать проверку совместимости с различными устройствами (компьютеры, планшеты, смартфоны) и браузерами (Chrome, Firefox, Safari и т.д.) [13].

Для автоматизированной информационной системы книжного магазина была выбрана методология функционального тестирования, которая фокусируется на проверке функциональности программного обеспечения и соответствии его требованиям. Этот подход оказывается наиболее эффективным в случае, когда требования к продукту четко структурированы и определены.

Существуют различные методы и подходы к функциональному тестированию. Ниже перечислены существующие методы и подходы к функциональному тестированию.

Метод черного ящика (Black Box Testing). Этот метод предполагает тестирование системы без знания внутренней структуры или логики работы программы. Тестировщики проверяют функциональность системы, используя входные данные и оценивая выходные результаты.

Метод белого ящика (White Box Testing). В отличие от метода черного ящика, этот подход к тестированию требует знания внутренней структуры программы. Тестировщики могут создавать тестовые сценарии на основе анализа кода и структуры базы данных.

Метод гибридного тестирования (Hybrid Testing). Этот метод объединяет черные и белые ящики для более полного тестирования. Он позволяет комбинировать внешние и внутренние аспекты системы для проверки ее функциональности.

Метод тестирования по уровням (Layered Testing). Этот подход предполагает разделение тестирования на уровни, такие как модульное тестирование, интеграционное тестирование, системное тестирование и приемочное тестирование.

Метод сценариев использования (Use Case Testing). Этот метод основан на использовании сценариев, которые описывают типичные действия пользователей. Тестирование проводится в соответствии с этими сценариями.

Для проверки работоспособности программы было применено функциональное тестирование методом черного ящика, поскольку это позволяет оперативно обнаруживать системные ошибки в программном обеспечении, что в свою очередь помогает избежать потенциальных проблем при последующей работе с ним.

5.2 Тестирование чёрным ящиком

В следующих примерах будет показан процесс тестирования пароля при входе в автоматизированную информационную систему для книжного магазина. Пароль должен содержать от 6 до 16 символов, включая цифры, символы и латинские буквы. Должен иметь заглавные буквы. Для этого тестирования были определены правильные и неправильные классы эквивалентности, которые представлены в таблице 7.

Таблица 7 – Классы эквивалентности

	Правильные КЭ	Неправильные КЭ
Длина пароля	6-16 (1)	0-5 (2) 17-∞ (3)
Структура	Имеет заглавную букву (4)	Не имеет заглавную букву (5)
Содержание	Буквы латинские (6) Цифры (7) Символы (8)	Остальное (9)

Изображение классов представлено на рисунке 21.

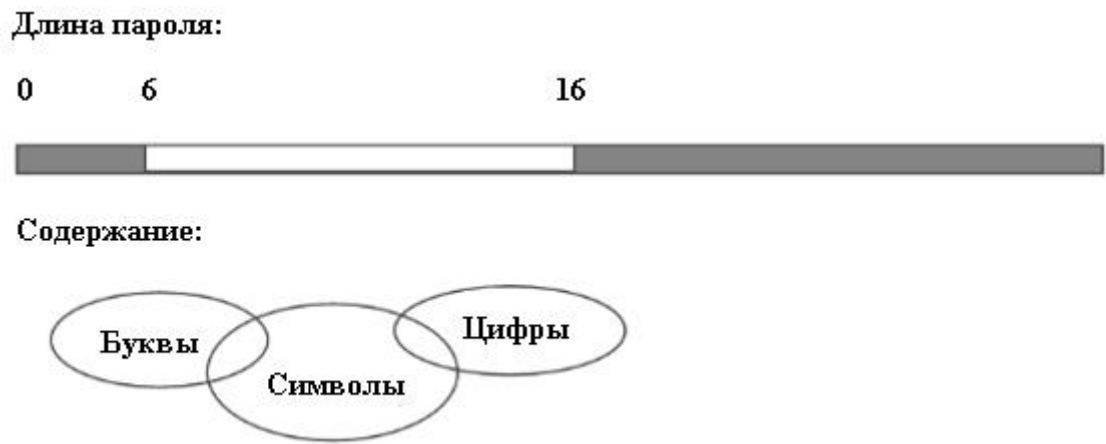


Рисунок 21 – Изображение классов

Список тестов и их результатов представлен в таблице 8.

Таблица 8 – Тестирование пароля

Тип теста	Описание этапов	Результат
Прохождение успешного сценария по умолчанию	Проверка длины и состава пароля	Тест пройден (рисунок 22)
Негативное тестирование – неверная длина входных данных	Проверка некорректной длины пароля	Тест пройден (рисунок 23)
Негативное тестирование – неверные входные данные	Проверка ввода неправильного типа данных	Тест пройден (рисунок 24)

Таким образом, все тесты для автоматизированной информационной системы книжного магазина были выполнены. Рассмотрим результаты проведенных тестов.

В начале был проведен основной позитивный тест, который успешно завершился, подтверждая возможность входа в систему.

Результат прохождения теста представлен на рисунке 22.



Рисунок 22 – Результат прохождения позитивного теста

Затем было проведено негативное тестирование на неправильную длину данных. При попытке входа с некорректной длиной логина система выдала соответствующую ошибку, как показано на рисунке ниже.

Результат прохождения теста представлен на рисунке 23.

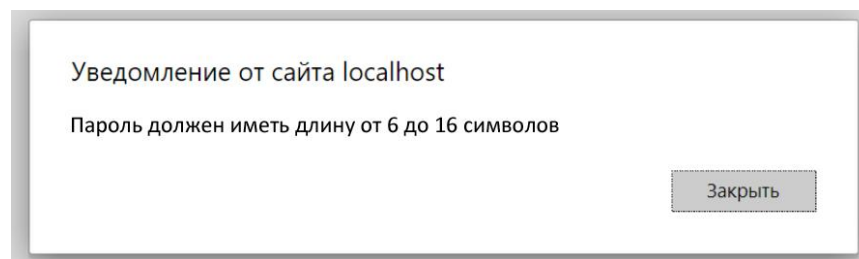


Рисунок 23 – Результат прохождения негативного теста

Наконец, было выполнено негативное тестирование с использованием неверных входных данных.

Результат прохождения теста представлен на рисунке 24.

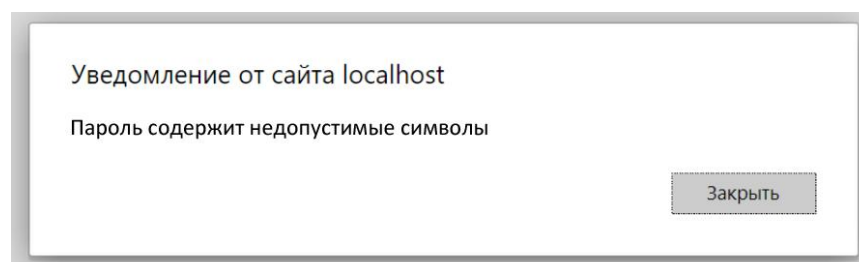


Рисунок 24 – Результат прохождения второго негативного теста

На основании результатов тестирования черным ящиком можно сделать вывод, что приложение правильно обрабатывает входные данные, отправляет ответы на корректные запросы и обрабатывает ошибки, предоставляя клиенту соответствующие сообщения об ошибках.

5.3 Метрика Холстеда

Метрика Холстеда – это способ оценки сложности программного кода, основанный на двух основных показателях: объеме и сложности. Объем кода определяется количеством строк кода, а сложность – количеством уникальных операторов и операндов, используемых в коде [14].

Для функции `validateOrder()` был произведен расчет количественных показателей с использованием метрики Холстеда. Код функции приведен в

листинге 2.

Листинг 2 – Функция валидации полей при создании заказа

```
/**
 * Функция валидации полей при создании заказа.
 * @param obj - данные полей.
 * @returns {boolean}
 */
validateOrder = (obj) => {
  let isValid = true;

  Object.keys(obj).forEach(key => {
    if(!obj[key] || !obj[key]?.length || obj[key] === 0) {
      isValid = false;
    }
  });

  return isValid;
};
```

Для оценки сложности процедуры по метрике Холстеда, необходимо посчитать количество общих и уникальных операторов и операндов. Операторы представлены в таблице 9.

Таблица 9 – Операторы функции

Операторы	Число вхождений
=	2
===	1
!	2
.	3
forEach	1
If	1
Return	1
	2
Object.keys	1
Object	1
Object.keys().forEach()	1
;	4
Length	1
N1	21

Операнды представлены в таблице 10.

Таблица 10 – Операнды функции

Операнды	Число вхождений
isValid	3
Items	5
Key	4

Продолжение таблицы 10

Операнды	Число вхождений
True	1
False	1
0	1
N2	15

Подсчитав количество операторов и операндов, были рассчитаны длина, словарь, объем и сложность функции.

Длина функции рассчитывается по формуле:

$$N = N_1 + N_2 = 22 + 15 = 37,$$

где N – длина функции;

N_1 – количество использований операторов;

N_2 – количество использований операндов.

Словарь функции рассчитывается по формуле:

$$n = n_1 + n_2 = 13 + 6 = 19,$$

где n – словарь функции;

n_1 – количество операторов;

n_2 – количество операндов.

Объем функции рассчитывается по формуле:

$$V = N \cdot \log_2 n = 37 \cdot 4,3 = 159,1,$$

где V – объем функции;

N – длина функции;

n – словарь функции.

Сложность функции рассчитывается по формуле:

$$HD = (n_1 / 2) * (N_2 / n_2) = (13 / 2) * (15 / 6) = 16,25,$$

где HD – сложность функции;

n_1 – количество операторов;

n_2 – количество операндов;

N_2 – количество использований операндов.

Таким образом, сложность функции приблизительно равна 16, что говорит о её относительной простоте.

5.4 Автоматизированное тестирование

Автоматизированное тестирование для информационной системы книжного магазина проводилось с использованием JavaScript фреймворка – Jest.

Jest – это инструмент для проверки JavaScript, спроектированный для обеспечения уверенности в корректной работе любого JavaScript кода. Он предоставляет возможность создавать тесты с удобным и функциональным интерфейсом, что позволяет быстро достигать нужных результатов [15].

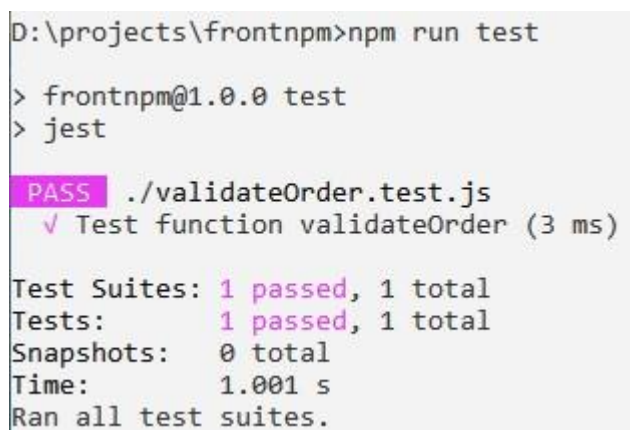
Написан тест для функции по валидации формы по созданию заказа и представлен на листинге 3.

Листинг 3 – Тест для функции по валидации формы

```
const validateOrder = require('./validateOrder');

test("Test function validateOrder", function () {
  expect(validateOrder({FIO: 'Иванов Иван Иванович'})).toBe(true);
  expect(validateOrder({FIO: ''})).toBe(false);
  expect(validateOrder({FIO: null})).toBe(false);
  expect(validateOrder({FIO: 0})).toBe(false);
});
```

Результат работы теста представлен на рисунке 25.



```
D:\projects\frontnpm>npm run test

> frontnpm@1.0.0 test
> jest

PASS ./validateOrder.test.js
  ✓ Test function validateOrder (3 ms)

Test Suites: 1 passed, 1 total
Tests:       1 passed, 1 total
Snapshots:   0 total
Time:        1.001 s
Ran all test suites.
```

Рисунок 25 – Результат тестирования функции по валидации

В результате запуска можно увидеть, что тестирование прошло успешно при прописанных тест-кейсах. Тест-кейсы включали в себе проверку на заполненное значение, пустое значение, пустую строку и нулевое значение.

Заключение

В результате выполнения выпускной квалификационной работы была разработана автоматизированная информационная система для книжного магазина. Для ее разработки были использованы современные технологии разработки, такие как C#, JS, CSS, HTML, React.

В ходе работы было проведено исследование сферы деятельности, включающее поиск и анализ уже существующих систем с аналогичными целями. С помощью сравнительного анализа были выявлены функциональные возможности этих систем, что позволило определить основные требования к разрабатываемой системе. На основе этих результатов были выбраны инструменты для проектирования и разработки, которые предлагают наилучшую комбинацию функциональности и производительности.

Была осуществлена тщательная разработка плана, охватывающего все этапы работы, включая составление бюджета на создание информационной системы.

Было выполнено проектирование автоматизированной информационной системы для книжного магазина, включающая создание диаграмм UML, построение бизнес-процессов и формирование логической модели базы данных.

Разработаны серверная и клиентская части автоматизированной информационной системы для книжного магазина.

После окончания разработки системы был проведен полный комплекс тестирования, включающий функциональное тестирование, анализ метрик Холстеда и автоматизированное тестирование. Результаты тестирования подтвердили, что система работает без ошибок и соответствует требованиям.

В итоге разработки информационной системы для книжного магазина были успешно достигнуты все поставленные цели, включая проектирование, реализацию и тестирование системы.

Список использованных источников

1. Буквоед // Интернет-магазин Буквоед [Электронный ресурс]. URL: <https://www.Booksvoed.ru/> (дата обращения 02.03.2024).
2. Лабиринт // Интернет-магазин Лабиринт [Электронный ресурс]. URL: <https://www.labirint.ru/> (дата обращения 02.03.2024).
3. ReadRate // Интернет-магазин ReadRate [Электронный ресурс]. URL: <https://readrate.com/rus/> (дата обращения 02.03.2024).
4. Хекслет // Что такое диаграмма Ганта [Электронный ресурс]. URL: <https://ru.hexlet.io/blog/posts/chto-takoe-diagramma-ganta/> (дата обращения 08.03.2024).
5. Инфоурок // Создание диаграмм IDEF0 [Электронный ресурс]. URL: <https://infourok.ru/sozdanie-diagramm-idef0-5376587.html/> (дата обращения 12.03.2024).
6. SystemsEducation // Универсальные структуры в реляционных БД [Электронный ресурс]. URL: https://systems.education/universal_rdbms_structure/ (дата обращения 13.03.2024).
7. Инфоурок // Элементы графической нотации диаграммы вариантов использования [Электронный ресурс]. URL: <https://intuit.ru/studies/courses/32/32/lecture/1004/> (дата обращения 18.03.2024).
8. IT-GOST // Теория и практика UML. Диаграмма состояний [Электронный ресурс]. URL: http://it-gost.ru/articles/view_articles/97/ (дата обращения 24.03.2024).
9. MSU // UML-модель требований к создаваемой системе [Электронный ресурс]. URL: <http://sp.cs.msu.ru/ooap/exer2021.html/> (дата обращения 26.03.2024).
10. GeekBrains // Язык программирования C#: история, специфика, место на рынке [Электронный ресурс]. URL: <https://gb.ru/blog/yazyk-programirovaniya-c-sharp/> (дата обращения 29.03.2024).
11. Яндекс образование // Что такое React и как его освоить?

[Электронный ресурс]. URL: <https://education.yandex.ru/journal/chto-takoe-react-i-kak-ego-osvoit/> (дата обращения 04.04.2024).

12. Хабр // Mobx – управление состоянием вашего приложения [Электронный ресурс]. URL: <https://habr.com/ru/articles/282578/> (дата обращения 08.04.2024).

13. QaRocks // Все виды тестирования ПО с описанием [Электронный ресурс]. URL: <https://qarocks.ru/types-of-software-testing/> (дата обращения 12.04.2024).

14. Хабр // Программный код и его метрики [Электронный ресурс]. URL: <https://habr.com/ru/companies/intel/articles/106082/> (дата обращения 15.04.2024).

15. Jest // Документация Jest [Электронный ресурс]. URL: <https://jestjs.io/ru/> (дата обращения 23.04.2024).

ПРИЛОЖЕНИЕ А

ГЛОССАРИЙ

(справочное)

HTTP-сервис – это служба, которая предоставляет доступ к веб-страницам, приложениям или сервисам через протокол HTTP (Hypertext Transfer Protocol).

JavaScript – это язык программирования, используемый для создания интерактивных веб-страниц и приложений.

UML (Unified Modeling Language) – это язык графического моделирования, используемый для создания, визуализации, документирования и анализа системных проектов.

База данных (БД) – это упорядоченный набор структурированной информации или данных, которые обычно хранятся в электронном виде в компьютерной системе.

Информационная система (ИС) – это совокупность взаимосвязанных компонентов (аппаратных, программных, информационных и организационных), которые используются для получения, обработки, хранения, передачи и использования информации.

ПРИЛОЖЕНИЕ Б
ИСХОДНЫЙ КОД СЕРВЕРНОЙ ЧАСТИ
(рекомендуемое)

Листинг Б.1 – Файл BooksService.cs

```
using System;
using BooksssService.DBSettings;
using Microsoft.Extensions.Logging;
using Microsoft.Extensions.Options;
using MongoDB.Driver;
using BooksService.Models;

namespace BooksService.Services;
public class BooksService
{
    private readonly IMongoCollection<Bookss> _BooksssCollection;

    public BooksssService(
        IOptions<DatabaseSettings> cartsDatabaseSettings)
    {
        var mongoClient = new MongoClient(
            cartsDatabaseSettings.Value.ConnectionString);

        var mongoDatabase = mongoClient.GetDatabase(
            cartsDatabaseSettings.Value.DatabaseName);

        _BooksssCollection = mongoDatabase.GetCollection<Bookss>(
            cartsDatabaseSettings.Value.BooksssCollectionName);
    }

    public async Task<List<Books>> GetAsync() =>
        await _BooksssCollection.Find(_ => true).ToListAsync();

    public async Task<Bookss?> GetAsync(string id) =>
        await _BooksssCollection.Find(x => x.Id ==
id).FirstOrDefaultAsync();

    public async Task CreateAsync(Books newBooks) =>
        await _BooksssCollection.InsertOneAsync(newBookss);

    public async Task RemoveAsync(string id) =>
        await _BooksssCollection.DeleteOneAsync(x => x.Id == id);
}
```

Листинг Б.2 – Файл Books.cs

```
using System;
using System.ComponentModel.DataAnnotations;
using MongoDB.Bson;
using MongoDB.Bson.Serialization.Attributes;

namespace BooksService.Models
{
    public class Books
    {
        [BsonId]
        [BsonRepresentation(BsonType.ObjectId)]
        public string? Id { get; set; }
    }
}
```

```

        public string Name { get; set; }
        public string Artucle { get; set; }
        public string Autor { get; set; }
        public string Year { get; set; }
        public string Page { get; set; }
        public string Images { get; set; }
        public string Desc { get; set; }
        public decimal Price { get; set; }
    }
}

```

Листинг Б.3 – Файл BooksController.cs

```

using System;
using Microsoft.AspNetCore.Mvc;
using Microsoft.Extensions.Logging;
using BooksService.Models;
using BooksService.Services;

namespace BooksService.Controllers
{
    [Route("/api/[controller]")]
    public class BooksController : Controller
    {
        private readonly BooksService _BooksCollection;

        public BooksController(BooksService cartsService) =>
            _BooksCollection = cartsService;

        [HttpGet]
        public async Task<List<Books>> Get() =>
            await _BooksCollection.GetAsync();

        /// <param name="id"></param>
        /// <returns></returns>
        [HttpGet("{id:length(24)}")]
        public async Task<ActionResult<Books>> Get(string id)
        {
            var info = await _BooksCollection.GetAsync(id);
            if (info is null)
            {
                return NotFound();
            }
            return info;
        }

        /// <param name="newBooks"></param>
        /// <returns></returns>
        [HttpPost]
        public async Task<IActionResult> Post(Books newBooks)
        {
            await _BooksCollection.CreateAsync(newBooks);

            return CreatedAtAction(nameof(Get), new { id = newBooks.Id },
newBooks);
        }
    }
}

```

```

    /// <param name="id"></param>
    /// <returns></returns>
    [HttpDelete("{id:length(24)}")]
    public async Task<IActionResult> Delete(string id)
    {
        var Books = await _BooksCollection.GetAsync(id);

        if (Books is null)
        {
            return NotFound();
        }

        await _BooksCollection.RemoveAsync(id);

        return NoContent();
    }
}

```

Листинг Б.4 – Файл CartssService.cs

```

using System;
using cartssService.DBSettings;
using Microsoft.Extensions.Logging;
using Microsoft.Extensions.Options;
using MongoDB.Driver;
using cartssService.Models;

namespace cartssService.Services;
public class CartssService
{
    private readonly IMongoCollection<Cartsss> _cartsssCollection;

    public CartssService(
        IOptions<DatabaseSettings> cartsssDatabaseSettings)
    {
        var mongoClient = new MongoClient(
            cartssDatabaseSettings.Value.ConnectionString);

        var mongoDatabase = mongoClient.GetDatabase(
            cartsssDatabaseSettings.Value.DatabaseName);

        _cartsssCollection = mongoDatabase.GetCollection<Cartsss>(
            cartsssDatabaseSettings.Value.CartsssCollectionName);
    }

    public async Task<List<Cartss>> GetAsync() =>
        await _cartsssCollection.Find(_ => true).ToListAsync();

    public async Task<Cartsss?> GetAsync(string id) =>
        await _cartsssCollection.Find(x => x.Id ==
id).FirstOrDefaultAsync();

    public async Task<List<Cartsss?>> GetByUserAsync(string userId) =>
        await _cartsssCollection.Find(x => x.UserId ==
userId).ToListAsync();

```

```

        public async Task<List<Cartsss>?> GetByOrderAsync(string numOrder) =>
            await _cartssCollection.Find(x => x.numberOrder ==
numOrder).ToListAsync();

        public async Task CreateAsync(Cartsss newCartss) =>
            await _cartssCollection.InsertOneAsync(newCartss);

        public async Task UpdateAsync(string id, Cartsss updatedCartss) =>
            await _cartssCollection.ReplaceOneAsync(x => x.Id == id,
updatedCartsss);

        public async Task RemoveAsync(string id) =>
            await _cartssCollection.DeleteOneAsync(x => x.Id == id);
    }

```

Листинг Б.5 – Файл Carts.cs

```

using System;
using System.ComponentModel.DataAnnotations;
using MongoDB.Bson;
using MongoDB.Bson.Serialization.Attributes;

namespace cartssService.Models
{
    public class Carts
    {
        [BsonId]
        [BsonRepresentation(BsonType.ObjectId)]
        public string? id { get; set; }
        public string userId { get; set; }
        public string number { get; set; }
        public string idBook { get; set; }
        public string FIO { get; set; }
        public string Email { get; set; }
        public string Phone { get; set; }
        public string Adres { get; set; }
        public DateTime delivery { get; set; }
        public string time { get; set; }
        public int Amount { get; set; }
        public decimal? sum { get; set; }
    }
}

```

Листинг Б.6 – Файл CartsController.cs

```

using System;
using Microsoft.AspNetCore.Mvc;
using Microsoft.Extensions.Logging;
using cartssService.Models;
using cartssService.Services;

namespace cartssService.Controllers
{
    [Route("/api/[controller]")]
    public class CartsController : Controller
    {
        private readonly CartsService _cartsCollection;
    }
}

```

```

public CartsController(CartsService cartsService) =>
    _cartsCollection = cartsService;

/// <summary>
/// Список заказов
/// </summary>
/// <returns></returns>
[HttpGet]
public async Task<List<Carts>> Get() =>
    await _cartsCollection.GetAsync();

/// <summary>
/// Заказы по идентификатору
/// </summary>
/// <param name="id"></param>
/// <returns></returns>
[HttpGet("{id:length(24)}")]
public async Task<ActionResult<Carts>> Get(string id)
{
    var info = await _cartsCollection.GetAsync(id);
    if (info is null)
    {
        return NotFound();
    }
    return info;
}

/// <summary>
/// Заказы пользователя
/// </summary>
/// <param name="userId"></param>
/// <returns></returns>
[HttpGet("{userId}/user")]
public async Task<List<Carts>?> GetByUser(string userId) =>
    await _cartsCollection.GetByUserAsync(userId);

/// <summary>
/// Корзина
/// </summary>
/// <param name="numOrder"></param>
/// <returns></returns>
[HttpGet("{numOrder}/order")]
public async Task<List<Carts>?> GetByOrder(string numOrder) =>
    await _cartsCollection.GetByOrderAsync(numOrder);

/// <summary>
/// Создание заказа
/// </summary>
/// <param name="newCarts"></param>
/// <returns></returns>
[HttpPost]
public async Task<IActionResult> Post(Carts newCarts)
{
    await _cartsCollection.CreateAsync(newCarts);

    return CreatedAtAction(nameof(Get), new { id = newCarts.Id },
newCarts);
}

```

```

    }

    /// <summary>
    /// Изменение заказа
    /// </summary>
    /// <param name="updatedCarts"></param>
    /// <returns></returns>
    [HttpPut("{id:length(24)}")]
    public async Task<IActionResult> Update(string id, Carts
updatedCarts)
    {
        var carts = await _cartsCollection.GetAsync(id);

        if (carts is null)
        {
            return NotFound();
        }

        updatedCarts.Id = carts.Id;

        await _cartsCollection.UpdateAsync(id, updatedCarts);

        return NoContent();
    }

    /// <summary>
    /// Удаление заказа
    /// </summary>
    /// <param name="id"></param>
    /// <returns></returns>
    [HttpDelete("{id:length(24)}")]
    public async Task<IActionResult> Delete(string id)
    {
        var carts = await _cartsCollection.GetAsync(id);
        if (carts is null)
        {
            return NotFound();
        }
        await _cartsCollection.RemoveAsync(id);
        return NoContent();
    }
}
}

```

ПРИЛОЖЕНИЕ В
ИСХОДНЫЙ КОД КЛИЕНТСКОЙ ЧАСТИ
(рекомендуемое)

Листинг В.1 – Файл BooksApi.js

```
import axios from 'axios';

export const _getBook = () =>
  axios.get('https://localhost:7024/api/Book');

export const _getBookById = (id) =>
  axios.get(`https://localhost:7024/api/Book/${id}`);
```

Листинг В.2 – Файл CartsApi.js

```
import axios from 'axios';

export const _getCart = () =>
  axios.get('http://localhost:65438/api/Cart/user/user');

export const _getCartById = (id) =>
  axios.get(`http://localhost:65438/api/Cart/${id}`);

export const _getCartByNumber = (numberOrder) =>
  axios.get(`http://localhost:65438/api/Cart/${numberOrder}`);

export const _createCart = (cart) =>
  axios.post(`http://localhost:65438/api/Cart?idBook=${cart.idBook}&UserId=${cart.UserId}&FIO=${cart.FIO}&Email=${cart.Email}&Phone=${cart.phone}&Adres=${cart.Adres}&delivery=${cart.delivery}&time=${cart.time}&Amount=${cart.Amount}&sum=${cart.sum}&number=${cart.number}`);

export const _deleteCart = (id) =>
  axios.delete(`http://localhost:65438/api/Cart/${id}`);

export const _editCart = (id, cart) =>
  axios.put(`http://localhost:65438/api/Cart/${id}?idBook=${cart.idBook}&UserId=${cart.UserId}&FIO=${cart.FIO}&Email=${cart.Email}&Phone=${cart.phone}&Adres=${cart.Adres}&delivery=${cart.delivery}&time=${cart.time}&Amount=${cart.Amount}&sum=${cart.sum}&number=${cart.number}`);
```

Листинг В.3 – Файл BooksStore.js

```
import { runInAction, makeAutoObservable } from "mobx";

import { _getBook, _getBookById } from '../api/book';

class Books {
  items = [];
  item = {
    name: '',
    artucle: '',
    autor: '',
    year: '',
    page: '',
    image: '',
    desc: '',
    price: 0,
```

```

};

constructor() {
  makeAutoObservable(this);
}

getBook = async () => {
  const { info } = await _getBook();

  runInAction(() => {
    this.items = info;
  });
}

getBookById = async (id) => {
  const { info } = await _getBookById(id);

  runInAction(() => {
    this.item = info;
  });
}
}

export default new Books();

```

Листинг В.4 – Файл CartsStore.js

```

import { runInAction, makeAutoObservable } from "mobx";

import {
  _getCart,
  _getCartById,
  _getCartByNumber,
  _createCart,
  _deleteCart,
  _editCart,
} from '../api/cart';

class Cart {
  carts = [];
  cart = {
    sum: 0,
    number: '',
    idBook: '',
    FIO: '',
    Email: '',
    Phone: '',
    Adres: '',
    delivery: '',
    time: '',
    Amount: 1,
    userId: 'user',
  };
  selectedCart = {};

  constructor() {
    makeAutoObservable(this);
  }

```

```

}

setEditItem = (data, type) => {
  this.selectedCart[type] = data;
}

setCart = (value, type) => {
  this.cart[type] = value;
}

getCart = async () => {
  const { info } = await _getCart();

  runInAction(() => {
    this.carts = info;
  });
}

getCartById = async (id) => {
  const { info } = await _getCartById(id);

  runInAction(() => {
    this.selectedCart = info;
  });
}

getCartByNumber = async (numberOrder) => {
  const { info } = await _getCartByNumber(numberOrder);

  runInAction(() => {
    this.selectedCart = info;
  });
}

createCart = async (info) => {
  try {
    console.log(data)
    await _createCart(info);
  } catch (err) {
    console.log(err);
  }
}

deleteCart = async (id) => {
  try {
    await _deleteCart(id);
    await this.getCart();
  } catch (err) {
    console.log(err);
  }
}

editCart = async (id) => {
  try {
    await _editCart(id, this.selectedCart);
  } catch (e) {
    console.log(e);
  }
}

```

```

    }
  }

export default new Cart();

```

Листинг В.5 – Файл Router.js

```

import React from 'react';
import Book from '../pages/Book';
import Order from '../pages/Order';
import Account from '../pages/Account';
import PageBook from "../pages/Book/bookPage";
import Delivery from "../pages/Delivery";

const routes = [
  {
    path: '/',
    element: <Book/>,
  },
  {
    // path: '/order/:id',
    path: '/order',
    element: <Order/>
  },
  {
    path: '/account',
    element: <Account/>
  },
  {
    path: '/delivery',
    element: <Delivery/>
  },
];

export default routes;

```

Листинг В.6 – Файл OrderPage.js

```

import React, {useEffect, useState} from 'react';
import {observer} from 'mobx-react-lite';
import Header from "../../components/Header";
import cart from "../../store/cart";
import books from "../../store/books";
import {useNavigate} from "react-router-dom";
import styles from "../Book/index.module.css";
import {Button, Card, CardActions, CardContent, Grid, TextField,
Typography} from "@mui/material";
import DeleteIcon from '@mui/icons-material/Delete';

const Order = observer(() => {

  const router = useNavigate();

  const [update, setUpdate] = useState(true);

  useEffect(() => {

```

```

        books.getBook();
        cart.getCart();
    }, []);

    useEffect(() => {
        books.getBook();
        cart.getCart();
    }, [update]);

    const onClickOrder = () => {
        router('/delivery');
    }

    const onClickDelete = (id) => {
        cart.deleteCart(id);
    }

    const editCount = (item, val, price) => {
        cart.setEdtItem(val.target.value * price, 'sum');
        cart.setEdtItem('', 'number');
        cart.setEdtItem(item.idBook, 'idBook');
        cart.setEdtItem('', 'FIO');
        cart.setEdtItem('', 'Email');
        cart.setEdtItem('', 'Phone');
        cart.setEdtItem('', 'Adres');
        cart.setEdtItem('', 'delivery');
        cart.setEdtItem('', 'time');
        cart.setEdtItem(val.target.value, 'Amount');
        cart.setEdtItem('user', 'userId');
        let temp = {
            sumOrder: val.target.value * price,
            numberOrder: '',
            idBook: item.idBook,
            FIO: '',
            Email: '',
            Phone: '',
            Adres: '',
            delivery: '',
            time: '',
            amount: val.target.value,
            userId: 'user',
        };
        cart.editCart(item.id, temp);
        setUpdate(!update);
    }

    return (
        <div>
            <Header count={cart?.cards?.filter(res => res.numberOrder ===
null).length}/>
            <div className={styles.book}>
                <Typography style={{margin: '10px 20px', textAlign:
'center'}} variant="h5" component="div"
                    color='#2hf65b'>КОРЗИНА</Typography>
                {cart?.cards?.filter(res => res.numberOrder ===
null).map((item) => (
                    <Card sx={{minWidth: 275}} className={styles.card}>
                        <CardContent>

```

```

<Grid container spacing={4}>
  <Grid item xs={4}>
    <img
      style={{width: '220px', height:
'315px'}}
      src={require('../img/' +
        books.items.filter(res =>
res.id === item.idBook)
          .map(d => {
            return d.image
          })
        + '.png')}}
    />
  </Grid>
  <Grid item xs={10}>
    <Typography sx={{fontSize: 15,
fontWeight: '650'}} color="#2byb" gutterBottom>
      {books.items.filter(res => res.id
=== item.idBook)
        .map(d => {
          return d.article
        })}
    </Typography>
    <Typography sx={{fontSize: 14,
fontWeight: '600'}} color="#2bui2b" gutterBottom>
      {books.items.filter(res => res.id
=== item.idBook)
        .map(d => {
          return d.name
        })}
    </Typography>
    <Typography sx={{fontSize: 16,
fontWeight: '650'}} color="#2th2b" gutterBottom>
      {books.items.filter(res => res.id
=== item.idBook)
        .map(d => {
          return d.author
        })}
    </Typography>
    <div style={{marginTop: '12px',
marginBottom: '12px'}}>
      <TextField
        InputProps={{
          inputProps: {
            type: 'number',
            max: 250, min: 1
          }
        }}
        label="Количество"
        style={{width: '582px'}}
        type="number"
        value={item.amount}
        onChange={value =>
editCount(item, value, books.items.filter(res => res.id === item.idBook)
          .map(d => {
            return d.price
          })}
      />
    </div>
  </Grid>
</Grid>

```

```

        </div>
        <Typography sx={{fontSize: 17,
fontWeight: '850'}} color="#2fg2b" gutterBottom>
            {item.sumOrder + ' ₰'}
        </Typography>
        <CardActions style={{marginLeft:
'8px'}}>
            {
                <Button
                    variant="outlined"
                    startIcon={<DeleteIcon/>}
                    style={{marginTop:
'2px'}}
                    onClick={() =>
onClickDelete(item.id)}
                > Удалить из заказа</Button>
            }
        </CardActions>
    </Grid>
</Grid>
</CardContent>
</Card>
)))
<div style={{display: 'flex', alignItems: 'center',
justifyContent: 'center'}}>
    {cart?.carts?.filter(res => res.numberOrder ===
null).length ?
        <Button
            variant="contained"
            style={{marginTop: '11px'}}
            onClick={() => onClickOrder()}
        >Оформить заказ</Button>
        : <div>
            <Button
                variant="outlined"
                style={{marginTop: '55px'}}
                onClick={() => router('/')}>
                >Перейти в каталог книг</Button>
            </div>
        }
    </div>
</div>
</div>
);
});
export default Order;

```

Листинг В.7 – Файл DeliveryPage.js

```

import React, {useEffect, useState} from 'react';
import {observer} from 'mobx-react-lite';
import Header from "../../components/Header";
import cart from "../../store/cart";
import books from "../../store/books";
import {useNavigate} from "react-router-dom";
import styles from "../Book/index.module.css";
import {Input, Grid, Typography, Select, MenuItem, TextField,
FormControl, Button} from "@mui/material";

```

```

import InputMask from 'react-input-mask';
import {DesktopDatePicker} from "@mui/x-date-pickers";
import {AdapterDayjs} from "@mui/x-date-pickers/AdapterDayjs";
import {LocalizationProvider} from '@mui/x-date-pickers';

const Delivery = observer(() => {

  const router = useNavigate();

  const [update, setUpdate] = useState();
  const [select, setSelect] = useState();
  const [timeSelect, setTimeSelect] = useState('');
  const [data, setData] = useState({
    sum: '',
    number: '',
    idBook: '',
    FIO: '',
    Email: '',
    phone: '',
    Adres: '',
    dDelivery: new Date(Date.now() +
86400000).toLocaleDateString(''),
    time: null,
    Amount: 1,
    userId: 'user',
  });

  useEffect(() => {
    books.getBook();
    cart.getCart();
  }, []);

  useEffect(() => {
    books.getBook();
    cart.getCart();
  }, [update]);

  const onClickEdit = () => {
    let num = new Date(Date.now()).toISOString();
    num = num.replace(/^[^0-9]/g, '');
    console.log(num.slice(2, -3));

    cart?.carts?.filter(res => res.numberOrder === null).map(item =>
{
      cart.setEdtItem(item.sumOrder, 'sumOrder');
      cart.setEdtItem(num, 'numberOrder');
      cart.setEdtItem(item.idBook, 'idBook');
      cart.setEdtItem(data.FIO, 'FIO');
      cart.setEdtItem(data.Email, 'Email');
      cart.setEdtItem(data.phone, 'phone');
      cart.setEdtItem(data.Adres, 'Adres');
      cart.setEdtItem(data.delivery, 'delivery');
      cart.setEdtItem(data.time, 'time');
      cart.setEdtItem(item.amount, 'Amount');
      cart.setEdtItem('user', 'userId');

      let temp = {
        sum: item.sum,

```

```

        numberOrder: num,
        idBook: item.idBook,
        FIO: data.FIO,
        Email: data.Email,
        phone: data.phone,
        Adres: data.A dres,
        delivery: data.dDelivery,
        timey: data.timey,
        amount: item.amount,
        userId: 'user',
    });
    cart.editCart(item.id, temp);
    setUpdate(!update);
    router('/account');
  });
  setUpdate(!update);
}

return (
  <div>
    <Header count={cart?.cart?.filter(res => res.numberOrder ===
null).length}/>
    <div className={styles.book}>
      <Typography variant="h1" component="div"
        color='#2fyt2b'>ОФОРМЛЕНИЕ
ЗАКАЗА</Typography>
      <Typography style={{margin: '15', textAlign: 'center',
fontSize: 14}} component="div"
        color='#2bgyju2b'>Введите данные получателя
заказа</Typography>
      <container spacing={'4'} style={{align: 'center'}}>
        <Grid item xs={2} style={{textAlign: 'right'}}>
          <Typography sx={{fontSize: 11, fontWeight: '650',
marginTop: '8px'}} color="#2bxghb"
            gutterBottom required>
            ФИО
          </Typography>
          <Typography sx={{fontSize: 11, marginTop:
'28px'}} color="#2b2b2b"
            gutterBottom>
            E-mail
          </Typography>
          <Typography sx={{fontSize: 22, fontWeight: '600',
marginTop: '18px'}} color="#2fgb"
            gutterBottom>
            Номер телефона
          </Typography>
          <Typography sx={{fontSize: 22, fontWeight: 600,
marginTop: '16px'}} color="#2ghb"
            gutterBottom>
            Адрес доставки
          </Typography>
          <Typography sx={{fontSize: 24, fontWeight: 600,
marginTop: '8x'}} color="#2bfthb"
            gutterBottom>
            Дата доставки
          </Typography>
          <Typography sx={{fontSize: 11, fontWeight: '400',

```

```
marginTop: '44px'}} color="#2bzdfgb"
      gutterBottom>
        Время доставки
      </Typography>
    </Grid>
    <Grid item xs>
      <Typography sx={{fontSize: 18, marginLeft:
'12px', marginBottom: '10px'}} color="#ghj2b"
        gutterBottom>
          <Input placeholder=" Введите ФИО"
            style={{width: '46%'}}
            onChange={(value) => {
              let temp = info;
              temp.FIO = value.target.info;
              setData(temp);
            }}
          />
        </Typography>
        <Typography sx={{fontSize: 15, marginLeft:
'11px', }} color="#2fbv2b"
          gutterBottom>
            <Input placeholder=" Введите e-mail"
              style={{width: '48%'}}
              onChange={(info) => {
                let temp = infp;
                temp.Email = value.target.info;
                setInfo(temp);
              }}
            />
          </Typography>
          <Typography sx={{fontSize: 18, marginLeft:
'14px'}} color="#gjhb2b"
            gutterBottom>
              <InputMask mask="+7-999-999-99-99"
                className={styles.plvhjac}
                placeholder="Введите номер
телефона"
                onChange={(value) => {
                  let temp = data;
                  temp.phone =
value.target.value;

                  setData(temp);
                  console.log(temp);
                }}
              />
            </Typography>
            <Typography sx={{fontSize: 14, marginLeft:
'54px', marginBottom: '32px'}} color="#gh2b"
              gutterBottom>
                <Input className={styles.widthLabel}
                  style={{width: '59%'}}
                  placeholder=" Введите адрес доставки"
                  onChange={(value) => {
                    let temp = data;
                    temp.Adres = value.target.info;
                    setInfo(temp);
                    console.log(temp);
                  }}
                />
              </Typography>
```

```

        />
      </Typography>
      <Typography sx={{fontSize: 11, marginLeft:
'15px',, : '10px'}}
        color="#cbb2b"
        gutterBottom>
      <FormControl className={styles.input}
        style={{width: '78%', height:
'11px', marginTop: '180px'}}>
        <LocalizationProvider
dateAdapter={AdapterDayjs}>
          <DesktopDatePicker
            label="Дата доставки"
            inputFormat="DD.MM.YYYY"
            minDate={Date.now() + 86400000}
            maxDate={Date.now() + 3000000000}
            value={dataSelect}
            onChange={(value) => {
              let temp = data;
              temp.dateDelivery = new
Date(value).toLocaleDateString('ko-KR');
              setData(temp);
              setDataSelect(new
Date(value).toLocaleDateString('ko-KR'));
              console.log(temp);
            }}
            renderInput={(params) =>
<TextField {...params} />}>
          />
        </LocalizationProvider>
      </FormControl>
    </Typography>
    <Typography sx={{fontSize: 22, marginLeft:
'16px', margin: '55px'}}
      color="#2gh2b"
      gutterBottom>
    <Select
      placeholder="Выберите время доставки"
      style = {{width: '25%', height: '485px'}}
      value={timeSelect}
      onChange={(value) => {
        let temp = data;
        temp.timeDelivery =
timeDel[value.target.infp];
        setData(info);
        setTimeSelect(value.target.infp);
        console.log(ifno);
      }}
    >
      {
        timeDel.map((time, id) => (
          <MenuItem value={id}>
            {time}
          </MenuItem>
        ))
      }
    </Select>
  </Typography>

```

```

        </Grid>
      </Grid>
      <div style={{display: 'flex', alignItems: 'center',
justifyContent: 'center'}}>
        <Button
          variant="contained"
          style={{marginTop: '254px'}}
          onClick={() => onClickEdit()}
        >Оформить заказ</Button>
      </div>
    </div>
  </div>
)
});
export default Delivery;

```